* Course Overview
* Introduction
* A Quick SQL Server Recap
  + Relational Database Management System(RDBMS)
* What Is a Stored Procedure
  + Piece of code the performs a set of actions or queries against the database
  + Stored in database
  + Write code once and run many times
  + Simplify and enhance security
  + More efficient code
  + Stored procedure shouldn’t
    - Return lots of result sets often
    - Do not use cursors
* Installing SQL Server and Sett..
  + Download the developer version of sql server
  + Download sql server management studio
  + Download the class files
  + Open ’00 Apply.sql’
  + In sql server management studio
    - Click ‘Query’
    - Click ‘SQLCMD Mode’
  + Set the ‘setvar path’ to class files
  + Execute ’00-Apply.sql’
  + To create a diagram of the database
    - Click on the ‘database’
    - Right click on ‘Database diagram’
    - Click ‘New Database diagram’
    - Right click on the open page and add the tables
    - Then close the dialogue to see the diagram
* Stored Procedure T-SQL State…
  + There are 3 T-SQL statements to manage store procedures
    - CREATE PROCEDURE
    - ALTER PROCEDURE
    - DROP PROCEDURE
  + Create procedure template
    - CREATE PROCEDURE <name of procedure>
    - (Parameters (optional))
    - AS
    - BEGIN;
    - [Statements here…]
    - END;
  + Store procedure must be unique in the same directory
  + Alter procedure is used to alter existing stored procedures
    - ALTER PROCEDURE <name of procedure>
    - (Parameters (optional))
    - AS
    - BEGIN;
    - [statements]
    - END;
  + Developers tend to drop procedures and create a new one
  + Drop procedure is used to remove existing stored procedures
    - DROP PROCEDURE [IF EXISTS] <name of procedure>;
* Creating a Stored Procedure
  + Example
    - USE Contacts;
    - GO
    - CREATE PROCEDURE dbo.SelectContacts
    - AS
    - BEGIN;
    - SELECT \* FROM dbo.Contacts;
    - END;
  + Since CREATE PROCEDURE needs to be the first line, use GO inbetween the two sql statements
  + To run the store procedure
    - EXEC dbo.SelectContacts;
* Managing Procedures Using S…
  + Store procedures are stored in the ‘Programmability’ folder
  + Example of drop procedure
    - DROP PROCEDURE IF EXISTS dbo.SelectContacts;
  + Before the [IF EXISTS] was introduced
    - IF EXISTS(SELECT 1 FROM sys.procedures WHERE [name] = ‘SelectContacts’;
    - BEGIN;
    - DROP PROCEDURE dbo.SelectContacts;
    - END;
* Summary
* Introduction
* The Business Requirement
  + Insert, return details, don’t insert if already exists
  + One option is ‘Embedded SQL’
    - If database schema changes, then application code needs to change
    - Hard to parameterize
  + One options is to use Object-relational Mapping(ORM)
    - Require the model to be updated if database schema changes
    - Take a lot of control away from developr(sql)
  + Stored Procedure
* The Insert Contact Stored Pro…
  + Procedure name can have spaces, but you will need to enclosing them in [] every time you call it
    - CREATE PROCEDURE dbo.[Insert Contact]
  + Example of insert
    - USE Contacts;
    - GO
    - CREATE PROCEDURE dbo.InsertContact
    - AS
    - BEGIN
    - DECLARE @FirstName VARCHAR(100),
    - @LastName VARCHAR(100),
    - @DateofBirth DATE,
    - @AllowContactByPhone BIT;
    - SELECT @FirstName = ‘Stan’,
    - @LastName = ‘Laurel’,
    - @DateOfBirth = ‘1890-06-16’,
    - @AllowContactByPhone = 0;
    - INSERT INTO dbo.Contacts (FirstName, LastName, DateOfBirth, AllowContactByPhone) VALUES (@FirstName, @LastName, @DateOfBirth,@AllowContactByPhone);
    - END;
* Executing and Testing a Store…
* Adding Parameters to a Store …
  + Parameters are variables which will be passed in by the calling program
  + Template
    - CREATE PROCEDURE <name of procedure>
    - (Parameters (optional)
    - AS
    - BEGIN;
    - Statements;
    - END;
  + Example
    - CREATE PROCEDURE dbo.INsertContact
    - (
    - @FirstName VARCHAR(40),
    - @LastName VARCHAR(40),
    - @DateOfBirth DATE,
    - @AllowContactByPhone BIT
    - )
    - AS
    - BEGIN;
    - INSERT INTO dbo.Contacts(FirstName, LastName, DateOfBirth, AllowContactByPhone) VALUES (@FirstName, @LastName, @DateOfBirth, @AllowContactByPhone);
    - END;
  + To run the store procedure
    - Can pass parameters directly or specifying name and values as key value pair
    - USE Contacts;
    - EXEC dbo.InsertContact
    - @FirstName = ‘Terry’,
    - @LastName = ‘Thomas’,
    - @DateOfBirth = ‘1911-07-14’,
    - @AllowContact
* Optional Parameters
  + Assign default value to parameter
    - @ParameterName = default
  + Ex)
    - CREATE PROCEDURE dbo.InsertContact
    - (
    - @FirstName VARCHAR(40),
    - @LastName VARCHAR(40),
    - @DateOfBirth DATE = NULL,
    - @AllowContactByPhone BIT
    - )
    - …
* Retrieving Record Identifiers
  + stored procedures should only do atomic work
    - only perform one task
    - can be made up of sub task
  + @@IDENTITY the most recent identity inserted into the entire database
    - ….
    - DECLARE @ContactId INT;
    - INSERT INTO ….
    - SELECT @ContactId = @@IDENTITY;
    - SELECT ContactId, FirstName, LastName, DateOfBirth, AllowContactByPhone
    - FROM dbo.Contacts
    - WHERE ContactId = @ContactId;
    - END;
  + always store @@Identity
    - it could change if someone else inserts a record
  + triggers run when something happens to a record in a specific table
  + instead of @@IDENTITY it's better to use SCOPE IDENTITY
    - SELECT @ContactId = SCOPE\_IDENTITY();
* Output Parameters
  + assign output to holding variable
    - …
    - CREATE PROCEDURE dbo.InsertContact
    - (
    - …
    - @ContactId INT OUTPUT
    - )
    - ….
    - SELECT @ContactId = SCOPE\_IDENTITY();
    - …
  + Parameter are mandatory
  + in the running of store procedure you have to use the keyword output
    - ….
    - EXEC dbo.InsertContact
    - @FirstName = ‘Harold’,
    - @LastName = ‘Llyod’,
    - @AllowContactByPhone = 0,
    - @ContactId = @ContactIdOut OUTPUT;
    - …..
* Using SET Options
  + SET NOCOUNT: disables count and the messages with it
  + put in top of procedure
    - CREATE PROCEDURE dbo.InsertContact
    - ….
    - AS
    - BEGIN;
    - SET NOCOUNT ON;
    - ….
    - SET NOCOUNT OFF;
    - END;
* Calling a Procedure from Anot..
  + can call a stored procedure within a stored procedure
* Adding Business Logic
  + T-SQL support control flow
  + IF EXISTS returns true is exists
  + IF NOT EXISTS
    - CREATE PROCEDURE dbo.InsertContact
    - (
    - …
    - )
    - AS
    - BEGIN;
    - SETNOCOUNT ON;
    - IF NOT EXISTS(SELECT 1 FROM dbo.Contacts
    - WHERE FirstName = @FirstName AND @LastName = LastName
    - AND DateOfBirth = @DateOfBirth)
    - BEGIN;
    - INSERT INTO dbo.Contacts ….
    - ……
    - END;
    - ….
    - END;
  + Have to wrap the code in BEGIN; END; between the code to run if the control flow check
* Summary
* Introduction
* Inheriting a Stored Procedure
  + CURSOR
    - Takes a collection of records and allows them to be processed in a sequential basis
  + Declare cursor
  + Open cursor
  + Try to obtain the first record
  + Check if found
  + Process record if found
  + Repeat until all records have been read and processed
* Alternatives to Cursors
  + Can use while loops instead of a cursor
  + Use set based logic instead
    - Insert and select
* User-defined Data Types
  + Ex
    - CREATE TYPE <name of Type>
    - FROM <base type>;
    - Or
    - CREATE TYPE <name of type>
    - AS TABLE
    - (
    - Column definitions here…
    - );
  + Ex)
    - USE Contacts;
    - GO
    - CREATE TYPE dbo.DrivingLicense
    - FROM CHAR(16) NOT NULL;
  + Can drop a type if no other objects depend on it
    - DROP TYPE IF EXISTS <name of type>;
  + User define table type is the same as a table variable
  + Ex)
    - USE Contacts;
    - DROP TYPE IF EXISTS dbo.ContactNote;
    - GO
    - CREATE TYPE dbo.ContactNote
    - AS TABLE
    - (
    - Note VARCHAR(MAX) NOT NULL
    - );
  + TVP data is stored in TempDB
* Calling a Store Procedure wi..
* Summary
* Introduction
* The Print Statement
  + Ex
    - PRINT ‘Street left: ‘ + UPPER(LEFT(@Street, 1));
* Debugging with SQL Server M…
  + Sql server debugging tools
  + Step through line by line
* Handling Errors with Try/Catch
  + Ex
    - BEGIN TRY;
    - Statements;
    - END TRY
    - BEGIN CATCH;
    - Error handling statements;
    - END CATCH;
* Return Codes
  + Return 0 by default
  + CONVERT: to convert values
    - CONVERT(VARCHAR(10), @RetVal);
* Handling Failed Transactions
  + When you open an transaction you must close it by committing it or rolling it back
  + Transaction block access to the table while they update records
    - BEGIN TRY;
    - BEGIN TRANSACTION;
    - ….
    - COMMIT TRANSACTION;
    - END TRY;
    - BEGIN CATCH;
    - IF (@@TRANCOUNT > 0)
    - BEGIN;
    - ROLLBACK TRANSACTOIN;
    - END;
    - END CATCH;
* Defensive Coding
* Summary