LAB 4

WORKING WITH DATA

TYPES

This lab contains the following exercises and activities:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Exercise 4.1 | | | Create Alias Data Types | |
| Exercise 4.2 | | Using Date and Time Data Types | |
| Exercise 4.3 | | Implementing a Sparse Column | |
| Exercise 4.4 | | Using Page and Row Compression | |
| Exercise 4.1 | | Create a Data Type using Object Explorer | |
| Scenario | | You have been tasked by your company to support the Enterprise Planners and the Developers by creating needed objects in SQL Server. The Enterprise Planners have completed their IDEF(0), IDEF(1X) and affinity analysis; the developers are designing the database table layouts (schema). Your job is to create needed objects. Your first task is to create a custom data type. | |
| Duration | | This task should take less than fifteen minutes. | |
| Setup | | This task requires little setup. All you need is access to a copy of SQL Server Enterprise Edition or Developer Edition and a computer that meets the requirements to run it. | |
| Caveats | | This task has no special concerns. | |
| Procedure | | Use SQL Server Management Studio’s Object Explore and included Query Editor to complete these tasks. | |
| Equipment Used | | Although several editions of SQL Server exist, you will be working with the either the Enterprise Edition or Developer Edition. | |
| Objective | | To understand two methods of creating alias data types for use in table definitions and other development tasks. | |
| Criteria for Completion | | You have completed this task when check Object Explorer and find your new definitions recorded in system tables. | |

 PART A: Creating an Alias Data type using the Object Explorer

1. Start **SQL Server Management Studio**. Connect to your default instance by assuring Database Engine, <YourServerName> and windows authentication are listed. Click **Connect**.

2. If Object Explorer is not visible, click **Object Explorer** on the View menu.

3. In Object Explorer, expand **Databases**, **AdventureWorks**, **Programmability** and **Types**.

4. Right-click **Types** and then click **New User-Defined Data Type**…

5. Enter the following information:

Schema: dbo

Name: CountryCode

Data type: Char

Length: 2

Allow NULLs: Selected

 PART B: Creating an Alias Data type using Transact-SQL Code

1. In SQL Server Management Studio, click the **New Query** button on the toolbar.

2. In the new, blank query window, type the following Transact-SQL code:

USE AdventureWorks

CREATE TYPE dbo.EmailAddress

FROM varchar(6)

NULL;

3. Click the **Execute** button on the toolbar or Press F5.

 PART C: Verifying Results

1. Right-click the User-defined Data types folder in Object Explore and then click Refresh.

2. Verify that both CountryCode and EmailAddress have both been added to the database.

|  |  |
| --- | --- |
| Exercise 4.2 | Using Date and Time Data Types |
| Scenario | SQL Server includes some new date and time related data types and you wish to use these new types but first you must understand how they differ from previous data types. |
| Duration | This task should take approximately 60 minutes. |
| Setup | For this task, you need access to the machine you installed SQL Server on. |
| Caveat | This task doesn’t have any caveats. |
| Procedure | In this task, you will develop various queries to explore how the data types can be used. |
| Equipment Used | For this task, you need access to the machine you installed SQL Server. |
| Objective | To explore the uses and features of date and time data types. |
| Criteria for Completion | This task is complete when you understand the new data types. |

 PART A: Create Some Date and Time Variables

1. Open **SQL Server Management Studio**. It does not matter which database is used in this exercise.

2. Open your **Query Editor**. You will be executing a series of queries in the steps below.

3. To use the system date and time, enter and execute the following code:

-- Code set 1

DECLARE @current\_date date = GETDATE();

DECLARE @current\_time time(7) = GETDATE();

DECLARE @current\_datetime datetime = GETDATE();

SELECT @current\_date AS 'Today',

@current\_time AS 'Right Now',

@current\_datetime AS 'Date & Time';

-- Now round the time value to whole seconds

DECLARE @short\_current\_time time(0) = @current\_time;

SELECT @short\_current\_time, @current\_time;

GO

|  |  |
| --- | --- |
| NOTE | Notice that you can set the precison for a time data type to any value from 0 through 7. |

4. To use fixed date and time values and specific parts of the data, enter and execute the following code:

-- Code set 2

-- Use the new date, time, and datetimeoffset data types

DECLARE @MyDate date = '07/13/2010';

DECLARE @MyTime time = '12:30:15';

DECLARE @MyDatetimeoffset datetimeoffset = '07/13/2010 12:30:15 -8:00';

SELECT DATENAME(month, @MyDate) AS 'Month Name';

SELECT DATENAME(hour, @MyTime) As 'Hour Name';

SELECT CONVERT(VARCHAR(30), @MyDatetimeoffset) AS 'DATETIMEOFFSET';

GO

|  |  |
| --- | --- |
| NOTE | Notice that you can now include a time zone value using the datetimeoffset data type. The DATENAME function has existed in SQL Server for a while and can be used to extract various parts of a date or time value. |

5. Suppose your database contains employee work hours collected by a timeclock. Can you calculate an amount of time worked? Enter and execute the following code to find out.

-- Code set 3

-- Calculate a time period with time data

DECLARE @starttime time = '09:00:00';

DECLARE @stoptime time = '17:00:00';

SELECT DATEDIFF(hour, @starttime, @stoptime) AS 'Work Hours';

GO

6. Suppose your desired date data includes historical dates from a few centuries ago. How far back in time can you go? Enter and execute the following code to test out a few dates.

-- Code set 4

-- Explore the datatime2 data type

DECLARE @mydatetime datetime = GETDATE();

DECLARE @mydatetime2 datetime2(7) = GETDATE();

SELECT @mydatetime, @mydatetime2;

SET @mydatetime = '1776/07/04';

SET @mydatetime2 = '1492/07/04';

SELECT @mydatetime, @mydatetime2;

GO

|  |  |
| --- | --- |
| NOTE | Notice that in the first select statement’s results you have two values that are the same except that the datatime2 data type allows for more precision in fractions of a second. The two fixed dates in 1776 and 1492 worked as well. |

7. How far back in time can you go with date data? Enter and execute the following code to find out.

-- Code set 5

DECLARE @mydatetime datetime = GETDATE();

DECLARE @mydatetime2 datetime2(7) = GETDATE();

SELECT @mydatetime, @mydatetime2;

SET @mydatetime = '1492/07/04';

SET @mydatetime2 = '1492/07/04';

SELECT @mydatetime, @mydatetime2;

DECLARE @myolddate date;

SET @mydatetime2 = '0001/01/01'

SET @myolddate = @mydatetime2;

SELECT @mydatetime2, @myolddate;

GO

8. The above code will fail because the basic datetime data type cannot handle years prior to 1753 and 1492 is way earlier then that. Change the code in some way to work around this problem and re-execute the code. You will then see that both the new date and datetime2 data types can store the date of 0001/01/01 which is the earliest possible date for datetime2 data.

9. Now experiment with date data formatting. What date does the six digit string of 07/04/10 represent?

July 4th, 2010?

April 7th, 2010?

November 4th, 2007?

April 10th in the year 0007?

Enter and execute the code below to use DATEFORMAT. Experiment and change around the code as you might like to see other options.

-- Code set 6

-- Set date format to month, day, year.

SET DATEFORMAT mdy;

GO

DECLARE @ambiguous\_date date = '07/04/10';

SELECT @ambiguous\_date AS DateVar;

-- Returns: 2010-07-04

-- Set date format to year, day, month.

SET DATEFORMAT ydm;

GO

DECLARE @datevar datetimeoffset = '1710/04/07 12:30:15 -08:00';

SELECT @datevar AS DateVar;

GO

10. If a user enters a two-digit year value in a date such as 8/31/56 – what century and 4-digit year will be used? 1956 or 2056? Enter and execute the following code to find out.

-- Code set 7

-- Experiments with Y2K dates and the century pivot value

SET DATEFORMAT mdy;

GO

DECLARE @nocentury49 datetime = '12/31/49'

DECLARE @nocentury50 date = '01/01/50'

DECLARE @withcentury date = '12/31/1949'

SELECT @nocentury49 AS 'W/O Century (49)',

@nocentury50 AS 'Without Century (50)',

@withcentury AS 'With Century (1949)'

SELECT DATEADD(dd, +1, @nocentury49) AS 'Increment 1 day',

DATEADD(dd, -1, @nocentury50) AS 'Decrement 1 day'

DECLARE @test\_AD\_2\_BC date = '01/01/2010'

SELECT DATEADD(yy, -2009, @test\_AD\_2\_BC) AS 'Lowest Possible Date'

GO

|  |  |
| --- | --- |
| Question 1 | What is the Year 2000 cutoff or pivot year?  **Answer: The year in a century above which will be treated as in the previous century and below which will be treated as in the current century using four digit dates. If you set the pivot year to 30, the four year century for today’s date will be 2010.** |

11. To determine the SQL Server pivot year setting, enter and execute the following code:

-- Code set 8

-- Determine the Pivot Year value

USE master;

GO

EXEC sp\_configure 'show advanced option', '1';

RECONFIGURE

EXEC sp\_configure 'two digit year cutoff';

SELECT value FROM sys.configurations WHERE name = 'two digit year cutoff'

GO

|  |  |
| --- | --- |
| Exercise 4.3 | Implementing a Sparse Column |
| Scenario | You want to add an email column to the existing Customers table. You have only a few email addresses initially so you want to use the sparse feature of SQL Server to reduce storage requirements. |
| Duration | This task should take approximately 30 minutes. |
| Setup | For this task, you need access to the machine you installed SQL Server on in Exercise 2.1 and the Sales database |
| Caveat | This task doesn’t have any caveats. |
| Procedure | In this task, you will set a column to be SPARSE in the Sales database. |
| Equipment Used | For this task, you need access to the machine you installed SQL Server on in Exercise 2.1 and the Sales database |
| Objective | To see how to create a column using the SPARSE option. |
| Criteria for Completion | This task is complete when the column has a SPARSE attribute. |

 PART A: Creating a Temporary Database for the Table

1. Open **SQL Server Management Studio**, and click on **New Query** to open a query window.

2. Enter and execute the following code to create a dabase and table for the exercise:

CREATE DATABASE SalesDemo;

GO

USE salesdemo

GO

CREATE TABLE customers(

CustID int NOT NULL,

Fname nvarchar(20),

Lname nvarchar(20),

Address nvarchar(20),

City nvarchar(20),

State nchar(2),

Zip nchar(5),

Phone nchar(10)

);

 PART B: Creating a New Column in the Table

1. Open **SQL Server Management Studio**, and in **Object Explorer**, expand **Databases** under your server.

2. Select **SalesDemo** as the database.

3. Insert some data into the table (feel free to add more data if you so desire):

USE salesdemo

GO

INSERT INTO customers VALUES (1, N'Jones', N'Jeremiah',

N'123 Maple St.', N'Boise', N'ID', N'87654', N'8015551212')

INSERT INTO customers VALUES (2, N'James', N'Jessie',

N'123 Maple St.', N'Stillwater', N'MN', N'55082', N'5025551212')

INSERT INTO customers VALUES (3, N'Cooper', N'D.B.',

N'c/o Postmaster', N'Ariel', N'WA', N'98603', null)

INSERT INTO customers VALUES (4, N'Brown', N'Paul',

N'4567 Green St.', N'Lexington', N'KY', N'40502', N'8595551234')

INSERT INTO customers VALUES (5, N'Medina', N'Jose',

N'789 Olive St.', N'Gila Bend', N'AZ', N'85337', N'5205551234')

INSERT INTO customers VALUES (6, N'Panther', N'Flora',

N'1234 Swamp Ln.', N'Panacea', N'FL', N'32346', N'8505551212')

INSERT INTO customers VALUES (7, N'Carter', N'Bill',

N'1 Peach Ave.', N'Macon', N'GA', N'31201', N'4781211212')

INSERT INTO customers VALUES (8, N'Washington', N'George',

N'3200 Mt. Vernon Hwy.', N'Mount Vernon', N'VA', N'22121', N'7035550011')

4. Enter the following code to add a new column to the existing table: Note that the key word SPARSE is included. This key word will cause the new column to become a sparse column.

ALTER TABLE customers

ADD Emailaddress varchar(128)SPARSE NULL;

GO

5. Create a little bit of data for this new column.

UPDATE customers SET Emailaddress = 'pbrown@someisp.com'

WHERE CustID = 4;

6. Close the query window.

|  |  |
| --- | --- |
| Question 2 | What data will exist in the Emailaddress column for those table rows that existed prior to this change?  **Answer: The Emailaddress column is defined as allowing nulls so regardless of whether or not the column is a SPARSE column, nulls will exist as the data for all rows other than the one row changed in this exercise.** |

 PART C: Verifying Results

1. In **Object Explorer**, expand the database name **SalesDemo** and then expand **Tables**.

2. Right click on **Customers**, select **Design**.

3. Select the new **Emailaddress** column from the **Design** window. Do this by clicking the left most column. Once this is done you will see a black right arrow next to Emailaddress.

4. In the **Column Properties** tab, scroll down until you see the **Is Sparse** entry.

5. Ensure that the Is Sparse property is set to Yes

6. In order to ensure that your modifications to the Customers table do not interfere with other exercises, you may wish to remove this new Emailaddress column. To do so, right click on the **Emailaddress** column name, select **Delete column** from the drop down list. The column should now disappear. Make sure that you close the window and resond with Yes to to the save changes pop-up window.

|  |  |
| --- | --- |
| Exercise 4.4 | Using Page and Row Compression |
| Scenario | To reduce the disk space used by a large table, you want to experiment with different compression options to see how much space can be saved. |
| Duration | This task should take approximately 30 minutes. |
| Setup | For this task, you need access to the machine you installed SQL Server on in Exercise 2.1 and the AdventureWorks database installed with the sample data. |
| Caveat | This task doesn’t have any caveats. |
| Procedure | In this task, you will change the compression options for a table. |
| Equipment Used | For this task, you need access to the machine you installed SQL Server on in Exercise 2.1 and the AdventureWorks database installed with the sample data |
| Objective | To set the recovery model for the AdventureWorks database. |
| Criteria for Completion | This task is complete when the AdventureWorks database is configured to use the Full recovery model as outlined in the details of this task. |

 PART A: Viewing an Existing Table Without Compression

1. Open **SQL Server Management Studio**, and in **Object Explorer**, expand **Databases** under your server.

2. Expand the **AdventureWorks** database and expand **Tables**.

3. Right click on **Production.TransactionHistoryArchive**, select **Storage** and then click on **Manage Compression**.

4. The **Data Compression Wizard** window will open. Click **Next** to continue.

5. The **Select Compression Type** window will now show you that the current compression type is ‘None’. Change this drop down box to **Page** and click on the **Calculate** button. It may take a few seconds to perform the necessary calculations.

6. Note the newly displayed values for **Current space** and **Requested compressed space**.

7. Now change the **Compression type** drop down box setting to **Row**.

8. Click on the **Calculate** button again and note the changed value under **Requested compressed space**.

9. Clickthe **Cancel** button to exit out of the wizard without actually setting a compression type.

|  |  |
| --- | --- |
| Question 3 | Which compression setting saves the most space? Is the space reduction meaningful?  **Answer: With small tables no meaningful space savings will occur.With larger tables space savings can become more meaningful.** |

 PART B: Adding PAGE or ROW Type Compression

1. Open **Management Studio**.

2. Open your **Query Editor**.

3. Run this code to create a new table:

USE SalesDemo

GO

CREATE TABLE MyNewTable(

CustomerID int NOT NULL,

Notes char(1000));

4. Now implement a compression type on this table. Choose one of the two commands below and run the code to setup compression on the table.

ALTER TABLE MyNewTable REBUILD WITH (DATA\_COMPRESSION = PAGE);

ALTER TABLE MyNewTable REBUILD WITH (DATA\_COMPRESSION = ROW);

 PART C: Removing PAGE or ROW Type Compression

1. Run this code to remove all compression from this table:

ALTER TABLE MyNewTable REBUILD WITH (DATA\_COMPRESSION = NONE);

2. Cleanup the database by dropping this new table from the database:

DROP TABLE MyNewTable;

 PART D: Sparse Columns and Compression

1. Try to implment either ROW or PAGE compression of the customers table in the sales database.

ALTER TABLE customers REBUILD WITH (DATA\_COMPRESSION = ROW);

2. A very strange and severe sounding error may occur. This is because the customers table may have a SPARSE column – Emailaddress. You cannot have both a SPARSE column and row/page compression. This column was added in a prior exercise. If you did not drop this column, then it’s presence as a SPARSE column will block the implementation of page or row compression. The reverse sequence of trying to add a SPARSE column to a compressed table will also generate an error.

3. Close the Query Window.