IMPLEMENTING A DATABASE DESIGN USING MICROSOFT SQL SERVER
RATIONALE

Why This Module

In today’s computerized environment, there is usually a back-end application and a front-end application. The front-end application interacts with the backend application. A front-end application, developed using front-end tools, can be a stand-alone application residing on a desktop or it can be Web-based. End-users use the front-end applications to interact with backend applications. Data is typically stored at the back-end and usually managed by an RDBMS.

Some very popular RDBMSs are Microsoft SQL Server and Oracle server. This module covers the features of an RDBMS with specific reference to Microsoft SQL Server. This module is necessary for any student embarking on a career in Information Technology.
OBJECTIVES

This course focuses on implementing a database design using Microsoft SQL Server. It begins with an introduction to databases, and moves on to other features of SQL Server. Structured Query Language (SQL) that is used to retrieve and maintain data is covered in great detail. The course also deals with other features of SQL Server, such as constraints, rules, defaults, indexes, views, stored procedures, triggers, and transactions.
ENTRY PROFILE

A student should have knowledge on the following topics before starting with the SQL Server module:

- Build Flowcharts – the student should be able to represent logic, sequence tasks for execution, and implement conditional operations and iteration. This will be essential for programming using SQL Server.

- Work in a Windows environment – the student should be able to navigate in a Windows environment. SQL Server is Windows-based and hence, knowledge of the same is critical.

- Create and maintain documents – the student should be able to use an editor and should be able to perform simple editing tasks. SQL server scripts will be stored in text files and hence, this knowledge is essential.
EXIT PROFILE

After completing this module, the student should be able to:

Implement and design a database using various features of Microsoft SQL Server. Some of the features that the student will learn are:

- Querying Databases
- Retrieving Result Sets
- Functions and Joins
- Subqueries
- Creating Tables and Enforcing Data Integrity
- Maintaining Databases
- Implementing Indexes
- Implementing Views and Batches
- Implementing Stored Procedures
- Implementing Cursors
- Implementing Triggers and Transactions
- The Enterprise Manager and Replication Strategy
FAQs

Q. Why should we use SQL Server?

SQL Server is fast becoming the preferred RDBMS next to Oracle server. Oracle rules the RDBMS market as of now.

Q. What are the new features of SQL Server 2000?

- **XML Support**
  The relational database engine can return data as eXtensible Markup Language (XML) documents. Additionally, XML can also be used to insert, update, and delete values in the database.

- **User-Defined Functions**
  The programmability of Transact-SQL can be extended by creating your own Transact-SQL functions. A user-defined function can return either a scalar value or a table.

- **Indexed Views**
  Indexed views can significantly improve the performance of an application where queries frequently perform certain joins or aggregations. An indexed view allows indexes to be created on views, where the result set of the view is stored and indexed in the database. Existing applications do not need to be modified to take advantage of the performance improvements with indexed views.

- **New Data Types**
  SQL Server 2000 introduces three new data types. The bigint data type is an 8-byte integer type. The sql_variant data type allows the storage of data values of different data types. The table data type allows applications to store results temporarily for later use. It is supported for variables, and as the return type for user-defined functions.

- **INSTEAD OF and AFTER Triggers**
  INSTEAD OF triggers are executed instead of the triggering action (for example, INSERT, UPDATE, DELETE). They can also be defined on views, in which case they greatly extend the types of updates a view can support. AFTER triggers fire after the triggering action. SQL Server 2000 introduces the ability to specify which AFTER triggers fire first and last.

- **Full-Text Search Enhancements**
  Full-text search now includes change tracking and image filtering. Change tracking maintains a log of all changes to the full-text indexed data. You can update the full-text index with these changes by flushing the log manually, on a schedule, or as they occur, using the background update index option. Image filtering allows you to index and query documents stored in image columns. The user provides the document type in a column that contains the file name extension that the document would have had if it were stored as a file in the file system. Using this information, full-text search is able to load the appropriate document filter to extract textual information for indexing.

- **Multiple Instances of SQL Server**
  SQL Server 2000 supports running multiple instances of the relational database engine on the same computer. Each computer can run one instance of the relational database engine from SQL Server version 6.5 or 7.0, along with one or more instances of the database engine from SQL Server 2000. Each instance has its own set of system and user databases. Applications can connect to each instance on a computer similar to the way they connect to instances of SQL Servers running on different computers. The SQL...
Server 2000 utilities and administration tools have been enhanced to work with multiple instances.

- **Index Enhancements**

  You can now create indexes on computed columns. You can specify whether indexes are built in ascending or descending order, and if the database engine should use parallel scanning and sorting during index creation.

  The CREATE INDEX statement can now use the tempdb database as a work area for the sorts required to build an index. This results in improved disk read and write patterns for the index creation step, and makes it more likely that index pages will be allocated in contiguous strips. In addition, the complete process of creating an index is eligible for parallel operations, not only the initial table scan.

- **Net-Library Enhancements**

  The SQL Server 2000 Net-Libraries have been rewritten to virtually eliminate the need to administer Net-Library configurations on client computers when connecting SQL Server 2000 clients to instances of SQL Server 2000. The new Net-Libraries also support connections to multiple instances of SQL Server on the same computer, and support Secure Sockets Layer encryption over all Net-Libraries. SQL Server 2000 introduces Net-Library support for Virtual Interface Architecture (VIA) system-area networks that provide high-speed connectivity between servers, such as between application servers and database servers.

- **64-GB Memory Support**

  Microsoft SQL Server 2000 Enterprise Edition can use the Microsoft Windows 2000 Advanced Windows Extension (AWE) API to support up to 64 GB of physical memory (RAM) on a computer.

- **Distributed Query Enhancements**

  SQL Server 2000 introduces a new OPENDATASOURCE function, which you can use to specify ad hoc connection information in a distributed query. SQL Server 2000 also specifies methods that OLE DB providers can use to report the level of SQL syntax supported by the provider and statistics on the distribution of key values in the data source. The distributed query optimizer can then use this information to reduce the amount of data that has to be sent from the OLE DB data source. SQL Server 2000 delegates more SQL operations to OLE DB data sources than earlier versions of SQL Server. Distributed queries also support the other functions introduced in SQL Server 2000, such as multiple instances, mixing columns with different collations in result sets, and the new bigint and sql_variant data types.

- **Updateable Distributed Partitioned Views**

  SQL Server 2000 introduces enhancements to distributed partitioned views. You can partition tables horizontally across several servers, and define a distributed partitioned view on each member server that makes it appear as if a full copy of the original table is stored on each server. Groups of servers running SQL Server that cooperate in this type of partitioning are called federations of servers. A database federation built using SQL Server 2000 databases is capable of supporting the processing requirements of the largest Web sites or enterprise-level databases.

- **Backup and Restore Enhancements**

  SQL Server 2000 introduces a new, more easily understood model for specifying backup and restore options. The new model makes it clearer that you are balancing increased or decreased exposure to losing work against the performance and log space requirements of different plans. SQL Server 2000 introduces support for recovery to specific points of work using named log marks in the transaction log, and the ability to do partial database restores.

  Users can define passwords for backup sets and media sets that prevent unauthorized users from accessing SQL Server backups.
Scalability Enhancements for Utility Operations

SQL Server 2000 enhancements for utility operations include faster differential backups, parallel Database Console Command (DBCC) checking, and parallel scanning. Differential backups can now be completed in a time that is proportional to the amount of data changed since the last full backup. DBCC can be run without taking shared table locks while scanning tables, thereby, enabling them to be run concurrently with update activity on tables. Additionally, DBCC now takes advantage of multiple processors, thus enabling near-linear gain in performance in relation to the number of CPUs (provided that I/O is not a bottleneck).

Text in Row Data

SQL Server 2000 supports a new text in row table option that specifies that small text, ntext, and image values be placed directly in the data row instead of in a separate page. This reduces the amount of space used to store small text, ntext, and image data values, and reduces the amount of disk I/O needed to process these values.
Objectives

Following are the objectives for the Microsoft exam on Designing and Implementing Databases with Microsoft SQL Server 2000 Enterprise Edition.

**Developing a Logical Data Model**

- Specify entity attributes.
- Specify degree of normalization.
- Specify attributes that uniquely identify records.
- Specify attributes that reference other entities.
- Specify scale and precision of allowable values for each attribute.
- Allow or prohibit NULL for each attribute.
- Specify allowable values for each attribute.

**Implementing the Physical Databases**

- Specify space management parameters. Parameters include autoshrink, growth increment, initial size, and maxsize.
- Specify file group and file placement. Considerations include logical and physical file placement.
- Specify transaction log placement. Considerations include bulk load operations and performance.
- Specify table characteristics. Characteristics include cascading actions, CHECK constraints, clustered, defaults, FILLFACTOR, foreign keys, nonclustered, primary key, and UNIQUE constraints.
- Specify schema binding and encryption for stored procedures, triggers, user-defined functions, and views.
- Specify recompile settings for stored procedures.
- Specify index characteristics. Characteristics include clustered, FILLFACTOR, nonclustered, and uniqueness.
- Support merge, snapshot, and transactional replication models.
- Design a partitioning strategy.
- Design and create constraints and views.
- Resolve replication conflicts.

**Retrieving and Modifying Data**

- Import and export data. Methods include the bulk copy program, the Bulk Insert task, and Data Transformation Services (DTS).
- Manipulate heterogeneous data. Methods include linked servers, OPENQUERY, OPENROWSET, and OPENXML.
- Retrieve, filter, group, summarize, and modify data by using Transact-SQL.
- Manage result sets by using cursors and Transact-SQL. Considerations include locking models and appropriate usage.
Extract data in XML format. Considerations include output format and XML schema structure.

**Programming Business Logic**
- Implement error handling in stored procedures, transactions, triggers, and user-defined functions.
- Pass and return parameters to and from stored procedures and user-defined functions.
- Validate data.
- Specify trigger actions.
- Design and manage transactions.
- Manage control of flow.
- Filter data by using stored procedures, triggers, user-defined functions, and views.

**Tuning and Optimizing Data Access**
- Analyze the query execution plan. Considerations include query processor operations and steps.
- Capture, analyze, and replay SQL Profiler traces. Considerations include lock detection, performance tuning, and trace flags.
- Create and implement indexing strategies. Considerations include clustered index, covering index, indexed views, nonclustered index, placement, and statistics.
- Improve index use by using the Index Tuning Wizard.
- Monitor and troubleshoot database activity by using SQL Profiler.

**Designing a Database Security Plan**
- Apply ownership chains.
- Use programming logic and objects. Considerations include implementing row-level security and restricting direct access to tables.

**Guidelines for Preparation**
- The first step towards preparing for the exam would be to solve all the problems whether Demos, Guided, or Unguided practices in the Student Guide.
- Refer to the SQL Server Books Online.
The Database Schemas

The tables in the schemas for Tebisco and GlobalToyz Universe were decided by identifying the entities from the respective case studies. The entities were identified by typically picking out the nouns from the case studies.

On reading the case study you may find that some tables in the schema may not match the case study exactly. Few extra tables have been added, few tables have been normalized, and few tables have been denormalized. This was done in order to be able to cover all required concepts and features of SQL Server in the course.
**Installation note - SQL Database Scripts**

To conduct the SQL 2000 course, the following databases need to be installed:

- Recruitment
- GlobalToyz

Note: Before you run these scripts, you need to edit the scripts and change the server name to the name of the machine on which SQL Server is installed. You should back up these databases so that they can be restored later.

A brief description of the various scripts and their content is given below.

<table>
<thead>
<tr>
<th>Scripts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateServerLogins.bat</td>
<td>This script creates 30 SQL Server logins for students. You need to edit the script and change the server name to the name of the machine on which SQL Server is installed. Running this script is a one-time activity.</td>
</tr>
<tr>
<td>InstallRecruitmentDB.bat</td>
<td>This script creates the Recruitment database and its objects. You must back up the database after creating it.</td>
</tr>
<tr>
<td>InstallGlobalToyzDB.bat</td>
<td>This script creates the GlobalToyz database and its objects. You must back up the database after creating it.</td>
</tr>
<tr>
<td>InstallRecruitmentDBForLesson5.bat</td>
<td>This script is to be used for Lesson 5.</td>
</tr>
<tr>
<td>INSTALLRECRUITMENTDBWITHOUTTCONSTRAINT.BAT</td>
<td>This script creates the Recruitment database without any constraints. You need to execute this script for demonstrations and practices of Lesson 6, 7, 8, 9, 10, 11, and 12.</td>
</tr>
<tr>
<td>INSTALLGLOBALTOYZDBWITHOUTCONSTRAINTS.BAT</td>
<td>This script creates the GlobalToyz database without any constraints. You need to execute this script for unguided practice 4, 5, 6, 7, and 8</td>
</tr>
</tbody>
</table>

Before every OCR and Unguided Practice session, the faculty needs to drop the database and restore the backed up copy using the Enterprise Manager of Microsoft SQL Server.

**Note:** You can also reinstall the database but it takes less time to restore a database than to install it.
Frequently Asked Questions About Scripts

Q) Which script needs to be executed first?
Answer) You need to execute the CREATE_SQLSERVERLOGINS.BAT before executing any other script. The contents of the script are:

```bash
osql -Usa -Psasa -Svandanak-f90 -iCreateSQLServerLogins.sql
```

You need to modify the script to be able to execute it on your machine. For example, if the name of the server on which SQL Server is installed is NIIT-RG-MR and the sa login does not have any password, you need to change the command to:

```bash
osql -Usa -P -SNIIT-RG-MR -iCreateSQLServerLogins.sql
```

Save the changes and execute the batch file. This batch-file would call the CreateSQLServerLogins script, which in turn would use the sp_addlogin username, password command to add logins to SQL Server. It would create 30 users with the names user1, user2, user3 ... user30. It would also create a faculty login. You should ask students to use one of these logins to connect to the SQL Server. Do not let them use the faculty or the sa login.

This script needs to be executed only once after the installation of the SQL server since these logins are used to connect to the SQL Server.

Q) How should I create the Recruitment database?
Answer) To create the Recruitment database, you can use the following scripts:

INSTALLRECRUITMENTDB.BAT. Its contents are:

```bash
osql -Usa -Psasa -SVANDANAK-f90 -iCreateRecruitmentDb.sql
osql -Usa -Psasa -SVANDANAK-f90 -iCreateUsersForRecruitmentDb.sql
for %%i in (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30) do osql -Uuser%%i -P -SVANDANAK-f90 -iCreateRecruitmentObjects.sql
osql -Ufaculty -P -SVANDANAK-f90 -iCreateRecruitmentObjects.sql
```

You need to modify the script to be able to execute the script. For example, If the name of your server is NIIT-RG-MR and the sa login does not have any password, you need to modify the script to:

```bash
osql -Usa -P -SNIIT-RG-MR -iCreateRecruitmentDb.sql
osql -Usa -P -SNIIT-RG-MR -iCreateUsersForRecruitmentDb.sql
for %%i in (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30) do osql -Uuser%%i -P -SNIIT-RG-MR -iCreateRecruitmentObjects.sql
osql -Ufaculty -P -SNIIT-RG-MR -iCreateRecruitmentObjects.sql
```
In case sa login of NIIT-RG-MR has a password say secret, you need to modify the script as follows:

```
  osql -Usa -Psecret -SNIIT-RG-MR -iCreateRecruitmentDb.sql
  osql -Usa -Psecret -SNIIT-RG-MR -iCreateUsersForRecruitmentDb.sql
  for %%i in (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30) do
     osql -Uuser%%i -P -SNIIT-RG-MR -iCreateRecruitmentObjects.sql
     osql -Ufaculty -P -SNIIT-RG-MR -iCreateRecruitmentObjects.sql
```

Important: Even though sa has the password as secret, the last two lines do not have any passwords. They just contain -P. This is because these would create the schema for all the users individually.

---

**Q)** What is the purpose and sequence of executing these scripts?

**Answer)** If you need to install databases with constraints, you should execute these scripts in the following sequence:

1) **CREATESQLSERVERLOGINS.BAT**
2) **INSTALLRECRUITMENTDB.BAT**
3) **INSTALLGLOBALTOYZDB.BAT**

In case you are working on Lesson 6 onwards, you need to execute the following scripts after having executed the **CREATESQLSERVERLOGINS.BAT** once:

4) **INSTALLRECRUITMENTDBWITHOUTCONSTRAINT.BAT**
5) **INSTALLGLOBALTOYZDBWITHOUTCONSTRAINTS.BAT**

---

**Q)** Why should I execute the batch file and not directly execute the sql scripts?

**Answer)** The batch file uses a for loop to create 30 copies of the same tables. If you execute the script by loading it in the Query analyzer, you would just get one copy of the tables. Thirty copies are required as all the users should have their own copy of the tables. In case you don't create 30 copies, if one person drops a table, the other person would not be able to use that table.

---

**Q)** How do I execute the .BAT files?

**Answer)** You can double-click the batch files to execute them. You can also access the command prompt, type the name of a batch file on the command prompt, and then press enter to execute it.

---

**Q)** What is the difference between the three batch files?

**INSTALLRECRUITMENTDB.BAT**

**INSTALLRECRUITMENTDBFORLESSON5.BAT**

**INSTALLRECRUITMENTDBWITHOUTCONSTRAINT.BAT**

**Answer)** **INSTALLRECRUITMENTDB.BAT**: would create the Recruitment database with constraints. All the tables will be created with primary key and foreign key relations.

**INSTALLRECRUITMENTDBFORLESSON5.BAT**: would drop the existing recruitment database and recreate it without any objects. This is required because we are creating tables in Lesson 5. If the tables already exist, an error would be generated.

**INSTALLRECRUITMENTDBWITHOUTCONSTRAINT.BAT**: would drop the existing recruitment database and recreate it without any constraints. All the tables will be created but they would not have primary key and foreign key relations. Any table can be dropped. Also, you can create indexes and constraints on the existing tables.
Q) What is the difference between INSTALLGLOBALTOYZDB.BAT and INSTALLGLOBALTOYZDBWITHOUTCONSTRAINTS.BAT files?
Answer) INSTALLGLOBALTOYZDB.BAT: would create the GlobalToyz database with constrains. You would not be able to drop any table as primary key and foreign key relations would exist.
INSTALLGLOBALTOYZDBWITHOUTCONSTRAINTS.BAT: would create the GlobalToyz database without any constrains. You can drop any table if the database is created using this batch file.

Q) What is the use of the following SQL scripts:
CREATEUSERSFORRECRUITMENTDB.SQL
CREATERECRUITMENTOBJECTS.SQL
CREATERECRUITMENTOBJECTSWITHOUTCONSTRAINT.SQL
CREATERECRUITMENTDB.SQL
Answer) CREATEUSERSFORRECRUITMENTDB.SQL: Assigns permissions to users to use the database.
CREATERECRUITMENTOBJECTS.SQL: Creates the Recruitment database objects and adds records in the database. The tables created contain constraints.
CREATERECRUITMENTOBJECTSWITHOUTCONSTRAINT.SQL: Creates Recruitment database objects and records in the database. The tables created do not contain constraints.
CREATERECRUITMENTDB.SQL: Creates a blank Recruitment database. It does not create any database objects.

Q) What is the use of the following SQL scripts:
CREATEUSERSFORGLOBALTOYZDB.SQL
CREATEGLOBALTOYZOBJECTSWITHOUTCONSTRAINTS.SQL
CREATEGLOBALTOYZOBJECTS.SQL
CREATEGLOBALTOYZDB.SQL
Answer) CREATEUSERSFORGLOBALTOYZDB.SQL: Assigns permission to users to use the GlobalToyz database.
CREATEGLOBALTOYZOBJECTSWITHOUTCONSTRAINTS.SQL: Creates the objects in GlobalToyz database without any constraints. Tables in the database can be dropped.
CREATEGLOBALTOYZOBJECTS.SQL: Creates the objects in GlobalToyz database with constraints. Tables in the database cannot be dropped.
CREATEGLOBALTOYZDB.SQL: Creates a blank GlobalToyz database after dropping any existing GlobalToyz database.

Q) What is present in the SQL files?
Answer) The SQL files contain SQL statements to create the Recruitment and the GlobalToyz database.

Q) On which machine should you execute these scripts?
Answer) You should execute these scripts on the machine where the SQL server is installed.
Q) Do I have to execute the script on all the machines?
Answer) No, you just need to install the script on one machine and other persons can use the Query Analyzer to connect to the SQL server and use the database. You need to install SQL Server on one machine and the Client components on rest of the machines. Only in case you have problems in networking and are not able to connect to the SQL server and the machine configuration is such that you can install SQL Server on all the machines should you execute the script on all the machines.

Q) How should I back up and restore the database?
Answer) To back up the database:
1) Invoke Enterprise Manager.
2) Right-Click the database to be backed up.
3) Select All Tasks.
4) Select the backup option.
The following figure would be displayed:

5) Click the Add button.
6) Enter the name of the backup file.
7) Click OK to start the backup.

To restore the data from the backup:
1) Delete the existing database.
2) Create a database with the same name that existed on the server previously.
3) Right-Click the database to be restored.
4) Select All Tasks.
5) Select the restore database option.

Following dialog box would be displayed:

![Restore database dialog box]

6) Click the From device option.
7) Click the Select Devices BUTTON to open the Choose Restore Device dialog box.
8) In the dialog box, select the name of the file that you used while creating the backup.
9) Click OK to return to the Restore database dialog box.
10) Click OK to start restoring the database.

Q) How do I change the password of sa or faculty login?

Answer) You can change the password of any user using the sp_password command. The syntax of the command is:

```
sp_password old_password, new_password, login_name
```

Example

```
sp_password null,'newpassword','faculty'
```

Q) If one user, say user1, drops a table, say Employee, then can another user, say user2, drop the same table?
Answer) Yes, if you execute the batch files in the proper sequence. This is possible as each table is referenced by servername.databasename.username.tablename. For Example, if you have a table called Employee in the Recruitment database on the outlookserver created by user1, you can reference it as outlookserver.recruitment.user1.employee. When you execute the batch file, 30 copies of the schema are created. Each user has his/her own copy.

Q) How do I to give a demonstration of locking? Do I need two machines to demonstrate locking?
Answer) No, you do not need two machines to give a demonstration of locking. You can open two connections to the SQL server on the same machine and then you can resize the two windows so that both are visible. Ask the students to assume that the first window is one computer and the second window is another computer.

Q) Can students use the Enterprise Manager without using the sa login?
Answer) Yes, students should use logins user1, user2, user3... etc. to connect to the SQL server from the Enterprise Manager. To change a user name while connecting to the server, right click the SQL server and click the Edit SQL Server Properties option. In the dialog box, enter the new user name with which you want to connect to the SQL server.

Q) What if the tables in a database are deleted by a student?
Answer) In case the tables are deleted, you can restore them from the backup or execute the scripts to recreate the database.

Q) What should I do if the table that needs to be created is already present in the database?
Answer) In case you have executed CREATERECRUITMENTDB.SQL or CREATEGLOBALTOYZDB.SQL, you would not be able to drop any tables without removing the constraints. In case you have executed CREATEGLOBALTOYOBJECTSWITHOUTCONSTRAINTS.SQL or CREATERECRUITMENTOBJECTSWITHOUTCONSTRAINT.SQL you can drop any table as the database does not contain any constraints.
## When to Execute Which Script

<table>
<thead>
<tr>
<th>Lesson Number</th>
<th>Script Name</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lesson 1, 2, 3, and 4</td>
<td>INSTALLRECRUITMENTDB.BAT</td>
<td>It would create the tables along with all the constraints. You would not be able to delete any table unless the relationships are deleted. You should use this script for these lessons as these lessons are related to queries and all the tables must exist for the queries to be executed.</td>
</tr>
<tr>
<td>Lesson 5</td>
<td>INSTALLRECRUITMENTDBFORLESSON5.BAT</td>
<td>This script would create a blank database called Recruitment. All the tables would be deleted from the database. This is required in lesson 5 as in this lesson you need to create tables. If the tables are already existing, you would not be able to create tables with the same name.</td>
</tr>
<tr>
<td>Lesson 6, 7, 8, 9, 10, 11, and 12</td>
<td>INSTALLRECRUITMENTDBWITHOUTCONSTRAINT.BAT</td>
<td>When you execute this script all the tables would get created but without any constraints. They would not have any primary key - foreign key relationship. This is done so that you can alter the structure of the tables. In case you are creating a table that already exists in the database, you need to delete it.</td>
</tr>
<tr>
<td>Unguided practice 1, 2 and 3</td>
<td>INSTALLGLOBALTOYZDB.BAT</td>
<td>When you execute this script, the GlobalToyz database would be created with all the constraints. In this case, you would not be able to delete any table without deleting the constraints first.</td>
</tr>
<tr>
<td>Unguided practice 4, 5, 6, 7, 8 and 9</td>
<td>INSTALLGLOBALTOYZDBWITHOUTCONSTRAINTS.BAT</td>
<td>When you execute this script, the GlobalToyz database would be created without any constraints. As the database would not have any constraints, you can drop any table at any time. You would also be able to create new constraints on the existing tables.</td>
</tr>
</tbody>
</table>
Lesson One

Experiences

**Simple Queries**

Stress on the fact that if the student wants to visualize the output required, it would be easier to create the query. Also, ask them to compare the output with the required results.

Stress on the fact that when specific columns are mentioned in the select list, then the output is nothing but a column subset. Similarly, when the WHERE clause is used, the output is a row subset.

Stress on writing the SQL statement in separate lines in the Query Analyzer window. It becomes easy to read and debug.

**Customized Column Headings**

Stress on the fact that customized column headings are used for better readability of the query output.

**Multiple Conditions**

Mention that multiple conditions can be used to retrieve data from a table, and you can do this by using the logical operators.

Cover an example where the AND and OR operators are used in a single query.

**FAQ**

Q: What is the difference between T-SQL and PL-SQL?

Transact-SQL (T-SQL) is a scripting language used in SQL Server 2000 for programming. T-SQL confirms to the ANSI SQL-92 standard published by American National Standards Institute (ANSI) and International Organization for Standardization (ISO) in the year 1992. PL-SQL on the other hand is used for programming in Oracle.

**Additional Inputs**

The different types of DML statements are SELECT, INSERT, UPDATE, and DELETE.

Operator precedence levels are as follows: for arithmetic operators, the precedence is ‘*’, ‘/', ‘-’, ‘%’, ‘+’. For logical operators, it is NOT, AND, and OR. As it is obvious, the precedence levels can be changed by the use of ‘( )’.

**Solutions: Just a Minute...**
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | Question: In Tebisco, there is a review by senior management on manpower. You need to write a query to display a list of the budgeted and the actual people available for various roles from the Recruitment database.  
Answer:  
Use GlobalToyz  
SELECT vDescription, iBudgetedStrength, iCurrentStrength  
FROM Recruitment |
| 2              | Question: The hiring charges of contract recruiters (charging more than 8 percent of the annual salary as their percentage) are under review. The names, addresses, and the hiring charges of these contract recruiters are required. Write a query to display the required information.  
Answer:  
SELECT cName, vAddress, siPercentageCharge  
FROM ContractRecruiter  
WHERE siPercentageCharge > 8 |
| 3              | In order to identify the successful candidates, who took the test between March 5, 2001 and March 12, 2001, a report displaying the names of candidates and their scores needs to be created. The format of the report is as follows:  

<table>
<thead>
<tr>
<th>vFirstName</th>
<th>vLastName</th>
<th>siTestScore</th>
<th>dTestDate</th>
</tr>
</thead>
</table>

SELECT  
vFirstName, vLastName, siTestScore, dTestDate  
FROM ExternalCandidate  
WHERE dTestDate between '03/05/01' and '03/12/01' |

---

**Lesson Two**

**Experiences**

**NULL and NOT NULL**

Mention the fact that while comparing a column for NULL or NOT NULL, the usual = operator cannot be used. You need to use the IS operator instead.

**ORDER BY Clause**
Tell the students that when the ORDER BY clause is used, only the output of the SELECT statement is sorted, while the data in the table remains as it is. Also, do not forget to mention that the default sort order is ASC.

**Aggregate Functions**

Tell the students that aggregate functions are used to count or to find out the average of a particular column, do not forget to tell the students that these should only be used on numeric columns. Besides, when an aggregate function is used on a column, NULL values are not considered.

**GROUP BY Clause**

Tell the students that the GROUP BY clause is used to group the output of the SELECT statement in several groups. If ALL is used, then it ignores the restriction provided by the WHERE clause. Just like the WHERE clause, you can use the HAVING clause with the GROUP BY clause.

**Examples**

Example of the usage of the HAVING clause:
The following code displays the titles and types of all titles whose type is either business or mod_cook grouped by type from the titles table:

```sql
select type, title from titles
group by type, title
having type = 'business'
or type = 'mod_cook'
```

**TOP**

Stress that the TOP keyword is used to retrieve the top few records, as they exist in the table. If you require the top few records with respect to a sort order of a particular column, then you need to include the ORDER BY…DESC keywords in the SELECT statement.

**COMPUTE BY clause**

Stress that the COMPUTE BY clause is used to generate totals and subtotals in a control break report.

**Additional Inputs**

Image, text, and ntext columns cannot be used in an ORDER BY clause.

When using DISTINCT, only the name of a column can be used, an arithmetic expression gives an error.

MAX and MIN functions cannot be used on the bit data type columns.

All columns mentioned in the GROUP BY clause have to be included in the SELECT list. If a WHERE clause is present, then the GROUP BY clause groups only those rows which are satisfied by the conditions used in the WHERE clause.

The TOP keyword displays records from the table as they exist in the table. Use the ORDER BY clause to display the top few records based on the values of a particular column.

With the TOP keyword you can use the WITH TIES keywords to display rows having equal values. This cannot be used without the ORDER BY clause.

The columns that are mentioned in the COMPUTE clause need to be part of the SELECT list. Use an ORDER BY clause with the COMPUTE BY clause so that rows are grouped together.

**Example**
**CUBE and ROLLUP example:**

```sql
SELECT pub_id, job_id, 'Avg' = AVG(job_lvl)
FROM employee
GROUP BY pub_id, job_id with cube
```

**Output of the above code:**

<table>
<thead>
<tr>
<th>Pub_id</th>
<th>job_id</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0736</td>
<td>5</td>
<td>175</td>
</tr>
<tr>
<td>0736</td>
<td>6</td>
<td>220</td>
</tr>
<tr>
<td>0736</td>
<td>7</td>
<td>195</td>
</tr>
<tr>
<td>0736</td>
<td>8</td>
<td>175</td>
</tr>
<tr>
<td>0736</td>
<td>9</td>
<td>170</td>
</tr>
<tr>
<td>0736</td>
<td>10</td>
<td>165</td>
</tr>
<tr>
<td>0736</td>
<td>11</td>
<td>150</td>
</tr>
<tr>
<td>0736</td>
<td>12</td>
<td>100</td>
</tr>
<tr>
<td>0736</td>
<td>13</td>
<td>100</td>
</tr>
<tr>
<td>0736</td>
<td>14</td>
<td>100</td>
</tr>
<tr>
<td>0736</td>
<td>NULL</td>
<td>155</td>
</tr>
<tr>
<td>0877</td>
<td>5</td>
<td>159</td>
</tr>
<tr>
<td>0877</td>
<td>6</td>
<td>140</td>
</tr>
<tr>
<td>0877</td>
<td>7</td>
<td>120</td>
</tr>
<tr>
<td>0877</td>
<td>8</td>
<td>100</td>
</tr>
<tr>
<td>0877</td>
<td>9</td>
<td>75</td>
</tr>
<tr>
<td>0877</td>
<td>10</td>
<td>75</td>
</tr>
<tr>
<td>0877</td>
<td>11</td>
<td>75</td>
</tr>
<tr>
<td>0877</td>
<td>12</td>
<td>35</td>
</tr>
<tr>
<td>0877</td>
<td>13</td>
<td>35</td>
</tr>
<tr>
<td>0877</td>
<td>14</td>
<td>35</td>
</tr>
<tr>
<td>0877</td>
<td>NULL</td>
<td>84</td>
</tr>
<tr>
<td>1389</td>
<td>5</td>
<td>150</td>
</tr>
<tr>
<td>1389</td>
<td>6</td>
<td>192</td>
</tr>
<tr>
<td>1389</td>
<td>7</td>
<td>135</td>
</tr>
<tr>
<td>1389</td>
<td>8</td>
<td>125</td>
</tr>
<tr>
<td>1389</td>
<td>9</td>
<td>78</td>
</tr>
<tr>
<td>1389</td>
<td>10</td>
<td>87</td>
</tr>
<tr>
<td>1389</td>
<td>11</td>
<td>112</td>
</tr>
<tr>
<td>1389</td>
<td>12</td>
<td>32</td>
</tr>
<tr>
<td>1389</td>
<td>13</td>
<td>64</td>
</tr>
<tr>
<td>1389</td>
<td>14</td>
<td>89</td>
</tr>
<tr>
<td>1389</td>
<td>NULL</td>
<td>106</td>
</tr>
<tr>
<td>1622</td>
<td>5</td>
<td>198</td>
</tr>
</tbody>
</table>
Now let use the same code ROLLUP:

```sql
SELECT pub_id, job_id, 'Avg' = AVG(job_lvl)
FROM employee
GROUP BY pub_id, job_id with rollup
```

And the output is:

```
Pub_id job_id  Avg
0736  5  175
0736  6  220
0736  7  195
```
<table>
<thead>
<tr>
<th>Code</th>
<th>Value1</th>
<th>Value2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0736</td>
<td>8</td>
<td>175</td>
</tr>
<tr>
<td>0736</td>
<td>9</td>
<td>170</td>
</tr>
<tr>
<td>0736</td>
<td>10</td>
<td>165</td>
</tr>
<tr>
<td>0736</td>
<td>11</td>
<td>150</td>
</tr>
<tr>
<td>0736</td>
<td>12</td>
<td>100</td>
</tr>
<tr>
<td>0736</td>
<td>13</td>
<td>100</td>
</tr>
<tr>
<td>0736</td>
<td>14</td>
<td>100</td>
</tr>
<tr>
<td>0736</td>
<td>NULL</td>
<td>155</td>
</tr>
<tr>
<td>0877</td>
<td>5</td>
<td>159</td>
</tr>
<tr>
<td>0877</td>
<td>6</td>
<td>140</td>
</tr>
<tr>
<td>0877</td>
<td>7</td>
<td>120</td>
</tr>
<tr>
<td>0877</td>
<td>8</td>
<td>100</td>
</tr>
<tr>
<td>0877</td>
<td>9</td>
<td>75</td>
</tr>
<tr>
<td>0877</td>
<td>10</td>
<td>75</td>
</tr>
<tr>
<td>0877</td>
<td>11</td>
<td>75</td>
</tr>
<tr>
<td>0877</td>
<td>12</td>
<td>35</td>
</tr>
<tr>
<td>0877</td>
<td>13</td>
<td>35</td>
</tr>
<tr>
<td>0877</td>
<td>14</td>
<td>35</td>
</tr>
<tr>
<td>0877</td>
<td>NULL</td>
<td>84</td>
</tr>
<tr>
<td>1389</td>
<td>5</td>
<td>150</td>
</tr>
<tr>
<td>1389</td>
<td>6</td>
<td>192</td>
</tr>
<tr>
<td>1389</td>
<td>7</td>
<td>135</td>
</tr>
<tr>
<td>1389</td>
<td>8</td>
<td>125</td>
</tr>
<tr>
<td>1389</td>
<td>9</td>
<td>78</td>
</tr>
<tr>
<td>1389</td>
<td>10</td>
<td>87</td>
</tr>
<tr>
<td>1389</td>
<td>11</td>
<td>112</td>
</tr>
<tr>
<td>1389</td>
<td>12</td>
<td>32</td>
</tr>
<tr>
<td>1389</td>
<td>13</td>
<td>64</td>
</tr>
<tr>
<td>1389</td>
<td>14</td>
<td>89</td>
</tr>
<tr>
<td>1389</td>
<td>NULL</td>
<td>106</td>
</tr>
<tr>
<td>1622</td>
<td>5</td>
<td>198</td>
</tr>
<tr>
<td>1622</td>
<td>NULL</td>
<td>198</td>
</tr>
<tr>
<td>1756</td>
<td>5</td>
<td>246</td>
</tr>
<tr>
<td>1756</td>
<td>NULL</td>
<td>246</td>
</tr>
<tr>
<td>9901</td>
<td>5</td>
<td>172</td>
</tr>
<tr>
<td>9901</td>
<td>NULL</td>
<td>172</td>
</tr>
<tr>
<td>9952</td>
<td>2</td>
<td>215</td>
</tr>
<tr>
<td>9952</td>
<td>3</td>
<td>200</td>
</tr>
<tr>
<td>9952</td>
<td>4</td>
<td>227</td>
</tr>
<tr>
<td>9952</td>
<td>NULL</td>
<td>214</td>
</tr>
<tr>
<td>Problem number</td>
<td>Solution</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
<td></td>
</tr>
</tbody>
</table>
| 1              | Question: Write a query to display the details of all the contract recruiters whose names begin with “J”.
Answer: 
Select * from ContractRecruiter
WHERE cName LIKE 'J%'
| 2              | Question: Write a query that displays a list of cities from where applications of external candidates have been received.
Answer: 
SELECT DISTINCT cCity
FROM ExternalCandidate
| 3              | Question: A list of external candidates who took a test, along with their test scores, is required. The average of the test scores needs to be printed at the bottom of the list.
Answer: 
SELECT vFirstName, siTestScore
FROM ExternalCandidate
COMPUTE AVG (siTestScore)

---

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | --Implement 1
Open Query Analyzer to connect to SQL Server |
| 2              | --Implement 2
SELECT * FROM Toys |
| 3              | --Implement 2
SELECT 'Recipient First Name' = vFirstName, 'Recipient Last Name' = vLastName, 'Address' = |
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.28</td>
<td>vAddress, 'Zip' = cZipCode FROM Recipient</td>
</tr>
</tbody>
</table>
| 4              | --Implement 4
|                | SELECT vFirstName, vLastName
|                | FROM Shopper
|                | WHERE cState='California' |
| 5              | --Implement 5
|                | SELECT * FROM Orders
|                | WHERE mTotalCost > 75 |
| 6              | --Implement 6
|                | SELECT * FROM Toys
|                | WHERE cBrandId = '004' |
| 7              | --Implement 7
|                | SELECT * FROM ORDERS
|                | WHERE dOrderDate = '05/22/01' |
| 8              | --Implement 8
|                | SELECT 'Order Number' = cOrderNo, 'Shipping Charges' = mShippingCharges, 'Gift Wrap Charges' = mGiftWrapCharges, 'Handling Charges' = mShippingCharges + mGiftWrapCharges
|                | FROM Orders |
| 9              | --Implement 9
|                | SELECT * from toys
|                | WHERE mtoyrate > 10 and mtoyrate < 20 |
| 10             | --Implement 10
|                | SELECT vFirstName, vLastName, vEmailId
|                | FROM Shopper
|                | WHERE cState = 'California' or cState = 'Illinois' |
| 11             | --Implement 11
|                | SELECT 'Order Number' = cOrderNo, 'Order Date' = dOrderDate, 'Shopper Id' = cShopperId, 'Total Cost' = mTotalCost
|                | FROM Orders
|                | WHERE dOrderDate = '2001/05/20' and mTotalCost > 75 |
| 12             | --implement 12
|                | select * from orderdetail
<p>|                | where vmmessage is NULL |</p>
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 13             | --Implement 13
SELECT 'Toy Name'=vToyName, 'Toy Rate'=mToyRate
FROM Toys
Order by mToyRate desc |
| 14             | --Implement 14
SELECT vToyName,siLowerAge From Toys WHERE
mToyRate<20
Order by mToyRate asc |
| 15             | --Implement 15
SELECT cOrderNo, cShopperID, mTotalCost
FROM Orders
Order by mTotalCost asc |
| 16             | --Implement 16
SELECT COUNT(*)
FROM Toys |
| 17             | --Implement 17
SELECT 'Maximum'=MAX (mToyRate), 'Minimum'=MIN (mToyRate), 'Average'=AVG (mToyRate)FROM Toys |
| 18             | --Implement 18
SELECT 'Total'=SUM (mTotalCost)
FROM Orders |
| 19             | -- Implement 19
SELECT TOP 5 cToyId, iTotalSold
FROM PickOfMonth
WHERE iYear = 2000
Order by iTotalSold desc |
| 20             | -Implement 20
SELECT * FROM toys
WHERE vToyName like '%$Racer%'
| 21             | --Implement 21
SELECT * FROM Shopper
WHERE vFirstName like 'S%'
| 22             | --Implement 22
SELECT 'Order Number' = cOrderNo, 'Toy Id' = cToyId, 'Wrapper Description'=vDescription
FROM OrderDetail JOIN Wrapper
ON OrderDetail.cWrapperId = Wrapper.cWrapperId |
| 23             | --Implement 23
The code returns an error, as cToyId is not part
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>of the GROUP BY clause.</td>
<td></td>
</tr>
<tr>
<td>The correct code for displaying the total value for each order is:</td>
<td></td>
</tr>
<tr>
<td>SELECT cOrderNo, SUM(mToyCost) FROM OrderDetail GROUP BY cOrderNo</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>--Implement 24</td>
</tr>
<tr>
<td>SELECT 'Order Number' = cOrderNo, 'Total Cost of Toy for an Order' = sum (mToyCost) from orderdetail group by cOrderNo having sum(mToyCost) &gt; 50</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>--Implement 25</td>
</tr>
<tr>
<td>The code returns an error, as cToyId is not ordered using the ORDER BY clause.</td>
<td></td>
</tr>
<tr>
<td>The correct code for displaying the total value for each order is:</td>
<td></td>
</tr>
<tr>
<td>SELECT cToyId,mToyCost FROM OrderDetail ORDER BY cToyId COMPUTE SUM(mToyCost) BY cToyId</td>
<td></td>
</tr>
<tr>
<td>The code returns an error, as mTotalCost, the column on which the aggregate function is used should be part of the select list.</td>
<td></td>
</tr>
<tr>
<td>The correct code for displaying the total value for each order is:</td>
<td></td>
</tr>
<tr>
<td>SELECT cCartId,mTotalCost FROM Orders ORDER BY cCartId COMPUTE AVG(mTotalCost) BY cCartId</td>
<td></td>
</tr>
</tbody>
</table>

Lesson Three

Experiences

**String Functions**

Mention that the string functions are used to manipulate data of character type columns.

**Date Functions**

Tell the students that the date manipulation is done using the date functions.

**Mathematical Functions**

Mathematical calculations and manipulation can be done using the mathematical functions.

**Wildcard Characters**

Mention that if a particular character pattern is being searched for, the wildcard characters can be used along with the LIKE keyword.
Joins

Tell the students that data from multiple tables can be retrieved using a join.

The most important thing is to use ANSI joins in the class and not T-SQL joins.

Mention the difference between the OUTER and the INNER joins. Also, mention the difference between the LEFT OUTER and RIGHT OUTER joins.

Inner Joins

The INNER JOIN returns all rows from both tables where there is a match. If there are rows in Supplier that do not have matches in Orders, those rows will not be listed. It is the most common type of join.

For Example,

```
SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers, orders
WHERE suppliers.supplier_id = orders.supplier_id;
```

The INNER JOIN operation can be used in any FROM clause to combine records from two tables. There must be a matching value in a field common to both tables.

An INNER JOIN cannot be nested inside a LEFT JOIN or RIGHT JOIN.

Equi Joins

Conceptually equijoin is same as inner join and is used to list all the columns from the joining tables. The equi join tables join tables across a foreign key link.

For Example,

```
SELECT * FROM authors A, titleauthor B WHERE A.au_id = B.au_id
```

Output:

```
au_id  au_first au_last  au_init  au_phone ao_adr1   ao_city  ao_state  ao_zip  au_ord
117  53-117  White   Johnathan  403-492-7392  1596 Biggie Rd  St. Louis  CA  94052  1  323-321-1776  94033  01  103
215  46-6995 Green  Margaret  415-986-7100  109 Eglis St, #111  Oakland  CA  94619  213-46-6995  986-0075  1  103
213  46-6995 Green  Margaret  415-986-7100  109 Eglis St, #111  Oakland  CA  94619  213-46-6995  986-0075  1  103
326  96-7796 Carson  Cheryl  415-549-2752  189 Damans Ln  Berkeley  CA  94705  289-79-7756  502-0155  1  103
407  41-2584 Olney  Michael  415-238-2445  22 Cleveland Ave, #1  San Jose  CA  95130  437-41-2584  621-1111  2  43
326  41-2584 Olney  Michael  415-238-2445  22 Cleveland Ave, #1  San Jose  CA  95130  437-41-2584  621-1111  2  43
194  40-5391 Straight  Dean  415-534-3419  1440 College Av  Oakland  CA  94609  1494-50-5001  307-6122  1  103
401  56-3208 Simms  Abraham  415-655-9095  2223 Palmer St  Berkeley  CA  94705  405-65-3208  954-1935  1  53
427  17-2313 Dull  Amy  415-838-7136  1410 Briscoe St  Pete Alto  CA  94031  418-17-2313  958-8889  1  53
472  27-2947 Ginglespa  Bluff  707-958-4444  160 Ben Park  Cotina  CA  95429  472-27-2947  4447-7777  3  33
486  26-1786 Locksley  Charles  415-959-4600  16 Broadway Av  San Francisco  CA  94102  486-26-1786  469-8999  1  103
486  26-1786 Locksley  Charles  415-959-4600  16 Broadway Av  San Francisco  CA  94102  486-26-1786  469-8999  1  103
686  60-1872 Ritchie-Halls  Reginald  951-748-6402  168 Hidemore Rd  Connolly  OR  97030  686-60-1872  732-2436  1  103
686  60-1872 Ritchie-Halls  Reginald  951-748-6402  168 Hidemore Rd  Connolly  OR  97030  686-60-1872  732-2436  1  103
712  37-2249 Yoderman  Mikes  415-536-4236  16 Saber Dr  Walnut Creek  CA  94596  712-37-2249  621-7777  1  43
712  37-2249 Yoderman  Mikes  415-536-4236  16 Saber Dr  Walnut Creek  CA  94596  712-37-2249  621-7777  1  43
722  61-5449 DeFrance  Michel  219-547-9602  321 Balding Pl  Gary  IN  46403  722-61-5449  469-8821  1  75
724  60-5991 McFarland  Besans  415-354-7348  141 Upland Nks  Oakland  CA  94612  724-60-5991  540-1111  55
724  60-5991 McFarland  Besans  415-354-7348  141 Upland Nks  Oakland  CA  94612  724-60-5991  540-1111  55
506  20-7291 Karsen  Lisa  415-534-3325  1720 McClusky St  Oakland  CA  94609  506-20-7291  321-7272  1  75
506  20-7291 Karsen  Lisa  415-534-3325  1720 McClusky St  Oakland  CA  94609  506-20-7291  321-7272  1  75
507  21-6964 Plantoff  Byrde  891-538-6965  1859 Allington Pl  Rockville  MD  20850  507-21-6964  843-7202  1  103
507  21-6964 Plantoff  Byrde  891-538-6965  1859 Allington Pl  Rockville  MD  20850  507-21-6964  843-7202  1  103
558  46-2035 Ringer  Anne  861-826-0752  167 Sevenrail Av  Salt Lake City  UT  84152  558-46-2035  826-3241  2  25
558  46-2035 Ringer  Anne  861-826-0752  167 Sevenrail Av  Salt Lake City  UT  84152  558-46-2035  826-3241  2  25
558  46-2035 Ringer  Anne  861-826-0752  167 Sevenrail Av  Salt Lake City  UT  84152  558-46-2035  826-3241  2  25
558  46-2035 Ringer  Anne  861-826-0752  167 Sevenrail Av  Salt Lake City  UT  84152  558-46-2035  826-3241  2  25

The output contains duplicate columns across join tables. To remove this ambiguity in the output, you can use Natural join.
NATURAL JOIN

A natural join can retrieve specific columns from the joining tables, thus removing the ambiguity. You can specify the required column list in the Select query.

For Example,

```
SELECT t.Title, p.Pub_Name FROM Titles t JOIN Publishers p
ON t.Pub_Id = p.Pub_Id
```

In the above example, t is an alias for the Titles table and p is an alias for the Publishers table. These aliases are used to distinguish the table to which the column Pub_Id belongs.

In ANSI SQL-92, NATURAL JOIN key word is used to join the tables. The output will include common columns having identical values and then the rest of the columns from the joining tables.

For example, if your STUDENT and CLASS table share a common STUDENT_NO column, you could NATURAL JOIN them thus;

```
SELECT student_name, class_name
FROM student NATURAL JOIN class
```

Examples & Analogies

SELF Join Example: The following example of self-join displays the names and states of all the stores that are located in the same state from the Stores table in the Pubs database.

```
select t1.stor_name, t1.state, t2.stor_name, t2.state
from stores t1 INNER JOIN stores t2
on t1.state = t2.state
where t1.stor_id > t2.stor_id
```

Output will be as follows:

```
Stor_name      State
News & Brews     CA
Barnum's      CA
Doc-U-Mat: Quality Laundry and Books WA
Eric the Read Books    WA
Fricative Bookshop    CA
Barnum's      CA
Fricative Bookshop    CA
News & Brews     CA
```

Additional Inputs

Text, ntext, and image data type columns cannot be joined directly.

An OUTER join can be performed between two tables only.

GROUP BY can have CUBE operators to generate aggregate values.

The HOST_ID() system function returns the identification number of you SQL Server terminal or workstation.

The SUSER_SNAME() system function takes the login identification number of the user as a parameter and returns the login name of the user.

The DB_NAME() system function takes the database identification number as a parameter and returns the name of the database. For example, SELECT DB_NAME(1) returns Master.
The OBJECT_ID() system function takes the object name of a database object as a parameter and returns its object id. For example, SELECT OBJECT_ID('Titles') returns 2121058592 when executed in the Pubs database.

### Solutions: Just a Minute...

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | Question: The names, addresses, and phone numbers of the recruitment agencies from Houston are required. However, only the first 10 characters of the address should be displayed.  
    Answer:  
    SELECT cName, SUBSTRING(vAddress,1,10), cPhone  
    FROM RecruitmentAgencies  
    WHERE cCity='Houston' |
| 2              | Question: A schedule for interviews is required. The name of the candidate, the interviewer’s employee code, the date, and the weekday of the interviews are to be printed in alphabetical order of candidate names in the following format:  
    Candidate Name  
    Interviewer  
    Date  
    Week Day  
    Answer:  
    SELECT 'Candidate Name' = vFirstName,  
    'Interviewer'=cInterviewer,  
    'Date'=datepart(dd,dInterviewDate),  
    'Week Day'=datename(dw, dInterviewdate)  
    FROM ExternalCandidate  
    Order by vFirstName |
| 3              | Question: The names of candidates and the newspapers they referred for recruitment advertisements are required for an analysis. A report displaying these details is to be generated.  
    Answer:  
    SELECT vFirstName, vLastName, cNewspaperName  
    FROM ExternalCandidate JOIN NewsAd  
    ON ExternalCandidate.cNewsAdNo = NewsAd.cNewsAdNo  
    JOIN Newspaper  
    ON NewsAd.cNewspaperCode = Newspaper.cNewspaperCode |

### Solutions: Unguided Practice 2
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1             | --Implement 1  
SELECT vToyName, 'Description' = Substring (vToyDescription, 1, 40), mToyRate  
from Toys |
| 2             | --Implement 2  
SELECT 'Order Number' = cOrderNo, 'Shipment Date' = dShipmentDate, 'Actual Delivery Date' = dActualDeliveryDate, 'Days in Transit' = datediff (dd, dShipmentDate, dActualDeliveryDate)  
FROM Shipment |
| 3             | --Implement 3  
SELECT 'Order Number' = cOrderNo, 'Days in Transit' = datediff (dd, dShipmentDate, dActualDeliveryDate)  
FROM Shipment  
WHERE cOrderNo = '000009' |
| 4             | --Implement 4  
SELECT 'Order Number' = cOrderNo, 'Shopper Id' = cShopperId, 'Day of Order' = datepart (dd, dOrderDate), 'Week Day' = datename(dw, dOrderDate)  
FROM Orders |
| 5             | --Implement 5  
SELECT vToyName, cCategory  
FROM Toys JOIN Category  
on Toys.cCategoryId = Category.cCategoryID |
| 6             | --Implement 6  
SELECT 'Toy Name'=vToyName, 'Brand'=cBrandName, 'Category'=cCategory from Toys join Category on Toys.cCategoryId = Category.cCategoryId join ToyBrand on Toys.cBrandId = ToyBrand.cBrandId |
| 7             | --Implement 7  
SELECT 'Order Number'=cOrderNo,'Toy ID'=cToyID,'Wrapper Description'=vDescription  
FROM OrderDetail JOIN Wrapper ON OrderDetail.cWrapperId=Wrapper.cWrapperId |
| 8             | --Implement 8  
SELECT 'Toy Name'=vToyName, 'Card ID'=cCartId  
FROM Toys LEFT OUTER JOIN ShoppingCart ON Toys.cToyId = ShoppingCart.cToyId |
| 9             | --Implement 9  
select Initials = left(vFirstName,1)+'.'+left(vLastName,1)+'.',vFirstName,vlastName from shopper |
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 10            | --Implement 10  
select cOrderNo,dOrderDate,'Quarter' = datepart(quarter,dOrderDate) from orders |
| 11            | --Implement 11  
select round(avg(mToyRate),0) from toys |
| 12            | --Implement 12  
select cOrderNo,cShopperId  
from Recipient r right outer join Shopper s  
on r.cState = s.cState |

**Lesson Four**

**Experiences**

**Unions**

Do not forget to mention the restrictions on unions to the students.

**Example**

The following example displays how the UNION operator can be used to insert records into a table. First, create two tables Products and Suppliers and insert three records into each of these tables, respectively.

```sql
create table products  
(pcode char(4), descr char(10))

create table suppliers  
(scode char(4), descr char(10))

insert products values('P001', 'Toothpaste')
insert products values('P002', 'Tootbrush')
insert products values('P003', 'Soap')

insert suppliers values('S001', 'Smith')
insert suppliers values('S002', 'John')
insert suppliers values('S003', 'Tim')
```

Next, create another table called prodsupp as given below:

```sql
create table prodsupp  
(code char(4), descr char(10))
```

Now, use the following code to insert the records of the Products and Suppliers tables into the prodsupp table:

```sql
insert prodsupp
```
select * from products
union
select * from suppliers

SELECT INTO
Please remember that in SQL 2000 the select into/bulkcopy statement given with SP_DBOPTION has no effect on whether you can create a table or not. The option has been provided for backward compatibility and affects only the amount of logging in transaction log for certain bulk operations like select into depending on the type of recovery model.

Subqueries
Mention that in a subquery if the inner query is returning only one value, then use the =, >, <, >=, and <= operators. If the inner query is returning more than one value, then use the IN operator.

In case of an error while executing a subquery, execute the inner query separately to check whether there is any error in the inner query or not.

Additional Inputs
The SELECT INTO clause cannot be used with the same statement as the COMPUTE or COMPUTE BY clause as a relational output is not produced.

You cannot use subqueries on columns that contain text and image.

Each nested subquery is evaluated only once.

If a subquery does not return a single value, you must introduce the subquery with the IN, EXISTS, ANY, or ALL keyword.

For a UNION to generate a result set, the corresponding columns should have equivalent data types. The ALL keyword when used with a UNION, displays duplicate rows if any.

In a subquery, if an ‘=” operator is being used, the inner query should return a single value. If the inner query is returning multiple values and the WHERE clause of the outer query has to match with all the values returned by the inner query, then use the IN keyword.

While creating a database, it is a good practice to have multiple data files if the size of the database is very big. You should also create the data files in different physical hard disk drives as this implements faster Input-Output. The data files should also be put in NTFS partitions for better performance. The following example creates a database Accounts with two data files on two separate disks and two log files on another disk.

    CREATE DATABASE Accounts
    ON
    PRIMARY ( NAME = Acct1,
    FILENAME = 'c:\AcctDat1.mdf',
    SIZE = 200MB,
    MAXSIZE = 400,
    FILEGROWTH = 20),
    ( NAME = Acct2,
    FILENAME = 'd:\AcctDat2.ndf',
    SIZE = 200MB,
    MAXSIZE = 400,
    FILEGROWTH = 20),
    LOG ON
    ( NAME = Acctlog1,
    FILENAME = 'e:\Acctlog1.ldf',
    SIZE = 200MB,
    MAXSIZE = 400,
    FILEGROWTH = 20),
    ( NAME = Acctlog2,
    FILENAME = 'e:\Acctlog2.ldf',
    SIZE = 200MB,
MAXSIZE = 400,
FILEGROWTH = 20)

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | Question: Write a query to copy all the contents of InternalCandidate to a table called backupInternalCandidate.  
Answer:  
SELECT * into backupinternalcandidate  
from InternalCandidate |
| 2              | Question: Display the list of college names, newspaper names, and their addresses in the following format:  
**Name**  
**Address**  
Answer:  
SELECT 'Name'=cCollegeName,  
'Address'=vCollegeAddress  
FROM College  
UNION  
SELECT cNewspaperName, vHOAddress  
FROM Newspaper |
| 3              | Question: What is the use of model database in SQL Server?  
Answer: When a new database is created, the contents of the model database are copied on to the new database. |
| 4              | Question: List the three types of operating system files that store the data and objects of the SQL Server database.  
Answer:  
Primary  
Secondary  
Transaction Log |
Lesson Five

Experiences

Table Creation

Before a table is created, tell your students that the database has to be created. The tables are created in a database. Tell the students that the database has already been created for them. This will not be covered in class. Normally, databases are created and managed by experienced database administrators.

While creating a table, it is a good practice to explicitly mention whether an attribute allows null or not null values.

Tell the students that it is a good practice to name the tables and attributes meaningfully, and that they should follow the recommended naming conventions. SQL Server does not allow keywords to be used as column names.

If a table already exists with a particular name, you will not be able to create another with the same name. You need to drop the table and re-create it again with the same name.

Table Deletion

Tell the students that once a table is dropped, it cannot be referred to unless recreated.

User-defined Data types

Explain to the students that a user-defined data type is created for an attribute if the attribute appears in more than one table. Besides, a user-defined data type is used to maintain consistency of the system data type used for an attribute across tables.

Tell the students that a user-defined data type needs to be created before creating the table.

Before dropping a user-defined data type, ensure that it is not being used in any table.

Stress again the need for naming the data types meaningfully. Also, ask them to follow the recommended naming conventions.

Primary Key

The easiest way to implement a primary key is by using a PRIMARY KEY constraint. It also ensures entity integrity by not allowing duplicate values and null values.

Primary key and entity integrity can also be implemented using a unique clustered index, provided the column is defined not to allow null values. Indexes are dealt with in more detail later in the course.

Unique Constraint

Mention the fact that if the column having a unique constraint defined on it allows NULLs, then only one row in the entire column can contain a NULL value.

The unique constraint can also be implemented using a unique clustered index on a column that allows NULL values. However, only one row can contain a NULL value.

Foreign Key

Stress on the fact that a foreign key constraint can refer to another column of another table or another column of the same table.

The easiest way to implement referential integrity is to define a PRIMARY KEY constraint on the parent table and a FOREIGN KEY constraint on the child table. If an index is used to implement the primary key and entity integrity on the parent table, then referential integrity can be
implemented using a trigger on the child table. Triggers are dealt with in more detail later in the course.

**Check Constraint**

A CHECK constraint on a column implements domain integrity on the column.

A column level CHECK constraint can refer to values of only that column. However, a table level constraint can refer to values of other columns of the same table.

**Default Constraint**

Stress the fact that a default constraint is used for easing data entry.

### Examples and Analogies

The following example creates a table level constraint, which compares two columns of the same table. The following code creates a table called QuantityStatus with two columns QOH and reord_lvl and a table level constraint that restricts the value entered in the QOH to be greater than or equal to the reord_lvl.

```sql
create table QuantityStatus
(qoh int,
 reord_lvl int,
CHECK (qoh >= reord_lvl))
```

### FAQ

Q.: Can I add or delete columns from a table that has already been created?

Yes, you can use the ALTER TABLE command to add or drop columns. Refer the Books Online for the syntax.

### Additional Inputs

When creating tables, these points can be discussed with students:
- There can be two billion tables per database.
- There can be 1024 columns per table.
- Identity columns can be created to ensure that SQL Server inserts unique values automatically. There can be only one column declared as an identity column. An identity column cannot be updated. It does not allow null values. It can be of the int, smallint, tinyint, and float data types.
- Before dropping a table, you must remove any references between the table and any other object.
- User-defined data types created in the model database are automatically included in all databases that are subsequently created.
- Each user-defined data type when created adds a row to the systypes table.
- The nullability of the column defined in the table, overrides that of the data type’s.
- Domain integrity is implemented using the CHECK constraint.
- Entity integrity is implemented using the PRIMARY KEY constraint.
- Referential integrity is implemented using the FOREIGN KEY and PRIMARY KEY constraints.
- User-defined integrity is implemented in the form of business rules using CHECK constraints or triggers.
Constraints can be added while creating the table. They can also be added later using the ALTER TABLE statement.

When a constraint is added in an existing table, by default, it checks the existing data.

The system tables which store constraint definitions are syscomments, sysreferences, and sysconstraints.

To view all the constraints on a table use sp_helpconstraint followed by the table name.

Only one PRIMARY KEY constraint can be defined per table. It automatically creates a unique clustered index if not specified. If a clustered index exists on a table, then it automatically creates a non-clustered index. Null values are not allowed in the column containing a primary key constraint. The index created by the primary key cannot be dropped using the drop index command. To remove it you need to drop the constraint.

A Unique constraint is used in a situation where there is already a primary key and another column needs to have a constraint, which will restrict duplicate values in the column. A Unique constraint allows NULL, so a column having such a constraint will allow only one NULL value if the column allows NULL.

When a CHECK constraint is applied on a column, it verifies the data every time an insertion or an updation happens. A table level check constraint can reference another column on the same table. It cannot be placed on columns having the IDENTITY property.

A default constraint only applies to insert statements. A default cannot be placed on columns with IDENTITY properties.

The ALTER TABLE statement can be used to add constraints after table creation.

### Solutions: Just a Minute...

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | Question: Draft the statement to remove the College table from the database.  
Answer: DROP TABLE College |
| 2              | Question: Which integrity ensures that the values in the foreign key match with the value of the corresponding primary key?  
Answer: Referential integrity |
| 3              | Question: Which constraint enforces domain integrity by restricting the value to be inserted in a column?  
Answer: The CHECK constraint |

### Solutions: Practice

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 5.P.1          | --Implement  
create table College  
( |
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
|               | `cCollegeCode    char(4) not null,`  
|               | `cCollegeName    char(30) not null,`  
|               | `vCollegeAddress  varchar(35),`  
|               | `cCity           char(20),`  
|               | `cState          char(20),`  
|               | `cZip            char(10),`  
|               | `cPhone          char(15)` |
|               | )  
|               | `--Verify`  
|               | `sp_help College`  
|               | `--Verify`  
|               | `Insert into college values ('0002','Cromwell College ','4010 Gartner Ave.','Abilene','Texas','79605-4123','(915)692-6628')`  
|               | `--Verify`  
|               | `SELECT * FROM College`  
| 5.P.2         | `-- Implement`  
|               | `-- Primary key, default, check constraints on College table.`  
|               | `create table College`  
|               | `{`  
|               | `cCollegeCode    char(4) constraint ct_pk primary key,`  
|               | `cCollegeName    char(35) not null,`  
|               | `vCollegeAddress  varchar(30),`  
|               | `cCity           char(20) constraint defCity default 'New Orleans',`  
|               | `cState          char(20),`  
|               | `cZip            char(10),`  
|               | `cPhone          char(15) constraint cph_ck check(cPhone like('([0-9][0-9][0-9][0-9]-[0-9][0-9][0-9][0-9])'))`  
|               | `)`  
|               | `CREATE TABLE CampusRecruitment`  
|               | `{`  
|               | `cCampusRecruitmentCode char(4) primary key,`  
|               | `cCollegeCode char(4) References College(cCollegeCode),`  
|               | `dRecruitmentStartDate datetime,`  
|               | `dRecruitmentEndDate datetime`  
|               | `)`  
|               | `--Verify checking for primary key`
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
|                | Insert into college values('0001','WestendCollege ','900N Main Ave','Hempstead','New York','11550-1648','(516)423-4591') Insert into college values('0001','WestendCollege ','900N Main Ave','Hempstead','New York','11550-1648','(516)423-4591') -- Verify checking for default Insert into college values('0002','WestendCollege ','900N Main Ave',DEFAULT,'New York','11550-1648','(516)423-4591') Insert into college values('0003','WestendCollege ','900N Main Ave','Hempstead','New York','11550-1648','(516)423-4591') -- Verify checking for check constraint Insert into college values('0004','WestendCollege ','900N Main Ave','Hempstead','New York','11550-1648','(516)AAA-4591') Insert into college values('0004','WestendCollege ','900N Main Ave','Hempstead','New York','11550-1648','(516)423-4591') -- Verify checking for primary key. Insert into campusrecruitment values('0001','0001','05/13/1999','05/28/1999') Insert into campusrecruitment values('0001','0002','05/13/1999','05/28/1999') -- Verify checking for foreign key. Insert into campusrecruitment values('0002','0003','05/13/1999','05/28/1999') Insert into campusrecruitment values('0003','0011','05/13/1999','05/28/1999') SELECT * from College select * from CampusRecruitment

Solutions: Unguided Practice 3
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | **Implement 1**  
SELECT * INTO PremiumToys  
FROM Toys  
WHERE mToyRate > $20 |
| 2              | **Implement 2**  
SELECT 'First Name'=vFirstName, 'Last Name'=vLastName, 'Address'=vAddress, 'City'=cCity  
FROM Shopper  
UNION  
SELECT vFirstName, vLastName, vAddress, cCity  
FROM Recipient |
| 3              | **Implement 3**  
SELECT vToyName FROM Toys  
WHERE cCategoryId IN (SELECT cCategoryId FROM CATEGORY  
WHERE cCategory = 'Stuffed Toys') |
| 4              | **Implement 4**  
create table Recipient  
(  
cOrderNo char(6) not null,  
vFirstName varchar(20) not null,  
vLastName varchar(20) not null,  
vAddress varchar(20) null,  
cCity char(15) null,  
cState char(15) null,  
cCountryId char(3) null,  
cZipCode char(10) null,  
cPhone char(15) null  
)  
sp_addtype typCountryID,"char(3)"
create table Recipient  
(  
cOrderNo char(6) not null,  
vFirstName varchar(20) not null,  
vLastName varchar(20) not null,  
vAddress varchar(20) null,  
cCity char(15) null,  
cState char(15) null,  
cCountryId char(3) null,  
cZipCode char(10) null,  
cPhone char(15) null  
) |
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cState   char(15) null,</td>
</tr>
<tr>
<td></td>
<td>cCountryId typCountryID null,</td>
</tr>
<tr>
<td></td>
<td>cZipCode  char(10) null,</td>
</tr>
<tr>
<td></td>
<td>cPhone    char(15) null</td>
</tr>
</tbody>
</table>
|                | )
<p>|                | create table Country |
|                | ( |
|                | cCountryId typCountryID, |
|                | cCountry char(25) , |
|                | ) |
| 6              | --Implement 6 |
|                | Drop table Recipient |
| 7 a)           | --Implement 7 |
|                | create table Category |
|                | ( |
|                | cCategoryId char(3) constraint pkCategoryId primary key, |
|                | cCategory char(20) constraint unqCategory unique, |
|                | vDescription varchar(100) null |
| 7 b)           | --Implement 7 |
|                | create table ToyBrand |
|                | ( |
|                | cBrandId char(3) constraint pkBrandId primary key, |
|                | cBrandName char(20) constraint unqBrandName unique |
| 7 c)           | --Implement 7 |
|                | create table Toys |
|                | ( |
|                | cToyId char(6) constraint pkToyid primary key clustered, |
|                | vToyName varchar(20) not null, |
|                | vToyDescription varchar(250) not null, |
|                | cCategoryId char(3) references Category(cCategoryId) , |
|                | mToyRate money not null, |</p>
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cBrandId   char(3),</td>
</tr>
<tr>
<td></td>
<td>imPhoto    image null,</td>
</tr>
<tr>
<td></td>
<td>siToyQoh   smallint constraint chkToyQoh check  (siToyQoh between 0 and 250),</td>
</tr>
<tr>
<td></td>
<td>siLowerAge smallint default 1,</td>
</tr>
<tr>
<td></td>
<td>siUpperAge smallint ,</td>
</tr>
<tr>
<td></td>
<td>siToyWeight smallint,</td>
</tr>
<tr>
<td></td>
<td>vToyImgPath varchar(50) null</td>
</tr>
<tr>
<td>7 d)</td>
<td>--Implement 7</td>
</tr>
<tr>
<td></td>
<td>alter table toys</td>
</tr>
<tr>
<td></td>
<td>add constraint fkBrandId FOREIGN KEY(cBrandId) references ToyBrand(cBrandId)</td>
</tr>
<tr>
<td></td>
<td>alter table toys</td>
</tr>
<tr>
<td></td>
<td>add constraint defUpperAge DEFAULT 1 FOR siUpperAge</td>
</tr>
<tr>
<td>8</td>
<td>--Implement 8</td>
</tr>
<tr>
<td></td>
<td>SELECT * INTO CaliforniaShopper</td>
</tr>
<tr>
<td></td>
<td>FROM Shopper</td>
</tr>
<tr>
<td></td>
<td>WHERE cState = 'California'</td>
</tr>
<tr>
<td>9</td>
<td>--Implement 9</td>
</tr>
<tr>
<td></td>
<td>DROP TABLE CaliforniaShopper</td>
</tr>
<tr>
<td>10</td>
<td>--Implement 10</td>
</tr>
<tr>
<td></td>
<td>select vFirstName,vLastName from shopper where cCity in ('Woodbridge','San Jose','Las Vegas')</td>
</tr>
<tr>
<td>11</td>
<td>--Implement 11</td>
</tr>
<tr>
<td></td>
<td>select vToyName from toys</td>
</tr>
<tr>
<td></td>
<td>where cToyId in (select cToyId from orderdetail where cWrapperId is NULL)</td>
</tr>
<tr>
<td>12</td>
<td>--Implement 12</td>
</tr>
<tr>
<td></td>
<td>select cOrderNo from orders</td>
</tr>
<tr>
<td></td>
<td>where exists(select * from shipment</td>
</tr>
<tr>
<td></td>
<td>where orders.cOrderNo = cOrderNo</td>
</tr>
<tr>
<td></td>
<td>and cDeliveryStatus = 's')</td>
</tr>
</tbody>
</table>

**Lesson Six**
**Experiences**

**Rules and Defaults**

CHECK constraints can also be implemented using rules. Similarly, default constraints can also be implemented using defaults. If the same check constraints or default constraints apply to columns in more than one table, then a rule or default can be created and bound to the columns.

**Inserting Rows**

While inserting rows into a table, remember to put the values for the columns with the char and varchar data types within single quotes.

While inserting values into a column, which has a default defined on it, if you want the default to be inserted for that column, you can use the DEFAULT keyword.

Tell the students that while entering data, the column-list needs to be specified when the data being entered is not in the order of the columns in the table or when the data is not being entered for all the columns.

Tell the students that SQL Server will automatically generate values for an Identity column. It’s not a good practice to insert values for the same.

Tell the students that while inserting partial data, the columns for which data is not being inserted should support NULL or default values.

You cannot insert rows into two tables using a single INSERT statement.

**Updating Rows**

Stress on the fact that whenever data is updated, the modified value should adhere to the business rules that were effective when the row was entered.

You cannot update columns from two different tables using a single UPDATE statement.

**Deleting Rows**

You cannot delete rows from two different tables using a single DELETE statement.

**Truncating Table**

The TRUNCATE TABLE command deletes data from a table page wise and this operation is not logged in the transaction log. This is the reason a delete operation in a transaction can be rolled back, but a TRUNCATE TABLE command cannot be rolled back. Transactions are dealt with in detail later in the course.

The WHERE clause cannot be used in the TRUNCATE TABLE statement.

Tell the students to follow naming conventions for constraints, rules, and defaults.

**FAQ**

Q: When would you use a CHECK constraint, and when would you use a rule?

If the same domain or user-defined integrity applies to multiple columns of the same table or different tables, then you should create rules. Otherwise, if the integrity applies only to one column of one table, use a constraint. The same is the case with defaults.

Q: If a rule and a constraint exist on a column, which one will be effective?

The rule is applied before the constraint. Let us understand this using an example:

```sql
Create table Products
(code char(4),
qoh int
CHECK (qoh > 10))
```
The above code creates a table called Products with a constraint that does not allow entries in the qoh column that are less than 10. Now, create a rule which restricts values below 100 in the same column.

```sql
CREATE RULE qoh_rule
AS
@qoh > 100
```

Now, bind the rule to the qoh column of the Products table.

```sql
EXEC sp_bindrule 'qoh_rule', 'Products.qoh'
```

Now, try to insert the value 12 in the qoh field of the Products table.

```sql
insert Products values('T001',12)
```

This returns an error message, which indicates that the rule bound to the qoh column restricted the entry of the value 12.

Q: What happens if the number of values supplied in the values list does not match the table definition?

The insert statement gives an error. You need to supply the column-list in your insert statement.

Q: How many tables can be updated in a single update statement?

Only one table.

### Additional Inputs

A rule and default can be separately created using the CREATE RULE and the CREATE DEFAULT statement, respectively. After creation, these need to be bound to the specified columns. If any of these are bound to a table, then before dropping the rules and defaults, unbind these rules and defaults.

The FUTUREONLY option of `sp_bindrule` applies to only user-defined data types. If this option is specified, then the rule does not apply for the data in those columns, which are already using this user-defined data type. It is only applicable to all those columns which will be created in future and which will use this data type.

While inserting rows, it is a good practice to mention the column list.

Data in a text or an image column can be entered using `Writetext()`. The following example explains the usage of the `Writetext()` function.

Create a table called supplier with attributes suppcode (char(4)) and suppdescr (text) and insert the three records as shown below:

<table>
<thead>
<tr>
<th>Suppcode</th>
<th>suppdescr</th>
</tr>
</thead>
<tbody>
<tr>
<td>P001</td>
<td>Supplies the raw materials in record time and is the most preferred supplier</td>
</tr>
<tr>
<td>P002</td>
<td>Supplies the raw materials in time and can be informed in case of emergency</td>
</tr>
<tr>
<td>P003</td>
<td>Has delayed supplies on a number of occasions so is not a preferred supplier</td>
</tr>
</tbody>
</table>

Now set the select into/bulk copy option for the database to true. Next, declare a variable `@suppdescr` of binary(16) data type and put the handle of the text data into this variable using the `TEXTPTR()` function. This handle belongs to the suppdescr for the supplier P001. Now, use the `Writetext()` to write this text to the Supplier table for supplier P003. The code for this is given below:

```sql
DECLARE @suppdescr binary(16)
SELECT @suppdescr = TEXTPTR(suppdescr)
FROM supplier
```
where suppcode = 'P003'

select @suppdescr

WRITETEXT supplier.suppdescr @suppdescr 'Supplies the raw materials in record time and is the most preferred supplier'

After this, remember to set the select into/bulk copy option for the database back to false.

Each update, delete or insert operation, if not used in a transaction, is automatically committed. Once committed, it cannot be rolled back. For example, if a row is deleted from a table outside an explicit transaction, the row cannot be retrieved back unless inserted again.

Data can be deleted from a table or a view using the DELETE statement. The same can be done using the truncate statement also. If a truncate statement is used, the data in the table gets deleted page wise. Furthermore, a truncate statement is not logged in the transaction log so it cannot be rolled back.

### Configuring the Resource for the Demonstrations

You may get an error when you try to create objects that already exist in the database. To avoid the error to occur, you can execute the script: INSTALLRECRUITMENTDBFORLESSON5.BAT. This script would delete the existing Recruitment database and create a blank recruitment database. In case you are unable to execute the script, you can create a blank database and ask students to create the tables in that database. Alternatively, students can create the tables by adding their user numbers to the table names. You also have a script called INSTALLRECRUITMENTDBWITHOUTCONSTRAINT.BAT that would install the Recruitment database without any constraints. This is done so that the indexes and constraints can be created without any restrictions. Also, you would be able to delete any table, as no relation would exist between them. The structure of the tables in INSTALLRECRUITMENTDB and INSTALLRECRUITMENTDBWITHOUTCONSTRAINT is same.

Only sa or members of the sysadmin group are allowed to execute the Index Tuning Wizard. The login faculty has been made member of the sysadmin group in the scripts.

You can execute the sp_password command to change the password of the faculty login:

**Syntax of the sp_password command:**

sp_password old_password, new_password, login_name

**Example**

sp_password null,'newpassword','faculty'

### Solutions: Just a Minute...

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | Question: Which system stored procedure is used to bind and unbind a rule?  
Answer:  
The sp_bindrule and sp_unbindrule system stored procedures are used to bind and unbind a rule. |
| 2              | Question: Which statement allows you to copy contents of one table into another table?  
Answer:  
The INSERT INTO statement allows you to copy contents of one table into another table. |
Lesson Seven

Experiences

Stress on the importance of naming conventions while creating indexes.

Stress on the advantages of using indexes.

Stress on the concept of creating appropriate indexes for appropriate columns.

Do mention that indexes are used to speed up queries and are created on columns that are used in joins, the WHERE clause, the ORDER BY clause, or the GROUP BY clause.

Note: To explain the diagram for clustered and non-clustered index, faculty needs to follow all the given steps in the student guide. Faculty also needs to ask students to do the same thing.

The diagram contained in the slide has simplified the Nonclustered index diagram given in the student guide. This diagram depicts the reference of only one record (EID E005) from Root Page to the Data Page.

Note: Faculty can ask students to note down the diagram for their reference.

Creating Clustered Indexes

Tell the students that when a clustered index is created, the whole table gets reorganized and ordered based on the column values on which the index has been created.

On inserting or updating the column on which a clustered index is created, the entire table gets reorganized again in the order of the column values.

Stress on creating clustered indexes before the non-clustered indexes.

Tell the students that for a clustered index, the leaf level and the data level pages are the same.

There can be only one clustered index on a table since the table is in the order of the column values on which the index has been created.

A clustered index is created on tables, which are updated rarely. In other words, it is created on look-up tables (master tables as they are known popularly). In other words, clustered indexes have to be created on columns of tables where rows are rarely inserted, deleted, or updated. These points are very important thumb rules for creating a clustered index.

A clustered index is generally created on columns that have unique values or on primary keys. Besides, the table has to be a look-up table and not a highly transacted table.

Creating Non-clustered Indexes

Tell the students that if the type of index is not mentioned in the CREATE INDEX statement, then a non-clustered index is created by default.

If an index has to be created on a column for speeding up queries, and if the values of that column are frequently modified (due to inserts, updates, and deletes), then a non-clustered index has to be created, even though the values of the column are unique (such as a primary key). Creating a clustered index on a highly transacted table will slow down the transactions since the table gets reorganized on every insert, update, or delete operation. These points are very important thumb rules for creating a non-clustered index.

A Unique index can be used to enforce uniqueness on a column.
**DBCC Showcontig**

DBCC Showcontig command provides information about data fragmentation in a table. Fragmentation happens in a table due to frequent insertion and deletion operations. Data fragmentation leads to slower I/O and results in slow server performance. Therefore, it is essential to know about fragmented data pages. Once fragmentation is identified as a problem, you can reindex the tables to remove fragmentation.

**Note:** This fragmentation is returning to the fragmentation of data pages within the SQL Server MDB file, and not of the physical file itself.

Following command can be used to get the information about fragmentation in authors table:

**DBCC Showcontig ‘authors’**

**Output:**

```
DBCC SHOWCONTIG scanning 'authors' table...
Table: 'authors' (1556968673); index ID: 1, database ID: 16
TABLE level scan performed.
- Pages Scanned........................: 18986
- Extents Scanned......................: 2443
-Extent Switches........................: 9238
-Avg. Pages per Extent.................: 7.8
-Scan Density [Best Count:Actual Count]...: 25.70% [2374:9239]
-Logical Scan Fragmentation ............: 44.58%
-Extent Scan Fragmentation .............: 87.07%
-Avg. Bytes Free per Page..............: 1658.7
-Avg. Page Density (full)...............: 79.51%
```

Explanation of the output. In authors table:

There were 18,986 pages examined to create the report.

Those pages existed within 2,443 extents, indicating that the table consumed approximately 97% (7.8 pages per extent on average) of the extents allocated for it.

While examining the pages for fragmentation, the server had to switch extent locations 9,238 times.

The Scan Density states, that how much a table is fragmented. If scan density is 100, it means the table is not fragmented.

The Logical Scan Fragmentation and Extent Scan Fragmentation are indications of how well the indexes are stored within the system, when a clustered index is present. The scan fragmentation (Logical & Extent) should be ignored for tables that do not have a clustered index. In both cases, a number close to 0 is preferable, to indicate no fragmentation.

**Conclusion**

After examining the output of DBCC SHOWCONTIG, we are able to identify several key issues. First, we saw that our table is heavily fragmented, and required defragmentation. Next, we are able to state the percentage of the allocated pages SQL Server is actually using.

DBCC DBREINDEX command rebuilds an index for a table and removes fragmentation.

**FAQ**

Q: Can all columns of a table be indexed?

Yes, but it is not very beneficial. Index only those columns, which are used in a WHERE clause, joins, the ORDER BY clause or the GROUP BY clause.

Q: Can a single column be indexed more than once?

Yes, but it is not very beneficial.

**Additional Inputs**
If a primary key constraint is not created, then a unique clustered index can be used to implement the primary key and entity integrity. However, a foreign key constraint cannot reference such a column since a primary key constraint is not defined. However, in such cases, referential integrity can be implemented from the child table using a trigger.

Each table can have only one clustered index and a maximum of 249 non-clustered indexes.

Clustered indexes need to be created on the same filegroup as the base table, but non-clustered indexes can be created on different filegroups placed on different physical drives, so as to take full advantage of parallel I/O happening through multiple disk controllers.

Multiple indexes on a column is not a very good practice.

Indexes created by creation of constraints like the Primary key or Unique cannot be dropped using the DROP INDEX statement. To drop it, you need to drop the constraint.

In a SELECT statement without a WHERE clause, indexes belonging to the column mentioned first in the select list are used.

A Unique index enforces uniqueness on the column for which it is mentioned. If nulls are allowed, it will allow only one null, as a second one will become a duplicate. A Unique index can be clustered or non-clustered, the default is clustered.

While creating an index, the fillfactor can be mentioned for it. It is a percentage to which the leaf level pages of the index are going to be filled up. Similarly, PAD INDEX can be used to mention how many of the non-leaf level pages are filled. PAD INDEX cannot be used without fillfactor. A separate percentage value cannot be mentioned for PAD INDEX as it takes the fillfactor value only. A low fillfactor is good for an OLTP environment, whereas a high fillfactor is suitable for a Decision support (DSS) environment.

Database Consistency Checker (DBCC) statements can be used by DBAs (Database Administrators) to check the databases for an instance of SQL Server 2000. These statements not only check and report the physical status of the database and its various objects, they can also be used for checking the logical status. The DBCC statements can be divided into four categories, namely Validation, Maintenance, Status, and Miscellaneous. Examples of Validation statements are DBCC CHECKFILEGROUP and DBCC CHECKCONSTRAINTS. Examples of Maintenance statements are DBCC DBREINDEX and DBCC INDEXDEFRAG. Examples of Status statements are DBCC SHOWCONTIG and DBCC SHOW_STATISTICS. Some examples of Miscellaneous are DBCC HELP and DBCC ROWLOCK.

**Solutions: Just a Minute...**

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | Question: 1. How many clustered indexes can be created per table?  
Answer:  
Only one clustered index can be created per table.  
Question: 2. Which index organizes the data logically but does not store the data physically?  
Answer:  
NonClustered index organizes the data logically but does not store the data physically. |
Question: Susan wants to minimize the amount of page splitting that occurs each time an index page is full. What should she use?

Answer:

FILLFACTOR

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1 a)           | --Implement 1
create table Category
{
    cCategoryId char(3),
    cCategory   char(20),
    vDescription varchar(100)
}
| 1 b)           | --Implement 1
create table ToyBrand
{
    cBrandId    char(3),
    cBrandName  char(20)
}
| 1 c)           | --Implement 1
create table Toys
{
    cToyId   char(6),
    vToyName varchar(20),
    vToyDescription varchar(250),
    cCategoryId char(3),
    mToyRate  money,
    cBrandId  char(3),
    imPhoto   image null,
    siToyQoh  smallint ,
    siLowerAge smallint ,
    siUpperAge smallint ,
    siToyWeight smallint ,
    vToyImgPath varchar(50) null
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>)</td>
</tr>
</tbody>
</table>
| 1 d)           | --Implement 1  
|                | create rule rulRate  
|                | as  
|                | @rate>0  
|                | sp_bindrule rulRate,'Toys.mToyRate'  
|                | create default rulWeight  
|                | As 1  
|                | sp_bindefault rulWeight,'Toys.siToyWeight'  |
| 1 e)           | --Implement 1  
|                | insert into ToyBrand  
|                | Values('001','Bobby')  
|                | insert into ToyBrand  
|                | values('002','Frances-Price')  
|                | insert into ToyBrand  
|                | values('003','The Bernie Kids')  
|                | insert into ToyBrand  
|                | values('004','Largo')  |
| 1 f)           | --Implement 1  
|                | insert into Category  
|                | values('001','Activity','Activity toys encourage the child’s social skills and interest in the world around them.')  
|                | insert into Category  
|                | values('002','Dolls','A wide range of dolls from all the leading brands.')  
|                | insert into Category  
|                | values('003','Arts And Crafts','Encourage children to create masterpieces with these incredible craft kits.')  |
| 1 g)           | --Implement 1  
|                | insert into toys  
<p>|                | values('000001','Robby the Whale','A giant Blue Whale with two heavy-duty handles that allow a child to ride on its back.','001',8.99,'001',NULL,50,3,9,1,null)  |</p>
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1 h)           | --Implement 1  
 update toys  
 set mToyRate=mToyRate+1  
 WHERE cToyId='000001' |
| 1 i)           | --Implement 1  
 DELETE ToyBrand  
 WHERE cBrandName='Largo' |
| 2              | --Implement 2  
 Create nonclustered index idxshopper  
 on Orders(cShopperId) |
| 3              | --Implement 3  
 CREATE unique clustered index idxToys  
 on Toys(cToyId) |
| 4              | --Implement 4  
 CREATE unique nonclustered index idxCategory  
 ON Category(cCategory) |
| 5              | --Implement 5  
 Declare @objid int  
 SET @objid = OBJECT_ID('Shopper')  
 DBCC SHOWCONTIG (@objid) |
| 6              | --Implement 6  
 Declare @objid int  
 SET @objid = OBJECT_ID('Wrapper')  
 DBCC SHOWCONTIG (@objid) |
| 7              | --Implement 7  
 CREATE NONCLUSTERED INDEX idx_cState  
 ON Recipient (cState) |
| 8              | --Implement 8  
 INSERT ShoppingCart VALUES('000010','000013',2) |

Lesson Eight

**Experiences**

**Create Views**
It is a good practice to follow naming conventions while creating views. Prefix the name using ‘vw’.

Views are used for two basic reasons – to simplify complex queries for users and to restrict users from viewing data directly from the table.

While explaining views to the students you can take the example of the small window on your class room door. If somebody peeps through it he or she will only be able to see the only a small portion of the class and will think that the visible portion is the entire class room. Similarly for a view which displays only say 4 columns of a table, the user will perceive that the table contains only 4 columns.

If you want to restrict users from seeing the definition of the view using the sp_helptext command, you can encrypt the definition using the WITH ENCRYPTION option in the CREATE VIEW statement.

You need to stress the fact that the views cannot store data by themselves, they get the data from the underlying base tables. A view is nothing but a query stored as an object.

Views are slow as the query is processed each time the view is used.

**Data Update through Views**

Updating rows using views affect the base table values.

Columns specified in the UPDATE statement must belong to a single base table. If columns belonging to two different tables are specified in the UPDATE statement, SQL server will return an error.

Columns not part of the view cannot be updated using the view.

The WITH CHECK OPTION forces data modification within the criteria specified by the WHERE clause of the SQL statement.

The ALTER VIEW statement can be used to change the view definition.

**Variable Declaration**

A variable can be declared using the DECLARE keyword. Follow the naming conventions as recommended.

**BEGIN and END Statements**

Tell your students that when more than one statement needs to be executed, the statements needs to be put within the BEGIN and END block, else only the first line of the SQL block gets executed.

Remember that a block of SQL statements executed together is called a batch.

Demonstrate a set of INSERT statements or SELECT statements as a batch.

**Conditional Execution**

One or more SQL statements can get executed depending on a condition. The IF statement can be used for doing so. If the same condition has to be tested for different values, tell your students to use the CASE construct instead of the IF statement.

The WHILE construct can be used to repeatedly execute a block of SQL statements.

**FAQs**

Q What are the restrictions on views?

The ORDER BY, COMPUTE or COMPUTE BY, and the INTO statements cannot be used in a CREATE VIEW statement.

Views cannot reference temporary tables.

The CREATE VIEW statement cannot be combined with other T-SQL statements.
Q: Can a batch be saved in the database like any other database object?
No. However, the script can be stored in a script file using Notepad.

**Additional Inputs**

Rows can be updated, deleted, and inserted using a view if they affect a single table.

If a computed column exists in the view definition, the header for the same needs to specified.

Computed columns, summary data, and aggregate values cannot be updated using a view.

Avoid using OUTER JOINS in views.

If a base table is dropped before dropping the view, it does not generate an error, but if the view is referenced after this, it generates an error. So before dropping the base table, drop the view first.

If a stored procedure is to be executed in a batch, execute it with the EXECUTE statement.

Multiple updates, inserts, and deletes can be done in a batch.

You can use the CASE statement in a DML statement when conditional query or updates have to be done.

In SQL Server 2000, you can create indexes on views. When a clustered index is created on a view, the data is stored in the database instead of online generation of the result-set. This improves data retrieval time. Just like clustered indexes on tables, every time there is a modification of data in the base table(s), the clustered index on the view also gets updated. As this updatation consumes a lot of time, it affects the database performance negatively. So such indexes should be created only on views that are read-intensive and not write-intensive.

**Solutions: Just a Minute...**

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | Question: Some queries to be executed are as follows:  
SELECT vFirstName, vDepartmentName  
FROM Employee JOIN Department  
ON Employee.cDepartmentCode = Department.cDepartmentCode  

SELECT vFirstName, cDesignation, vDepartmentName  
FROM Employee JOIN Department  
ON Employee.cDepartmentCode = Department.cDepartmentCode  

SELECT vFirstName, vAddress, cCity, cZip, cDesignation, vDepartmentName, vDepartmentHead  
FROM Employee JOIN Department  
ON Employee.cDepartmentCode = Department.cDepartmentCode  
WHERE cCity = 'Columbus'  
Simplify the task of executing these queries.  
Answer: |
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1.57 | Create view vwEmployeeDepartment as
| | SELECT vFirstName, vAddress, cCity, cZip, cDesignation, vDepartmentName, vDepartmentHead |
| | FROM Employee JOIN Department ON Employee.cDepartmentCode=Department.cDepartmentCode |
| | --Verify |
| | SELECT vFirstName, vDepartmentName |
| | FROM vwEmployeeDepartment |
| | SELECT vFirstName, cDesignation, vDepartmentName |
| | FROM vwEmployeeDepartment |
| | SELECT vFirstName, vAddress, cCity, cZip, cDesignation, vDepartmentName, vDepartmentHead |
| | FROM vwEmployeeDepartment WHERE cCity = 'Columbus' |

2 | Question: Fill in the blanks:
| | A group of SQL statements submitted together to SQL server for execution is called a ________.
| | ________ declared within a batch is lost when the execution of the batch is over.
| | Answer:
| | Batch
| | Local variable

**Solutions: Practice**

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>--Implement</td>
</tr>
<tr>
<td></td>
<td>UPDATE vwNewspaperNewsad SET cZip='88993-4532' WHERE cNewspaperName='Daily News'</td>
</tr>
<tr>
<td></td>
<td>UPDATE vwNewspaperNewsad SET dAdStartDate='01/09/99'</td>
</tr>
</tbody>
</table>
Lesson Nine

Experiences

Procedures

Discuss the benefits of procedures – modularity, speed, security, reduced network congestion and consistency of usage across applications and users.

Tell the students that since stored procedures have so many benefits, all operations and transactions from the client such as queries, updation, insertion, and deletion of rows are done using stored procedures. Even if the insert, update, delete, or query operation is very simple, a stored procedure must be created. This improves performance of the application. Hence, programmers simply execute the stored procedures, which are stored at the backend instead of sending SQL statements from the client.

If the definition of the stored procedure needs to be modified, then use the ALTER PROCEDURE statement.

Types of Procedures

Tell your students that an extended stored procedure is used to activate a function in a DLL residing outside the database.

A temporary stored procedure is accessible to a user as long as he or she does not log out.

A temporary stored procedure can be of two types, local or global. A global stored procedure is available to all users as long as the last user has not logged out. The name of a global stored procedure is prefixed by ##.

Tell the students that the difference between a system-defined stored procedure and a user-defined stored procedure is that a system defined stored procedure is provided by SQL Server and a user-defined stored procedure is what the user creates.

Extended Stored Procedure

The extended stored procedure name is prefixed with "xp ". All the extended stored procedures physically reside in a DLL file. All the DLL’s for extended stored procedure resides in C:\Program Files\Microsoft SQL Server\80\Tools\Binn folder.

For example: To run any DOS command from SQL Analyzer window, following command can be used:

```
xp_cmdshell 'dir'
```

Note: As students are not aware of DOS, so to handle this problem, faculty can give a small demo of DOS command on the command prompt.

To find out free space on the drive, following command is used:

```
xp_fixeddrives
```

To search for a file “test.java” on C drive, use the following command:
xp_fileexists 'c:\test.java'

**Parameters**

After explaining the two types of parameters, tell the students that a procedure with output parameters is typically executed from another calling procedure.

Mention that the ‘return’ keyword is used when a single value needs to be returned to the calling program.

Output parameters are used when multiple values of any data type have to be returned.

Tell the students that apart from explicitly executing a stored procedure it can be executed automatically, for example, at the start-up of SQL Server. This can be done by using a system stored procedure called `sp_procoption` for only objects of master database, which are owned by dbo. The syntax for `sp_procoption` procedure is shown below:

```sql
sp_procoption @ProcName = 'procedure', @OptionName = 'option', @OptionValue = 'value'
```

where,

- `@ProcName` defines the procedure for which to set the option
- `@OptionName` defines the option to set for the procedure that can only be ‘startup’, which will set the procedure for autoexecution
- `@OptionValue` defines whether to set the option as true or false

**Procedure Usage**

Mention in the class that if data changes and indexes are updated, the execution plan for the SQL statements in a stored procedure will become outdated. In this kind of a situation, the SQL statement needs to be recompiled every time the stored procedure gets executed. This is done using the WITH RECOMPILE option of the CREATE PROCEDURE statement.

Remember to use naming conventions while creating procedures.

**FAQ**

Q: What are the different types of procedures?

System stored procedures, Local stored procedures, Temporary stored procedures, Remote stored procedures, and Extended stored procedures.

**Additional Inputs**

Stored procedures can be nested upto 32 levels.

When designing an application, stored procedures can significantly reduce the network requirements. Use stored procedures for long, complicated, and frequently repeated queries. This reduces the traffic from the client to the server because only the stored procedure name and its associated parameters are passed across the network to the server, where it is executed. In addition, multi-step queries that perform additional filtering or processing based upon the response to initial queries, run much more efficiently as a stored procedure. By using a stored procedure, it is not necessary to pass the results of the initial query to the client in order that a second query can be passed to the server.

The `sp_depends` stored procedure can be used to find out the dependencies of a stored procedure. The syntax is as given below:

```sql
sp_depends object_name
```

Extended stored procedures can be used to execute commands that are external to SQL Server. These commands can be operating system level commands or those created by a user using a programming language such as c or c++. These commands need to be created as part of a DLL for
execution using extended stored procedures. Before executing these DLLs need to be registered using sp_addextendedproc stored procedure. MS SQL Server 2000 provides you with some inbuilt extended stored procedure example xp_cmdshell and xp_readmail. For example the following command will display the files and directory listings from the current directory:

```sql
xp_cmdshell 'dir'
```

The command can be executed only from the master database.

Notice that if the number of statements after else is only one then there is no need to use BEGIN…END.

Notice that whenever a procedure gets called from within another procedure you have to use the EXEC keyword.

The [:;number] of the CREATE PROCEDURE procedure_name [:;number] syntax allows creating multiple procedures with the same name. The main advantage of this is that the procedures can be managed together. For example, you can drop all the procedures together in a single command.

### Solutions: Just a Minute...

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | **Question:** What are the benefits of using stored procedures?  
|                | **Answer:**  
|                | Improved performance  
|                | Reduction in network congestion  
|                | Better consistency  
|                | Better security mechanism  |
| 2              | **Question:** The query to obtain a list of candidates and their recruitment agencies is:  
|                | `SELECT 'Candidate Name'= vFirstName, 'Recruitment Agency'= cName  
|                | FROM ExternalCandidate JOIN RecruitmentAgencies  
|                | ON ExternalCandidate.cAgencyCode = RecruitmentAgencies.cAgencyCode`  
|                | Create a stored procedure for the same.  
|                | **Answer:**  
|                | Create procedure prcCandidateRecruitment  
|                | As  
|                | BEGIN  
|                | SELECT "Candidate Name"=vFirstName, "Recruitment Agency"=cName  
|                | FROM ExternalCandidate JOIN RecruitmentAgencies  
|                | ON ExternalCandidate.cAgencyCode=RecruitmentAgencies.cAgencyCode  
|                | END  
|                | --Verify  
<p>|                | <code>sp_helpText prcCandidateRecruitment</code> |</p>
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 3              | Question: You need to modify a stored procedure. Which command will you use to modify the procedure?  
Answer: ALTER PROCEDURE |

**Solutions: Unguided Practice 5**

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | --Implement 1  
Create view vwShopperToy  
as  
SELECT Shopper.vFirstName,  
vToyName,siQty,mToyCost  
FROM Shopper JOIN Orders  
ON Shopper.cShopperId= Orders.cShopperId  
JOIN OrderDetail  
ON Orders.cOrderNo=OrderDetail.cOrderNo  
JOIN Toys  
ON OrderDetail.cToyId=Toys.cToyId  
--Verify  
SELECT * from vwShopperToy |
| 2              | --Implement 2  
Update vwOrderWrapper  
SET siQty=2  
FROM vwOrderWrapper  
WHERE cOrderNo='000001'  

Update vwOrderWrapper  
SET mWrapperRate=mWrapperRate + 1  
FROM vwOrderWrapper  
WHERE cOrderNo='000001' |
| 3              | --Implement 3  
DECLARE @deliverystatus char(1)  
SELECT @deliverystatus=cDeliveryStatus FROM Shipment WHERE cOrderNo='000003'  
IF @deliverystatus='d'  
    Print 'The order has been delivered' |
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ELSE</strong></td>
<td>Print 'The order has been shipped but not delivered'</td>
</tr>
</tbody>
</table>
| **4**          | `--Implement 4
While(SELECT Avg(mToyRate+0.5) from toys)<22.5
BEGIN
Update toys
set mToyRate=mToyRate+.5
END` |
| **5**          | `--Implement 5
While(SELECT Avg(mToyRate+0.5) from toys)<24.5
BEGIN
Update toys
set mToyRate=mToyRate+.5
IF (SELECT MAX(mToyRate+0.5) FROM toys)]>=53
BREAK
ELSE
CONTINUE
END
--Verify
SELECT MAX(mToyRate) from Toys` |
| **6**          | `--Implement 6
Create procedure prcDisplayToys
as
SELECT vToyName,vToyDescription,mToyRate
FROM Toys
--Verify
exec prcDisplayToys` |
| **7**          | `--Implement 7
CREATE PROCEDURE prcDisplayShopper
as
SELECT vFirstName, vLastName, vEmailId
FROM Shopper
--Verify
exec prcDisplayShopper` |
| **8**          | `--Implement 8
CREATE PROCEDURE prcDisplayPrice
@ToyId char(6)
as
SELECT vToyName,mToyRate` |
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
|               | FROM Toys  
|               | WHERE cToyId=@ToyId  
|               | SELECT * FROM Toys  
|               | --Verify  
|               | exec prcDisplayPrice '000001'  
| 9             | --Implement 9  
|               | CREATE PROCEDURE prcAddBrand  
|               | @BrandId char(3),  
|               | @BrandName char(20)  
|               | as  
|               | INSERT ToyBrand  
|               | VALUES(@BrandID,@BrandName)  
|               | --Verify  
|               | exec prcAddBrand '009','Fun World'  
| 10            | --Implement 10  
|               | CREATE PROCEDURE prcAddCategory  
|               | @CategoryId char(3),  
|               | @Category char(20),  
|               | @Description varchar(100)  
|               | as  
|               | INSERT Category  
|               | VALUES (@CategoryId,@Category,@Description)  
|               | --Verify  
|               | exec prcAddCategory '018','Electronic Games','These games contain a screen with which children interact.'  
| 11            | --Implement 11  
|               | drop procedure prcAddCategory  
| 12            | --Implement 12  
|               | CREATE PROCEDURE prcCharges  
|               | @OrderNo char(6),  
|               | @ShippingCharges money OUTPUT,  
|               | @GiftWrapCharges money OUTPUT  
|               | as  
|               | SELECT  
|               | @ShippingCharges=mShippingCharges,@GiftWrapCharges=mGiftWrapCharges  
|               | FROM Orders  

©NIIT
Implementing a Database Design Using Microsoft SQL Server 1.63
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| **13** | WHERE cOrderNo=@OrderNo  
--Verify  
DECLARE @Shipping money  
DECLARE @Gift money  
exec prcCharges '000003',@Shipping OUTPUT,@Gift OUTPUT  
SELECT @Shipping,@Gift |
| **14** | --Implement 14  
CREATE PROCEDURE prcHandlingCharges @OrderNo char(6)  
as  
DECLARE @HandlingCharges money  
DECLARE @Shipping money  
DECLARE @Gift money  
exec prcCharges @OrderNo,@Shipping OUTPUT,@Gift OUTPUT  
SELECT @HandlingCharges=@Shipping+@Gift  
SELECT @HandlingCharges  
RETURN  
--Verify  
exec prcHandlingCharges '000001' |
| **15** | --Implement 15  
CREATE PROCEDURE prcAddCategory @CategoryId char(3),  
@Category char(20), @Descrip char(30)  
as  
INSERT Category VALUES(@CategoryId,@Category,@Descrip)  
--Verify  
exec prcAddCategory '017','War Games','A wide range of toy guns' |
Lesson Ten

Experiences

Triggers
Stress on the fact that triggers are used when complex business rules have to be implemented. While constraints can be used to maintain referential integrity, triggers can also be used if required.

Mention that triggers are a special type of stored procedure, but cannot be executed explicitly.

Mention that the overhead involved with a trigger is very high, but the functionality provided is also very good.

You also need to touch upon the cascade delete, restrict delete, and nullify delete rules in your class. If a record is deleted from the master table, then the corresponding records from the transaction table also get deleted. This is the cascade delete rule. In the restrict delete rule, if an open transaction exists in the transaction table, then the corresponding records in the parent table cannot be deleted. In the nullify delete rule, if a record is deleted from a parent table, then the corresponding values in the foreign key column of the child tables is replaced by NULL.

Discuss the concept of nesting levels of triggers.

Follow naming conventions for triggers. Prefix a trigger name with ‘trg’.

Multiple Triggers
Tell the students that multiple triggers for a DML operation can be created on the same table. For instance, in a table called TAB1 you can create two triggers TRIG1 and TRIG2, both for update operation. Also, mention that this facility was not available in the earlier versions of SQL Server. The benefit of using multiple triggers is that you can implement multiple business rules using different triggers. However, you can incorporate all the rules in a single trigger. Having multiple triggers helps in maintenance, readability, and documentation. The triggers are executed in the order they have been created.

Instead Of Triggers
Tell the students that in SQL Server 2000, you have a new variation of the database object trigger. This type of trigger is used to update the base tables of a view when a view is created on multiple tables. This type of trigger is particularly useful for validating insert values before inserting in the base tables. The instead of triggers can be created on tables or views. In case a table contains primary key or foreign key constraints that implement with cascade delete or cascade update functionality, then the instead of delete and instead of update triggers cannot be defined on them.

FAQ
Q: When does a trigger get executed?
After a DML (update, insert, or delete) transaction.

Q: If there exists a trigger and a rule, which will get executed first?
The rule will get executed first.

Additional Inputs
The maximum nesting level for triggers is 32.
You cannot create triggers on system tables.
Triggers unlike stored procedures do not return values or result sets.

If multiple business rules need to be applied when a DML operation is underway use multiple triggers for implementing the same. For example, if three columns are being updated and different business rules have to be applied for each, use three different update triggers for each business rule.
SQL Server allows recursive triggers. Recursion occurs when the same trigger gets executed again and again. There are two types of recursion, direct and indirect. For example, if an application updates table T3, the trigger TRIG3 defined on the table for update gets executed. This trigger again does an updation on the table T3, thereby, re-executing the trigger TRIG3. This is an example of direct recursion. If an application updates table T3, the trigger TRIG3 defined on the table for update gets executed. This trigger updates another table T4, this executes trigger TRIG4 defined for update on the table. TRIG4 updates table T3 thereby executing TRIG3. This is an example of indirect recursion.

To enable recursive triggers for a particular database, issue the following command:
```
sp_dboption <databasename>, 'recursive triggers', True.
```

To switch between the implicit and the explicit modes, use the SET IMPLICIT_TRANSACTIONS {ON | OFF} statement

In the implicit mode the following statements trigger off a transaction: ALTER TABLE, INSERT, OPEN, CREATE, DELETE, REVOKE, DROP, SELECT, FETCH, TRUNCATE TABLE, GRANT, UPDATE.

The number of open transactions per connection is stored in the system variable @@TRANCOUNT. Every new transaction i.e. every BEGIN TRANSACTION increments the value of this system variable by one and every COMMIT TRANSACTION or ROLLBACK TRANSACTION decrements the value by one. In the implicit mode, every issue of the above mentioned commands automatically generates a BEGIN TRANSACTION.

### Solutions: Just a Minute…

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | Question:  
|                | a. What are magic tables?  
|                | Answer:  
|                | Magic tables are conceptual tables and are similar in structure to the table on which the trigger is defined. Whenever a trigger fires in response to the INSERT, DELETE, or UPDATE statement, two special tables are created. These are the inserted and the deleted tables. These tables are referred to as the magic tables.  
|                | Question:  
|                | b. Which statement is used to create triggers?  
|                | Answer:  
|                | CREATE TRIGGER |
| 2              | Question: When is a DELETE trigger fired?  
|                | Answer:  
|                | A DELETE trigger is fired whenever an attempt is made to delete a row from the trigger table. |
| 3              | Question: Which statement is used to recreate a trigger?  
|                | Answer:  
|                | ALTER TRIGGER |
| 4              | Question: How are triggers used to maintain integrity and consistency of data? |
Problem number | Solution
---|---
| Answer: Using triggers you can implement referential integrity and thus maintain integrity and consistency of data.

Solutions: Practice

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 10.P.1 | **--Implement**
Create trigger trgInsertEmployee
on Employee
for insert
as
update Position
set iCurrentStrength= iCurrentStrength + 1
from inserted JOIN Position
ON
Position.cPositionCode=inserted.cCurrentPosition

print 'Position table updated'
return

select cCurrentPosition from Employee where cEmployeeCode='000016'
**--Verify**
insert Employee
values('000016','Linda','White','000007','1543 Joviality Drive','Huston','Texas','43232-1223','001','(623)234-2342','Graduate','4/9/71','F','0001','Assistant Manager','linaday@gmail.com','0001','Houston',NULL,getdate(),NULL,'234-34-2325',NULL)
select * from position

| 10.P.2 | **--Implement**
create trigger trgEmployeeResignation
on Employee
for update
as
if update(dResignationDate)
Begin
Update Position
Lesson Eleven

Experiences

Transaction

Stress on the fact that a transaction is an atomic unit of work, which either happens completely or does not happen at all. If an insert operation in one table and two update operations in two different tables constitute a logical unit of work, then the three operations can be termed as a transaction. If only one insert happens and the other two updates do not happen, the transaction is not complete and may result in inconsistency of data. Hence, it is essential that all the operations happen or none of them happens at all. Explicit statements like BEGIN TRANSACTION and COMMIT TRANSACTION ensure that all statements in a transaction are completed successfully or do not take place at all in case there is a system crash while the transaction is running.

By default SQL Server uses a row level lock.

Tell the students that the transactions should be as short as possible, and table level locks should be avoided as this locks the entire table.

If multiple transactions refer to the same tables, then refer them in a specific order to minimize deadlocks.

While creating a transaction, follow the naming conventions. Prefix the transaction name with a ‘trn’.

Transaction Log

Tell the students that the transaction log is like a huge ‘security register’ where any activity on the database gets recorded. The log is used to roll forward or rollback transactions in case of a system failure. Tell them that the transaction log plays a big role in transaction management.
**Transaction Modes**

There are two types of transaction modes, explicit and implicit. The default is implicit, this can be changed by using BEGIN TRANSACTION, COMMIT TRANSACTION, COMMIT WORK, ROLLBACK TRANSACTION, and ROLLBACK WORK.

SAVE TRANSACTION can be used to save transactions to a certain point. Tell the students when SAVE TRANSACTION is used, and if we rollback a transaction then the transaction rolls back only up till the save point.

**Distributed Transactions**

Unlike normal transactions, a distributed transaction is processed on more than one database server.

**Concurrency Problems**

Find below the examples for each of the concurrency problems:

**Lost updates**

Lost updates occur when two or more transactions select the same row and then update the row based on the value originally selected. Each transaction is unaware of the other transaction. The last update overwrite updates made by the other transaction, which results in lost data.

Let us assume that both Sam and Anne are simultaneously trying to update the price of all the “Business” books in the Titles table. Sam is trying to update the price by 10% while Anne is trying to update the price by 15%.

Now, the table will get updated by the changes of the query that will get completed last. That means, if Sam’s query is executed later than Anne’s query, then the price column in titles table will get increased by 10% and the changes made by Anne’s query will be lost.

**Uncommitted dependency**

Uncommitted dependency occurs when a second transaction selects a row that is being updated by another transaction. The second transaction is reading data that has not been committed yet and may be changed by the transaction updating the row.

Let us assume User A and B are working on titles table. User A had increased the price of title_id ‘BU1032’ by Rs. 10. But user A does not commit the transaction. Now User B tries to execute a query on title_id ‘BU1032’. User B is accessing old record as the transaction handle by user A is not yet complete. Therefore user B also updates the price of title_id ‘BU1032’ by Rs. 5. These transactions will update the record by Rs 15. Such kind of problems leads to inconsistency in the table.

**Inconsistent Analysis**

Inconsistent analysis occurs when a second transaction accesses the same row several times and reads different data each time. Inconsistent analysis is similar to uncommitted dependency in that another transaction is changing the data that a second transaction is reading. However, in inconsistent analysis, the data read by the second transaction was committed by the transaction that made the change. Also, inconsistent analysis involves multiple reads (two or more) of the same row and each time the information is changed by another transaction; thus, the term nonrepeatable read.

For example:

Let us assume that you are accessing the online reservation system to check the status of your ticket. The site showed the status as ‘Waiting’. Just a little later, when you refreshed the page, you found that the status is ‘confirmed’. This shows that while you were browsing thru the information, some procedure was updating the record information.

**Phantom reads**
Phantom reads occur when an insert or delete action is performed against a row that belongs to a range of rows being read by a transaction. The transaction's first read of the range of rows shows a row that no longer exists in the second or succeeding read, as a result of a deletion by a different transaction. Similarly, as the result of an insert by a different transaction, the transaction's second or succeeding read shows a row that did not exist in the original read.

For Example:

You are accessing online catalog of a book store. You found a book name “You can win” in your initial search where you are looking for titles having “win” in their title name. But subsequent search of the same query did not show “You can win” in the output. The reason can be that some procedure might have deleted the title from the table.

Note: Though these scenarios will help you to explain the concept, please clarify to the students that these situations occur in remote multi-user environment. They will not find similar scenario while working in the MR.

LOCKS

Discuss the types of locks in detail. Also, discuss when each of these locks are used by SQL Server.

SQL Server uses row lock by default.

The concept of deadlock needs to be explained to the students.

Tell the students that if the deadlock priority is set low for a transaction, then the transaction has a higher probability of becoming the deadlock victim in a deadlock situation.

Mention that lock_timeout is used on a transaction if you do not want a transaction to wait for an indefinite period. Using lock_timeout you can mention in milliseconds how long a transaction will wait for a lock to open.

Tell the students that whenever an ad hoc DML statement is executed, SQL Server, by default, treats the statement as a transaction and commits the transaction. This mode is called the auto commit mode.

Tell the students that although SQL Server 2000 uses dynamic locking, it is still very important for you to differentiate between the different locking modes. The different types of locks used by SQL Server 2000 are shared, update, exclusive, intent, schema, and bulk-update.

Tips on handling demo related to Locks Implementation

Use the example given below to demonstrate locks implementation. This example shows the implementation of Exclusive locks.

1. Open two SQL Analyzer windows.
2. In first window, execute the following commands:
   
   ```
   Begin Tran
   Select * from position
   INSERT INTO RecruitmentAgencies
   VALUES('0010','Head Hunters','223 Hill Street','Cleveland','Ohio','44167-5943','(440)345-8872','(440)345-8943',7,1000)
   ```
3. In second window, execute the following commands:
   
   ```
   Begin Tran
   Select * from RecruitmentAgencies
   Insert into position values('0999','Sales and Marketing',8,2006,2)
   ```

Output:
Notice that the result does not appear in the second window and the query appears to be in the processing stage.

**Reason:**

In the first SQL analyzer window, the query is trying to insert a record in RecruitmentAgencies table. When a record is getting inserted in a table, Exclusive lock is implemented. Since the end transaction statement is not encountered, the lock is not released and hence you are not able to do query on the RecruitmentAgencies table from the second SQL analyzer window.

**Solution:**

Now to release the lock either execute the command Rollback or Commit in the first SQL Query analyzer window. Observe that the output appears in the second window.

### Examples & Analogies

#### Deadlock

Suppose a delete trigger called Trigger1 has been defined on a table called Table1 and there is a delete trigger called Trigger2 on another table called Table2. Trigger1 deletes a row on Table2 and Trigger2 deletes a row on Table1. If you delete a row in Table1, then Trigger1 will try to obtain an exclusive lock on Table2 and Trigger2, which will be fired now, will try to acquire an exclusive lock on Table1 that is already locked. Both these transactions wait for the other to release the locks imposed by them. This causes a stalemate because neither application can release its locks and finish its session while waiting for the other application to release its locks. SQL Server automatically fixes this by choosing one application, forcing it to release the lock and allowing the other session to continue. By setting the DEADLOCK PRIORITY, you can decide which session is more likely to be the next deadlock looser. SQL Server will release the lock of the session that has lower priority. Set the DEADLOCK PRIORITY using the following syntax: SET DEADLOCK_PRIORITY {LOW | NORMAL}.

### FAQ

**Q:** What are the different types of transaction modes available in SQL Server 2000?

Implicit and explicit.

**Q:** How does the transaction log help in transaction management?

Every transaction is recorded in the transaction log to maintain database consistency and aid in recovery.

**Q:** When will SQL Server use an update lock?

When updating rows, SQL Server first searches for the records and uses a shared lock in the process. Once the records are located, the shared lock is upgraded to an exclusive lock. If another transaction has applied a shared lock on the resource, the shared lock imposed while searching for the records cannot be upgraded. To avoid this SQL Server uses an update lock while updating records.

**Q:** In case there are a lot of transactions, which lock will you request for while updating a row in a table?

You should request for an intent lock. Intent locks impose locks higher up in the hierarchy and only those locks are compared instead of comparing locks in the lower level.

### Additional Inputs

Timeouts can be used to prevent deadlocks.

For multiple transactions running simultaneously on a SQL Server, you can define their isolation level to balance between concurrency and data integrity. By choosing the right transaction isolation level can improve performance of the SQL Server queries. There are four transaction isolation levels:
- **Read Committed**  This is the default isolation level.
- **Read Uncommitted**  The restriction in this isolation level is the least as there are no shared or exclusive locks. This allows data updates before the transaction is over.
- **Repeatable Read**  In this isolation level, rows can be added but existing data cannot be updated.
- **Serializable**  Data integrity is the highest in this isolation level but concurrency between transactions is very low. Data involved in this transaction isolation level is locked. Transactions with this isolation level execute one by one.

The syntax for setting transaction isolation levels is shown below:

```sql
SET TRANSACTION ISOLATION LEVEL {READ COMMITTED|READ UNCOMMITTED|REPEATABLE READ|SERIALIZABLE}
```

When an INSENSITIVE cursor is created SQL Server stores the result set of the cursor in a temporary table in the tempdb database. This result set does not get updated with changes made to the base table(s). Apart from this the cursor itself is a read-only cursor i.e. it cannot be updated. The following code declares an INSENSITIVE cursor:

```sql
DECLARE curPublishers CURSOR
FOR SELECT * FROM publishers
OPEN curPublishers
FETCH NEXT FROM curPublishers
```
## Solutions: Just a Minute...

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Question: Identify the following properties of a single unit of work:</td>
</tr>
<tr>
<td></td>
<td>a. Any data modification made by concurrent transactions must be isolated from the modifications made by other concurrent transactions.</td>
</tr>
<tr>
<td></td>
<td>b. All the data modifications are performed or none of them are performed.</td>
</tr>
<tr>
<td></td>
<td>c. Any change in data by a completed transaction remains permanently in effect in the system.</td>
</tr>
<tr>
<td></td>
<td>d. All data is in a consistent state after a transaction is completed successfully.</td>
</tr>
<tr>
<td></td>
<td>Answers:</td>
</tr>
<tr>
<td></td>
<td>a. Isolation</td>
</tr>
<tr>
<td></td>
<td>b. Atomicity</td>
</tr>
<tr>
<td></td>
<td>c. Durability</td>
</tr>
<tr>
<td></td>
<td>d. Consistency</td>
</tr>
<tr>
<td>2</td>
<td>Question: Which of the concurrency problems does the following refer to:</td>
</tr>
<tr>
<td></td>
<td>a. When two or more transactions try to modify the same row that is based on the originally selected value.</td>
</tr>
<tr>
<td></td>
<td>b. When a document is distributed to people that contains information which no longer exists in the original document.</td>
</tr>
<tr>
<td></td>
<td>Answer:</td>
</tr>
<tr>
<td></td>
<td>a. Lost updates</td>
</tr>
<tr>
<td></td>
<td>b. Uncommitted Dependency</td>
</tr>
<tr>
<td>3</td>
<td>Question: What is a deadlock? How can you customize deadlocks?</td>
</tr>
<tr>
<td></td>
<td>Answer:</td>
</tr>
<tr>
<td></td>
<td>A deadlock is a situation in which two users (or transactions) have locks on separate objects, and each user is waiting for a lock on the other’s object. It usually occurs in a multi-user environment.</td>
</tr>
<tr>
<td></td>
<td>The SET DEADLOCK_PRIORITY command is used to customize deadlocks.</td>
</tr>
</tbody>
</table>

### Configuring the Resource for the Demonstrations

Note: Login as the same user by opening two different Instances for Locks demonstration. You can execute the transaction of user1 in the first window and transaction of user2 in the second window. You will observe that when the transaction of user1 in the first window is not committed, the transaction in the second window of user2 would also not get committed.
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 1              | ---Implement 1  
declare @fname char(20)  
declare @lname char(20)  
declare @testscore int  
declare @msg varchar(120)  
declare candidatechk cursor  
for  
select vfirstname,vlastname,sitestscore from externalcandidate  
open candidatechk  
fetch candidatechk into @fname,@lname,@testscore  
Print 'First Name Last Name Marks Result'  
Print '--------------------------------------------------------------------------------'  
While (@@fetch_status = 0)  
begin  
  if(@testscore >= 90)  
  begin  
    SELECT @msg = 'Selected for the Interview'  
  end  
  else  
  if(@testscore < 90)  
  begin  
    SELECT @msg = 'Not Selected for the Interview'  
  end  
  print @fname + @lname +  
CONVERT(char,@testscore)+@msg  
  fetch next FROM candidatechk INTO  
@fname,@lname,@testscore  
end  
close candidatechk  
deallocate candidatechk |
| 2              | --Implement 2  
CREATE PROCEDURE prcGenOrder  
@OrderNo char(6)OUTPUT |
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>as</td>
<td>SELECT @OrderNo=Max(cOrderNo) FROM Orders</td>
</tr>
<tr>
<td></td>
<td>SELECT @OrderNo=</td>
</tr>
<tr>
<td></td>
<td>CASE</td>
</tr>
<tr>
<td></td>
<td>WHEN @OrderNo &gt;=0 and @OrderNo&lt;9 Then</td>
</tr>
<tr>
<td></td>
<td>'00000'+Convert(char,@OrderNo+1)</td>
</tr>
<tr>
<td></td>
<td>WHEN @OrderNo&gt;=9 and @OrderNo&lt;99 Then</td>
</tr>
<tr>
<td></td>
<td>'0000'+Convert(char,@OrderNo+1)</td>
</tr>
<tr>
<td></td>
<td>WHEN @OrderNo&gt;=99 and @OrderNo&lt;999 Then</td>
</tr>
<tr>
<td></td>
<td>'000'+Convert(char,@OrderNo+1)</td>
</tr>
<tr>
<td></td>
<td>WHEN @OrderNo&gt;=999 and @OrderNo&lt;9999 Then</td>
</tr>
<tr>
<td></td>
<td>'00'+Convert(char,@OrderNo+1)</td>
</tr>
<tr>
<td></td>
<td>WHEN @OrderNo&gt;=9999 and @OrderNo&lt;99999 Then</td>
</tr>
<tr>
<td></td>
<td>'0'+Convert(char,@OrderNo+1)</td>
</tr>
<tr>
<td></td>
<td>WHEN @OrderNo&gt;=99999 Then</td>
</tr>
<tr>
<td></td>
<td>Convert(char,@OrderNo+1)</td>
</tr>
<tr>
<td></td>
<td>END</td>
</tr>
<tr>
<td></td>
<td>RETURN</td>
</tr>
<tr>
<td></td>
<td>CREATE PROCEDURE prcOrder(@CartId char(6),@ShopperId char(6))</td>
</tr>
<tr>
<td></td>
<td>As</td>
</tr>
<tr>
<td></td>
<td>Begin transaction</td>
</tr>
<tr>
<td></td>
<td>DECLARE @Order char(6)</td>
</tr>
<tr>
<td></td>
<td>exec prcGenOrder @Order OUTPUT</td>
</tr>
<tr>
<td></td>
<td>SELECT @Order</td>
</tr>
<tr>
<td></td>
<td>INSERT Orders</td>
</tr>
<tr>
<td></td>
<td>VALUES(@Order,getdate(),@CartId,@ShopperId,null,null,null,null,null,null)</td>
</tr>
<tr>
<td></td>
<td>INSERT INTO OrderDetail(cOrderNo,cToyId,siQty)</td>
</tr>
<tr>
<td></td>
<td>SELECT @Order,cToyId,siQty FROM ShoppingCart</td>
</tr>
<tr>
<td></td>
<td>WHERE cCartId = @CartId</td>
</tr>
<tr>
<td></td>
<td>Update OrderDetail</td>
</tr>
<tr>
<td></td>
<td>SET mToyCost=mToyRate*siQty</td>
</tr>
<tr>
<td></td>
<td>FROM OrderDetail JOIN Toys</td>
</tr>
<tr>
<td>Problem number</td>
<td>Solution</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| 3             | ON OrderDetail.cToyId=Toys.cToyId  
WHERE cOrderNo=@Order  
Commit transaction  
Return  

--Verify  
exec prcOrder '000001','000001' |

|               | **--Implement 3**  
CREATE PROCEDURE prcOrderDetail  
@OrderNo char(6),  
@ToyId char(6),  
@WrapperId char(3)  
as  
BEGIN TRANSACTION  
Update OrderDetail  
SET cGiftWrap='Y', cWrapperId=@WrapperId  
WHERE cOrderNo=@OrderNo and cToyId=@ToyId  

Update Orders  
Set  
mGiftWrapCharges=mGiftWrapCharges+mWrapperRate *siQty  
FROM Orders JOIN OrderDetail  
ON Orders.cOrderNo=OrderDetail.cOrderNo  
JOIN Wrapper  
ON OrderDetail.cWrapperId=Wrapper.cWrapperId  
and Orders.cOrderNo=@OrderNo and cToyId=@ToyId  
COMMIT TRANSACTION  
RETURN  

--Verify  
exec prcOrderDetail '000010','000020','005' |

| 4             | **--Implement 4**  
CREATE Trigger trgOrderQty  
ON OrderDetail  
FOR UPDATE |
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As</td>
</tr>
<tr>
<td></td>
<td>UPDATE OrderDetail</td>
</tr>
<tr>
<td></td>
<td>SET mToyCost=INSERTED.siQty*mToyRate</td>
</tr>
<tr>
<td></td>
<td>FROM OrderDetail</td>
</tr>
<tr>
<td></td>
<td>JOIN Inserted</td>
</tr>
<tr>
<td></td>
<td>ON OrderDetail.cOrderNo=Inserted.cOrderNo</td>
</tr>
<tr>
<td></td>
<td>JOIN Toys</td>
</tr>
<tr>
<td></td>
<td>ON Inserted.cToyId=Toys.cToyId</td>
</tr>
<tr>
<td></td>
<td>--Verify</td>
</tr>
<tr>
<td></td>
<td>Update OrderDetail</td>
</tr>
<tr>
<td></td>
<td>SET siQty=3</td>
</tr>
<tr>
<td></td>
<td>WHERE cOrderNo='000002'</td>
</tr>
<tr>
<td></td>
<td>SELECT * FROM OrderDetail</td>
</tr>
<tr>
<td></td>
<td>SELECT * FROM Toys</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5</th>
<th>--Implement 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>declare @orddate char(15)</td>
</tr>
<tr>
<td></td>
<td>declare @totalsales money</td>
</tr>
<tr>
<td></td>
<td>declare @msg varchar(120)</td>
</tr>
<tr>
<td></td>
<td>declare ordercheck cursor for</td>
</tr>
<tr>
<td></td>
<td>select dorderdate,sum(mtotalcost) from orders</td>
</tr>
<tr>
<td></td>
<td>group by dorderdate</td>
</tr>
<tr>
<td></td>
<td>open ordercheck</td>
</tr>
<tr>
<td></td>
<td>fetch ordercheck</td>
</tr>
<tr>
<td></td>
<td>into @orddate,@totalsales</td>
</tr>
<tr>
<td></td>
<td>WHILE (@@fetch_status = 0)</td>
</tr>
<tr>
<td></td>
<td>BEGIN</td>
</tr>
<tr>
<td></td>
<td>IF(@totalsales &lt; 170)</td>
</tr>
<tr>
<td></td>
<td>BEGIN</td>
</tr>
<tr>
<td></td>
<td>SELECT @msg = 'Low sales on ' + @orddate + ' Sale value ' + CONVERT(char(10), @totalsales)</td>
</tr>
<tr>
<td></td>
<td>END</td>
</tr>
<tr>
<td></td>
<td>ELSE IF(@totalsales &gt; 170)</td>
</tr>
<tr>
<td></td>
<td>BEGIN</td>
</tr>
<tr>
<td></td>
<td>SELECT @msg = 'High sales on ' + @orddate + ' Sale value ' + CONVERT(char(10), @totalsales)</td>
</tr>
<tr>
<td></td>
<td>END</td>
</tr>
<tr>
<td></td>
<td>END</td>
</tr>
<tr>
<td></td>
<td>END</td>
</tr>
<tr>
<td>Problem number</td>
<td>Solution</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td>1.78</td>
<td>Implement a Database Design Using Microsoft SQL Server ©NIIT</td>
</tr>
</tbody>
</table>
| 6 | --Implement 6  
declare @orddate char(15)  
declare @totalshippingcharges money  
declare @msg varchar(120)  
declare ordershipcharge cursor for  
select dorderdate,sum(mshippingcharges) from orders  
group by dorderdate  
open ordershipcharge  
fetch ordershipcharge  
into @orddate,@totalshippingcharges  
While (@@fetch_status = 0)  
begin  
    select @msg = @orddate + CONVERT(char(10), @totalshippingcharges)  
    select @msg  
fetch next FROM ordershipcharge INTO @orddate,@totalshippingcharges  
end  
close ordershipcharge  
deallocate ordershipcharge |
Lesson One

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>SELECT 'Candidate Code' = cCandidateCode, 'Candidate Name' = vFirstName + vLastName FROM ExternalCandidate</td>
</tr>
<tr>
<td>1b</td>
<td>SELECT 'Department Code' = cDepartmentCode, 'Department Name' = vDepartmentName FROM Department</td>
</tr>
<tr>
<td>2a</td>
<td>SELECT cToyId, mToyRate, vToyName FROM Toys WHERE vToyName = 'Baby Minnie' or vToyName = 'Dune Racer'</td>
</tr>
<tr>
<td>2b</td>
<td>SELECT * FROM Toys WHERE vToyName = 'Large Duck'</td>
</tr>
<tr>
<td>2c</td>
<td>SELECT * FROM Toys WHERE mToyRate &gt; 19.0000 and mToyRate &lt; 30.0000</td>
</tr>
<tr>
<td>2d</td>
<td>SELECT 'Toy ID' = cToyId, 'Month' = siMonth, 'Year' = iYear, 'Total Toys Sold' = iTotalSold FROM PickOfMonth</td>
</tr>
</tbody>
</table>

Lesson Two

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SELECT * from department where vDepartmentName Like 'T%'</td>
</tr>
<tr>
<td>2</td>
<td>Check the output in the Query Analyzer.</td>
</tr>
<tr>
<td>3</td>
<td>Select * from Department WHERE vDepartmentHead IS NULL</td>
</tr>
<tr>
<td>4</td>
<td>Select Top 5 * from ExternalCandidate WHERE cRating &gt;= 5 and cRating &lt;= 7 Order by vFirstName</td>
</tr>
<tr>
<td>5</td>
<td>Select cCategoryId, cToyId, vToyName, mToyRate FROM Toys ORDER BY cCategoryid COMPUTE AVG (mToyRate) BY cCategoryId</td>
</tr>
</tbody>
</table>
### Lesson Three

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.</td>
<td>The compute by list does not match the order by list.</td>
</tr>
</tbody>
</table>

### Lesson Three

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SELECT DISTINCT 'Employee Name'= vFirstName + vLastName, 'Department Name '= vDepartmentName, 'Annual Salary'=mAnnualSalary From Employee join Department on Employee.cDepartmentCode = Department.cDepartmentCode join Annualsalary on Employee.cEmployeeCode = Annualsalary.cEmployeeCode</td>
</tr>
<tr>
<td>2</td>
<td>It displays the redundant column data from the three tables.</td>
</tr>
<tr>
<td>3</td>
<td>Select cCountry, cEmployeeCode, vFirstName, vLastName From Country left outer join Employee On Country.cCountryCode = Employee.cCountryCode</td>
</tr>
<tr>
<td>4</td>
<td>select s1.vFirstName, s1.vLastName, s1.cCity, s2.vFirstName,s2.vLastName from shopper s1 join shopper s2 on s1.cCity=s2.cCity and s1.cShopperId&gt;s2.cShopperId</td>
</tr>
</tbody>
</table>

### Lesson Four

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>All queries in an SQL statement containing a UNION operator must have equal number of expressions in their target lists.</td>
</tr>
<tr>
<td>2</td>
<td>When two tables are joined using the UNION operator, the data is displayed in the sequence in which the fields are mentioned.</td>
</tr>
<tr>
<td>3</td>
<td>The initial size and the maximum size of the database have not been mentioned.</td>
</tr>
</tbody>
</table>
## Lesson Five

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>sp_addtype Code,'char(4)', 'NOT NULL'</code></td>
</tr>
<tr>
<td>2</td>
<td><code>ALTER TABLE OrderDetail ADD CONSTRAINT chkQty CHECK (siQty &gt; 0)</code></td>
</tr>
</tbody>
</table>

## Lesson Six

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>CREATE RULE rulCollegeCode AS @CollegeCode LIKE '[0-9][0-9][0-9][0-9]' sp_bindrule rulCollegeCode, 'College.cCollegeCode'</code></td>
</tr>
<tr>
<td>2</td>
<td><code>CREATE DEFAULT defBrandId AS '001' sp_bindefault defBrandId, 'Toys.cBrandId'</code></td>
</tr>
<tr>
<td>3</td>
<td><code>SELECT * INTO ToysToOrder FROM Toys WHERE siToyQoH &lt; 40</code></td>
</tr>
</tbody>
</table>

## Lesson Seven

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>CREATE NONCLUSTERED INDEX idxEmployeeFirstName ON Employee (vFirstName)</code></td>
</tr>
<tr>
<td>2</td>
<td><code>CREATE UNIQUE NONCLUSTERED INDEX idxSocialSecurityNo ON Employee (cSocialSecurityNo)</code></td>
</tr>
<tr>
<td>3</td>
<td><code>CREATE NONCLUSTERED INDEX idxRequisition ON Requisition (cRequisitionCode, cPositionCode)</code></td>
</tr>
</tbody>
</table>

## Lesson Eight

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>CREATE VIEW vwToyDescriptionRate WITH ENCRYPTION AS SELECT vToyDescription, mToyRate FROM Toys</code></td>
</tr>
<tr>
<td>2</td>
<td><code>CREATE VIEW vwShopper AS SELECT vFirstName, vLastName, vEmailId from</code></td>
</tr>
</tbody>
</table>
Lesson Nine

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CREATE PROCEDURE prcEmployeeName AS BEGIN SELECT vFirstName, vLastName FROM Employee END</td>
</tr>
<tr>
<td>2</td>
<td>CREATE PROC prcShopperDetails @ShopperState char(15) AS BEGIN SELECT vFirstName, vLastName, vAddress, cState FROM Shopper WHERE cState = @ShopperState END</td>
</tr>
<tr>
<td>3</td>
<td>ALTER PROCEDURE prcCategoryDetails WITH RECOMPILE AS SELECT cCategory, vDescription FROM Category</td>
</tr>
</tbody>
</table>

Lesson Ten

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CREATE TRIGGER trgInsertOrderDetail ON OrderDetail FOR insert AS IF NOT EXISTS (SELECT * FROM Toys JOIN Inserted on Toys.cToyId = Inserted.cToyId) BEGIN PRINT 'Toy not found in the Toys table.' ROLLBACK TRANSACTION END RETURN</td>
</tr>
<tr>
<td>Problem number</td>
<td>Solution</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| 2             | CREATE TRIGGER trgDeleteCollege  
ON College  
FOR delete  
AS  
IF EXISTS (SELECT * FROM ExternalCandidate JOIN  
Deleted on  
    ExternalCandidate.cCollegeCode =  
    Deleted.cCollegeCode)  
BEGIN  
    ROLLBACK TRANSACTION  
END  
RETURN |
| 1             | BEGIN TRANSACTION  
UPDATE Toys  
SET mToyRate = mToyRate + 5  
IF (SELECT SUM(mToyRate) FROM Toys) > 100  
BEGIN  
    ROLLBACK TRANSACTION  
END  
ELSE  
BEGIN  
    COMMIT TRANSACTION  
END |
| 2             | declare @cToyId char(6)  
declare @siQty int  
declare Toychk cursor  
for select cToyId,siQty  
from ShoppingCart  
open Toychk  
fetch Toychk into @cToyId,@siQty  
While (@@fetch_status = 0)  
begin  
    Update Toys  
    set siToyQoh = siToyQoh - @siQty  
    from Toys  
    where cToyId = @cToyId  
    fetch next FROM Toychk into @cToyId,@siQty  
end  
close Toychk  
deallocate Toychk |
| 3             | CREATE Trigger trgOrderDetail  
ON OrderDetail  
for insert  
as  
UPDATE Toys  
SET siToyQoh=siToyQoh-siQty  
FROM Toys  
JOIN INSERTED  
ON Toys.cToyId=Inserted.cToyId |
<table>
<thead>
<tr>
<th>Problem number</th>
<th>Solution</th>
</tr>
</thead>
</table>
|               | --Verify  
insert into OrderDetail  
values('00009', '000001', 2, 'N', NULL, NULL, 39.9800)  
select * from toys |
ADDITIONAL REFERENCE

- Mike Gunderloy and Joseph L Jorden, Mastering SQL Server 2000
- Marci Frohock Garcia, Jamie Reding, Edward Whalen, Steve Adrien DeLuca, Microsoft® SQL Server™ 2000 Administrator's Companion
- Kalen Delaney, Inside Microsoft® SQL Server™ 2000
- Graeme Malcolm, Programming Microsoft® SQL Server™ 2000 with XML
- Rebecca Riordan, Microsoft® SQL Server™ 2000 Programming Step by Step
- Joyjit Mukherjee, MCDBA Administering SQL Server 2000 Study Guide (Exam 70-228)
- Sharon Bjeletich, et al. ,McSe Microsoft Sql Server 2000 Database Design and Implementation : Readiness Review Exam 70-229 (Pro-Certification)
LIST OF WEB SITES

- SQL Magazine: http://www.sqlmag.com/
- Find SQL Server add-ons, patches, service packs, tools, and trial versions from Microsoft: http://www.microsoft.com/sql/downloads/default.asp
- Professional Association for SQL Server. http://www.sqlpass.org/
## SESSION PLAN: SQL 2000

<table>
<thead>
<tr>
<th>Cycle #</th>
<th>Activity/Problem No.</th>
<th>Duration (In Mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cycle 1</strong></td>
<td>OCR2</td>
<td></td>
</tr>
<tr>
<td>OCR2</td>
<td>Lesson 1: Objectives</td>
<td>5</td>
</tr>
<tr>
<td>OCR2</td>
<td>Getting Started</td>
<td>10</td>
</tr>
<tr>
<td>OCR2</td>
<td>1.D.1</td>
<td>25</td>
</tr>
<tr>
<td>OCR2</td>
<td>1.D.2</td>
<td>10</td>
</tr>
<tr>
<td>OCR2</td>
<td>1.D.3</td>
<td>10</td>
</tr>
<tr>
<td>OCR2</td>
<td>Arithmetic Operators</td>
<td>7</td>
</tr>
<tr>
<td>OCR2</td>
<td>Search Based on Conditions</td>
<td>5</td>
</tr>
<tr>
<td>OCR2</td>
<td>1.D.4</td>
<td>15</td>
</tr>
<tr>
<td>OCR2</td>
<td>More on Operators</td>
<td>15</td>
</tr>
<tr>
<td>OCR2</td>
<td>Summary</td>
<td>10</td>
</tr>
<tr>
<td>OCR2</td>
<td><strong>Total</strong></td>
<td><strong>112</strong></td>
</tr>
<tr>
<td><strong>Cycle 2</strong></td>
<td>OCR2</td>
<td></td>
</tr>
<tr>
<td>OCR2</td>
<td>Lesson 2: Objectives</td>
<td>5</td>
</tr>
<tr>
<td>OCR2</td>
<td>2.D.1</td>
<td>10</td>
</tr>
<tr>
<td>OCR2</td>
<td>2.D.2</td>
<td>10</td>
</tr>
<tr>
<td>OCR2</td>
<td>2.D.3</td>
<td>10</td>
</tr>
<tr>
<td>OCR2</td>
<td>2.D.4</td>
<td>15</td>
</tr>
<tr>
<td>OCR2</td>
<td>2.D.5</td>
<td>20</td>
</tr>
<tr>
<td>OCR2</td>
<td>Grouping Result Sets</td>
<td>5</td>
</tr>
<tr>
<td>OCR2</td>
<td>2.D.6</td>
<td>20</td>
</tr>
<tr>
<td>OCR2</td>
<td>Summary</td>
<td>10</td>
</tr>
<tr>
<td>OCR2</td>
<td><strong>Total</strong></td>
<td><strong>105</strong></td>
</tr>
<tr>
<td><strong>Cycle 3</strong></td>
<td>OCR2</td>
<td></td>
</tr>
<tr>
<td>OCR2</td>
<td>Lesson 3: Objectives</td>
<td>5</td>
</tr>
<tr>
<td>OCR2</td>
<td>3.D.1</td>
<td>15</td>
</tr>
<tr>
<td>OCR2</td>
<td>3.D.2</td>
<td>12</td>
</tr>
<tr>
<td>OCR2</td>
<td>3.D.3</td>
<td>15</td>
</tr>
<tr>
<td>OCR2</td>
<td>Joins</td>
<td>10</td>
</tr>
<tr>
<td>OCR2</td>
<td>3.D.4</td>
<td>12</td>
</tr>
<tr>
<td>OCR2</td>
<td>3.D.5</td>
<td>10</td>
</tr>
<tr>
<td>OCR2</td>
<td>More on Joins</td>
<td>15</td>
</tr>
<tr>
<td>OCR2</td>
<td>Summary</td>
<td>10</td>
</tr>
<tr>
<td>OCR2</td>
<td><strong>Total</strong></td>
<td><strong>104</strong></td>
</tr>
<tr>
<td><strong>Cycle 4</strong></td>
<td>OCR1</td>
<td></td>
</tr>
<tr>
<td>OCR1</td>
<td>Lesson 4: Objectives</td>
<td>5</td>
</tr>
<tr>
<td>OCR1</td>
<td>Subqueries</td>
<td>20</td>
</tr>
<tr>
<td>OCR1</td>
<td>4.D.1</td>
<td>20</td>
</tr>
<tr>
<td>OCR1</td>
<td>More on Subqueries</td>
<td>30</td>
</tr>
<tr>
<td>OCR1</td>
<td>4.D.2</td>
<td>15</td>
</tr>
<tr>
<td>OCR1</td>
<td>4.D.3</td>
<td>15</td>
</tr>
<tr>
<td>OCR1</td>
<td>Summary</td>
<td>10</td>
</tr>
<tr>
<td>Cycle #</td>
<td>Activity/Problem No.</td>
<td>Duration (In Mins)</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>OCR2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lesson 4: Databases</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Recap</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>110</td>
</tr>
<tr>
<td>OCR1</td>
<td>Lesson 5: Objectives</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>5.D.1</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>5.P.1</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>5.D.2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>User-Defined Datatypes</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>5.D.3</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Data Integrity</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Creating Constraints</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>5.D.4</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>5.P.2</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>118</td>
</tr>
<tr>
<td>OCR1</td>
<td>Lesson 6: Objectives</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6.D.1</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>6.D.2</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Maintaining Data</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6.D.3</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>6.D.4</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>6.D.5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6.D.6</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Truncating a Table</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>115</td>
</tr>
<tr>
<td>OCR2</td>
<td>Lesson 7: Objectives</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Getting Started</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>7.D.1</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Index Tuning Wizard</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>7.D.2</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Index Enhancements</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>115</td>
</tr>
<tr>
<td>OCR1</td>
<td>Lesson 8: Objectives</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Getting Started</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>8.D.1</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Altering, Dropping, and Renaming Views</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>8.D.2</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>8.P.1</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Programming in SQL Server</td>
<td>10</td>
</tr>
<tr>
<td>Cycle #</td>
<td>Activity/Problem No.</td>
<td>Duration (In Mins)</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td></td>
<td>8.D.3</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>More Constructs</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>116</strong></td>
</tr>
<tr>
<td>OCR2</td>
<td>Lesson 9: Objectives</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Getting Started</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>9.D.1</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>9.D.2</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td><strong>9.D.3</strong></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>9.D.4</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td><strong>9.D.5</strong></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>9.D.6</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>110</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle8</th>
<th>OCR1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lesson 10: Objectives</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>10.D.1</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>10.P.1</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>10.D.2</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10.D.3</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10.P.2</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10.D.4</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10.D.5</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Enforcing Data Integrity Through Triggers</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OCR2</th>
<th>Lesson 11: Objectives</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11.D.1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>11.D.2</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>11.D.3</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>11.D.4</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Deadlock</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Cursors</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>11.D.5</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>11.P.1</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>115</strong></td>
</tr>
</tbody>
</table>
