

MS SQL server 2012 for BI



Copyright © 2011 IGATE Corporation. All rights reserved. No part of this publication shall be reproduced in any way, including but not limited to photocopy, photographic, magnetic, or other record, without the prior written permission of IGATE Corporation.  
IGATE Corporation considers information included in this document to be Confidential and Proprietary.

## Course Goals and Non Goals

### ➤ Course Goals

- Understanding database concepts
- How to use T-SQL statements for implementing database concepts
- Writing Stored procedures, Functions and triggers



## Pre-requisites

- Fair knowledge of Oracle and SQL

## Course Goals and Non Goals

### ➤ Course Non Goals

- Database administration will not be covered
- Different services Like SSIS,SSRS,SSAS will not be covered



## Intended Audience

- Developers.



## Day Wise Schedule

### ➤ Day 1

- Lesson 1: Introduction to SQL Server 2012
- Lesson2 : Working with Data Types, DDL,DML and DCL statements
- Lesson 3: Beginning with Transact-SQL

### ➤ Day 2

- Lesson 3: Beginning with Transact-SQL
- Lesson 4: Working with Joins and Subqueries
- Lesson 5: Stored Procedures and functions

## Table of Contents

### ➤ Lesson 1: Introduction to SQL Server 2012

What is SQL Server?

SQL Server Components

SQL Server Authentication Modes

SQL Server Services

SQL Server Version History

## Table of Contents

### ➤ Lesson 1: Introduction to SQL Server 2012 (Cont.)

- SQL Server Key Business Scenarios
- SQL Server Editions
- SQL Server Tool & Utilities
- SQL Server System Databases
- SQL Server – Database Objects
- SQL Server system Tables
- SQL Server - Metadata Retrieval
- SQL Server - Features

## Table of Contents

➤ **Lesson 2: Working with Data Types, Tables & Data Integrity**

Working With Data Types

Working With Schema

Working With Tables

Sequence

Implementing Data Integrity

➤ **Lesson 3: Beginning with Transact-SQL**

Transact-SQL

System Functions

Advanced T-SQL Queries

Advanced T-SQL Statements

Other T-SQL Statements



## Table of Contents

### ➤ Lesson 4: Working with Joins and Subqueries

What are Joins?

Types of joins

Subqueries

#### Set Operators

UNION, UNION ALL, EXCEPT, INTERSECT

### ➤ Lesson 5: Procedures and Functions

Stored Procedure

Implementing Stored Procedure

Exception handling using TRY-CATCH

Implementing User Defined Functions

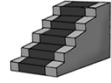
## References

- Microsoft SQL Server 2012 : A Beginner's Guide  
By Dusan Petković
- What's new in SQL Server 2012 :Rachel Clements
- Microsoft SQL Server 2012 – Database Development  
By - Tobias Themström  
Ann Weber  
Mike Hitek  
and GrandMasters
- Microsoft SQL Server 2012 – MSDN Library
- Microsoft SQL Server 2012 – TechNet Library



### Next Step Courses

- **SQL server 2012 services like SSIS,SSRS,SSAS**
- **Database Administration using SQL server 2012**



**IGATE**  
Speed. Agility. Imagination

June 21, 2015

Proprietary and Confidential

- 12 -

## Other Parallel Technology Areas

- Oracle
- MySQL

# SQL Server 2012 – Database Development

Lesson 01: Introduction to SQL Server 2012

June 22, 2015

Proprietary and Confidential

- 1 -

IGATE  
Speed. Agility. Imagination.

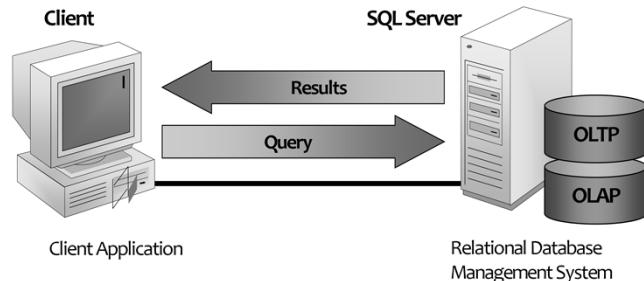
## Lesson Objectives

➤ **In this lesson, you will learn:**

- What Is SQL Server?
- SQL Server Components
- SQL Server Services
- SQL Server Key Business Scenarios
- SQL Server Versions, Editions and Tools
- Exploring new features in SSMS 2008 and SSMS 2012
- SQL Server Databases
- Features of SQL Server



## Introduction to SQL Server 2012



June 22, 2015 | Proprietary and Confidential | - 3 -

**IGATE**  
Speed. Agility. Imagination.

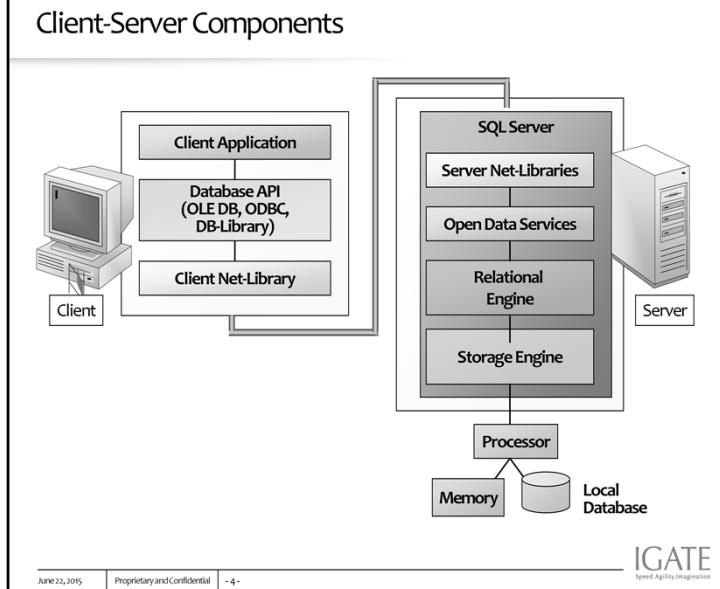
### What is SQL Server 2008?

SQL Server 2008 is a database that follows the relational model of data management system.

The emphasizes of relational Model is that the data is fundamentally organized in table which can also be termed as a collection of rows and columns .The relational concept maintains the relationships amongst data by adhering to the rules defined by the database administrator. In situations of power failure, which is normally termed as crash situations, the data should be free of corruption (invalid).

SQL Server 2008 is a Relational Database Management System (RDBMS) that provides:

- Maintaining Relationship among Data stored in the Database
- Ensuring Data is stored correctly and rules defining relationship are not violated.
- Recovering all data to a point of consistency , in the event of failure.



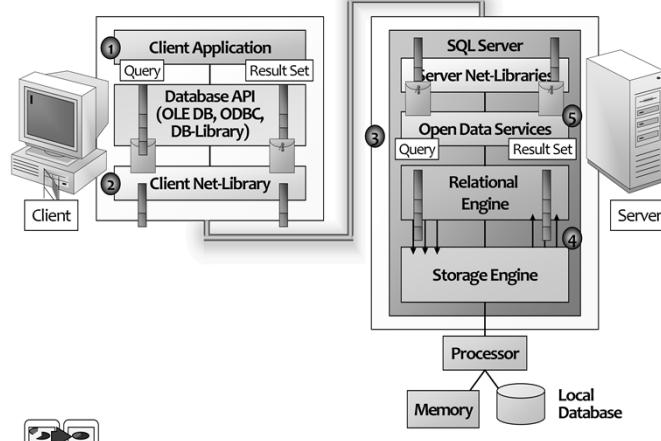
### The Net-Library

The Net-Library abstraction layer enables SQL Server to read from and write to many different network protocols, and each such protocol (such as TCP/IP sockets) can have a specific driver. The Net-Library layer makes it relatively easy to support many different network protocols without having to change the core server code.

### Open Data Services

Open Data Services (ODS) functions as the client manager for SQL Server; it is basically an interface between server Net-Libraries and server-based applications, including SQL Server. ODS manages the network: it listens for new connections, cleans up failed connections, acknowledges "attentions" (cancellations of commands), coordinates threading services to SQL Server, and returns result sets, messages, and status back to the client.

### Client-Server Communication Process



June 22, 2015 | Proprietary and Confidential | - 5 -

**IGATE**  
Speed. Agility. Imagination.

Figure shows the path from the SQL Server client application to the SQL Server engine and shows where the Net-Library interface fits in. On the server side, ODS provides functionality that mirrors that of ODBC, OLE DB, or DB-Library at the client. Calls exist for an ODS server application to describe and send result sets, to convert values between datatypes, to assume the security context associated with the specific connection being managed, and to raise errors and messages to the client application.

ODS uses an event-driven programming model. Requests from servers and clients trigger events that your server application must respond to. Using the ODS API, you create a custom routine, called an *event handler*, for each possible type of event. Essentially, the ODS library drives a server application by calling its custom event handlers in response to incoming requests.

ODS server applications respond to the following events:

**Connect events** When a connect event occurs, SQL Server initiates a security check to determine whether a connection is allowed. Other ODS applications, such as a gateway to DB/2, have their own logon handlers that determine whether connections are allowed. Events also exist that close a connection, allowing the proper connection cleanup to occur.

**Language events** When a client sends a command string, such as an SQL statement, SQL Server passes this command along to the command parser. A different ODS application, such as a gateway, would install its own handler that accepts and is responsible for execution of the command.

**Remote stored procedure events** These events occur each time a client or SQL Server directs a remote stored procedure request to ODS for processing.

### The Relational Engine and the Storage Engine

The SQL Server database engine is made up of two main components, the relational engine and the storage engine. Unlike in earlier versions of SQL Server, these two pieces are clearly separated, and their primary method of communication with each other is through OLE DB. The relational engine comprises all the components necessary to parse and optimize any query. It requests data from the storage engine in terms of OLE DB rowsets and then processes the rowsets returned. The storage engine comprises the components needed to actually access and modify data on disk.

### Communication Between the Relational Engine and the Storage Engine

The relational engine uses OLE DB for most of its communication with the storage engine. The following description of that communication is adapted from the section titled "Database Server" in the SQL Server Books Online. It describes how a SELECT statement that processes data from local tables only is processed: The relational engine compiles the **SELECT** statement into an optimized execution plan. The execution plan defines a series of operations against simple rowsets from the individual tables or indexes referenced in the **SELECT** statement. (Rowset is the OLE DB term for a result set.) The rowsets requested by the relational engine return the amount of data needed from a table or index to perform one of the operations used to build the **SELECT** result set. For example, this **SELECT** statement requires a table scan if it references a table with no indexes:

```
SELECT * FROM ScanTable
```

The relational engine implements the table scan by requesting one rowset containing all the rows from ScanTable. This next SELECT statement needs only information available in an index:

```
SELECT DISTINCT LastName  
FROM Northwind.dbo.Employees
```

The relational engine implements the index scan by requesting one rowset containing the leaf rows from the index that was built on the LastName column.

The following SELECT statement needs information from two indexes:

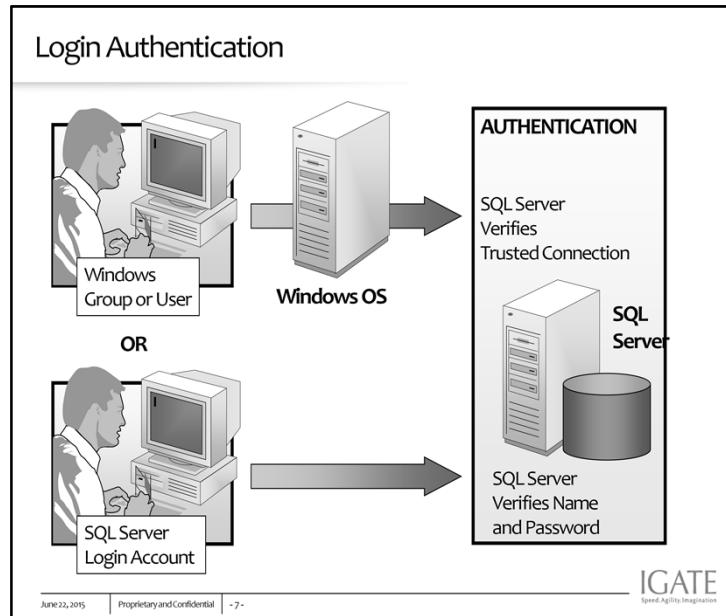
```
SELECT CompanyName, OrderID, ShippedDate  
FROM Northwind.dbo.Customers AS Cst  
JOIN Northwind.dbo.Orders AS Ord  
ON (Cst.CustomerID = Ord.CustomerID)
```

The relational engine requests two rowsets: one for the clustered index on Customers and the other for one of the nonclustered indexes on Orders.

The relational engine then uses the OLE DB API to request that the storage engine open the rowsets.

As the relational engine works through the steps of the execution plan and needs data, it uses OLE DB to fetch the individual rows from the rowsets it requested the storage engine to open. The storage engine transfers the data from the data buffers to the relational engine.

The relational engine combines the data from the storage engine rowsets into the final result set transmitted back to the user.



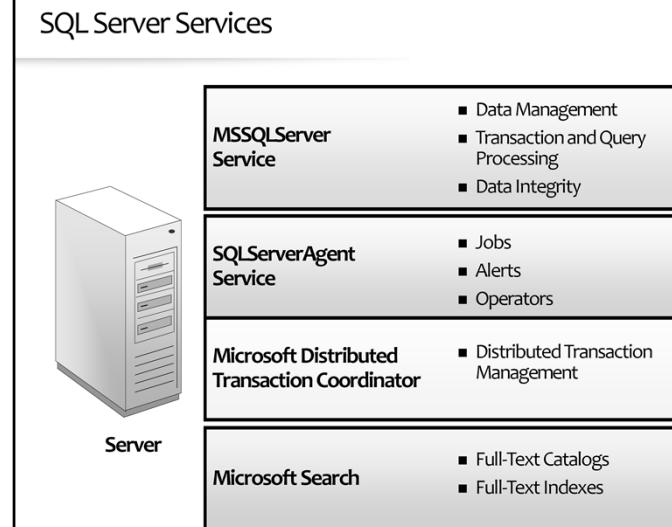
### Authentication Modes

Microsoft® SQL Server can operate in one of two security (authentication) modes:

#### **Windows Authentication Mode (Windows Authentication)**

Windows Authentication mode allows a user to connect through a Microsoft Windows OS user account.

**Mixed Mode (Windows Authentication and SQL Server Authentication)** Mixed Mode allows users to connect to an instance of SQL Server using either Windows Authentication or SQL Server Authentication. Users who connect through a Windows user account can make use of trusted connections in either Windows Authentication Mode or Mixed Mode.



## SQL Server Services

Services are internal processes, which SQL Server invokes to perform specific actions. The major four services of SQL-Server are as follow:

### **MSSQL Server Service**

This Service is used to manage the database and security aspect of SQL Server.

### **Microsoft Search Service**

This Service is for enabling the English Query Search.

### **SQL Server Agent**

This service is for scheduling and managing tasks on SQL Server.

### **Microsoft Distributed Transaction Coordinator Service (MSDTC)**

This service is used for Distributed Data Management , which is an environment where data is stored at multiple locations which can be like Oracle.

## SQL Server Versions

SQL Server 1.0 (1989)	Developed by Microsoft, Sybase, and Ashton-Tate for OS/2
SQL Server 4.2 (1992)	Developed for Windows NT 3.1
SQL Server 6.0 (1995)	First version designed specifically for Windows NT
SQL Server 7.0 (1999)	Total rewrite of code base resulted in performance and scalability improvements
SQL Server 2000	Further improvements in performance, scalability, and reliability
SQL Server 2005	New and improved features: Integration Services, Analysis Services, Notification Services, Reporting Services, and XML support
SQL Server 2008	Microsoft SQL Server 2008 is a powerful and reliable data management system that delivers a rich set of features, data protection, and performance for embedded application clients, light Web applications, and local data stores. Designed for easy deployment and rapid prototyping.
SQL Server 2012	SQL Server Database Engine introduces new features and enhancements that increase the power and productivity of architects, developers, and administrators who design, develop, and maintain data storage systems.

## SQL Server – Key Business Scenarios

**Business Intelligence** Scalable business intelligence platform optimized for data integration, reporting, and analysis

**Data Warehousing** Comprehensive and scalable platform for faster storage and more data

**OLTP** Scalable, high-performance database engine for mission-critical, highly available, secure applications

**Application Development** Comprehensive programming framework, including ADO.NET Entity Framework and LINQ

**Server Consolidation** Flexible, manageable, high-performance consolidation solutions

June 22, 2015

Proprietary and Confidential

- 10 -

**IGATE**  
Speed. Agility. Imagination.

SQL Server delivers a trusted, productive, and intelligent enterprise data platform.

### **Business Intelligence –**

SQL Server provides business intelligence functionality in various ways:

- SQL Server Database Engine provides a scalable, high-performance data-storage engine that is suitable for extremely large volumes of data.
- SQL Server Integration Services provides a comprehensive platform for extract, load, and transform (ETL) operations that enables you to incorporate disparate data into your data warehouse and business applications.
- SQL Server Analysis Services provides an online analytical processing (OLAP) engine that supports complex analytics.
- SQL Server Reporting Services makes it easy to create, publish, and distribute detailed reports.

You can easily use the 2007 Microsoft Office system to enable business users to access corporate SQL Server 2008 data. You can use the Microsoft Office Excel spreadsheet software to perform OLAP analysis, view reports, and mine data; Microsoft Office Word to view reports; and Microsoft Office SharePoint® Server 2007 to host reports.

**Data Warehousing** – SQL Server includes scalability features that you can use to enhance your data warehousing solutions:

- SQL Server Database Engine provides functionality to load, query, compress, and back up data.
- SQL Server Integration Services provides lookup performance and pipeline performance features that enable you to quickly load data into your data warehouse.
- SQL Server Analysis Services provides improved query processing and the ability to back up larger databases
- SQL Server Reporting Services provides improved reporting scalability.

**OLTP** - SQL Server includes a range of features that improve the OLTP functionality of the database engine. It enables you to build scalable, high-performing, highly available, secure, and manageable database applications through the use of new and improved features such as variable-sized data types, dynamic allocation of Address Windowing Extensions (AWE), database mirroring, clustering, replication, event auditing, transparent data encryption, management policies, and centralized server administration and management.

#### Features of SQL Server

- **Scalability**
- **High level of security**
- **High Availability**
- **Manageability**
- **Programmability**
- **Replication**
- **Support for data migration and analysis**
- **Service-oriented architecture**
- **Support for Web services**
- **Reporting**
- **Data Integration**
- **Data Analysis**

**Application Development** - SQL Server supports the use of new programmability technologies that make developers more productive and make their applications more scalable. The introduction of the Microsoft ADO.NET Entity Framework and LINQ will simplify data access to a variety of data sources. By using the Microsoft .NET Framework common language runtime (CLR) inside the SQL Server Database Engine, developers can write easily maintained code that runs directly against the data.

**Server Consolidation** - SQL Server supports consolidation through several techniques, including running multiple instances of SQL Server on one physical computer and using Microsoft Virtual Server to run multiple virtual operating systems on one physical computer.

## SQL Server – Editions

Enterprise	For large-scale, business-critical applications
------------	---

Standard	For small to medium departmental applications
----------	---

Workgroup	For small-scale branch applications
-----------	-------------------------------------

Web	For low-cost, large-scale Web applications
-----	--

Express	For entry-level and learning applications
---------	---

Compact	For embedded databases
---------	------------------------

Developer	For development and testing
-----------	-----------------------------

June 22, 2015

Proprietary and Confidential

- 13 -

**IGATE**  
Speed. Agility. Imagination.

## Tools & Utilities

- **SQL Server management Studio**
  - Tool for developing and managing the SQL Server database objects.
- **Business Intelligence Development Studio**
  - Tool for developing business intelligence solutions, data analysis, reports, and SQL Server 2012 Integration Services (SSIS) packages.

## Tools & Utilities

### ➤ **SQL Server Configuration manager**

- Helps the database administrators to manage the services associated with the SQL Server.
- Administrators can start, pause, resume or stop the services by using this tool.
- Used to manage the network connectivity configuration from the SQL Server client computers.

## Tools & Utilities

### ➤ Database Engine Tuning Advisor

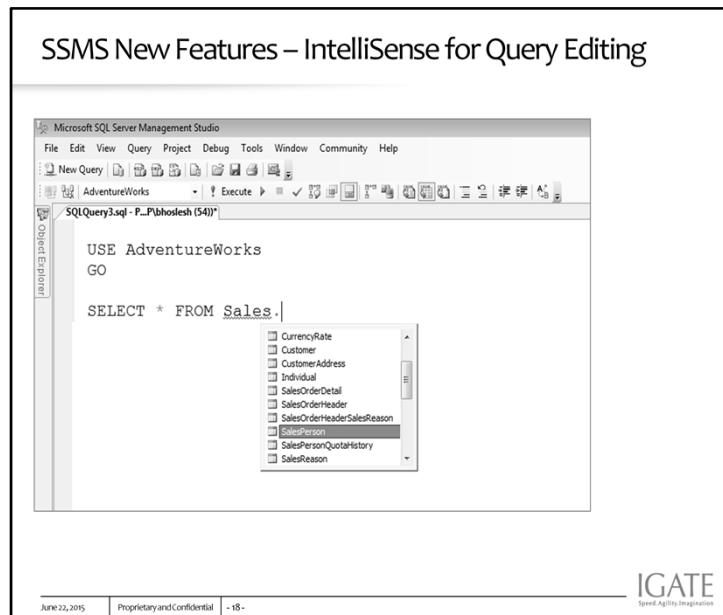
- GUI tool used to provide tuning recommendations.
- Acts as a tuning advisory.
- Tuning information is stored in the msdb database

### ➤ SQL Server Profiler

- Graphical user interface to SQL Trace for monitoring an instance of the SQL Server Database Engine or Analysis Services.
- Used to capture and save data about each event to a file or table to analyze later.

## SQL Server – SSMS New Features

- The SQL Server 2008 Management Studio (SSMS) has several new features which can help you administer your server, write queries or develop applications
- You can use these Features to work more efficiently
  - IntelliSense for Query Editing
  - Multi Server Query
  - Query Editor Regions
  - Object Explorer Enhancements



### IntelliSense

The Database Engine Editor now provides IntelliSense functionality such as word completion, error underlining, parameter help, colorizing, Quick Info, outlining, and syntax pair matching. IntelliSense is provided for frequently used Transact-SQL elements. It will be extended to other Transact-SQL elements in future releases.

## SSMS New Features – Multiserver Query

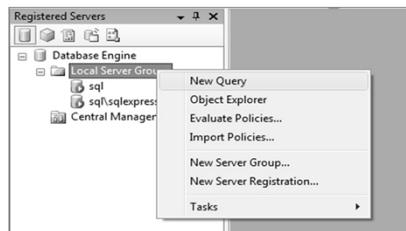


Figure1

```

SELECT SERVERPROPERTY('Edition') AS Edition,
       SERVERPROPERTY('ProductLevel') AS ProductLevel,
       SERVERPROPERTY('ProductVersion') AS ProductVersion
GO

```

Server Name	Edition	ProductLevel	ProductVersion
sql\sqlexpress	Express Edition	SP2	9.00.3042.00
sql	Developer Edition	RTM	10.0.1787.0

Figure2

**IGATE**  
Speed. Agility. Imagination.

### MultiServer Query

Usually DBA don't manage only one database; they have many servers to manage. There are cases when DBA has to check the status of all the servers.

SSMS 2008 has a feature to run a query on different servers from one query editor window. First of all make sure that you registered all the servers under your registered server. Once they are registered Right Click on server group name and click New Query as shown in the image Figure1 above.

Now in the opened query window run the following query

```

SELECT
    SERVERPROPERTY('Edition') AS Edition,
    SERVERPROPERTY('ProductLevel') AS ProductLevel,
    SERVERPROPERTY('ProductVersion') AS ProductVersion

```

Query above will give the result shown in the image above. Note that we have only three columns in the SELECT but our output contains four columns. The very first column is the “Server Name” and it is added by SQL Server to identify rows belonging to a specific server.

## SSMS New Features – Query Editor Region

```
SQLQuery7.sql - sql...s (SQL\Pinhal (54))*
-- Fully Expanded Region Code Snippet
CREATE PROCEDURE MyTest -- Region Type 1
AS
BEGIN -- Region Type 2
    SELECT 1
    BEGIN -- Region Type 2
        SELECT 2;
        SELECT * -- Region Type 3
        FROM sys.objects;
        SELECT * -- Region Type 3
        FROM sys.columns;
        SELECT 3;
    END
END
GO

-- Fully Collapsed Region Code Snippet
CREATE PROCEDURE MyTest -- Region Type 1...
GO
```

June 22, 2015

Proprietary and Confidential

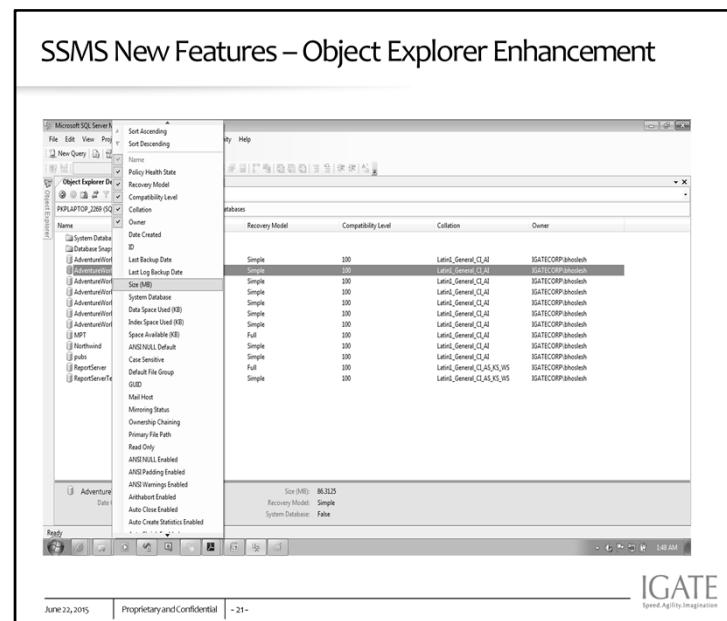
- 20 -

**IGATE**  
Speed. Agility. Imagination.

### Query Editor Regions

This feature exists in many programming languages already but now it is newly introduced in SSMS 2008. There are cases when T-SQL code is longer than hundreds of lines and after a while it keeps on getting confusing.

Regions can be collapsed as well as expanded by clicking the small sign of ‘-’ or ‘+’ besides them. The following image shows a fully expanded region code snippet and a fully collapsed code snippet.



## Object Explorer Enhancements

Object Explorer Detail view can be enabled by either going to Menu >> View >> Object Explorer Detail or pressing F7. In Object Explorer Detail the new feature is Object Search. Enter any object name in the object search box and the searched result will be displayed in the same window as Object Explorer Detail.

## Notes Pages

June 22, 2015

Proprietary and Confidential

- 22 -

**IGATE**  
Speed. Agility. Imagination.

### **SQL Server 2012 – SSMS New Features**

SQL Server 2012 brings some of the most desired enhancements in the Management Studio to simplify manageability.

The new SQL Server Management Studio is built on Visual Studio 2012. As it opens, you soon recognize the familiar dark blue theme of Visual Studio.

You can drag and move out Query Window from the main work area.

You can even drag out the ToolBox from the main IDE and place it elsewhere on the desktop or on another monitor

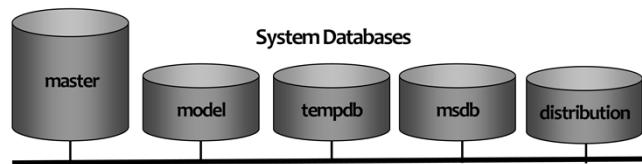
Another important feature is *Insert Snippet* which you can find on menu via *Edit -> IntelliSense*. It allows you to inject variety of code snippets onto your Query Window:

## Demo

### ➤ SSMS New Features



## SQL Server 2012 – System Databases



June 22, 2015

Proprietary and Confidential

- 24 -

**IGATE**  
Speed. Agility. Imagination.

### Types of Databases

SQL Server databases are categorized as below :

System databases.

User-defined databases.

Figure above displays the two categories of databases.

### System Databases

A system database is created to support the operation of the SQL server. When SQL Server is installed , four system databases – Master, Model, Tempdb, Msdb and two user databases are created automatically.

These databases are discussed as :

#### Master database

The master database is the basic database used by SQL Server in its operation. The master database controls the user databases and the operation of SQL Server as a whole. The default size of this database is 12.25 MB. This database contains pointers to all other databases stored on your system. It also includes server-wide information such as login information, system stored procedures and related services. Since the information stored in the master database is very critical in nature, no user is allowed direct access to the master database, which is updated on a continuous basis.

**Model database**

The model database is the template database for user-defined databases in SQL Server. This database consists of the system tables, which should belong to every user databases. The structure of the model database is copied into each new database that is created by the database administrator (DBA). Hence, whenever a new database is created its size should be at least the same as size of model database, so that the structure of the model database can be accustomed in new database.

**Mssdb database**

The msdb database contains task scheduling, event handling, replication, alert management and system operator information (all these keywords would be discussed in detail in the later chapters).

**Tempdb database**

The tempdb database provides storage area from temporary tables, as also the result of sort operations, join operations, and other activities that require temporary space to execute. There is only one tempdb database regardless of the number of the databases stored on the server. No special permission is required to use this database. They are deleted when the user logs out of the SQL Server session.

There are two types of temporary tables that are supported by SQL Server as mentioned below:

**Global temporary tables:** These tables are available to all the users that are connected and are prefixed with double-hash signs(##), to the name of the table.

**Local temporary tables:** These tables are only visible to the user who have created these and are prefaced with a single hash (#) sign.

The information about the various system databases can be summarized in Table below

Table: System Databases

Databases	Class	Size (in MB)
Master	System databases	12.25
Model	System databases	1.5
Tempdb	System databases	8.5
Msdb	System databases	

**User Databases**

A user database is a database that is created by the Database Administrator or a user. User defined databases are those created as per the needs of the users of SQL Server.

An example of the sample user database is the **AdventureWorks** database.

## Database Objects

➤ Some major database objects are as follows

- Tables
- Views,
- Indexes,
- Triggers,
- Procedures / functions,
- Constraints
- Rules

**And so on...**

June 22, 2015 | Proprietary and Confidential | - 26 -

**IGATE**  
Speed. Agility. Imagination.

A database can also be defined as a collection of interrelated data. The data is stored in the form of different objects that allow the users to store and retrieve information. These objects help in structuring different objects that allow the users to store and retrieve information. These objects help in structuring data and in defining data integrity mechanisms. The different database objects are:

Database Object	Description
Table	It is a fundamental method of storing data in the form of a collection of columns and rows (records).
Data Types	An identifier that specifies the nature of data stored in a column.
that can be	
Defaults	Specifies a value to be assigned by SQL Server for a column if the user does not enter any value
for a column if	
Index	A structure that helps faster access to records of a table based on the values of one or more columns. It contains pointers to the records where those values occur.
a table	
contains	
those values	
Constraint	Is a restriction that is enforced by SQL Server to limit the values that the user can enter into a column.
limit the	
Rule	Specifies acceptable values that can be inserted into a particular column.
into a	
Stored Proc	A pre-complied collection of SQL statements, which are used to perform specific tasks.
are	
Trigger	A special type of stored procedure that is automatically executed by SQL Server in response to an DML statement.
automatically	
statement.	
View	A virtual table that provides an alternate way to look at data from one or more tables in a database.
look at	

## System Tables

- System Tables Store Information (Metadata) about the System and Database Objects
- Database Catalog Stores Metadata about a specific database
- System Catalog Stores Metadata about the entire system and all other databases
  - SYS.DATABASES
  - SYS.OBJECTS
  - SYS.TABLES
  - SYS.PROCEDURES
  - SYS.INDEXES

June 22, 2015

Proprietary and Confidential

- 27 -

IGATE  
Speed. Agility. Imagination.

### System Catalog:

The system tables that SQL server needs to configure and manage itself are collectively grouped together and called the “System Catalog”. The **system catalog** consists primarily of the system tables contained in the master table.

### Database Catalog

The system tables that are needed to define the structure of a new user database are collectively known as the “Database Catalog”. The database catalog consists primarily system tables contained in the model database.

The various system tables in all the SQL Server system databases are discussed below as follows:

#### Some MASTER system Tables

**Syslockinfo** - Contains information on all granted, converting, and waiting lock requests.

**Syscacheobjects** - Contains information about how the cache is used.

**Syslogins** - Contains one row for each login account.

**Syssqlservers** - Contains one row for each database on SQL server 2008.

**Model System Tables**

Name	Utility
<b>Syscolumns</b>	Contains one row for every column in every table and view, and a row for each parameter in a stored procedure.
<b>Syscomments</b>	Contains entries for each view, rule, default, trigger, CHECK constraint , DEFAULT constraint , and stored procedure.
<b>Sysfiles</b>	Contains one row for each file in a database
<b>Sysindexes</b>	Contains one row for each index and table in the database
<b>Sysobjects</b>	Contains one rows for each object (Constraint , default, log, rule, stored procedure, and so on ) created within a database. Syscomments and Sysobjects, combined together, give detailed information on ny object in SQL Server.
<b>Syspermissions</b>	Contains information about permissions granted and denied to users , groups and roles in the database.
<b>Systypes</b>	Contains one row for each system-supplied and each user-defined data type
<b>Sysusers</b>	Contains one row for each windows user, windows group, SQL server user, or SQL Server role in the database

**MSDB System Tables**

Name	Utility
<b>Sysalerts</b>	Contains one row for each alert. An alert is a message sent in response to an event.
<b>Syscategories</b>	Contains the categories used by SQL Server Enterprise Manager to organize jobs, alerts, and operators.
<b>Syobs</b>	Stores the information for each scheduled job to be executed by the SQL Server Agent
<b>Sysoperators</b>	Contains one row for each SQL Server operator.
<b>Sysnotification</b>	Contains one row for each notification to an operator

**TEMPDB System Tables**

Name	Utility
<b>Local Temporary tables</b>	These tables are only visible to the user who have created these and are prefaced with a single hash (#) sign.
<b>Global Temporary tables</b>	These tables are available to all the users that are connected and are prefaced with two-hash signs(##).

## Metadata Retrieval

### ➤ System Stored Procedures

```
EXEC sp_help Employees
```

### ➤ System and Metadata Functions

```
SELECT USER_NAME(10)
```

### ➤ Information Schema Views

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
```

June 22, 2015

Proprietary and Confidential

- 29 -



### System Stored Procedures

Many of your administrative activities in SQL Server are performed through a special kind of procedure known as a **system stored procedure**. System stored procedures are created and stored in the **master** database and have the **sp\_** prefix. System stored procedures can be executed from any database without having to qualify the stored procedure name fully using the database name **master**.

It is strongly recommended that you do not create any stored procedures using **sp\_** as a prefix. SQL Server always looks for a stored procedure beginning with **sp\_** in this order:

The stored procedure in the **master** database.

The stored procedure based on any qualifiers provided (database name or owner).

The stored procedure using **dbo** as the owner, if one is not specified.

Therefore, although the user-created stored procedure prefixed with **sp\_** may exist in the current database, the **master** database is always checked first, even if the stored procedure is qualified with the database name.

### Important

If any user-created stored procedure has the same name as a system stored procedure, the user-created stored procedure will never be executed.

### System Procedures

**sp\_add\_data\_file**, **sp\_recover\_suspect\_db**,  
**sp\_add\_log\_file**, **sp\_recover\_suspect\_db**  
**sp\_helpconstraint**, **sp\_addextendedproc**, **sp\_helpdb**,  
**sp\_addextendedproperty**, **sp\_helpdevice**, **sp\_helpextendedproc**,  
**sp\_addmessage**, **sp\_helpfile**, **sp\_addtype**

## Summary

➤ **In this lesson, you have learnt:**

- SQL Server 2012 database follows the relational model of data management system
- SQL Server databases are categorized as:
  - System databases.
  - User-defined databases
- SQL Server offers various Tools and Utilities to work with



## Review Question

- **Question 1:** List the two authentication modes available in MS SQL Server.
- **Question 2:** List the System Databases in SQL Server.



# SQL Server 2012 – Database Development

Lesson 02: Working with Data Types, DDL, DML and DCL statements

June 22, 2015

Proprietary and Confidential

- 1 -

**IGATE**  
Speed. Agility. Imagination.

## Lesson Objectives

➤ In this lesson, you will learn:

- Working with Data Types
- Working with SQL Server Schemas
- Working with DDL, DML, DCL statements
- Implementing Data Integrity



---

June 22, 2015

Proprietary and Confidential

- 2 -

**IGATE**  
Speed. Agility. Imagination

## Lesson Objectives

Designing, creating and maintaining tables are one of the very important tasks a database developer performs while maintaining databases for any enterprise. In this particular lesson, you walk through these and other tasks related to tables. You also take a look at the Data Types available in SQL Server 2008. You will be also introduced to implementing declarative data integrity in your tables.

## Introduction to Data Types in SQL Server

- SQL Server has an extensive list of data types to choose from, including some that are new to respected versions.
- In SQL Server, each column, local variable, expression and parameter has a related data type
- A data type is an element that specifies the kind of data that the item can hold
- SQL Server introduces new additions of data types that can simplify your database development code
- You can also define your own data types in SQL Server
- These User Defined Data Types (UTDs)/ Alias Data Types are based on the system-supplied data types

June 22, 2015 | Proprietary and Confidential | - 3 -



## Introduction to Data Types in SQL Server

SQL Server 2008 has an extensive list of data types to choose from, including some that are new to SQL Server 2008. While working with SQL server data, a developer extensively works with objects like table column, local variable, expression, parameter and so on.

At the same time before you can create a table, you must define the data types for the data that will be stored in that table. Data types specify the type of information (characters, numbers, or dates) that a column can hold, as well as how the data is stored. SQL Server 2008 supplies more than 30 specific system data types. It also supports alias data types, which are user-defined data types based on system data types.

You should take into consideration following important attributes while assigning data type to an Object.

The type of data contained by the Object

The length or size of the stored value

The precision of the number (Numeric Data Types only)

The scale of the number (Numeric Data Types only)

## What are System-Supplied Data Types?

### ➤ Integers (whole number)

- int -2<sup>31</sup> to 2<sup>31</sup>-1
- smallint -2<sup>15</sup> to 2<sup>15</sup>-1
- bigint -2<sup>63</sup> to 2<sup>63</sup>-1
- tinyint 0 to 255
- bit 1 or 0

### ➤ numeric (Fixed precision)

- decimal -10<sup>38</sup>+1 to 10<sup>38</sup>-1
- numeric Equivalent to decimal

### ➤ Approximate numeric

- float -1.79E + 308 to 1.79E + 308
- real -3.40E + 38 to 3.40E + 38

June 22, 2015

Proprietary and Confidential

+ 4 -



Transact-SQL has these base data types :

Exact Numerics

Integers

**bigin** - Integer (whole number) data from -9223372036854775808 through 9223372036854775807.

**int** - Integer (whole number) data from -2,147,483,648 through 2,147,483,647.

**smallint** - Integer data from 2<sup>15</sup> (-32,768) through 2<sup>15</sup> - 1 (32,767).

**tinyint** - Integer data from 0 through 255.

**bit** - Integer data with either a 1 or 0 value.

**decimal** and **numeric**

**decimal** - Fixed precision and scale numeric data from -10<sup>38</sup> +1 through 10<sup>38</sup> -1.

**numeric** - Functionally equivalent to decimal.

Approximate Numerics

**float** - Floating precision number data from -1.79E + 308 through 1.79E + 308.

**real** - Floating precision number data from -3.40E + 38 through 3.40E + 38.

## What are System-Supplied Data Types?

➤ **Monetary (with accuracy to a ten-thousandth of a monetary unit)**

- money -2^63 to 2^63 -1
- smallmoney -214,748.3648 through +214,748.3647

➤ **Date and Time**

- datetime
- smalldatetime

➤ **Character(s)**

- char Fixed-length non-Unicode character data
- varchar Variable-length non-Unicode data (max length 8,000)
- 2^31 -1)

June 22, 2015

Proprietary and Confidential



**money and smallmoney**

**money** - Monetary data values from -

922,337,203,685,477.5808 through  
+922,337,203,685,477.5807, with accuracy to a ten-thousandth of a monetary unit.

**smallmoney** - Monetary data values from -214,748.3648 through +214,748.3647, with accuracy to a ten-thousandth of a monetary unit.

**datetime and smalldatetime**

**datetime** - Date and time data from January 1, 1753, through December 31, 9999, with an accuracy of three-hundredths of a second, or 3.33 milliseconds.

**range**, a larger default fractional precision, and optional user-specified precision.

**smalldatetime** - Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute.

**Character Strings**

**char** - Fixed-length non-Unicode character data with a maximum length of 8,000 characters.

**varchar** - Variable-length non-Unicode data with a maximum of 8,000 characters.

## What are System-Supplied Data Types?

### ➤ Binary

- Binary Fixed-length binary data ,max of 8000 bytes
- Varbinary Varying length binary data ,max of 8000 bytes

### ➤ Large Objects

- Image
- Text

### ➤ Unicode Data type storage is two times the byte size

- Nchar
- Nvarchar , nvarchar(max)
- ntext

June 22, 2015

Proprietary and Confidential

- 6 -



### Binary Strings

binary - Fixed-length binary data with a maximum length of 8,000 bytes.

varbinary - Variable-length binary data with a maximum length of 8,000 bytes.

### Large Objects

text - Variable-length non-Unicode data with a maximum length of  $2^{31} - 1$  (2,147,483,647) characters.

image - Variable-length binary data with a maximum length of  $2^{31} - 1$  (2,147,483,647) bytes.

### Unicode Data types

Unicode is a character encoding system for representing characters from different language like traditional Chinese, kanji etc .

Unicode provides a unique number for every character. The numbers occupy two bytes . The unicode data type supported in SQL Server are

Nchar - Fixed-length Unicode data with a maximum length of 4,000 characters.

Nvarchar - Variable-length Unicode data with a maximum length of 4,000 characters.

nvarchar(max) - Variable-length Unicode data with a maximum length of 230 characters

ntext - Variable-length Unicode data with a maximum length of 1,073,741,823 characters.

## What are System-Supplied Data Types?

### ➤ Other Data Types

- cursor - A reference to a cursor ,used only in procedures
- table - A special data type used to store a result set for later processing
- Sql\_variant Stores values of various SQL Server-supported data types, except text, ntext, and timestamp.
- timestamp - A database-wide unique number that gets updated every time a row gets updated
- uniqueidentifier - A globally unique identifier (GUID)
- xml datatype:
- xml data type lets you store XML documents and fragments

June 22, 2015

Proprietary and Confidential

- 7 -



### Other Data Types

**cursor** - A reference to a cursor, used only in procedures

**sql\_variant** - A data type that stores values of various SQL Server-supported data types, except text, ntext, timestamp, and **sql\_variant**.

**table** - A special data type used to store a result set for later processing .  
**timestamp** - A database-wide unique number that gets updated every time a row gets updated.

**uniqueidentifier** - A globally unique identifier (GUID).

Microsoft SQL Server introduces the max specifier. This specifier expands the storage capabilities of the varchar, nvarchar, and varbinary data types.  
**varchar(max)**,

**nvarchar(max)**, and **varbinary(max)** are collectively called large-value data types. You can use the large-value data types to store up to  $2^{31}-1$  bytes of data.

**Example:**

```
Create table cust(custcode int primary key,custname varchar(50)
comment varchar(max))
```

**xml datatype:**

The **xml** data type lets you store XML documents and fragments in a SQL Server database. An XML fragment is an XML instance that is missing a single top-level element. You can create columns and variables of the **xml** type and store XML instances in them. Note that the stored representation of **xml** data type instances cannot exceed 2 GB.

## What are System-Supplied Data Types?

- **Large value data types:**
  - varchar(max)
  - nvarchar(max)
  - varbinary(max)
- **For non-unicode data MAX allows upto  $2^{31}-1$  bytes**
- **For unicode data MAX allows upto  $2^{30}-1$  bytes**
- **XML**
  - To store data in XML Format
  - XML DML for performing insert, update and deletes on the XML based columns

June 22, 2015

Proprietary and Confidential

- 8 -



You can optionally associate an XML schema collection with a column, a parameter, or a variable of the xml data type. The schemas in the collection are used to validate and type the XML instances. In this case, the XML is said to be typed.

The xml data type and associated methods help integrate XML into the relational framework of SQL Server.

**Example:**

```
CREATE TABLE T1(Col1 int primary key, Col2 xml)
```

## SQL Server - New Data Types

### ➤ SQL Server Date Data Types

- DATE
- TIME
- DATETIME2
- DATETIMEOFFSET

June 22, 2015

Proprietary and Confidential

- 9 -



### SQL Server Date Data Types

Previous versions of SQL Server offered two date data types: datetime and smalldatetime. These data types were sufficient for most applications, but could be cumbersome in certain cases. For instance, both data types have a date and time portion, which is great if that's what you want, but cumbersome if all you need to store is just the date or just the time.

Similarly, the date ranges imposed by these two data types - 1753-01-01 to 9999-12-31 for datetime and 1900-01-01 to 2079-06-06 for smalldatetime - are insufficient for a small percentage of applications. SQL Server 2008 remedies these ills by keeping the datetime and smalldatetime and introducing four new date data types: time, date, datetime2, and datetimeoffset.

As you can probably guess by their names, the time and date data types track just a time and just a date portion, respectively. The datetime2 data type is like the datetime data type, but has a larger window of dates (from 0001-01-01 to 9999-12-31) and greater precision on the time portion.

The datetimeoffset data type has the same range and precision as the datetime2 data type, but enables an offset from UTC to be specified, which makes it easier to record and display times respective to the end user's timezone.

## SQL Server - Date Data Types

Data Type	Format	Range	Storage Size (bytes)
time	hh:mm:ss[.nnnnnnn]	00:00:00.000000 through 23:59:59.999999	5
date	YYYY-MM-DD	0001-01-01 through 9999-12-31	3 bytes, fixed
smalldatetime	YYYY-MM-DD hh:mm:ss	1900-01-01 through 2079-06-06	4 bytes, fixed.
datetime	YYYY-MM-DD hh:mm:ss[.nnn]	1753-01-01 through 9999-12-31	8 bytes
datetime2	YYYY-MM-DD hh:mm:ss[.nnnnnnn]	0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999	6 bytes for precisions less than 3; 7 bytes for precisions 3 and 4. All other precisions require 8 bytes.
datetimeoffset	YYYY-MM-DD hh:mm:ss[.nnnnnnn] [+/-]hh:mm	0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999 (in UTC)	10 bytes, fixed is the default with the default of 100ns fractional second precision

June 22, 2015

Proprietary and Confidential


  
Speed. Agility. Imagination.

## TIME

The datatype TIME is primarily used for storing the time of a day. This includes Hours, minutes, Seconds etc. It is based on a 24-hour clock. The datatype TIME can store seconds up to the fraction of 9999999.

**Syntax:** time [ (fractional second precision) ]

**Usage:** DECLARE @MyTime time(7)

```
CREATE TABLE Table1 ( Column1 time(7) )
```

## DATE

This data type is useful to store the dates without the time part, we can store dates starting from 00001-01-01 through 9999-12-31 i.e. January 1, 1 A.D. through December 31, 9999 A.D. It supports the Gregorian Calendar and uses 3 bytes to store the date.

**Syntax:** date

**Usage:** DECLARE @MyDate date

```
CREATE TABLE Table1 ( Column1 date )
```

**DATETIME2**

This is a new data type introduced in SQL Server and this date/time data type is introduced to store the high precision date and time data. The data type can be defined for variable lengths depending on the requirement. This data type also follows the Gregorian Calendar. The Time Zone can't be specified in this data type. This is still useful because it gives you a complete flexibility to store the date time data as per your requirement.

**Syntax:** datetime2 [ (fractional seconds precision) ]

**Usage:** DECLARE @MyDatetime2 datetime2(7)

```
CREATE TABLE Table1 ( Column1 datetime2(7) )
```

**SMALLDATETIME and DATETIME**

Microsoft SQL Server continues to support existing data types such as datetime and smalldatetime.

**SMALLDATETIME:** Defines a date that is combined with a time of day. The time is based on a 24-hour day, with seconds always zero (:00) and without fractional seconds. The date range for the datatype smalldatetime is from 1900-01-01 through 2079-06-06 and time range supported is 00:00:00 through 23:59:59.

**Syntax:** smalldatetime

**Usage:** DECLARE @MySmalldatetime smalldatetime

```
CREATE TABLE Table1 ( Column1 smalldatetime )
```

**DATETIME:** This data type Defines a date that is combined with a time of day with fractional seconds that is based on a 24-hour clock. The date range for the datatype datetime is from January 1, 1753, through December 31, 9999 and time range supported is 00:00:00 through 23:59:59.997.

**Syntax:** datetime

**Usage:** DECLARE @MyDatetime datetime

```
CREATE TABLE Table1 ( Column1 datetime )
```

**DATETIMEOFFSET**

This is the new data type that is included in SQL Server and this data type is the most advanced in the league. We can store high precision date/ time with the Date Time Offset. We can't store the Time Zone like Eastern Time, Central Time etc. in the data type but can store the offset -5:00 for EST and -6:00 CST and so on. The data type is not Day light saving aware.

The date range is between 0001-01-01 and 9999-12-31 or January 1, 1 A.D. through December 31, 9999 A.D. and the Time Range is between 00:00:00 and 23:59:59.9999999. The offset range is between -14:00 through +14:00. The precision of the data type can be set manually and it follows the Gregorian Calendar.

**Usage:** datetimeoffset [ (fractional seconds precision) ]

**Usage:** DECLARE @MyDatetimeoffset datetimeoffset(7)

```
CREATE TABLE Table1 ( Column1 datetimeoffset(7) )
```

**Examples:**  
**DATE & TIME Data Type**

Use AdventureWorks

Go

Create Schema Trade

Go

```
CREATE TABLE Trade.BankTran
```

(

TransID	BIGINT IDENTITY(1,1) PRIMARY KEY ,
AccountNo	INT,
BankTranType	VARCHAR(50),
TransDate	DATE,
TransTime	Time(7)

)

Go

```
INSERT INTO Trade.BankTran
```

(AccountNo,BankTranType,TransDate,TransTime)
Values(101,'Transfer',sysdatetime(),sysdatetime())

Go

```
INSERT INTO Trade.BankTran
```

(AccountNo,BankTranType,TransDate,TransTime)
Values(102,'Transfer',sysdatetime(),sysdatetime())

Go

```
SELECT * FROM Trade.BankTran
```

**Result:**

TransID	AccountNo	BankTranType	TransDate	TransTime
---------	-----------	--------------	-----------	-----------

1	101	Transfer	2012-04-22	18:13:59.4657106
2	102	Transfer	2012-04-22	18:13:59.4807114

(2 row(s) affected)

**Examples:****Comparison between DATETIME2 & DATETIME data type with example:**

```
USE AdventureWorks
```

```
GO
```

```
CREATE TABLE TimeTable
```

```
(
```

```
    FirstDate DATETIME,
```

```
    LastDate DATETIME2(4)
```

```
)
```

```
GO
```

```
DECLARE @Intveral INT
```

```
SET @Intveral = 1000
```

```
WHILE (@Intveral > 0)
```

```
BEGIN
```

```
    INSERT TimeTable (FirstDate, LastDate)
```

```
        VALUES (SYSDATETIME(), SYSDATETIME())
```

```
    SET @Intveral = @Intveral - 1
```

```
END
```

```
GO
```

```
SELECT * FROM TimeTable
```

```
GO
```

**Output:**

FirstDate	LastDate
2012-04-22 19:54:52.700	2012-04-22 19:54:52.69
2012-04-22 19:54:52.703	2012-04-22 19:54:52.70
2012-04-22 19:54:52.703	2012-04-22 19:54:52.70
2012-04-22 19:54:52.707	2012-04-22 19:54:52.70
2012-04-22 19:54:52.707	2012-04-22 19:54:52.70
2012-04-22 19:54:52.707	2012-04-22 19:54:52.70
2012-04-22 19:54:52.707	2012-04-22 19:54:52.70
2012-04-22 19:54:52.710	2012-04-22 19:54:52.70
2012-04-22 19:54:52.710	2012-04-22 19:54:52.70
2012-04-22 19:54:52.713	2012-04-22 19:54:52.71
2012-04-22 19:54:52.713	2012-04-22 19:54:52.71
2012-04-22 19:54:52.717	2012-04-22 19:54:52.71

2012-04-22 19:54:52.700	2012-04-22 19:54:52.69
2012-04-22 19:54:52.703	2012-04-22 19:54:52.70
2012-04-22 19:54:52.703	2012-04-22 19:54:52.70
2012-04-22 19:54:52.707	2012-04-22 19:54:52.70
2012-04-22 19:54:52.707	2012-04-22 19:54:52.70
2012-04-22 19:54:52.707	2012-04-22 19:54:52.70
2012-04-22 19:54:52.710	2012-04-22 19:54:52.70
2012-04-22 19:54:52.710	2012-04-22 19:54:52.70
2012-04-22 19:54:52.713	2012-04-22 19:54:52.71
2012-04-22 19:54:52.713	2012-04-22 19:54:52.71
2012-04-22 19:54:52.717	2012-04-22 19:54:52.71

Note that when using the datetime data type is rounded to increments of .000, .003, or .007 seconds. However the datetime2 has a larger date range, a larger default fractional precision, and optional user-specified precision.

**Examples:**

**DATETIMEOFFSET:**

```
SELECT  
    CAST(SYSDATETIMEOFFSET() AS datetimeoffset(7)) AS  
        'datetimeoffset'
```

**Output:**

```
datetimeoffset  
-----  
2012-04-22 20:28:16.1262803 +05:30  
(1 row(s) affected)
```

**SYSDATETIMEOFFSET():** This function returns a datetimeoffset(7) value that contains the date and time of the computer on which the instance of SQL Server is running. The time zone offset is included.

**NOTE – IST**, which is GMT plus 5.30 hrs, came into existence in 1905.

## Spatial Data Type – What is Spatial Data?

- **Information about the location and shape of a geometric object:**
  - Store locations
  - Sales regions
  - Customer sites
  - Area within a specific distance of a location
- **Two types:**
  - Planar (or Euclidean) data for coordinate points on a flat, bounded surface. Distances are measured directly between points
  - Geodetic (or ellipsoidal) data for latitude and longitude points on the surface of the Earth. Distances are measured taking into account the curvature of the ellipsoidal surface



**IGATE**  
Speed. Agility. Imagination

### What is Spatial Data?

Spatial data is information that stores the location and shape of geometric objects, for example, a building, a road, a river, a city, a continent, or an ocean.

The object can be identified by a single point or a set of points that create a geometric shape. Spatial data is rapidly becoming more important to many industries. You can use it to locate stores, identify sales regions, pinpoint customer sites, and generate areas that are local to a location.

There are two types of spatial data: planar and geodetic. Planar, or Euclidean, spatial data views the Earth as if it is projected onto a planar surface like a map. Geodetic, or ellipsoidal, data views the Earth as an ellipsoid and works with information such as latitude and longitude coordinates.

The key difference between planar and ellipsoidal data is the way that operations on the data are performed. For example, distances between planar points are measured directly between the points by using simple trigonometry, whereas distances between geodetic data points take into account the curvature of the ellipsoidal surface.

## The geometry and geography Data Types

- **SQL Server supports two spatial data types:**
  - geometry for planar spatial data
  - geography for ellipsoidal spatial data
- **Both data types:**
  - Are implemented as .NET Framework common language runtime types
  - Can store points, lines, and areas
  - Provide members to perform spatial operations
- **Common uses:**
  - Geometry - localized geospatial data such as street maps
  - Geography - locations on the Earth's surface and integration
  - geospatial systems

June 22, 2015

Proprietary and Confidential

- 16 -



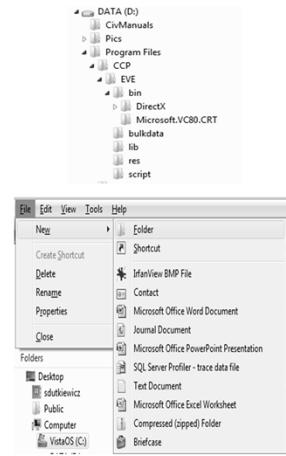
### Types of Spatial Data

There are two types of spatial data. The **geometry** data type supports planar, or Euclidean (flat-earth), data. The geometry data type conforms to the Open Geospatial Consortium (OGC) Simple Features for SQL Specification version 1.1.0.

In addition, SQL Server supports the **geography** data type, which stores ellipsoidal (round-earth) data, such as GPS latitude and longitude coordinates.

## SQL Server 2008 – Hierarchyid Data Types

- SQL Server 2008 introduces a new system-provided data type “Hierarchyid” to encapsulate hierarchical relationships
- We can use hierarchyid as a data type to create tables with a hierarchical structure
- It is designed to store values that represent the position of nodes of a hierarchical tree structure
- This type is internally stored as a VARBINARY value
- Examples where the hierarchyid type makes it easier to store and query hierarchical data include the following:
  - Organizational structures
  - A set of tasks that make up a larger projects (like a GANTT chart)
  - File systems (folders and their sub-folders)
  - A graphical representation of links between web pages



**IGATE**  
Speed. Agility. Imagination

June 22, 2015 | Proprietary and Confidential | - 17 -

### Hierarchyid

While hierarchical tree structures are commonly used in many applications, SQL Server has not made it easy to represent and store them in relational tables. In SQL Server 2008, the HIERARCHYID data type has been added to help resolve this problem. It is designed to store values that represent the position of nodes of a hierarchical tree structure.

The new HIERARCHYID data type in SQL Server 2008 is a system-supplied CLR UDT that can be useful for storing and manipulating hierarchies. This type is internally stored as a VARBINARY value that represents the position of the current node in the hierarchy (both in terms of parent-child position and position among siblings).

Unlike standard data types, the HIERARCHYID data type is a CLR user-defined type, and it exposes many methods that allow you to manipulate the date stored within it. For example, there are methods to get the current hierarchy level, get the previous level, get the next level, and many more. In fact, the HIERARCHYID data type is only used to store hierarchical data; it does not automatically represent a hierarchical structure. It is the responsibility of the application to create and assign HIERARCHYID values in a way that represents the desired relationship. Think of a HIERARCHYID data type as a place to store positional nodes of a tree structure, not as a way to create the tree structure.

The HIERARCHYID data type makes it easier to express these types of relationships without requiring multiple parent/child tables and complex joins.

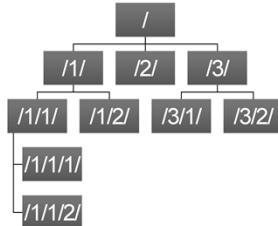
## Hierarchyid – Key Properties & Limitations

### ➤ Key Properties

- Extremely compact
- Comparison is in depth-first order
- Support for arbitrary insertions and deletions

### ➤ Limitations

- Does not automatically represent a tree
- No guarantee that hierarchyid values in a column would be unique
- Hierarchical relationships represented by hierarchyid values are not enforced like a foreign key relationship



June 22, 2015 Proprietary and Confidential +18 -

**IGATE**  
Speed. Agility. Imagination

### Key Properties of hierarchyid

A value of the hierarchyid data type represents a position in a tree hierarchy. Values for hierarchyid have the following properties:

**Extremely compact**

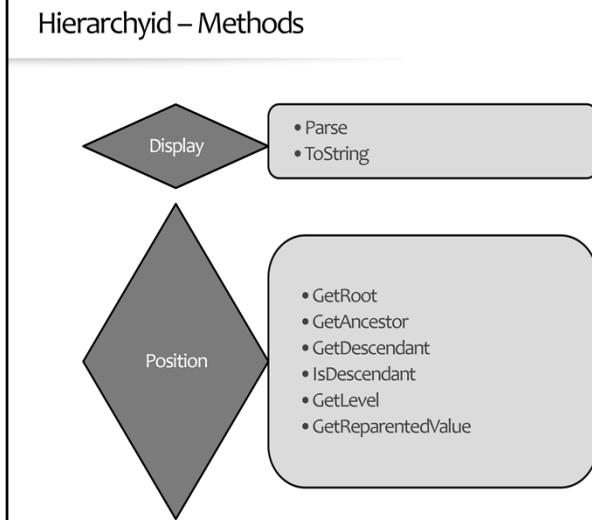
The average number of bits that are required to represent a node in a tree with n nodes depends on the average fanout (the average number of children of a node). For small fanouts, (0-7) the size is about  $6 * \log_2 n$  bits, where A is the average fanout. A node in an organizational hierarchy of 100,000 people with an average fanout of 6 levels takes about 38 bits. This is rounded up to 40 bits, or 5 bytes, for storage.

**Comparison is in depth-first order**

Given two hierarchyid values a and b,  $a < b$  means a comes before b in a depth-first traversal of the tree. Indexes on hierarchyid data types are in depth-first order, and nodes close to each other in a depth-first traversal are stored near each other. For example, the children of a record are stored adjacent to that record.

**Support for arbitrary insertions and deletions**

By using the GetDescendant method, it is always possible to generate a sibling to the right of any given node, to the left of any given node, or between any two siblings. The comparison property is maintained when an arbitrary number of nodes is inserted or deleted from the hierarchy. Most insertions and deletions preserve the compactness property. However, insertions between two nodes will produce hierarchyid values with a slightly less compact representation.



## Hierarchyid Methods

**Parse:** Converts a string representation of a hierarchy to a Hierarchyid value. Static.

**ToString:** Returns a string that contains the logical representation of this Hierarchyid .

**GetRoot:** Returns the root Hierarchyid node of this hierarchy tree. Static.

**GetAncestor:** Returns a Hierarchyid that represents the nth ancestor of this HierarchyID node.

**GetDescendant:** Returns a child node of this Hierarchyid node.

**IsDescendant:** Returns true if the passed-in child node is a descendant of this HierarchyID node.

**GetLevel:** Returns an integer that represents the depth of this Hierarchyid node in the overall hierarchy.

**GetReparentedValue:** Returns a node whose path from the root is the path to newRoot, followed by the path from oldRoot to this.

**Example:**

-- Create database and table

```
CREATE DATABASE HierarchyDB
```

```
GO
```

```
USE HierarchyDB
```

```
GO
```

```
IF ( Object_id('HierarchyTab') > 0 )
DROP TABLE HierarchyTab
```

```
GO
```

```
CREATE TABLE HierarchyTab
(
    NodeId INT IDENTITY(1, 1)
    ,NodeDepth VARCHAR(100) NOT NULL
    ,NodePath HIERARCHYID NOT NULL
    ,NodeDesc VARCHAR(100)
)
```

```
GO
```

-- Creating constraint on hierarchy data type.

```
ALTER TABLE HierarchyTab ADD CONSTRAINT U_NodePath
UNIQUE CLUSTERED (NodePath)
```

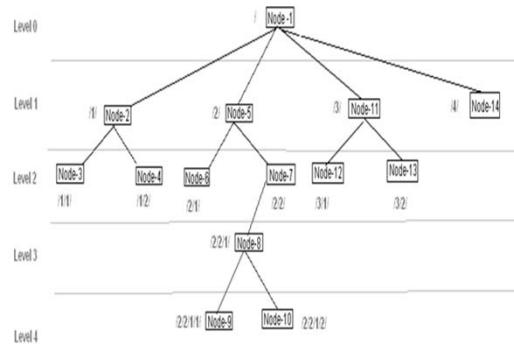
```
GO
```

-- Inserting data in above created table.

```
INSERT INTO HierarchyTab(NodeDepth,NodePath,NodeDesc)
VALUES
('1',HIERARCHYID::Parse('/'),'Node-1'),
('1.1',HIERARCHYID::Parse('/1/'),'Node-2'),
('1.1.1',HIERARCHYID::Parse('/1/1/'),'Node-3'),
('1.1.2',HIERARCHYID::Parse('/1/2/'),'Node-4'),
('1.2',HIERARCHYID::Parse('/2/'),'Node-5'),
('1.2.1',HIERARCHYID::Parse('/2/1/'),'Node-6'),
('1.2.2',HIERARCHYID::Parse('/2/2/'),'Node-7'),
('1.2.2.1',HIERARCHYID::Parse('/2/2/1/'),'Node-8'),
('1.2.2.1.1',HIERARCHYID::Parse('/2/2/1/1/'),'Node-9'),
('1.2.2.1.2',HIERARCHYID::Parse('/2/2/1/2/'),'Node-10'),
('1.3',HIERARCHYID::Parse('/3/'),'Node-11'),
('1.3.1',HIERARCHYID::Parse('/3/1/'),'Node-12'),
('1.3.2',HIERARCHYID::Parse('/3/2/'),'Node-13'),
('1.4',HIERARCHYID::Parse('/4/'),'Node-14')
```

```
GO
```

### Logical Representation of the table :



### Working with Methods to display data from the table:

#### -- GetRoot()

```

SELECT
HIERARCHYID::GetRoot() AS RootNode,
HIERARCHYID::GetRoot().ToString() AS RootNodePath
GO

```

#### Result :

RootNode	RootNodePath
0x	/

#### -- ToString()

```

SELECT
NodePath.ToString() AS NodeStringPath
,Nodeld
,NodeDepth
,NodePath
,NodeDesc
FROM HierarchyTab
GO

```

**Result :**

NodeStringPath	NodeId	NodeDepth	NodeDesc
	NodePath		
/	1	1	0x
	Node-1		
/1/	2	1.1	0x58
	Node-2		
/1/1/	3	1.1.1	0x5AC0
	Node-3		
/1/2/	4	1.1.2	0x5B40
	Node-4		
/2/	5	1.2	0x68
	Node-5		
/2/1/	6	1.2.1	0x6AC0
	Node-6		
/2/2/	7	1.2.2	0x6B40
	Node-7		
/2/2/1/	8	1.2.2.1	0x6B56
	Node-8		
/2/2/1/1/	9	1.2.2.1.1	
	0x6B56B0	Node-9	
/2/2/1/2/	10	1.2.2.1.2	
	0x6B56D0	Node-10	
/3/	11	1.3	0x78
	Node-11		
/3/1/	12	1.3.1	0x7AC0
	Node-12		
/3/2/	13	1.3.2	0x7B40
	Node-13		
/4/	14	1.4	0x84
	Node-14		

## Introduction to Alias Data Types (User Defined Data Types)

- An alias data type is a user defined custom data type based on system-supplied data type
- In SQL Server alias data types offers a great deal of data consistency while working with various tables or databases
- You should create an alias data type when:
  - You need to define a commonly used data element with specific format
  - Database tables must store the same type of data in a column
  - These columns have identical data type, length & nullability
- Example

```
CREATE TYPE EmailAddress
FROM varchar(30) NOT NULL;
```

June 22, 2015 | Proprietary and Confidential | -23- | IGATE | Speed. Agility. Imagination.

### Introduction to Alias Data Types (User Defined Data Types) :

An alias data type is a user defined custom data type based on system supplied data type. An alias data type allows you to refine a data type further to guarantee consistency when working with common data elements in various tables or databases. It provides you with a convenient way to standardize the usage of native data types for columns that have same domain of possible values. Assume that you need to store many email addresses in your database, in various tables. As there is no single & definitive way to store Email Address, it is very hard to maintain consistency in all those tables which store email address in a column. However, if the Email Address will be used regularly throughout the database, you could define a EmailAdress alias data type and use that instead. This also makes it easier to understand the object definitions and code in your database.

1. Name
2. System data type upon which the new data type is based
- Nullability (whether the data type allows null values)

Note - When nullability is not explicitly defined, it will be assigned based on the ANSI null default setting for the database or connection.

## Dropping Alias Data Types

- You can drop alias data type by using Object Explorer in SQL Server Management Studio
- You can also use DROP TYPE Transact-SQL Statement to remove the alias data type from a database
- The DROP TYPE statement removes an alias data type from the current database
- You cannot drop a user-defined type until all references to that type have been removed
- Example

```
DROP TYPE EmailAddress
```

June 22, 2015

Proprietary and Confidential

- 24 -



Note that when you create an alias data type, you can specify its nullability. An alias data type created with the NOT NULL option can never be used to store a NULL value.

It is important to specify the appropriate nullability when you create a data type, because it can be a lengthy procedure to change a data type. You must use the DROP TYPE statement to drop the data type and then re-create a new data type to replace it.

Because you cannot drop a data type that is used by tables in the database, you also need to ALTER every table that uses the data type first.

**Tip** – Alias data types that you create in the model database are automatically included in all databases that are subsequently created. However, if the data type is created in a user-defined database, the data type exists only in that user-defined database.

### Dropping Alias Data Types:

You can drop alias data types by using Object Explorer in SQL Server Management studio or by using DROP TYPE Transact-SQL statement.

The DROP TYPE statement will not execute when :

There are tables in the database that contain columns of the alias data type or the user-defined type.

There are computed columns, CHECK constraints, schema-bound views, and schema-bound functions whose definitions reference the alias or user-defined type.

3. There are functions, stored procedures, or triggers created in the database, and these routines use variables and parameters of the alias or user-defined type.

## Demo

- Creating alias data types (UDTs)
- Dropping alias data types (UDTs)



June 22, 2015

Proprietary and Confidential

IGATE  
Speed. Agility. Imagination

## Data Types in SQL Server –Best Practices

- If column length varies, use a variable data type
- Use tinyint appropriately
- For numeric data types, commonly use decimal
- If storage is greater than 8000 bytes, use text or image
- Use money for currency
- Do not use float or real as primary keys
- If your character data type columns use the same or a similar number of characters consistently, use fixed length data types (char, nchar)
- Choose the smallest numeric or character data type required to store the data
- If you are going to be using a column for frequent sorts, consider an integer-based column rather than a character-based column

## What is a Database Schema?

- A Database Schema is a way to logically group SQL Server objects such as tables, views, stored procedures etc
- A schema is a distinct namespace, a container of SQL Server objects, distinct from users those who have created those objects
- In earlier releases of SQL Server, an object's namespace was determined by the user name of its owner
- An object owned by a database user is no longer tied to that user
- By introducing Schemas in database objects can now be manipulated independently of user
- A SQL Server user can be defined with default schema
- If no default schema is defined, sql server will assume dbo as the default schema

June 22, 2015

Proprietary and Confidential

- 27 -



### What is a Database Schema?

Microsoft introduced the concept of database schemas. A schema is an independent entity- a container of objects distinct from the user who created those objects. A schema is a distinct namespace to facilitate the separation, management, and ownership of database objects. It removed the tight coupling of database objects and owners.

In earlier releases of SQL Server, database object owners and users were the same things. This meant that if, say, a user creates a table in the database, that user cannot be deleted without deleting the table or first transferring it to another user. SQL Server

introduced the concept of database schemas and the separation between database objects and ownership by users. An object owned by a database user is no longer tied to that user. The object now belongs to a schema – a container that can hold many database objects. The schema owner may own one or many schemas. This concept creates opportunities to expose database objects within a database for consumption yet protect them from modification, direct access using poor query techniques, or removal by users other than the owner.

When a schema is created, it is owned by a principal. A principal is any entity or object that has access to SQL Server resources. These are:

- Windows domain logins
- Windows local logins
- SQL Server logins
- Windows groups
- Database roles
- Server roles
- Application roles

## Creating Database Schema

➤ Example : Creating Database Schema

```
Use AdventureWorks
GO
CREATE SCHEMA Sales
GO
```

➤ Example : Assigning a Default Schema

```
ALTER USER Anders WITH DEFAULT_SCHEMA = Sales
```

June 22, 2015

Proprietary and Confidential

- 28 -


  
Speed. Agility. Imagination

### Creating Database Schema

As shown on the above slide, you can create a database schema by using CREATE SCHEMA Transact-SQL Statement.

### The dbo Schema

Every database contains a schema named dbo. The dbo schema is the default schema for all users who do not have any other explicitly defined default schema.

### How Object Name Resolution Works?

When a database contains multiple schemas, object name resolution can become confusing. For example, a database might contain two tables named Order in two different schemas, Sales and dbo. The qualified names of the objects within the Database are unambiguous: Sales.Order and dbo.Order, respectively. However, the use of the unqualified name Order can produce unexpected results. You can assign users a default schema to control how unqualified object names are resolved.

**How Name Resolution Works?**

SQL Server 2008 uses the following process to resolve an unqualified object name:

1. If the user has a default schema, SQL Server attempts to find the object in the default schema.
2. If the object is not found in the user's default schema, or if the user has no default schema, SQL Server attempts to find the object in the dbo schema.

For example, a user with the default schema Person executes the following Transact-SQL statement.

**SELECT \* FROM Contact**

SQL Server 2008 will first attempt to resolve the object name to Person.Contact.

If the Person schema does not contain an object named Contact, SQL Server will attempt to resolve the object name to dbo.Contact.

If a user with no defined default schema executes the same statement, SQL Server will immediately resolve the object name to dbo.Contact.

## Demo

### ➤ Creating Database Schema



June 22, 2015

Proprietary and Confidential

- 30 -

**IGATE**  
Speed. Agility. Imagination

## Create Table

- Once you define all the data types in your database, you can create the Tables to store your business data
- The Table is the most important and central object of any RDBMS
- These tables may be temporary, lasting only for a single interactive SQL Session
- They also can be permanent, lasting weeks or for months
- Creating a database table involves :
  - Naming the table
  - Defining the columns
  - Assigning properties to the column
- In SQL Server, you can use CREATE TABLE Transact-SQL Statement for creating table

June 22, 2015

Proprietary and Confidential



### Create table in SQL Server 2008

After you have designed the database , the tables that will store the data in the database can be created. The data is usually stored in permanent tables. Tables are stored in the database files until they are deleted and are available to any user who has the appropriate permissions.

#### Temporary Tables

You can also create temporary tables. Temporary tables are similar to permanent tables, except temporary tables are stored in tempdb and are deleted automatically when no longer in use.

The two types of temporary tables, local and global, differ from each other in their names, their visibility, and their availability. Local temporary tables have a single number sign (#) as the first character of their names; they are visible only to the current connection for the user; and they are deleted when the user disconnects from

instances of SQL Server. Global temporary tables have two number signs (##) as the first characters of their names; they are visible to any user after they are created; and they are deleted when all users referencing the table disconnect from SQL Server.

#### Example

If you create a table named employees, the table can be used by any person who has the security permissions in the database to use it, until the table is deleted. If you create a local temporary table named #employees, you are the only person who can work with the table, and it is deleted when you disconnect.

## Creating Table

### Given a Table Structure for Employee

Column Name	Data Type	Nullability
Employee_Code	Int	NOT NULL
Employee_Name	Varchar(40)	NOT NULL
Employee_DOB	Datetime	NOT NULL
Employee_EmailID	Varchar(20)	NULL

```
CREATE TABLE Employee
(
    Employee_Code int NOT NULL,
    Employee_Name varchar(40) NOT NULL,
    Employee_DOB datetime NOT NULL,
    Employee_EmailID varchar(20) NULL
)
```

June 22, 2015 | Proprietary and Confidential | - 32 -

If you create a global temporary table named ##employees, any user in the database can work with this table. If no other user works with this table after you create it, the table is deleted when you disconnect. If another user works with the table after you create it, SQL Server deletes it when both of you disconnect.

### Table Properties

You can define up to 1,024 columns per table. Table and column names must follow the rules for identifiers; they must be unique within a given table, but you can use the same column name in different tables in the same database. You must also define a data type for each column.

Although table names must be unique for each owner within a database, you can create multiple tables with the same name if you specify different owners for each. You can create two tables named employees and designate Jonah as the owner of one and Sally as the owner of the other. When you need to work with one of the employees tables, you can distinguish between the two tables by specifying the owner with the name of the table.

### Tables & Columns - Naming Guidelines

1. Use Pascal Casing (also known as upper camel casing)
2. Avoid abbreviations
3. A long name that users understand is preferred over a short name that users might not understand

## Adding and dropping a column

**ADD**

```
ALTER TABLE CategoriesNew
ADD Commission money null
```

Customer_name	Sales_amount	Sales_date	Customer ID	Commission

**DROP**

```
ALTER TABLE CategoriesNew
DROP COLUMN Sales_date
```

**IGATE**  
Speed. Agility. Imagination

### Modifying tables in SQL Server

After a table is created, you can change many of the options that were defined for the table when it was originally created. You can modify tables in many different ways.

You can modify tables to make changes to the columns, constraints, and indexes associated with tables. You can use ALTER TABLE statement to implement most common modifications to table

Using ALTER TABLE Columns can be added, modified, or deleted. For example, the column name, length, data type, precision, scale, and nullability can all be changed, although some restrictions exist. PRIMARY KEY and FOREIGN KEY constraints can be added or deleted. UNIQUE and CHECK constraints and DEFAULT definitions (and objects) can be added or deleted.

#### Adding & Dropping Columns :

When you use the ALTER TABLE statement to add a column, the new column is added at the end of the table.

There are also some things you need to consider with regard to the null option

specified for a new column. In the case of a column that allows nulls, there is no real

Issue. SQL Server adds the column and allows a NULL value for all rows. If NOT

NULL is specified, however, the column must be an identity column or have a default specified. Note that even if a default is specified, if the column allows nulls, the column does not be populated with the default. You use the WITH VALUES clause as part of the default specification to override this and populate the column with the default.

With some restrictions, columns can also be dropped from a table.

The following columns cannot be dropped:

A column in a schema-bound view

An indexed column

A replicated column

A column used in a CHECK, FOREIGN KEY, UNIQUE, or PRIMARY KEY constraints

A column that is associated with a default or bound to a default object

A column that is bound to a rule

## Special Types of Columns in SQL Server

### ➤ Computed Columns

```
CREATE TABLE Marks
( Test1 int, Test2 int,
  TestAvg AS (Test1 + Test2)/2 )
```

### ➤ Identity Columns

```
CREATE TABLE Printer
(PrinterId int IDENTITY (1000, 1) NOT NULL)
```

### ➤ Uniqueidentifier Columns

```
CREATE TABLE Customer ( CustomerID uniqueidentifier NOT NULL)
INSERT INTO Customer Values (NewID())
```

June 22, 2015

Proprietary and Confidential



#### Special Types of Columns in SQL Server

Computed Columns - A computed column is a column whose value is calculated based on other columns. Generally speaking, the column is a virtual column because it is calculated on-the-fly, and no value is stored in the database table. With SQL Server 2008, you have an option of actually storing the calculated value in the database. You do so by marking the column as persisted. If the computed column is persisted, you can create an index on this column as well.

Identity Columns - You can use the Identity property to create columns that contain system-generated sequential values that identify each row inserted into a table. Having SQL Server automatically provide key values can reduce costs and improve performance. It can be assigned only to columns that are of the following types:

decimal  
int  
numeric  
smallint  
bigint  
tinyint

Only one identity column can exist for each table, and that column cannot allow nulls. When implementing the IDENTITY property, you supply a seed and an increment. The seed is the starting value for the numeric count, and the increment is the amount by which it grows. A seed of 10 and an increment of 10 would produce values of 10, 20, 30, 40, and so on. If not specified, the default seed value is 1, and the increment is 1.

Uniqueidentifier Columns - Columns defined with the uniqueidentifier data type can be used to store globally unique identifiers (GUIDs), which are guaranteed to be universally unique. You can generate a value for a uniqueidentifier column by using the NEWID Transact-SQL function.

## SEQUENCE

- SQL Server 2012 introduces Sequence as database object.
- Sequence is an object in each database and is similar to IDENTITY in its functionality.
- It can have start value, incrementing value and an end value defined in it.
- It can be added to a column whenever required rather than defining an identity column individually for tables.
- Limitations of using the Identity property can be overcome by the introduction of this new object SEQUENCE.

## SEQUENCE

- **CREATESEQUENCE SQ1 AS INT**
- **START WITH 100**
- **INCREMENT BY 25**
- **MINVALUE 100**
- **MAXVALUE 200**
- **CYCLE**
- **CACHE 10**
- **GO**
- **SELECT NEXT VALUE FOR SQ1**

June 22, 2015

Proprietary and Confidential


  
Speed. Agility. Imagination

**CREATE SEQUENCE [schema\_name . ] sequence\_name [ AS [ built\_in\_integer\_type | user-defined\_integer\_type ] ][ START WITH <constant> ][ INCREMENT BY <constant> ][ { MINVALUE [ <constant> ] } | { NOMINVALUE } ][ { MAXVALUE [ <constant> ] } | { NOMAXVALUE } ][ CYCLE | { NOCYCLE } ][ { CACHE [ <constant> ] } | { NO CACHE } ][ ; ]**

sequence\_name Specifies the unique name by which the sequence is known in the database. Type is **sysname**.

[ built\_in\_integer\_type | user-defined\_integer\_type]A sequence can be defined as any integer type. The following types are allowed.

**tinyint** - Range 0 to 255

**smallint** - Range -32,768 to 32,767

**int** - Range -2,147,483,648 to 2,147,483,647

**bigint** - Range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807  
**decimal** and **numeric** with a scale of 0.

Any user-defined data type (alias type) that is based on one of the allowed types.

If no data type is provided, the **bigint** data type is used as the default.

**START WITH <constant>**The first value returned by the sequence object.

The **START** value must be a value less than or equal to the maximum and greater than or equal to the minimum value of the sequence object. The default start value for a new sequence object is the minimum value for an ascending sequence object and the maximum value for a descending sequence object.

**INCREMENT BY <constant>**Value used to increment (or decrement if negative) the value of the sequence object for each call to the **NEXT VALUE FOR** function. If the increment is a negative value, the sequence object is descending; otherwise, it is ascending. The increment cannot be 0. The default increment for a new sequence object is 1.

## SEQUENCE

- `CREATE SEQUENCE mySeq`
- `START WITH 10`
- `INCREMENT BY 5;`
- `SELECT NEXT VALUE FOR mySeq;`
- `SELECT NEXT VALUE FOR mySeq;`
- `SELECT NEXT VALUE FOR mySeq;`
- `create table myTable`
- `(sid int,`
- `Sname varchar(15))`
- `INSERT INTO myTable(sid, sname) VALUES`
- `(NEXT VALUE FOR mySeq, 'Tom');`
- `select * from myTable`
- `INSERT INTO myTable(sid, sname) VALUES`
- `(NEXT VALUE FOR mySeq, 'Moody');`

June 22, 2015

Proprietary and Confidential



[ MINVALUE <constant> | NO MINVALUE ]Specifies the bounds for the sequence object. The default minimum value for a new sequence object is the minimum value of the data type of the sequence object. This is zero for the `tinyint` data type and a negative number for all other data types.

[ MAXVALUE <constant> | NO MAXVALUE]Specifies the bounds for the sequence object. The default maximum value for a new sequence object is the maximum value of the data type of the sequence object.

[ CYCLE | NO CYCLE ]Property that specifies whether the sequence object should restart from the minimum value (or maximum for descending sequence objects) or throw an exception when its minimum or maximum value is exceeded. The default cycle option for new sequence objects is NO CYCLE. Note that cycling restarts from the minimum or maximum value, not from the start value.

[ CACHE [<constant>] | NO CACHE ]Increases performance for applications that use sequence objects by minimizing the number of disk IOs that are required to generate sequence numbers. Defaults to CACHE.

For example, if a cache size of 50 is chosen, SQL Server does not keep 50 individual values cached. It only caches the current value and the number of values left in the cache. This means that the amount of memory required to store the cache is always two instances of the data type of the sequence object.

## Identity Vs Sequence

- ToDo Assignment

## Demo

### ➤ Creating Table



June 22, 2015

Proprietary and Confidential

- 40 -

**IGATE**  
Speed. Agility. Imagination

## Enforcing Data Integrity

- The term **Data Integrity** refers to correctness and completeness of the data in a database
- To preserve the consistency and accuracy of data, every RDBMS imposes one or more data integrity constraints in a database
- These constraints restricts the wrong or invalid data values that can be inserted or updated in a database
- How and what kind of integrity is enforced in the database depends on the type integrity being enforced
- Enforcing data integrity ensures quality of the data in the database
- The quality of data refers to :
  - Accuracy
  - Completeness
  - Consistent
  - Correct

June 22, 2015 Proprietary and Confidential - 41 -



An important step in database planning is deciding the best way to enforce the integrity of the data. Data integrity refers to the consistency and accuracy of data that is stored in a database.

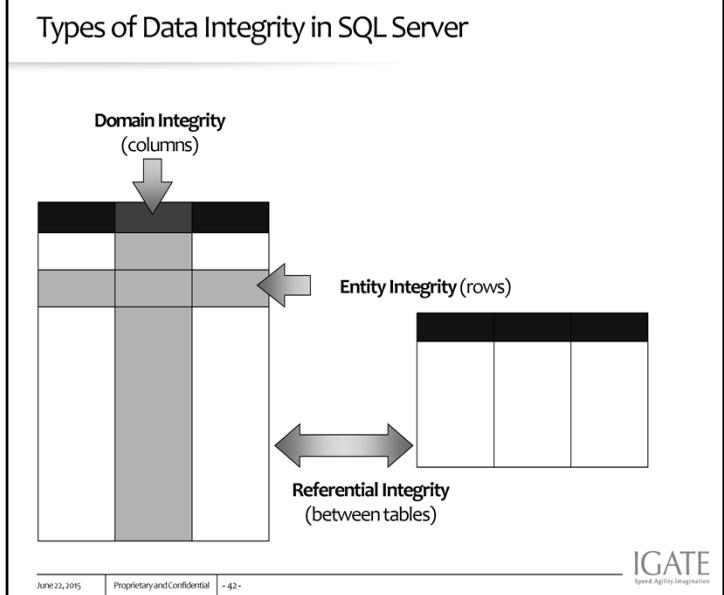
### What Is Data Integrity?

The term *data integrity* refers to the correctness and completeness of the data in a database. When the contents of a database are modified with the INSERT, DELETE, or UPDATE statements, the integrity of the stored data can be lost in many different ways.

For example:

- Invalid data may be added to the database, such as an Employee that belongs to a nonexistent department.
- Existing data may be modified to an incorrect value, such as reassigning an Employee to a nonexistent project.
- Changes to the database may be lost due to a system error or power failure.
- Changes may be partially applied, such as adding an order for a product without adjusting the quantity available for sale.

One of the important roles of a relational DBMS is to preserve the integrity of its stored data to the greatest extent possible. Enforcing data integrity ensures the quality of the data in the database.



### Types of Data Integrity in SQL Server

Enforcing data integrity guarantees the quality of the data in the database. For example, if an employee is entered with an employee ID value of 123, the database should not permit another employee to have an ID with the same value. If you have an employee\_rating column intended to have values ranging from 1 to 5, the database should not accept a value outside that range. If the table has a dept\_id column that stores the department number for the employee, the database should permit only values that are valid for the department numbers in the company.

Two important steps in planning tables are to identify valid values for a column and to decide how to enforce the integrity of the data in the column. Data integrity falls into the following categories:

- Entity integrity
- Domain integrity
- Referential integrity

## Ways of implementing Data Integrity in SQL Server

### ➤ Declarative Data Integrity

- Criteria defined in object definitions
- SQL Server enforces automatically
- Implement by using constraints, defaults, and rules
- Simplest integrity check

### ➤ Procedural Data Integrity

- Implement by using triggers, stored procedures & application code
- Data validation criteria will be defined in script
- Allows more advanced data integrity checks

---

June 22, 2015

Proprietary and Confidential



There are two ways to implement data integrity in SQL Server, either using declarative data integrity or procedural data integrity.

Declarative Data Integrity – It is a set of rules that are applied to a table and its columns using the CREATE TABLE or ALTER TABLE statements. These rules are called constraints, rules and defaults. This is the preferred and simplest method of enforcing integrity because it has low overhead and requires little or no custom programming.

Procedural Data Integrity - Procedural integrity can be implemented with stored procedures, triggers, and application code. It allows more advanced integrity check as You can implement the custom code in many different ways to enforce the integrity of your data.

## Determining the type of constraint to be used

Type of integrity	Constraint type
Domain	DEFAULT
	CHECK
	REFERENTIAL
Entity	PRIMARY KEY
	UNIQUE
Referential	FOREIGN KEY
	CHECK

June 22, 2015

Proprietary and Confidential


  
Speed. Agility. Imagination.

## Determining the type of constraint to be used

### Entity Integrity

Entity integrity defines a row as a unique entity for a particular table. Entity integrity enforces the integrity of the identifier columns or the primary key of a table, through UNIQUE indexes, UNIQUE constraints or PRIMARY KEY constraints.

### Domain Integrity

Domain integrity is the validity of entries for a specific column. You can enforce domain integrity to restrict the type by using data types, restrict the format by using CHECK constraints and rules, or restrict the range of possible values by using FOREIGN KEY constraints, CHECK constraints, DEFAULT definitions, NOT NULL definitions, and rules.

### Referential Integrity

Referential integrity preserves the defined relationships between tables when rows are entered or deleted. In SQL Server, referential integrity is based on relationships between foreign keys and primary keys or between foreign keys and unique keys, through FOREIGN KEY and CHECK constraints. Referential integrity makes sure that key values are consistent across tables. This kind of consistency requires that there are no references to nonexistent values and that if a key value changes, all references to it change consistently throughout the database. When you enforce referential integrity, SQL Server prevents users from doing the following:

Adding or changing rows to a related table if there is no associated row in the

primary table.

Changing values in a primary table that causes orphaned rows in a related table.

Deleting rows from a primary table if there are matching related rows.

## Creating Constraints

- Use CREATE TABLE or ALTER TABLE
- Can Add Constraints to a Table with existing Data
- Can Place Constraints on Single or Multiple Columns
  - Single column, called column-level constraint
  - Multiple columns, called table-level constraint

June 22, 2015

Proprietary and Confidential



You must declare constraints that operate on more than one column as table-level constraints.

For example, the following create table statement has a check constraint that operates on two columns, pub\_id and pub\_name:

```
create table my_publishers
(
    pub_id char(4), pub_name varchar(40),
    constraint my_chk_constraint check (pub_id in ("1234", "4321",
    "1212") or pub_name not like "Bad Books")
)
```

You can declare constraints that operate on just one column as column-level constraints, but it is not required. For example, if the above check constraint uses only one column (pub\_id), you can place the constraint on that column:

```
create table my_publishers
(
    pub_id char(4) constraint my_chk_constraint check (pub_id in
    ("1389", "0736", "0877")),
    pub_name varchar(40)
)
```

In either case, the constraint keyword and accompanying constraint\_name are optional.

### Consideration for using constraints

- Can be changed without recreating a table
- Require error-checking in applications and transactions
- Verify existing data

## Types constraints

- **DEFAULT Constraints**
- **CHECK Constraints**
- **PRIMARY KEY Constraints**
- **UNIQUE Constraints**
- **FOREIGN KEY Constraints**
- **Cascading Referential Integrity**

June 22, 2015 | Proprietary and Confidential | - 47 -



SQL Server supports the following classes of constraints:  
NOT NULL specifies that the column does not accept NULL values.

CHECK constraints enforce domain integrity by limiting the values that can be put in a column. A CHECK constraint specifies a Boolean (evaluates to TRUE, FALSE, or unknown) search condition that is applied to all values that are entered for the column. All values that evaluate to FALSE are rejected. You can specify multiple CHECK constraints for each column.

UNIQUE constraints enforce the uniqueness of the values in a set of columns.

In a UNIQUE constraint, no two rows in the table can have the same value for the columns. Primary keys also enforce uniqueness, but primary keys do not allow for NULL as one of the unique values.

PRIMARY KEY constraints identify the column or set of columns that have values that uniquely identify a row in a table. No two rows in a table can have the same primary key value. You cannot enter NULL for any column in a primary key. We recommend using a small, integer column as a primary key. Each table should have a primary key. A column or combination of columns that qualify as a primary key value is referred to as a candidate key.

FOREIGN KEY constraints identify and enforce the relationships between tables.

## DEFAULT constraints

- Apply only to INSERT statements
- Only one DEFAULT constraint per column
- Cannot be used with IDENTITY property or rowversion data type
- Allow some system-supplied values

```
USE Northwind
ALTER TABLE dbo.Customers
ADD
CONSTRAINT DF_contactname DEFAULT 'UNKNOWN'
FOR ContactName
```

June 22, 2015 | Proprietary and Confidential | - 48 -



Defaults specify what values are used in a column if you do not specify a value for the column when you insert a row. Defaults can be anything that evaluates to a constant, such as a constant, built-in function, or mathematical expression.

To apply defaults, create a default definition by using the DEFAULT keyword in CREATE TABLE. This assigns a constant expression as a default on a column.

Each column in a record must contain a value, even if that value is NULL. There may be situations when you must load a row of data into a table but you do not know the value for a column, or the value does not yet exist. If the column allows for null values, you can load the row with a null value. Because nullable columns may not be desirable, a better solution could be to define, where appropriate, a DEFAULT definition for the column. For example, it is common to specify zero as the default for numeric columns, or N/A as the default for string columns when no value is specified.

When you load a row into a table with a DEFAULT definition for a column, you implicitly instruct the SQL Server 2008 Database Engine to insert a default value in the column when a value is not specified for it.

Note: You can also use the DEFAULT VALUES clause of the INSERT STATEMENT to explicitly instruct the Database Engine to insert a default value for a column.

If a column does not allow for null values and does not have a DEFAULT definition, you must explicitly specify a value for the column, or the Database Engine returns an error that states that the column does not allow null values.

The value inserted into a column that is defined by the combination of the DEFAULT definition and the nullability of the column can be summarized as shown in the following table.

Column definition	No entry, no DEFAULT	No entry, DEFAULT
-------------------	----------------------	-------------------

Enter a null value

Allows null values	NULL	Default
value	NULL	
Disallow null values	Error	Default
value	Error	

## CHECK constraints

- Are used with INSERT and UPDATE statements
- Can reference other columns in the same table
- Cannot:
  - Be used with the rowversion data type
  - Contain subqueries

```
USE Northwind  
ALTER TABLE dbo.Employees  
ADD  
CONSTRAINT CK_birthdate  
CHECK(BirthDate > '01-01-1900' AND BirthDate < getdate())
```

June 22, 2015 | Proprietary and Confidential | - 49 -



You can declare a **check** constraint to limit the values users insert into a column in a table. Check constraints are useful for applications that check a limited, specific range of values. A **check** constraint specifies a *search\_condition* that any value must pass before it is inserted into the table. A *search\_condition* can include:

A list of constant expressions introduced with **in**

A range of constant expressions introduced with **between**

A set of conditions introduced with **like**, which may contain wildcard characters

An expression can include arithmetic operations and Transact-SQL built-in functions. The *search\_condition* cannot contain subqueries, a set function specification, or a target specification.

## PRIMARY KEY constraints

- Only one PRIMARY KEY constraint per table
- Values must be unique
- Null values are not allowed

```
USE Northwind  
ALTER TABLE dbo.Customers  
ADD  
CONSTRAINT PK_Customers  
PRIMARY KEY NONCLUSTERED (CustomerID)
```

June 22, 2015 | Proprietary and Confidential | - 50 -



A table typically has a column or combination of columns that contain values that uniquely identify each row in the table. This column, or columns, is called the primary key (PK) of the table and enforces the entity integrity of the table. You can create a primary key by defining a PRIMARY KEY constraint when you create or modify a table.

A table can have only one PRIMARY KEY constraint, and a column that participates in the PRIMARY KEY constraint cannot accept null values. Because PRIMARY KEY constraints guarantee unique data, they are frequently defined on an identity column.

When you specify a PRIMARY KEY constraint for a table, the SQL Server 2008 Database Engine enforces data uniqueness by creating a unique index for the primary key columns. This index also permits fast access to data when the primary key is used in queries. Therefore, the primary keys that are chosen must follow the rules for creating unique indexes.

If a PRIMARY KEY constraint is defined on more than one column, values may be duplicated within one column, but each combination of values from all the columns in the PRIMARY KEY constraint definition must be unique.

## UNIQUE constraints

- Allow One Null Value
- Allow Multiple UNIQUE Constraints on a Table
- Defined with One or More Columns

```
USE Northwind  
ALTER TABLE dbo.Suppliers  
ADD  
CONSTRAINT U_CompanyName  
UNIQUE NONCLUSTERED (CompanyName)
```

June 22, 2015 | Proprietary and Confidential | - 51 -



You can use UNIQUE constraints to make sure that no duplicate values are entered in specific columns that do not participate in a primary key. Although both a UNIQUE constraint and a PRIMARY KEY constraint enforce uniqueness, use a UNIQUE constraint instead of a PRIMARY KEY constraint when you want to enforce the uniqueness of a column, or combination of columns, that is not the primary key.

Multiple UNIQUE constraints can be defined on a table, whereas only one PRIMARY KEY constraint can be defined on a table.

Also, unlike PRIMARY KEY constraints, UNIQUE constraints allow for the value NULL. However, as with any value participating in a UNIQUE constraint, only one null value is allowed per column.

A UNIQUE constraint can be referenced by a FOREIGN KEY constraint.

## FOREIGN KEY constraints

- Must Reference a PRIMARY KEY or UNIQUE Constraint
- Provide Single or Multicolumn Referential Integrity
- Do Not Automatically Create Indexes
- Users Must Have SELECT or REFERENCES Permissions on Referenced Tables
- Use Only REFERENCES Clause Within Same Table

```
USE Northwind
ALTER TABLE dbo.Orders
ADD CONSTRAINT FK_Orders_Customers
FOREIGN KEY(CustomerID)
REFERENCES dbo.Customers(CustomerID)
```

June 22, 2015 | Proprietary and Confidential | +52 -

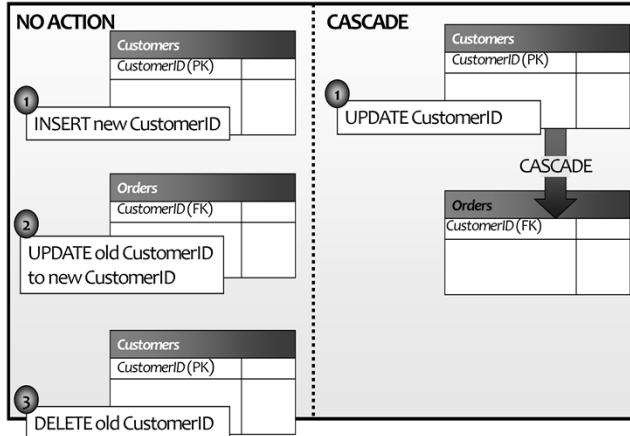


A foreign key (FK) is a column or combination of columns that is used to establish and enforce a link between the data in two tables. You can create a foreign key by defining a FOREIGN KEY constraint when you create or modify a table.

In a foreign key reference, a link is created between two tables when the column or columns that hold the primary key value for one table are referenced by the column or columns in another table. This column becomes a foreign key in the second table.

A FOREIGN KEY constraint does not have to be linked only to a PRIMARY KEY constraint in another table; it can also be defined to reference the columns of a UNIQUE constraint in another table. A FOREIGN KEY constraint can contain null values; however, if any column of a composite FOREIGN KEY constraint contains null values, verification of all values that make up the FOREIGN KEY constraint is skipped. To make sure that all values of a composite FOREIGN KEY constraint are verified, specify NOT NULL on all the participating columns.

## Cascading Referential Integrity Constraints



June 22, 2015 | Proprietary and Confidential | - 53 -

IGATE  
Speed. Agility. Imagination

### Cascading Referential Integrity Constraints

Cascading referential integrity constraints allow you to define the actions SQL Server takes when a user attempts to delete or update a key to which existing foreign keys point.

The REFERENCES clauses of the CREATE TABLE and ALTER TABLE statements support ON DELETE and ON UPDATE clauses:

```
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE | NO ACTION } ]
```

NO ACTION is the default if ON DELETE or ON UPDATE is not specified. NO ACTION specifies the same behavior that occurs in earlier versions of SQL Server.

CASCADE allows deletions or updates of key values to cascade through the tables defined to have foreign key relationships that can be traced back to the table on which the modification is performed. CASCADE cannot be specified for any foreign keys or primary keys that have a timestamp column.

#### ON DELETE CASCADE

Specifies that if an attempt is made to delete a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted. If cascading referential actions have also been defined on the target tables, the specified cascading actions are also taken for the rows deleted from those tables.

#### ON UPDATE CASCADE

Specifies that if an attempt is made to update a key value in a row, where the key value is referenced by foreign keys in existing rows in other tables, all of the foreign key values are also updated to the new value specified for the key. If cascading referential actions have also been defined on the target tables, the specified cascading actions are also taken for the key values updated in those tables.

## Demo

- Demonstration on all types of SQL Server Constraints



June 22, 2015

Proprietary and Confidential

**IGATE**  
Speed. Agility. Imagination

## Disabling Constraints

- Disabling constraint checking on existing data
- Disabling constraint checking when loading new data

---

June 22, 2015

Proprietary and Confidential



Sometimes you need to perform some actions that require the FOREIGN KEY or CHECK constraints be disabled, for example, your company do not hire foreign employees, you made the appropriate constraint, but the situation was changed and your boss need to hire the foreign employee, but only this one. In this case, you need to disable the constraint by using the ALTER TABLE statement. After these actions will be performed, you can re-enable the FOREIGN KEY and CHECK constraints by using the ALTER TABLE statement.

## Disabling Constraint Checking on Existing Data

- Applies to CHECK and FOREIGN KEY constraints
- Use WITH NOCHECK option when adding a new constraint
- Use if existing data will not change
- Can change existing data before adding constraints

```
USE Northwind  
ALTER TABLE dbo.Employees  
    WITH NOCHECK  
        ADD CONSTRAINT FK_Employees_Employees  
        FOREIGN KEY (ReportsTo)  
        REFERENCES dbo.Employees(EmployeeID)
```

June 22, 2015 | Proprietary and Confidential | - 56 -

**IGATE**  
Speed. Agility. Imagination

### WITH CHECK | WITH NOCHECK

Specifies whether the data in the table is or is not validated against a newly added or re-enabled FOREIGN KEY or CHECK constraint. If not specified, WITH CHECK is assumed for new constraints, and WITH NOCHECK is assumed for re-enabled constraints.

If you do not want to verify new CHECK or FOREIGN KEY constraints against existing data, use WITH NOCHECK. We do not recommend doing this, except in rare cases. The new constraint will be evaluated in all later data updates. Any constraint violations that are suppressed by WITH NOCHECK when the constraint is added may cause future updates to fail if they update rows with data that does not comply with the constraint.

The query optimizer does not consider constraints that are defined WITH NOCHECK. Such constraints are ignored until they are re-enabled by using ALTER TABLE table CHECK CONSTRAINT ALL.

## Disabling constraint checking when loading new data

- Applies to CHECK and FOREIGN KEY constraints
- Use when:
  - Data conforms to constraints
  - You load new data that does not conform to constraints

```
USE Northwind  
ALTER TABLE dbo.Employees  
NOCHECK  
CONSTRAINT FK_Employees_Employees
```

## Using DEFAULTS & RULES

➤ As independent objects they:

- Are defined once
- Can be bound to one or more columns or user-defined data types

```
CREATE DEFAULT phone_no_default
AS '(000)000-0000'
GO
EXEC sp_bindefault phone_no_default,
'Customers.Phone'
```

```
CREATE RULE regioncode_rule
AS @regioncode IN ('IA', 'IL', 'KS', 'MO')
GO
EXEC sp_bindrule regioncode_rule,
'Customers.Region'
```

June 22, 2015 | Proprietary and Confidential | -58-

**IGATE**  
Speed. Agility. Imagination

### Using Defaults and Rules

#### Default :-

When bound to a column or an alias data type, a default specifies a value to be inserted into the column to which the object is bound (or into all columns, in the case of an alias data type), when no value is explicitly supplied during an insert.

```
CREATE DEFAULT [ schema_name . ] default_name
AS constant_expression [ ; ]
```

A default name can be created only in the current database. Within a database, default names must be unique by schema. When a default is created, use sp\_bindefault to bind it to a column or to an alias data type.

#### Rules :-

Creates an object called a rule. When bound to a column or an alias data type, a rule specifies the acceptable values that can be inserted into that column.

A column or alias data type can have only one rule bound to it. However, a column can have both a rule and one or more check constraints associated with it. When this is true, all restrictions are evaluated.

A rule can be created only in the current database. After you create a rule, execute sp\_bindrule to bind the rule to a column or to alias data type. A rule must be compatible with the column data type. For example, "@value LIKE A%" cannot be used as a rule for a numeric column. A rule cannot be bound to a text, ntext, image, varchar(max), nvarchar(max), varbinary(max), xml, CLR user-defined type, or timestamp column. A rule cannot be bound to a computed column.

## Demo

- Disabling Constraints
- Working with DEFAULTS & RULES



June 22, 2015

Proprietary and Confidential

- 59 -

**IGATE**  
Speed. Agility. Imagination

### Deciding - Enforcement method to use

Data integrity components	Functionality	Performance costs	Before or after modification
Constraints	Medium	Low	Before
Defaults and rules	Low	Low	Before
Triggers	High	Medium-High	After
Data types, Null/Not Null	Low	Low	Before



## TRUNCATE

- Removes all rows from a table
- TRUNCATE TABLE is similar to the DELETE statement with no WHERE clause

```
TRUNCATE TABLE Employees
```

## Dropping table

- At times you need to delete a table, for example when you want to implement a new design or free up space in the database
- You can use **DROP TABLE** Transact-SQL statement to drop the table from Database

```
DROP TABLE Employee
```

- You can reference multiple tables in a single **DROP TABLE** command by separating the table names with comma

June 22, 2015

Proprietary and Confidential



### Dropping table SQL Server 2008

At times you need to delete a table (for example, when you want to implement a new design or free up space in the database). When you delete a table, its structural definition, data, full-text indexes, constraints, and indexes are permanently deleted from the database, and the space formerly used to store the table and its indexes is made available for other tables. You can explicitly drop a temporary table if you do not want to wait until it is dropped automatically.

A big consideration when dropping a table is the table's relationship to other tables. If a foreign key references the table that you want to drop, the referencing table or foreign key constraint must be dropped first. In a database that has many related tables, this can get complicated. Fortunately, a few tools can help you through this. The system Stored procedure `sp_helpconstraint` is one of the tools. This procedure lists all the foreign key constraints that reference a table.

The other approach is to right-click the table in Object Explorer and choose **View Dependencies**.

The dialog that appears gives you the option of viewing the objects that depend on the table or viewing the objects on which the table depends.

#### To delete a table - **DROP TABLE**

A. Drop a table in the current database  
**DROP TABLE titles**

This example removes the titles table and its data and indexes from the current database.

B. Drop a table in another database  
**DROP TABLE pubs.dbo.authors**

This example drops the authors table in the pubs database. It can be executed from any database.

## Commonly used T-SQL Statement

➤ **Transact-SQL statements can be used for:**

- Creating a database
- Creating and altering table
- Insert, update & delete data in the table
- Reading data from the table

## Data Manipulation Language

- **INSERT** - Adds a new row to a table
- **UPDATE** - Changes existing data in a table
- **DELETE** - Removes rows from a table
- **MERGE** - Merges the data

## Using INSERT

- **INSERT statement adds one or more new rows to a table**

```
INSERT [INTO] table or view [(column_list)] data_values
```

```
INSERT INTO MyTable (PriKey, Description)
VALUES (123, 'A description of part 123.')
GO
```

- **INSERT using SELECT statement**

```
INSERT INTO MyTable (PriKey, Description)
SELECT ForeignKey, Description
FROM SomeTable
GO
```

June 22, 2015 | Proprietary and Confidential | - 66 -



The **INSERT** statement adds one or more new rows to a table. In a simplified treatment, **INSERT** has the following form:

**INSERT [INTO] table\_or\_view [(column\_list)] data\_values**

The **INSERT** statement inserts **data\_values** as one or more rows into the specified table or view. **column\_list** is a list of column names, separated by commas, that can be used to specify the columns for which data is supplied. If **column\_list** is not specified, all the columns in the table or view receive data.

When **column\_list** does not specify all the columns in a table or view, either the default value, if a default is defined for the column, or **NULL** is inserted into any column that is not specified in the list. All columns that are not specified in the column list must either allow for null values or have a default value assigned. **INSERT** statements do not specify values for the following types of columns because the SQL Server 2008 Database Engine generates the values for these columns: 1.Columns with an **IDENTITY** property that generates the values for the column. 2.Columns that have a default that uses the **NEWID** function to generate a unique GUID value.

**CREATE TABLE** *dbo.T1*

```
( column_1 int IDENTITY, column_2 varchar(30)
    CONSTRAINT default_name DEFAULT ('my column default')
    INSERT INTO dbo.T1 (column_2) VALUES ('Explicit value');
    INSERT INTO T1 DEFAULT VALUES;
    If we have to insert into column_1 value explicitly into this table
    then we need to SET IDENTITY_INSERT ON
    SET IDENTITY_INSERT dbo.T1 ON
    INSERT INTO dbo.t1 (column_1,column_2) values(3,'some data');
```

## UPDATE statement

- UPDATE statement can change data values in single rows, groups of rows, or all the rows in a table or view

```
UPDATE products  
SET unitprice=unitprice*1.1  
WHERE categoryid=2  
GO
```

June 22, 2015 | Proprietary and Confidential | - 67 -



The UPDATE statement can change data values in single rows, groups of rows, or all the rows in a table or view. It can also be used to update rows in a remote server. An UPDATE statement referencing a table or view can change the data in only one base table at a time.

The UPDATE statement has the following major clauses:

### SET

Contains a comma-separated list of the columns to be updated and the new value for each column, in the form *column\_name = expression*. The value supplied by the expressions includes items such as constants, values selected from a column in another table or view, or values calculated by a complex expression.

### FROM

Identifies the tables or views that supply the values for the expressions in the SET clause, and optional join conditions between the source tables or views.

### WHERE

Specifies the search condition that defines the rows from the source tables and views that qualify to provide values to the expressions in the SET clause.

## DELETE statement

- Removes one or more rows in a table or view
- A simplified form of the DELETE syntax is:

```
DELETE table_or_view  
FROM table_sources  
WHERE search_condition
```

- If a WHERE clause is not specified, all the rows in table\_or\_view are deleted

June 22, 2015

Proprietary and Confidential

**IGATE**  
Speed. Agility. Imagination

The DELETE statement removes one or more rows in a table or view.

A simplified form of the DELETE syntax is:

```
DELETE table_or_view  
FROM table_sources  
WHERE search_condition
```

The parameter table\_or\_view names a table or view from which the rows are to be deleted. All rows in table\_or\_view that meet the qualifications of the WHERE search condition are deleted. If a WHERE clause is not specified, all the rows in table\_or\_view are deleted. The FROM clause specifies additional tables or views and join conditions that can be used by the predicates in the WHERE clause search condition to qualify the rows to be deleted from table\_or\_view. Rows are not deleted from the tables named in the FROM clause, only from the table named in table\_or\_view.

Any table that has all rows removed remains in the database. The DELETE statement deletes only rows from the table; the table must be removed from the database by using the DROP TABLE statement.

## MERGE statement (SQL Server 2008 onwards)

- In a typical database application, quite often you need to perform INSERT, UPDATE and DELETE operations on a TARGET table by matching the records from the SOURCE table
- To accomplish this, In previous versions of SQL Server, we had to write separate statements to INSERT, UPDATE, or DELETE data based on certain conditions
- Though it seems to be straight forward at first glance, but it becomes cumbersome when you have do it very often or on multiple tables
- Even the performance degrades significantly with this approach
- Now you can use MERGE SQL command to perform these operations in a single statement
- Using MERGE statement we can include the logic of such data modifications in one statement that even checks when the data is matched then just update it and when unmatched then insert it

June 22, 2015 | Proprietary and Confidential | - 69 -



### MERGE Statement

In a typical database application, quite often you need to perform INSERT, UPDATE and DELETE operations on a TARGET table by matching the records from the SOURCE table. For example, a products dimension table has information about the products; you need to sync-up this table with the latest information about the products from the source table. You would need to write separate INSERT, UPDATE and DELETE statements to refresh the target table with an updated product list or do lookups. Though it seems to be straight forward at first glance, but it becomes cumbersome when you have do it very often or on multiple tables, even the performance degrades significantly with this approach.

With SQL Server 2008, now you can use MERGE SQL command to perform these operations in a single statement. The MERGE statement basically merges data from a source result set to a target table based on a condition that you specify and if the data from the source already exists in the target or not. The new SQL command combines the sequence of conditional INSERT, UPDATE and DELETE commands in a single atomic statement, depending on the existence of a record.

## MERGE statement

➤ **The Merge statement comprises Five clauses**

- MERGE - Used to specify the Target tables to be inserted/updated/deleted
- USING - Used to specify the Source table
- ON - Used to specify the join condition on source and target tables
- WHEN- Used to specify the action to be taken place based on the result of the ON clause
- OUTPUT - Used to return the value of Insert/Update/Delete operations using the INSERTED / DELETED magic tables

➤ **MERGE Statement - Syntax**

```

MERGE <target_table> [AS TARGET]
USING <table_source> [AS SOURCE]
ON <search_condition>
[WHEN MATCHED THEN <merge_matched> ]
[WHEN NOT MATCHED [BY TARGET] THEN <merge_not_matched> ]
[WHEN NOT MATCHED BY SOURCE THEN <merge_matched> ];

```

June 22, 2015

Proprietary and Confidential

- 70 -


  
IGATE
   
Speed. Agility. Imagination

### The Clauses used in MERGE Statement

The Merge statement uses two tables (Source and Target tables). The Source table is specified with USING clause. The Target table is specified with MERGE INTO clause. Merge statement is similar to OUTER joins. We can use condition with Merge statement like WHEN MATCHED THEN, WHEN NOT MATCHED [BY TARGET] THEN , WHEN NOT MATCHED BY SOURCE THEN.

If the record is already there in target table then we can update the data, If its not there in target table then we an insert.

The OUTPUT clause can be used with Merge statement to return an output as INSERTED / DELETED like magic tables. We can use \$action function to identity that what kind of action taken place like INSERT / UPDATE / DELETE.

## MERGE statement

### ➤ MERGE Statement - Example

```
MERGE INTO dbo.Customers AS TGT
USING dbo.CustomersStage AS SRC
ON TGT.custid = SRC.custid
WHEN MATCHED THEN
UPDATE SET
TGT.companyname = SRC.companyname,
TGT.phone = SRC.phone,
TGT.address = SRC.address
WHEN NOT MATCHED THEN
INSERT(custid, companyname, phone, address)
VALUES(SRC.custid, SRC.companyname, SRC.phone, SRC.address)
WHEN NOT MATCHED BY SOURCE THEN
DELETE
OUTPUT
$action, deleted.custid AS del_custid, inserted.custid AS ins_custid;
```

## MERGE statement

➤ **The given example does the following**

- MERGE statement defines the Customers table as the target for the modification
- CustomersState table as the source
- The MERGE condition matches the custid attribute in the source with the custid attribute in the target
- When a match is found in the target, the target customer's attributes are overwritten with the source customer attributes
- When a match is not found in the target, a new row is inserted into the target, using the source customer attributes
- When a match is not found in the source, the target customer row is deleted

## Demo

### ➤ Using MERGE Statement



June 22, 2015

Proprietary and Confidential

IGATE  
Speed. Agility. Imagination

## DCL Statements

### ➤ DCL (Data Control Language)

- DCL Statement is used for securing the database
- DCL Statement control access to database
- DCL statements supported by T-SQL are GRANT , REVOKE and DENY

## DCL Statements

- **GRANT** authorizes one or more users to perform an operation or a set of operations on an object.
  - GRANT PRIVILEGES ON object-name TO endusers;
  - grant select,update on My\_table to some\_user, another\_user;
  - grant insert (empno, ename, job) on emp to endusers;
- **REVOKE** removes or restricts the capability of a user to perform an operation or a set of operations.
  - REVOKE PRIVILEGES ON object\_name FROM endusers.
  - revoke insert, delete on emp from enduser.
- **DENY** denies permission to a user/group , even though he may belong to a group/role having the permission
  - DENY SELECT ON Employees TO Ruser1

## Summary

➤ **In this lesson, you have learnt:**

- Different datatypes to store numeric, character, monetary and date time data
- New data types in SQL Server 2008 - Hierarchyid & Spatial
- Creating UDTs
- Creating Tables ,Manipulating Data and Assigning and revoking privileges.
- MERGE provides an efficient way to perform multiple DML operations
- Implement constraints like – Primary key, Foreign key, check, default and unique
- Alter table to change structure, enable/disable constraints



**IGATE**  
Speed. Agility. Imagination

## Review Question

- Question 1: To create a user defined data type we use \_\_\_\_.
- Question 2: If we want the column data to have unique values with null then we should use \_\_\_\_\_ constraint.
- Question 3: \_\_\_\_\_ and \_\_\_\_\_ are objects like constraints that are bound to the table.
- Question 4: \_\_\_\_\_ option is used to create a constraint that should not be temporarily checked for.
- Question 5: Use \_\_\_\_\_ as a data type to create tables with a hierarchical structure.
- Question 6: \_\_\_\_\_ Used to return the value of Insert/Update/Delete operations using the INSERTED / DELETED magic tables



IGATE  
Speed. Agility. Imagination

# SQL Server 2012 – Database Development

Lesson 03: Beginning with Transact-SQL

June 22, 2015

Proprietary and Confidential

- 1 -

**IGATE**  
Speed. Agility. Imagination

## Lesson Objectives

➤ **In this lesson, you will learn:**

- Transact-SQL Programming Language
- Types of Transact-SQL Statements
- Transact-SQL Syntax Elements
- Restricting rows
- Operators
- Functions – String, Date, Mathematical, System, Others
- Grouping and summarizing data



**IGATE**  
Speed. Agility. Imagination

June 22, 2015

Proprietary and Confidential

- 2 -

## T-SQL Languages

- SQL is a special-purpose language used to define, access, and manipulate data
- SQL is nonprocedural language, it only describes the necessary components like tables and desired results without specifying exactly how those results should be computed
- SQL comes in many flavors
- Microsoft SQL Server makes use of Transact-SQL

## T-SQL Sub Language

- The Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
- Transactional Control Language (TCL)

## T-SQL Sub Language - DDL

- It is the subset of SQL which contains the commands used to create and destroy databases and database objects
- DDL includes the commands for handling tasks such as creating tables, indexes, views, and constraints
- The commands are
  - CREATE
  - ALTER
  - DROP
  - TRUNCATE

## T-SQL Sub Language - DML

- It is the subset of SQL used to access and manipulate data contained within the data structures previously defined via DDL
- The Commands are:
  - INSERT
  - UPDATE
  - DELETE
  - MERGE
  - [SELECT]

## T-SQL Sub Language - DCL

- It is the subset of SQL Commands that control a database, including administering privileges and committing data
- It is used to create roles, permissions, and to control access to database.
- The Commands are
  - GRANT
  - REVOKE
  - DENY

## T-SQL Sub Language - Transactional Control Language

- It is used to manage different transactions occurring within a database.
- The commands are
  - COMMIT
  - ROLLBACK
  - SAVE TRANSACTION

## Transact-SQL Programming Language

- Transact SQL, also called T-SQL, is Microsoft's extension to the ANSI SQL language
- Structured Query Language(SQL), is a standardized computer language that was originally developed by IBM
- T-SQL expands on the SQL standard to include procedural programming, local variables, various support functions for string processing, date processing, mathematics, etc
- Transact-SQL is central to using SQL Server
- All applications that communicate with SQL Server do so by sending Transact-SQL statements to the server, regardless of the user interface of the application

---

June 22, 2015

Proprietary and Confidential

- 9 -



### Transact-SQL Programming Language

**T-SQL (Transact SQL)** is a proprietary SQL extension used by Microsoft SQL Server. T-SQL adds extensions for procedural programming.

Transact-SQL is central to using SQL Server. All applications that communicate with an instance of SQL Server do so by sending Transact-SQL statements to the server, regardless of the user interface of the application.

## Querying Data – SELECT Statement

- The Transact-SQL language has one basic statement for retrieving information from a database: the **SELECT** statement
- With this statement, it is possible to query information from one or more tables of a database
- The result of a **SELECT** statement is another table, also known as a result set
- **SELECT Statement Syntax**

```
SELECT [DISTINCT][TOP n] <columns>
      [FROM] <table names>
      [WHERE] <criteria that must be true for a row to be chosen>
      [GROUP BY] <columns for grouping aggregate functions>
      [HAVING] <criteria that must be met for aggregate functions>
      [ORDER BY] <optional specification of how the results should be sorted>
```

June 22, 2015 | Proprietary and Confidential | - 10 -

IGATE  
Speed. Agility. Imagination

## Querying Data – SELECT Statement

You wouldn't have any reason to store data if you didn't want to retrieve it. Relational databases gained wide acceptance principally because they enabled users to query, or access, data easily, without using predefined, rigid navigational paths. Indeed, the acronym SQL stands for Structured Query Language. Microsoft SQL Server provides rich and powerful query capabilities.

The **SELECT** statement is the most frequently used SQL command and is the fundamental way to query data. The SQL syntax for the **SELECT** statement is intuitive—at least in its simplest forms—and resembles how you might state a request in English. As its name implies, it is also structured and precise. However, a **SELECT** statement can be obscure or even tricky. As long as you write a query that's syntactically correct, you get a result.

The basic form of **SELECT**, which uses brackets ([ ]) to identify optional items, is shown in the slide.

Notice that the only clause that must always be present is the verb **SELECT**; the other clauses are optional. For example, if the entire table is needed, you don't need to restrict data using certain criteria, so you can omit the **WHERE** clause.

## SELECT statements primary properties

- Number and attributes of the columns
- The following attributes must be defined for each result set column
  - The data type of the column.
  - The size of the column, and for numeric columns, the precision and scale.
  - The source of the data values returned in the column
- Tables from which the result set data is retrieved
- Conditions that the rows in the source tables must meet
- Sequence in which the rows of the result set are ordered

## SELECT statements primary properties

The full syntax of the SELECT statement is complex, but most SELECT statements describe four primary properties of a result set:

- The number and attributes of the columns in the result set.
- The following attributes must be defined for each result set column:
  - The data type of the column.
  - The size of the column, and for numeric columns, the precision and scale.
  - The source of the data values returned in the column.
- The tables from which the result set data is retrieved, and any logical relationships between the tables.
- The conditions that the rows in the source tables must meet to qualify for the SELECT. Rows that do not meet the conditions are ignored.
- The sequence in which the rows of the result set are ordered.

## SELECT statement

- Simple query that retrieves specific columns of all rows from a table

```
Use pubs
GO
SELECT au_lname, au_fname,
city, state, zip
FROM authors
GO
```

- Using WHERE clause

```
SELECT au_lname, au_fname, city, state, zip
FROM authors
WHERE au_lname='Ringer'
GO
```

June 22, 2015 | Proprietary and Confidential | -12-

IGATE  
Speed. Agility. Imagination

## SELECT Statement

Here's a simple query that retrieves all columns of all rows from the *authors* table in the *pubs* sample database:

```
SELECT au_lname, au_fname, city, state, zip FROM authors
```

You can specify a query from one table return only certain columns or rows that meet your stated criteria. In the select list, you specify the exact columns you want, and then in the WHERE clause, you specify the criteria that determine whether a row should be included in the answer set. Still using the *pubs* sample database, suppose we want to find the first name and the city, state, and zip code of residence for authors whose last name is *Ringer*:

```
SELECT au_lname, au_fname, city, state, zip FROM authors
WHERE au_lname='Ringer'
```

Retrieving only Anne Ringer's data requires an additional expression that is combined (AND'ed) with the original. In addition, we'd like the output to have more intuitive names for some columns, so the query would be:

```
SELECT 'Last Name'=au_lname, 'First'=au_fname, city, state, zip FROM authors
WHERE au_lname='Ringer' AND au_fname='Anne'
```

## SELECT statement – Order By Clause

- Specifies the sort order used on columns returned in a SELECT statement

```
USE Northwind  
GO  
SELECT ProductId,ProductName,UnitPrice  
FROM Products  
ORDER BY ProductName ASC  
GO
```

June 22, 2015 | Proprietary and Confidential | - 13 -

IGATE  
Speed. Agility. Imagination.

## SELECT Statement – Order By Clause

Specifies the sort order used on columns returned in a SELECT statement. Default is ASCending sort , DESC is used to sort in a descending manner

```
SELECT ProductId,ProductName,UnitPrice  
FROM Products  
ORDER BY ProductName DESC
```

ORDER BY is the last operation to be done , before the result set is submitted to the client , therefore ORDER BY cannot be used in certain places like subquery , SELECT INTO etc .

## Use of DISTINCT

- DISTINCT is used to eliminate duplicate rows
- Precedes the list of columns to be selected from the table(s)
- The DISTINCT considers the values of all the columns as a single unit and evaluates on a row-by-row basis to eliminate any redundant rows

### Example

```
SELECT DISTINCT Region  
FROM Northwind.dbo.Employees
```

## Demo

### ➤ Using SELECT Statement



---

June 22, 2015

Proprietary and Confidential

- 15 -

**IGATE**  
Speed. Agility. Imagination

## Use of Operators

- An operator is a symbol specifying an action that is performed on one or more expressions.
- Arithmetic Operators
- Logical Operators
- Assignment Operator
- String Concatenation Operator
- Comparison Operators
- Compound Assignment Operator

---

June 22, 2015

Proprietary and Confidential

- 16 -



## Use of Operators

An operator is a symbol specifying an action that is performed on one or more expressions. The following tables lists the operator categories that SQL Server 2008 uses.

- Arithmetic Operators
- Logical Operators
- Assignment Operator
- String Concatenation Operator
- Comparison Operators
- Compound Assignment Operator
- Bitwise Operators
- Unary Operators

## Arithmetic Operators

+ (Add)	Addition
- (Subtract)	Subtraction
* (Multiply)	Multiplication
/ (Divide)	Division
% (Modulo)	Returns the integer remainder of a division

June 22, 2015

Proprietary and Confidential

- 17 -



Arithmetic operators perform mathematical operations on two expressions of one or more of the data types of the numeric data type category.

### Operator Meaning

+ (Add)	Addition
- (Subtract)	Subtraction
* (Multiply)	Multiplication
/ (Divide)	Division
% (Modulo)	Returns the integer remainder of a division.

For example,  $12 \% 5 = 2$  because the remainder of 12 divided by 5 is 2. The plus (+) and minus (-) operators can also be used to perform arithmetic operations on **datetime** and **smalldatetime** values.

## Logical Operators

Operator	Meaning
• ALL	TRUE if all of a set of comparisons are TRUE
• AND	TRUE if both Boolean expressions are TRUE
• ANY	TRUE if any one of a set of comparisons are TRUE
• BETWEEN	TRUE if the operand is within a range
• EXISTS	TRUE if a subquery contains any rows
• IN	TRUE if the operand is equal to one of a list of expressions
• LIKE	TRUE if the operand matches a pattern.
• NOT	Reverses the value of any other Boolean operator
• OR	TRUE if either Boolean expression is TRUE.

June 22, 2015

Proprietary and Confidential

- 18 -



Logical operators test for the truth of some condition. Logical operators, like comparison operators, return a **Boolean** data type with a value of TRUE, FALSE, or UNKNOWN.

Operator	Meaning
ALL	TRUE if all of a set of comparisons are TRUE.
AND	TRUE if both Boolean expressions are TRUE.
ANY	TRUE if any one of a set of comparisons are TRUE.
BETWEEN	TRUE if the operand is within a range.
EXISTS	TRUE if a subquery contains any rows.
IN	TRUE if the operand is equal to one of a list of expressions.
LIKE	TRUE if the operand matches a pattern.
NOT	Reverses the value of any other Boolean operator.
OR	TRUE if either Boolean expression is TRUE.
SOME	TRUE if some of a set of comparisons are TRUE.

Example for IN:

```
USE Northwind
GO
```

```
SELECT FirstName, LastName, Title
FROM Employees
WHERE Title IN ('Design Engineer', 'Tool Designer', 'Marketing
Assistant'); GO
```

## Like Operators

- **Pattern:** The pattern to search for in match\_expression.

### Wildcard character

% Any string of zero or more characters.

### Example

WHERE title LIKE "%computer%" finds all book titles with the word 'computer' anywhere in the book title

\_ (underscore) Any single character.

WHERE au\_fname LIKE '\_ean' finds all four-letter first names that end with ean, such as Dean or Sean.

June 22, 2015 | Proprietary and Confidential | - 19 -



Determines whether a given character string matches a specified pattern. A pattern can include regular characters and wildcard characters. During pattern matching, regular characters must match exactly the characters specified in the character string. Wildcard characters, however, can be matched with arbitrary fragments of the character string. Using wildcard characters makes the LIKE operator more flexible than using the = and != string comparison operators.

For example to list out all titles which starts with a The , the query would be

```
SELECT title,price  
FROM pubs.dbo.titles  
WHERE title LIKE 'The%'
```

The output will not be the same if the pattern is given as

```
SELECT title,price  
FROM pubs.dbo.titles  
WHERE title LIKE '%The%'
```

This would list all titles have the somewhere - beginning , middle or end

## Working with NULL Values

- NULL values are treated differently from other values
- NULL is used as a placeholder for unknown or inapplicable values
- We have to use the IS NULL and IS NOT NULL operators to test for NULL Values

```
SELECT LastName,FirstName,Address  
      FROM Persons  
     WHERE Address IS NULL
```

```
SELECT LastName,FirstName,Address  
      FROM Persons  
     WHERE Address IS NOT NULL
```

## Assignment Operator

- Can create a variable
- Sets a value returned by an expression
- Can be used to establish the relationship between a column heading and the expression that defines the values for the column

```
USE AdventureWorks;
GO
SELECT FirstColumnHeading = 'xyz',
SecondColumnHeading = ProductID
FROM Products;
GO
```

June 22, 2015 | Proprietary and Confidential | - 21 -

IGATE  
Speed. Agility. Imagination

The equal sign (=) is the only Transact-SQL assignment operator. In the following example, the @MyCounter variable is created, and then the assignment operator sets @MyCounter to a value returned by an expression.

```
DECLARE @MyCounter INT;
SET @MyCounter = 1;
```

The assignment operator can also be used to establish the relationship between a column heading and the expression that defines the values for the column. The following example displays the column headings FirstColumnHeading and SecondColumnHeading. The string xyz is displayed in the FirstColumnHeading column heading for all rows. Then, each product ID from the Product table is listed in the SecondColumnHeading column heading.

```
USE AdventureWorks;
GO
SELECT FirstColumnHeading = 'xyz',
SecondColumnHeading = ProductID
FROM Products;
GO
```

## Comparison Operator

- = (Equals) Equal to
- > (Greater Than) Greater than
- < (Less Than) Less than
- >= (Greater Than or Equal To) Greater than or equal to
- <= (Less Than or Equal To) Less than or equal to
- <> (Not Equal To) Not equal to
- != (Not Equal To) Not equal to
- <!> (Not Less Than) Not less than
- >!> (Not Greater Than) Not greater than

June 22, 2015 | Proprietary and Confidential | + 22 -

IGATE  
Speed. Agility. Imagination

Comparison operators test whether two expressions are the same. Comparison operators can be used on all expressions except expressions of the **text**, **ntext**, or **image** data types. The following table lists the Transact-SQL comparison operators.

Operator	Meaning
=	(Equals) Equal to
>	(Greater Than) Greater than
<	(Less Than) Less than
>=	(Greater Than or Equal To) Greater than or equal to
<=	(Less Than or Equal To) Less than or equal to
<>	(Not Equal To) Not equal to
!=	(Not Equal To) Not equal to (not SQL-92 standard)
<!>	(Not Less Than) Not less than (not SQL-92 standard)
>!>	(Not Greater Than) Not greater than (not SQL-92 standard)
Example:	
use Northwind;	
go	
SELECT ProductID,ProductName, UnitPrice AS Price	
FROM Products	
WHERE SupplierID = 7 AND UnitsInStock < 25	
ORDER BY ProductName ASC ;	

## Compound Assignment Operators

- `+=` Plus Equals**
- `-=` Minus Equals**
- `*=` Multiplication Equals**
- `/=` Division Equals**
- `%=` Modulo Equals**

---

June 22, 2015

Proprietary and Confidential

- 23 -



## Compound Assignment Operators

Compound assignment operators help abbreviate code that assigns a value to a column or a variable. The new operators are:

- `+=` (plus equals)
- `-=` (minus equals)
- `*=` (multiplication equals)
- `/=` (division equals)
- `%=` (modulo equals)

You can use these operators wherever assignment is normally allowed—for example, in the SET clause of an UPDATE statement or in a SET statement that assigns values to variables. The following code example demonstrates the use of the `+=` operator:

```
DECLARE @price AS MONEY = 10.00;
```

```
SET @price += 2.00;
```

```
SELECT @price;
```

This code sets the variable `@price` to its current value, 10.00, plus 2.00, resulting in 12.00.

## Demo

- Using LIKE operator
- Working with commonly used operators



## Using System Functions

- String Functions
- Date and Time Functions
- Mathematical Functions
- Aggregate Functions
- System Functions

---

June 22, 2015

Proprietary and Confidential

- 25 -



## Using System Functions

1. String Functions - Perform operations on a string (**char** or **varchar**) input value and return a string or numeric value.
2. Date and Time Functions - Perform operations on a date and time input values and return string, numeric, or date and time values.
3. Mathematical Functions - Perform calculations based on input values provided as parameters to the functions, and return numeric values.
4. Aggregate Functions - You can use aggregate functions to calculate and summarize data.
5. System Functions - Perform operations and return information about values, objects, and settings in an instance of SQL Server.

## Using System Functions – String Functions

- **STR** - Returns character data converted from numeric data
- **REPLACE** - Replaces all occurrences of a specified string value with another string value
- **LEFT** - Returns the left part of a character string with the specified number of characters
- **RIGHT** - Returns the right part of a character string with the specified number of characters

June 22, 2015 | Proprietary and Confidential | - 26 -



### Examples

The following example replaces the string code in abcdefghi with xxx.

```
SELECT REPLACE('abcdefgihcde','cde','xxx');
```

Here is the result set.

abxxxfghixxx

The following example converts an expression that is made up of five digits and a decimal point to a six-position character string. The fractional part of the number is rounded to one decimal place.

```
SELECT STR(123.45, 6, 1)
```

Here is the result set.

123.5 (1 row(s) affected)

The following example returns the five leftmost characters of each product name.

```
USE Northwind;
```

```
SELECT LEFT(ProductName, 5) FROM Products ORDER BY ProductID;
```

## Using System Functions – String Functions

- **SUBSTRING** - Returns part of a character, binary, text, or image expression
- **LEN** - Returns the number of characters of the specified string expression, excluding trailing blanks
- **REVERSE** - Returns the reverse of a character expression
- **LOWER** - Returns a character expression after converting uppercase character data to lowercase
- **UPPER** - Returns a character expression with lowercase character data converted to uppercase
- **+ -** used for concatenating strings

---

June 22, 2015

Proprietary and Confidential

- 27 -



The following example returns the five rightmost characters of the first name for each contact

```
USE AdventureWorks  
GO  
SELECT RIGHT(FirstName, 5) AS 'First Name'  
FROM Person.Contact  
WHERE ContactID < 5  
ORDER BY FirstName; GO
```

Here is the result set.

First Name

-----

erine  
stavo  
berto  
Kim

```
use Northwind;
```

```
Go
```

```
SELECT LastName, SUBSTRING(FirstName, 1, 1) AS Initial  
FROM Employees WHERE LastName like 'S%'
```

```
SELECT LEN(CompanyName) AS Length, CompanyName, city  
FROM Suppliers  
WHERE Country = 'USA'
```

Some more string functions are:

ASCII, NCHAR, SOUNDEX, CHAR, PATINDEX, SPACE,  
CHARINDEX, QUOTENAME, DIFFERENCE, STUFF, REPLICATE,  
UNICODE, LTRIM, RTRIM.

All built-in string functions are deterministic. This means they  
return the same value any time they are called with a specific  
set of input values

Example of string concatenation

```
SELECT TitleOfCourtesy + ' ' + LastName + ', ' + FirstName  
FROM northwind.dbo.Employees  
ORDER BY TITLE
```

## Using System Functions – Date Functions

- **GETDATE** - Returns the current database system timestamp as a datetime value without the database time zone offset
- **GETUTCDATE** - Returns the current database system timestamp as a datetime value. The database time zone offset is not included
- **CURRENT\_TIMESTAMP** - Returns the current database system timestamp as a datetime value without the database time zone offset
- **SYSDATETIME** - Returns a datetime2(7) value that contains the date and time of the computer on which the instance of SQL Server is running. The time zone offset is not included

---

June 22, 2015

Proprietary and Confidential

- 29 -



### Date Functions Examples

#### GETDATE

```
SELECT GETDATE()
```

Result - 2012-03-10 13:56:36.620

#### GETUTCDATE

```
SELECT GETUTCDATE()
```

Result - 2012-03-10 13:56:36.620

#### CURRENT\_TIMESTAMP

```
SELECT CURRENT_TIMESTAMP
```

Result - 2012-03-10 13:56:36.620

#### SYSDATETIME

```
Result - 2012-04-10 14:04:46.50
```

## Using System Functions – Date Functions

- **SYSDATETIMEOFFSET** - Returns a datetimeoffset(7) value that contains the date and time of the computer on which the instance of SQL Server is running. The time zone offset is included.
- **SYSUTCDATETIME** - Returns a datetime2(7) value that contains the date and time of the computer on which the instance of SQL Server is running. The date and time is returned as UTC time (Coordinated Universal Time).

---

June 22, 2015

Proprietary and Confidential

- 30 -

**IGATE**  
Speed. Agility. Imagination

**SYSDATETIMEOFFSET** – DATETIMEOFFSET Defines a date that is combined with a time of a day that has time zone awareness and is based on a 24-hour clock

```
SELECT SYSDATETIMEOFFSET()
```

Result - 2012-03-10 14:06:56.7418989 +05:30

**SYSUTCDATETIME**

```
SELECT SYSUTCDATETIME()
```

Result - 2012-03-10 08:39:06.94

## Date Functions – To retrieve Date & Time Parts

- **DATENAME** - Returns a character string that represents the specified datepart of the specified date
- **DATEPART** - Returns an integer that represents the specified datepart of the specified date

datepart	Return value – DATEPART	Return Value - DATENAME
year, yyyy, yy	2007	2007
quarter, q, qd, q	4	4
month, mm, m	9	October
dayofyear, dy, y	303	303
day, dd, d	30	30
week, wk, ww	44	44
weekday, dw	3	Tuesday
hour, hh	12	12
minute, n	15	15
second, ss, s	32	32
millisecond, ms	123	123
microsecond, mcs	123456	123456

June 22, 2015

Proprietary and Confidential

- 31 -



### Examples

The following example extracts the month name from the date returned by GETDATE.

```
SELECT DATENAME(month, GETDATE()) AS 'Month Name';
Result
```

Month Name

---

February

The following example extracts the month name from a column.

```
USE AdventureWorks
GO
```

```
SELECT StartDate, DATENAME(month,StartDate) AS
StartMonth FROM Production.WorkOrder WHERE
WorkOrderID = 1; GO
```

Result

StartDate StartMonth

---

2001-07-04 00:00:00.000 July

**Try this:**

```
SELECT DATENAME(YEAR,GETDATE())
SELECT DATEPART(YEAR,GETDATE())
SELECT DATENAME(MONTH,GETDATE())
SELECT DATEPART(MONTH,GETDATE())
```

## Using System Functions – Date Functions

- **DATEDIFF** - Returns the count (signed integer) of the specified datepart boundaries crossed between the specified startdate and enddate.
- **DATEADD** - Returns a specified date with the specified number interval (signed integer) added to a specified datepart of that date.

June 22, 2015 | Proprietary and Confidential | + 32 +



### Examples

#### **DATEDIFF**

```
CREATE TABLE dbo.Duration
(
startDate datetime2 ,endDate datetime2
)
INSERT INTO dbo.Duration(startDate,endDate)
VALUES('2007-05-06 12:10:09','2007-05-07 12:10:09')
```

```
SELECT DATEDIFF(day,startDate,endDate) AS 'Duration'
FROM dbo.Duration;
-- Returns: 1
```

#### **DATEADD-**

The following example adds 2 days to each OrderDate to calculate a new PromisedShipDate.

```
USE AdventureWorks;
GO
SELECT SalesOrderID
    ,OrderDate
    ,DATEADD(day,2,OrderDate) AS PromisedShipDate
FROM Sales.SalesOrderHeader;
```

## Using System Functions – Mathematical Functions

- **ABS** - A mathematical function that returns the absolute (positive) value of the specified numeric expression
- **RAND** - Returns a random float value from 0 through 1
- **ROUND** - Returns a numeric value, rounded to the specified length or precision
- **SQRT** - Returns the square root of the specified float value

---

June 22, 2015

Proprietary and Confidential

- 33 -



The following scalar functions perform a calculation, usually based on input values that are provided as arguments, and return a numeric value:

**ABS** - A mathematical function that returns the absolute (positive) value of the specified numeric expression.

**RAND** - Returns a random **float** value from 0 through 1.

**RAND ( [ seed ] )**

**ROUND** - Returns a numeric value, rounded to the specified length or precision. **ROUND ( numeric\_expression , length [ ,function ] )**

**SQRT** - Returns the square root of the specified float value.

**SQRT ( float\_expression )**

Examples:

**SELECT ABS(-1.0), ABS(0.0), ABS(1.0);**

Here is the result set.

1.0	.0	1.0
-----	----	-----

**SELECT ROUND(123.4545, 2); GO (Output: 123.4500)**

**SELECT ROUND(123.45, -2);GO (Output: 100.00)**

**Some more mathematical functions are:**

DEGREES, ACOS, EXP, ASIN, FLOOR, SIGN, ATAN, LOG, SIN, ATN2, LOG10, CEILING, PI, SQUARE, COS, POWER, TAN, COT, RADIANS

## Using System Functions – Aggregate Functions

- The aggregate functions are: sum, avg, count, min, max, and count(\*)
- Aggregate functions are used to calculate and summarize data

```
USE pubs  
GO
```

```
SELECT AVG(price * 2)  
FROM titles  
GO
```

```
SELECT MAX(price) as Maxprice,MIN(price) as Minprice  
FROM titles  
GO
```

June 22, 2015 | Proprietary and Confidential | - 34 -

IGATE  
Speed. Agility. Imagination

### Using aggregate functions

The aggregate functions are: **sum**, **avg**, **count**, **min**, **max**, and **count(\*)**. You can use aggregate functions to calculate and summarize data.

Examples:

```
use pubs;  
select sum(ytd_sales)  
from titles;
```

Note that there is no column heading for the aggregate column.

An aggregate function takes as an argument the column name on whose values it will operate. You can apply aggregate functions to all the rows in a table, to a subset of the table specified by a **where** clause, or to one or more groups of rows in the table. From each set of rows to which an aggregate function is applied, Adaptive Server generates a single value.

Here is the syntax of the aggregate function:

**aggregate\_function ( [all | distinct] expression )**

Expression is usually a column name. However, it can also be a constant, a function, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators. You can also use a **case** expression or subquery in an expression.

## System Functions – To retrieve System Information

- **CURRENT\_TIMESTAMP** - Returns the current date and time, equivalent to GETDATE.
- **CURRENT\_USER** - Returns the name of the current user, equivalent to USER\_NAME().
- **HOST\_ID & HOST\_NAME** - Returns the workstation identification number and name.

The following functions perform operations on and return information about values, objects, and settings in SQL Server 2008

**CURRENT\_TIMESTAMP** - Returns the current date and time. This function is the ANSI SQL equivalent to GETDATE.

**CURRENT\_USER** - Returns the name of the current user. This function is equivalent to USER\_NAME().

**HOST\_ID** - Returns the workstation identification number. The workstation identification number is the process ID (PID) of the application on the client computer that is connecting to SQL Server.

**HOST\_NAME** - Returns the workstation name.

## System Functions – To retrieve System Information

- **CAST** and **CONVERT** - Explicitly converts an expression of one data type to another
- **CAST ( expression AS data\_type [ (length) ] )**
- **CONVERT ( data\_type [ ( length ) ], expression [ , style ] )**

### Example

```
Select CONVERT(char,100) --converts 100 to '100'
```

```
Select CAST(100 as char)
```

June 22, 2015 | Proprietary and Confidential | - 36 -

**IGATE**  
Speed. Agility. Imagination

**CAST** and **CONVERT** - Explicitly converts an expression of one data type to another. **CAST** and **CONVERT** provide similar functionality.

### Syntax

**Syntax for CAST:** `CAST ( expression AS data_type [ (length) ] )`

**Syntax for CONVERT:** `CONVERT ( data_type [ ( length ) ], expression [ , style ] )`

## Demo

- Working with commonly used Functions



**IGATE**  
Speed. Agility. Imagination

June 22, 2015

Proprietary and Confidential

- 37 -

## Organizing Query result into Groups

- Using group by clause
- Group by clause divides the output of a query into groups
- Can group by one or more column names

### Example

lists no of employees in each region

```
SELECT Region, count(EmployeeID)
FROM Northwind.dbo.Employees
GROUP by REGION
GO
```

June 22, 2015 | Proprietary and Confidential | - 38 -

IGATE  
Speed. Agility. Imagination

### Organizing query results into groups: the **group by** clause

The **group by** clause divides the output of a query into groups. You can group by one or more column names, or by the results of computed columns using numeric datatypes in an expression. When used with aggregates, **group by** retrieves the calculations in each subgroup, and may return multiple rows. The maximum number of columns or expressions you can use in a **group by** clause is 16. You cannot **group by** columns of text or image datatypes. While you can use **group by** without aggregates, such a construction has limited functionality and may produce confusing results. The following example groups the results by title type:

**select type, advance from titles group by type,advance**

type	advance
business	5,000.00
business	5,000.00
business	10,125.00
business	5,000.00
mod_cook	0.00
mod_cook	15,000.00
UNDECIDED	NULL
popular_comp	7,000.00
popular_comp	8,000.00
popular_comp	NULL

.....

## Groups By

- SQL standards for group by are more restrictive
- SQL standard requires that:
  - Columns in a select list must be in the group by expression or they must be arguments of aggregate functions
  - A group by expression can only contain column names in the select list

```
USE pubs;
GO
select pub_id, type, avg(price), sum(ytd_sales)
from titles
group by pub_id, type
GO
```

## Using Aggregation with Groups

```
USE pubs;
GO
select type, sum(advance)
from titles
group by type;
GO
```

June 22, 2015 | Proprietary and Confidential | - 40 -

IGATE  
Speed. Agility. Imagination

With an aggregate for the *advance* column, the query returns the sum for each group:

**select type, sum(advance) from titles group by type**

type	
UNDECIDED	NULL
business	25,125.00
mod_cook	15,000.00
popular_comp	15,000.00
psychology	21,275.00
trad_cook	19,000.00

The summary values in a **group by** clause using aggregates are called vector aggregates, as opposed to scalar aggregates, which result when only one row is returned.

## Selecting Group

- Use the **having** clause to display or reject rows defined by the **group by** clause
- Example

```
USE pubs;
GO
select type
from titles
group by type
having count(*) > 1
GO
```

```
USE pubs;
GO
select type
from titles
where count(*) > 1
GO
```

June 22, 2015 | Proprietary and Confidential | - 41 -

IGATE  
Speed. Agility. Imagination

### Selecting groups of data: the **having** clause

Use the **having** clause to display or reject rows defined by the **group by** clause. The **having** clause sets conditions for the **group by** clause in the same way **where** sets conditions for the **select** clause, except **where** cannot include aggregates, while **having** often does.

#### How the **having**, **group by**, and **where** clauses interact

When you include the **having**, **group by**, and **where** clauses in a query, the sequence in which each clause affects the rows determines the final results:

The **where** clause excludes rows that do not meet its search conditions.

The **group by** clause collects the remaining rows into one group for each unique value in the **group by** expression. Aggregate functions specified in the select list calculate summary values for each group.

The **having** clause excludes rows from the final results that do not meet its search conditions.

The following query illustrates the use of **where**, **group by**, and **having** clauses in one **select** statement containing aggregates:

```
select stor_id, title_id, sum(qty)
from salesdetail
where title_id like 'PS%'
group by stor_id, title_id having sum(qty) > 200
```

## Grouping Sets

- SQL Server 2008 introduces several extensions to the GROUP BY clause that enable you to define multiple groupings in the same query
- We can use grouping set for single result set instead of using UNION ALL with multiple queries for various grouping sets for various calculations
- SQL Server optimizes the data for access and grouping

---

June 22, 2015

Proprietary and Confidential

- 42 -



## Grouping Sets

SQL Server 2008 introduces several extensions to the GROUP BY clause that enable you to define multiple groupings in the same query. We can use grouping set for single result set instead of using UNION ALL with multiple queries for various grouping sets for various calculations. SQL Server optimizes the data for access and grouping.

Without the extensions, a single query normally defines one “grouping set” (a set of attributes to group by) in the GROUP BY clause. If you want to calculate aggregates for multiple grouping sets, you usually need multiple queries. If you want to unify the result sets of multiple GROUP BY queries, each with a different grouping set, you must use the UNION ALL set operation between the queries.

You might need to calculate and store aggregates for various grouping sets in a table. By preprocessing and materializing the aggregates, you can support applications that require fast response time for aggregate requests. However, the aforementioned approach, in which you have a separate query for each grouping set, is very inefficient. This approach requires a separate scan of the data for each grouping set and expensive calculation of aggregates.

## Grouping Sets

### ➤ Example – Grouping Sets equivalent to UNION ALL

```
SELECT customer, NULL as year,  
SUM(sales) FROM T GROUP BY  
customer UNION ALL SELECT  
NULL as customer, year,  
SUM(sales) FROM T GROUP BY  
year
```

```
SELECT customer, year,  
SUM(sales) FROM T GROUP  
BY GROUPING SETS  
((customer), (year))
```

---

June 22, 2015

Proprietary and Confidential

- 43 -

**IGATE**  
Speed. Agility. Imagination

With the new GROUPING SETS subclause, you simply list all grouping sets that you need. Logically, you get the same result set as you would by unifying the result sets of multiple queries.

However, with the GROUPING SETS subclause, which requires much less code, SQL Server optimizes data access and the calculation of aggregates. SQL Server will not necessarily need to scan data once for each grouping set; plus, in some cases it calculates higher-level aggregates based on lower-level aggregates instead of re-aggregating base data.

## Demo

### ➤ Working with Group By



June 22, 2015

Proprietary and Confidential

- 44 -

**IGATE**  
Speed. Agility. Imagination

## Pagination

- SQL Server 2012 enables Pagination.
- The OFFSET with FETCH commands can limit the number of rows sent to the client.
- Instead of using the TOP command for paging, these new commands can be used to reduce complexity.
- These new commands will normally be accompanied with ORDER BY.

---

June 22, 2015

Proprietary and Confidential

- 45 -



OFFSET and FETCH are two new clauses introduced in SQL Server 2012. Used together in your queries, they allow you to extract a portion of rows from your result set. A great practical use for this is in applications and websites where you wish to display just a small selection of records.

## Pagination

```
USE NORTHWIND
```

```
SELECT *  
FROM Products  
ORDER BY ProductName  
OFFSET 5 ROWS  
FETCH NEXT 10 ROWS ONLY
```

```
USE AdventureWorks
```

```
SELECT ProductID, Name  
FROM  
    AdventureWorks.Production.Produ  
ct  
ORDER BY NAME  
OFFSET 10 ROWS  
FETCH NEXT 5 ROWS ONLY
```

## Introduction

- SET operators are mainly used to combine the same type of data from two or more tables into a single result
- SET Operators supported in SQL Server are
  - UNION /UNION ALL
  - INTERSECT
  - EXCEPT

June 22, 2015

Proprietary and Confidential

- 47 -

**IGATE**  
Speed. Agility. Imagination

### **UNION**

Combine two or more result sets into a single set, without duplicates.

### **UNION ALL**

Combine two or more result sets into a single set, including all duplicates.

### **INTERSECT**

Takes the data from both result sets which are in common.

### **EXCEPT**

Takes the data from first result set, but not the second (i.e. no matching to each other)

## Introduction (Contd...)

### ➤ UNION /UNION ALL

- Combine two or more result sets into a single set, without duplicates.
- The number of columns have to match
- UNION ALL works exactly like UNION except that duplicates are NOT removed

```
SELECT ProductID,ProductName FROM Products  
WHERE categoryID=1234  
UNION  
SELECT ProductID,ProductName FROM Products  
WHERE categoryID=5678  
GO
```

June 22, 2015 | Proprietary and Confidential | - 48 -

**IGATE**  
Speed. Agility. Imagination

The above query will list all products belonging to category 1234 and 5678

The number of columns in the SELECT clauses must be same . In case the result has to be sorted on any order , the last query can only have Order by clause

## Introduction (Contd...)

### ➤ INTERSECT

- Takes the data from both result sets which are in common.
- All the other conditions remain same

```
SELECT CustomerID FROM Customers  
INTERSECT  
SELECT CustomerID FROM Orders
```

### ➤ EXCEPT

- Takes the data from first result set, which is not available in the second
- It is like a complement operation

```
SELECT CustomerID FROM Customers  
EXCEPT  
SELECT CustomerID FROM Orders
```



Here in this example using an INTERSECT operation I can find out all Customers who have placed orders

Using an EXCEPT operation , I can find out all Customers who havent placed any Orders

## Rules of Set Operation

- The result sets of all queries must have the same number of columns.
- In every result set the data type of each column must match the data type of its corresponding column in the first result set.
- In order to sort the result, an ORDER BY clause should be part of the last statement.
- The records from the top query must match the positional ordering of the records from the bottom query.
- The column names or aliases of the result set are given in the first select statement

## Summary

- Transact-SQL is central to using SQL Server
- SELECT is the basic & commonly used data retrieval statement used in SQL Server
- SQL Server provides variety of System functions which can help us in performing our day to day activities
- For example, String Functions, Date Functions, Mathematical functions, Aggregate Functions and so on
- We can make use of Group By to divide the SQL query result into groups
- We can use grouping set for single result set instead of using UNION ALL with multiple queries for various grouping sets for various calculations



## Summary

➤ **In this lesson, you have learnt:**

- Joins are used to fetch data from more than 2 tables.
- Join types: equijoin, non-equijoin, outer join, self join
- Data manipulation can be done using insert update and delete statements
- Subquery is query within a query or nested query
- Subquery types: single value, multivalue and exists



### Review Question

- Question 1: \_\_\_\_\_ is central to using SQL Server
- Question 2: \_\_\_\_\_ returns the current database system timestamp as a datetime value without the database time zone offset
- Question 3: We can use \_\_\_\_\_ for single result set instead of using UNION ALL with multiple queries



**IGATE**  
Speed. Agility. Imagination

## Review Question

- **Question 1:** The Set operation that will show all the rows from both the resultsets including duplicates is \_\_\_\_\_
  - Option 1: Union All
  - Option 2: Union
  - Option 3: Intersect
  - Option 4: Minus
- **Question 2:** The Except operator returns \_\_\_\_\_



## SQL Server 2012 – Database Development

Lesson 04: Working with Joins and Subqueries

June 22, 2015

Proprietary and Confidential

- 1 -

**IGATE**  
Speed. Agility. Inspiration.

## Lesson Objectives

➤ In this lesson, you will learn:

- What are joins?
- Types of Joins
- Using Subqueries
- Restrictions on Subqueries



## Joins

- Retrieving data from several tables ,based on a relationship between certain columns in these tables
- Most of the time it would be the Foreign key
- One can join maximum 256 tables in single query

By using joins, you can retrieve data from two or more tables based on logical relationships between the tables. Joins indicate how SQL Server should use data from one table to select the rows in another table.

A join condition defines the way two tables are related in a query by:

A join operation compares two or more tables (or views) by specifying a column from each, comparing the values in those columns row by row, and linking the rows that have matching values. It then displays the results in a new table. The tables specified in the join can be in the same database or in different databases.

When you join two or more tables, the columns being compared must have similar values--that is, values using the same or similar datatypes.

There are several types of joins, such as equijoins, natural joins, and outer joins. The most common join, the equijoin, is based on equality

If the joining table have the same column names the query engine would report an error because of ambiguity . Therefore one has to qualify the column names with Table names as given in the example.

## Types of Joins

- **INNER join**
  - Equijoin
  - Nonequijoin
- **Outer join**
  - LEFT outer
  - RIGHT outer
  - FULL outer
- **Self join**
- **Cross join**

June 22, 2015

Proprietary and Confidential

- 4 -

**IGATE**  
Speed. Agility. Imagination

### Inner Join

Retrieves records from multiple tables based on the equality on the values of the common column. Inner joins are commutative in nature i.e same results come when A is joined with B or B is joined with A .

### Self Join

A table joined with itself.

### Outer Join

Displays the result set containing all the rows from one table and the matching rows from the another table. The table from where all the rows are retrieved is called OUTER table and other table is called as inner table . Based on the direction of the OUTER table , the outer joins can be LEFT , RIGHT FULL outer joins have both matched and unmatched rows of all tables

### Cross Join (Cartesian Product)

Cross Join happens when each row from one table is joined with all the row of the other table. This typically can be done by removing the where clause

The syntax of all the joins are explained in the next section

## Introducing T-SQL JOIN Syntax

```
SELECT <<Column List>>
FROM Table name1
[INNER] JOIN Table name2
ON join criteria
WHERE condition criteria
```

OR

```
SELECT <<Column List>>
FROM Table name1, Table name2
WHERE Join criteria
AND condition criteria
```

## Inner Join

```
USE Northwind
```

```
GO
```

```
SELECT ProductID,ProductName,CategoryName  
FROM Products,Categories  
WHERE Products.CategoryID=Categories.CategoryID  
GO
```

---

June 22, 2015

Proprietary and Confidential

- 6 -

**IGATE**  
Speed. Agility. Imagination

## INNER JOIN – More than 2 tables

- Number of joins = number of tables - 1

```
USE northwind  
GO
```

```
SELECT categoryname, description,  
productname, productid, companyname, suppliers.city  
FROM products, categories, suppliers  
WHERE Products.categoryid = Categories.categoryid  
AND Products.supplierid = Suppliers.supplierid  
AND Suppliers.city = 'London'  
ORDER by productname  
GO
```

When more than 2 tables need to be joined, the number of joins to be created will be one less than the total number of tables to be joined.

Example :-

```
SELECT categoryname, description, productname, productid,  
companyname, Suppliers.city  
FROM products  
INNER JOIN categories  
ON Products.categoryid = Categories.categoryid  
INNER JOIN Suppliers  
ON Products.supplierid = Suppliers.supplierid  
WHERE Suppliers.city = 'London'  
ORDER by Product.productname
```

An inner join is a join in which the values in the columns being joined are compared using a comparison operator. In the SQL-92 standard, inner joins can be specified in either the FROM or WHERE clause. This is the only type of join that SQL-92 supports in the WHERE clause. Inner joins specified in the WHERE clause are known as old-style inner joins.

This inner join is known as an equijoin. It returns all the columns in both tables, and returns only the rows for which there is an equal value in the join column.

Example:

```
select column1 FROM Table1  
INNER JOIN Table2  
ON Table1.col1=Table2.col1
```

## OUTER JOIN

- Inner joins eliminates rows which doesn't have a match in the joining tables
- Outer joins returns all rows from one of the joining tables and matched rows from the others
- Outer joins can be further classified as
  - LEFT OUTER JOIN or LEFT JOIN
  - RIGHT OUTER JOIN or RIGHT JOIN
  - FULL OUTER JOIN or FULL JOIN

Inner joins return rows only when there is at least one row from both tables that matches the join condition. Inner joins eliminate the rows that do not match with a row from the other table. Outer joins, however, return all rows from at least one of the tables (or views) mentioned in the FROM clause, as long as those rows meet any WHERE or HAVING search conditions. All rows are retrieved from the left table referenced with a left outer join, and all rows from the right table referenced in a right outer join. All rows from both tables are returned in a full outer join.

Microsoft SQL Server 2008 uses these SQL-92 keywords for outer joins specified in a FROM clause.

- LEFT OUTER JOIN or LEFT JOIN
- RIGHT OUTER JOIN or RIGHT JOIN
- FULL OUTER JOIN or FULL JOIN

## Using Left Outer Joins

- **Left Outer** - all records from left table and corresponding matching records from right

### T-SQL Syntax

```
SELECT categoryname, description,
productname, productid
FROM CATEGORIES
LEFT OUTER JOIN PRODUCTS
ON PRODUCTS.categoryid = CATEGORIES.categoryid
GO
```

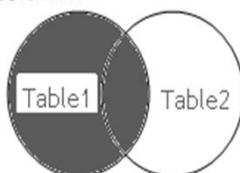
## Using Left Outer Joins

Consider a join of the Products table and the Categories table on their CategoryID columns.

To include all category id, regardless of whether a product exists for that category or not, use an SQL-92 left outer join. The following is the query:

```
SELECT categoryname, description,
productname, productid
FROM CATEGORIES
LEFT OUTER JOIN PRODUCTS
ON PRODUCTS.categoryid = CATEGORIES.categoryid
GO
```

The LEFT OUTER JOIN includes all rows in the categories table in the results, whether or not there is a match on the CategoryID column in the Products table. Notice that in the results where there is no matching categoryID for a product, the row contains a null value in the productname and productID column.



## RIGHT OUTER Join

### ➤ T-SQL Syntax

```
SELECT categoryname, description,
productname, productid
FROM CATEGORIES
RIGHT OUTER JOIN PRODUCTS
ON PRODUCTS.categoryid = CATEGORIES.categoryid
GO
```

A right outer join (or right join) closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the right table is retrieved with corresponding match from the left table.

## FULL OUTER JOIN

- FULL OUTER JOIN - includes all rows from both tables

```
USE Northwind;
GO
SELECT categoryname, description,
productname, productid
FROM CATEGORIES
FULL OUTER JOIN PRODUCTS
ON PRODUCTS.categoryid = CATEGORIES.categoryid
GO
```

June 22, 2015

Proprietary and Confidential

- 12 -

IGATE  
Speed. Agility. Imagination

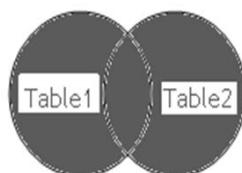
### Using Full Outer Joins

To retain the non-matching information by including non-matching rows in the results of a join, use a full outer join. SQL Server provides the full outer join operator, FULL OUTER JOIN, which includes all rows from both tables, regardless of whether or not the other table has a matching value.

Consider a join of the Product table and the Categories table on their categoryID columns. The results show only the Products that have sales orders on them. The SQL-92 FULL OUTER JOIN operator indicates that all rows from both tables are to be included in the results, regardless of whether there is matching data in the tables.

You can include a WHERE clause with a full outer join to return only the rows where there is no matching data between the tables.

There is no T-SQL equivalent of FULL JOIN .



## SELF JOIN

- A self join is when a table joins with itself
- Self join is possible when the table has references to itself  
For example to query employees along with managers who are also employees

```
SELECT emp.EmployeeID, emp.EmployeeName,mgr.EmployeeName  
FROM Employee emp  
INNER JOIN Employee mgr  
ON emp.ManagerID = mgr.EmployeeID
```

A table can be joined to itself in a self-join. This typically happens , when the records in a table has a reference in the same table itself . For example Given an employee table with the following structure

EmployeeID  
EmployeeName  
SupervisorID  
Basic

Here supervisor is also an employee , therefore to list all employees along with manager names , one has to do a self join. Appropriate aliases are used to distinguish the columns .

A self join is a join in all possible ways except that it joins to another instance of itself

## CROSS JOIN

- Cross join does not have a WHERE clause
- Result set is the number of rows in the first table multiplied by the number of rows in the second table- a Cartesian product

```
USE Northwind;
GO
SELECT categoryname, description, productname, productid
FROM CATEGORIES ,PRODUCTS
GO
```

```
USE Northwind;
GO
SELECT categoryname, description, productname, productid
FROM CATEGORIES CROSS JOIN PRODUCTS
ORDER BY CategoryName
```

June 22, 2015

Proprietary and Confidential

IGATE  
Speed. Agility. Imagination

A cross join that does not have a WHERE clause produces the Cartesian product of the tables involved in the join. The size of a Cartesian product result set is the number of rows in the first table multiplied by the number of rows in the second table.

## Introduction

- A sub query is an SQL statement that is used within another SQL statement
- subqueries are used to handle query requests that are expressed as the results of other queries
- Subquery can be embedded in WHERE /HAVING statement

```
USE Northwind;
GO
SELECT EmployeeID,EmployeeName
FROM Employees
WHERE Region=
(SELECT Region from Employees
WHERE EmployeeID=12345)
```

### How subqueries work

Subqueries, also called inner queries, appear within a **where** or **having** clause of another SQL statement or in the select list of a statement. You can use subqueries to handle query requests that are expressed as the results of other queries. A statement that includes a subquery operates on rows from one table, based on its evaluation of the subquery's **select** list, which can refer either to the same table as the outer query, or to a different table. In Transact-SQL, a subquery can also be used almost anywhere an expression is allowed, if the subquery returns a single value. A **case** expression can also include a subquery.

For example, this subquery lists the employee details of all employees who belong to the same region as employee

12345

```
USE Northwind;
GO
SELECT EmployeeID,EmployeeName
FROM Employees
WHERE Region=
(SELECT Region from Employees
WHERE EmployeeID=12345)
```

**select** statements that contain one or more subqueries are sometimes called nested queries or nested select statements.

You can formulate as joins many SQL statements that include a subquery. Other questions can be posed only with subqueries. Some people find subqueries easier to understand. Other SQL users avoid subqueries whenever possible. You can choose whichever formulation you prefer. The result of a subquery that returns no values is NULL. If a subquery returns NULL, the query failed.

## Subquery restrictions

- Subquery\_select\_list can consist of only one column name
- Subqueries can be nested inside the where or having clause
- Subquery can appear almost anywhere an expression can be used, if it returns a single value
- Subqueries cannot manipulate their results internally i,e ORDER BY Clause cannot be used inside the subquery
- If the sub query returns more than 1 row then outer query has to use appropriate operator like IN , ANY etc

A subquery is subject to the following restrictions:

1. The subquery\_select\_list can consist of only one column name, except in the **exists** subquery, where an (\*) is usually used in place of the single column name. Do not specify more than one column name. Qualify column names with table or view names if there is ambiguity about the table or view to which they belong.
2. Subqueries can be nested inside the **where** or **having** clause of an outer **select**, **insert**, **update**, or **delete** statement, inside another subquery, or in a select list. Alternatively, you can write many statements that contain subqueries as joins; Adaptive Server processes such statements as joins.
3. In Transact-SQL, a subquery can appear almost anywhere an expression can be used, if it returns a single value.
4. The select list of an inner subquery introduced with a comparison operator can include only one expression or column name, and the subquery must return a single value. The column you name in the **where** clause of the outer statement must be join-compatible with the column you name in the subquery select list.
5. Subqueries cannot manipulate their results internally, that is, a subquery cannot include the **order by** clause, the **compute** clause, or the **into** keyword.

## Types of Subqueries

- Single Row Sub query
- Multi Row Sub query
- Sub query for Existence
- Correlated Sub Query

June 22, 2015

Proprietary and Confidential

- 18 -

**IGATE**  
Speed. Agility. Imagination

6. Correlated (repeating) subqueries are not allowed in the **select** clause of an updatable cursor defined by **declare cursor**.
7. There is a limit of 16 nesting levels.
8. The maximum number of subqueries on each side of a union is 16.
9. The **where** clause of a subquery can contain an aggregate function only if the subquery is in a **having** clause of an outer query and the aggregate value is a column from a table in the **from** clause of the outer query.

There are three basic types of subqueries.

### Single ROW subquery

Such subquery returns only 1 value . Therefore the operator which can be used

=, >, >=, <, <=, and <>.

The subquery can be used in WHERE and HAVING clause

### Multiple ROW Subqueries

Subqueries which return more than 1 row . Some operators that can be used with multiple-row subqueries are:

IN, equal to any member in the list

ANY, compare values to each value returned by the subquery.

ALL , All the values

The examples of each type is explained in the next few sections

## Multi Row Subquery

- Subqueries returns a list of zero or more values and can include a GROUP BY or HAVING clause
- >ALL means greater than every value
- >ANY means greater than at least one value

```
SELECT ProductID,ProductName,UnitPrice  
FROM PRODUCTS WHERE SupplierID IN  
(SELECT SupplierID  
FROM Suppliers  
WHERE CITY='NEW York')
```

Comparison operators that introduce a subquery can be modified by the keywords ALL or ANY. SOME is an SQL-92 standard equivalent for ANY.

Subqueries introduced with a modified comparison operator return a list of zero or more values and can include a GROUP BY or HAVING clause. These subqueries can be restated with EXISTS.

Using the > comparison operator as an example, >ALL means greater than every value. In other words, it means greater than the maximum value. For example, >ALL (1, 2, 3) means greater than 3. >ANY means greater than at least one value, that is, greater than the minimum. So >ANY (1, 2, 3) means greater than 1.

For a row in a subquery with >ALL to satisfy the condition specified in the outer query, the value in the column introducing the subquery must be greater than each value in the list of values returned by the subquery.

Similarly, >ANY means that for a row to satisfy the condition specified in the outer query, the value in the column that introduces the subquery must be greater than at least one of the values in the list of values returned by the subquery.

## Multi Row Subquery

```
SELECT ProductID,ProductName,SupplierID  
FROM Products  
WHERE UnitPrice > ALL  
(Select UnitPrice  
FROM Products  
WHERE SupplierID=10098)
```

In this example you are listing all those products whose price is above all the price of products supplied by supplier 10098

## EXISTS

- EXISTS checks for a existence of a condition
- The EXISTS condition is considered "to be met" if the subquery returns at least one row.

```
SELECT SupplierID  
FROM suppliers  
WHERE EXISTS  
(select 'A'  
from orders  
where suppliers.supplier_id =  
orders.supplier_id);
```

Subquery that is introduced with **exists** is different from other subqueries, in these ways:

- The keyword **exists** is not preceded by a column name, constant, or other expression.
- The subquery **exists** evaluates to TRUE or FALSE rather than returning any data.

Since EXISTS evaluates to TRUE /FALSE for a condition , there is no need to return all the columns , a single column or a value can be returned back which will improve performance.

## Summary

➤ **In this lesson, you have learnt:**

- Joins are used to fetch data from more than 2 tables.
- Join types: equijoin, non-equijoin, outer join, self join
- Subquery is query within a query or nested query
- Subquery types



## Review Question

- Question 1: If we do not include a join condition then the result leads to \_\_\_\_\_
- Question 2: \_\_\_\_\_ return all rows from at least one of the tables in the FROM clause
- Question 3: When subquery depends on the outer query for its execution then it is \_\_\_\_\_ subquery.
- Question 4: Subquery \_\_\_\_\_ evaluates to TRUE or FALSE rather than returning any data



# SQL Server 2012 – Database Development

Lesson 05: Procedures and Functions

June 22, 2015

Proprietary and Confidential

-1-

**IGATE**  
Speed. Agility. Imagination.

## Lesson Objectives

- Database programming
- Creating, Executing, Modifying, and Dropping Stored Procedures and Functions
- Implementing Exception Handling



## Overview

- **Introduction to Stored Procedures**
- **Creating, Executing, Modifying, and Dropping Stored Procedures**
- **Using Parameters in Stored Procedures**
- **Executing Extended Stored Procedures**
- **Handling Error Messages**

## Definition

- Named Collections of pre compiled Transact-SQL Statements
- Stored procedures can be used by multiple users and client programs leading to reuse of code
- Abstraction of code and better security control
- Reduces network work and better performance
- Can accept parameters and return value or result set

In the simplest terms, a stored procedure is a collection of compiled T-SQL commands that are directly accessible by SQL Server. The commands placed within a stored procedure are executed as one single unit, or **batch**, of work. In addition to SELECT, UPDATE, or DELETE statements, stored procedures are able to call other stored procedures, use statements that control the flow of execution, and perform aggregate functions or other calculations.

Any developer with access rights to create objects within SQL Server can build a stored procedure.

If the stored procedure is on same machine it is called as **Local procedure**, otherwise if it is stored at different machine then it is called as remote procedure.

## Types

➤ **T-SQL supports the following types of procedure**

– **System-**

- Procedures pre-built in SQL Server itself
- Available in master database
- Name starts with sp\_

➤ **Temporary**

- name starts with #(Local) or ## (Global) and stored in tempdb
- Available only for that session

➤ **Extended**

- execute routines written in programming languages like C,C++,C# or VB.NET
- May have names starting with xp\_

June 22, 2015

Proprietary and Confidential

- 5 -

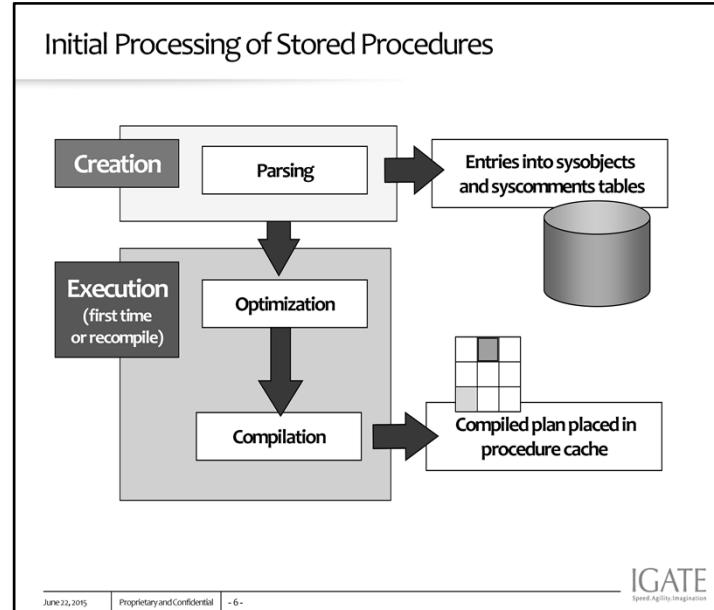
**IGATE**  
Speed. Agility. Imagination

### Types of stored procedure

**1.System procedure :** The name of all system procedure starts with a prefix of sp\_, within SQL Server. One cannot modify any system stored procedure that belongs to SQL Server, as this could corrupt not only your database, but also other databases

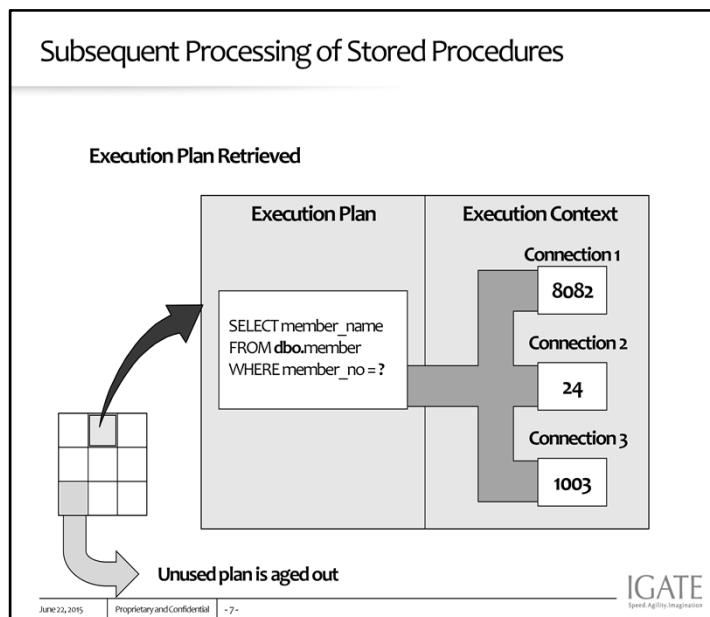
**2.Temporary Procedure :** The Database Engine supports two types of temporary procedures: local and global. A local temporary procedure is visible only to the user that created it. A global temporary procedure is available to all currently connected users. Local temporary procedures are automatically dropped at the end of the current session. Global temporary procedures are dropped at the end of the last session using the procedure. To create local temporary procedure name should start with # for global temporary procedure name should start with ##.

**3.Extended procedure :** Extended stored procedures let you create your own external routines in a programming language such as C,C++,.NET etc. The extended stored procedures appear to users as regular stored procedures and are executed in the same way. Parameters can be passed to extended stored procedures, and extended stored procedures can return results and return status. Extended stored procedures are DLLs that an instance of SQL Server can dynamically load and run. Extended stored procedures run directly in the address space of an instance of SQL Server and are programmed by using the SQL Server Extended Stored Procedure API.



When you create a stored procedure  
SQL server parses your code and make entries in appropriate system table.  
SQL server analyzes and optimizes the queries within stored procedure and generates an execution plan. An execution plan holds the instructions to process the query. These instruction include which order to access the table in, which indexes, access methods and join algorithm to use and so on.  
SQL server generates multiple permutation of execution plan and choose the one with lowest cost

After execution of query or procedure we can see the execution plan in result window by clicking on display estimated execution plan button in the tool bar.



3. Stored procedures can reuse a previously cached execution plan, which saves the resources involved in generating a new execution plan
4. But if changes have been made to database objects (like adding or dropping column or index) or changes to set option that affect query results or if plan is removed from cache after a while for lack of reuse, causes recompilation and SQL server will generate a new one when the procedure is invoked again.

## Advantages

- Share Application Logic across multiple clients
- Shield Database Schema Details (Abstraction)
- Provide Security Mechanisms
- Reduce Network Traffic
- Improve Performance

1. Share application logic- Same procedure can be executed n times by multiple clients .
2. Shield Database schema details - Even If user does not have access to tables he can use tables through procedure.
3. Provide security mechanism - Restricted access can be given to tables and user also does not get to know which tables are used to get the data.
4. Reduced Network traffic - Assume that you are using some front end application and calling stored procedure. Then instead of sending SQL statements to server front end, will pass only function call with parameters and call gets executed at server side and result gets transferred to front end application.
5. Improve performance - If in case there is any change in procedure (like change in tables or logic changes) . It does not affect front end application if we are using stored procedure. Which improves performance.

## CREATE PROCEDURE Statement

### ➤ Syntax

```
CREATE { PROC | PROCEDURE } [schema_name.]procedure_name [{  
    @parameter data_type } [ VARYING ] [ =default ] [ [ OUT [ PUT  
    ] ] [ ,...n ]  
    AS  
    { <sql_statement> [;][,...n] }  
    <methodSpecifier> [;]  
    Return <value>
```

June 22, 2015

Proprietary and Confidential

- 9 -

IGATE  
Speed. Agility. Imagination.

CREATE { PROC | PROCEDURE } [schema\_name.]procedure\_name  
[ { @parameter [ type\_schema\_name. ] data\_type } [ [ OUT [  
PUT ] ] [ ,...n ] { [ BEGIN ] statements [ END ] }

schema\_name

Is the name of the schema to which the procedure belongs.

procedure\_name

Is the name of the new stored procedure. Procedure names must comply with the rules for identifiers and must be unique within the schema.

You should not use the prefix sp\_ in the procedure name. This prefix is used by SQL Server to designate system stored procedures.

Local or global temporary procedures can be created by using one number sign (#) before procedure\_name (#procedure\_name) for local temporary procedures, and two number signs for global temporary procedures (##procedure\_name). Temporary names cannot be specified for CLR stored procedures.

The complete name for a stored procedure or a global temporary stored procedure, including ##, cannot exceed 128 characters. The complete name for a local temporary stored procedure, including #, cannot exceed 116 characters.

**@ parameter**

Is a parameter in the procedure. One or more parameters can be declared in a CREATE PROCEDURE statement. The value of each declared parameter must be supplied by the user when the procedure is called, unless a default for the parameter is defined or the value is set to equal another parameter. A stored procedure can have a maximum of 2,100 parameters.

Specify a parameter name by using an at sign (@) as the first character. The parameter name must comply with the rules for identifiers. Parameters are local to the procedure; the same parameter names can be used in other procedures. By default, parameters can take the place only of constant expressions; they cannot be used instead of table names, column names, or the names of other database objects.

**OUTPUT**

Indicates that the parameter is an output parameter. The value of this option can be returned to the calling EXECUTE statement. Use OUTPUT parameters to return values to the caller of the procedure. **text**, **ntext**, and **image** parameters cannot be used as OUTPUT parameters, unless the procedure is a CLR procedure. An output parameter that uses the OUTPUT keyword can be a cursor placeholder, unless the procedure is a CLR procedure.

The maximum size of a Transact-SQL stored procedure is 128 MB.

A user-defined stored procedure can be created only in the current database. Temporary procedures are an exception to this because they are always created in **tempdb**. If a schema name is not specified, the default schema of the user that is creating the procedure is used.

## Example

### ➤ Code Snippet

```
USE AdventureWorks2012;
GO
CREATE PROCEDURE HumanResources.uspGetEmployeesTest2
@LastName nvarchar(50),
@FirstName nvarchar(50) AS
SET NOCOUNT ON;
SELECT FirstName, LastName, Department FROM
HumanResources.vEmployeeDepartmentHistory WHERE FirstName = @FirstName AND
LastName = @LastName AND EndDate IS NULL;
GO
```

June 22, 2015

Proprietary and Confidential

- 11 -

**IGATE**  
Speed. Agility. Inspiration.

```
USE Northwind
GO
CREATE PROC dbo.OverdueOrders
AS
BEGIN
    SELECT COUNT(Order_id)
    FROM dbo.Orders
    WHERE RequiredDate < GETDATE() AND ShippedDate IS
NULL
END
GO
```

## Executing Stored Procedures

### ➤ Code Snippet

```
EXECUTE HumanResources.uspGetEmployeesTest2 N'Ackerman', N'Pilar';
-- Or
EXEC HumanResources.uspGetEmployeesTest2 @LastName = N'Ackerman', @FirstName = N'Pilar';
GO
-- Or
EXECUTE HumanResources.uspGetEmployeesTest2 @FirstName = N'Pilar', @LastName =
N'Ackerman';
GO
```

**Note:** You need to have execute permission for the procedure to execute it

June 22, 2015

Proprietary and Confidential

- 12 -

**IGATE**  
Speed. Agility. Imagination

Execute statement:

Executes a command string or character string within a Transact-SQL batch, or one of the following modules: system stored procedure, user-defined stored procedure, scalar-valued user-defined function, or extended stored procedure.

You can use insert with the results of stored procedure or dynamic execute statement taking place of values clause. Execute should return exactly one result set with types that match the table you have set up for it.

Executing a Stored Procedure by Itself

```
EXEC OverdueOrders
```

Executing a Stored Procedure Within an INSERT Statement

```
INSERT INTO Customers
EXEC EmployeeCustomer
```

e.g If you want to store results of executing sp\_configure stored procedure in temporary table

```
Create table #config_out
(
Name_col varchar(50),
MinvalInt,
Maxval int, configval int,
Runval int
)
```

```
insert #config_out
Exec sp_configure
```

## Altering and Dropping Procedures

### ➤ Altering Stored Procedures

- Include any options in ALTER PROCEDURE
- Does not affect nested stored procedures

```
ALTER { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]  [ {  
    @parameter [ type_schema_name. ] data_type }      [ VARYING ] [ = default ] [ OUT ]  
    [ OUTPUT ] [ READONLY ]  ][ ,...n ] [ WITH <procedure_option> [ ,...n ] ] [ FOR REPLICATION ]  
    AS { [ BEGIN ] sql_statement [ ; ] [ ...n ] [ END ] } { ; } <procedure_option> ::=  [ ENCRYPTION  
    ] [ RECOMPILE ] [ EXECUTE AS Clause ]
```

DROP PROCEDURE <stored procedure name>;

June 22, 2015

Proprietary and Confidential

- 13 -

**IGATE**  
Speed. Agility. Imagination.

### sp\_depends

Is system procedure which Displays information about database object dependencies, such as: the views and procedures that depend on a table or view, and the tables and views that are depended on by the view or procedure. References to objects outside the current database are not reported.

## Demo

### ➤ Creating Stored Procedures



June 22, 2015

Proprietary and Confidential

- 14 -

**IGATE**  
Speed. Agility. Imagination

## Stored Procedures Using Parameters

- Stored procedures can take parameters OR arguments and return value
- Parameters can of the following type
- **INPUT**
  - Default Type
  - IN or INPUT keyword is used to define variables of IN type
  - Used to pass a data value to the stored procedure
- **OUTPUT**
  - Allow the stored procedure to pass a data value or a back to the caller.
  - OUT keyword is used to identify output parameter

---

June 22, 2015

Proprietary and Confidential

- 15 -

**IGATE**  
Speed. Agility. Imagination

Parameters are used to exchange data between stored procedures and functions that called the stored procedure or function:

Input parameters allow the caller to pass a data value to the stored procedure or function.

Output parameter

Output parameters allow the stored procedure to pass a data value or a cursor variable back to the caller.

Parameters to procedure can be passed in the following manner

Passing values by parameter name

If you don't want to send all parameter values in same sequence as they are defined in create or procedure, you can pass them by parameter name Useful when many default values are defined , So required to pass very few parameters.

EXECUTE mytables @type='S'

Passing values by position

If you want to pass all parameters in the same sequence as they are defined in create or alter procedure. Then use this method.

If a parameter has default values , if default value is assigned to the INPUT parameters . DEFAULT keyword is used during execution to indicate DEFAULT value

CREATE PROCEDURE mytables ( @type char(2)='U')

AS

SELECT count( id) FROM sysobjects WHERE type = @type  
GO

EXECUTE mytables

EXECUTE mytables DEFAULT

EXECUTE mytables 'P'

## Stored Procedures Using Parameters

```

CREATE PROCEDURE usp_ProductCountByCategory(
    @_catid INT,
    @_Prodcount INT OUT
)
AS
BEGIN
    IF @_catid IS NULL OR @_catid < 0
        RETURN -1;
    SELECT @_Prodcount = COUNT(ProductID) FROM Products
    WHERE CategoryID = @_catid
END

```

➤ To execute

```

DECLARE @prodcount INT
EXEC usp_ProductCountByCategory 1234, @prodcount OUT

```

June 22, 2015 | Proprietary and Confidential | - 16 -

**IGATE**  
Speed. Agility. Imagination

In this example the procedure usp\_ProductCountByCategory takes a category id as input and returns the count of products belonging to that category as output. To execute the procedure having OUT variables, one has to declare the Variable first and then pass that variable to the procedure using the OUT keyword. The procedure also handles erroneous situation of invalid inputs by returning -1. More about return statement is discussed later in the session.

As with all Good programming practices, input values must be validated and all variables must be initialized

In case the procedure has default values then the procedure can be defined as

```

CREATE PROCEDURE usp_ProductCountByCategory (
    @_catid INT = 345,
    @_Prodcount INT OUT
)

```

And to execute with default value it would be

```
EXEC usp_ProductCountByCategory DEFAULT, @pcount OUTPUT
```

Example of INPUT and OUTPUT Parameter:-

The following example shows a procedure with an input and an output parameter. The @SalesPerson parameter would receive an input value specified by the calling program. The SELECT statement uses the value passed into the input parameter to obtain the correct SalesYTD value. The SELECT statement also assigns the value to the @SalesYTD output parameter, which returns the value to the calling program when the procedure exits.

```

USE AdventureWorks2012;
GO
IF OBJECT_ID('Sales.uspGetEmployeeSalesYTD', 'P') IS NOT NULL
    DROP PROCEDURE Sales.uspGetEmployeeSalesYTD;
GO
CREATE PROCEDURE Sales.uspGetEmployeeSalesYTD @SalesPerson nvarchar(50),
    @SalesYTD money OUTPUT
AS
SET NOCOUNT ON;
SELECT @SalesYTD = SalesYTD FROM Sales.SalesPerson AS sp JOIN HumanResources.vEmployee AS e ON
e.BusinessEntityID = sp.BusinessEntityID WHERE LastName = @SalesPerson;
RETURN
GO

```

The following example calls the procedure created in the first example and saves the output value returned from the called procedure in the @SalesYTD variable, which is local to the calling program.

- Declare the variable to receive the output value of the procedure.

```
DECLARE @SalesYTDBySalesPerson money;
```

- Execute the procedure specifying a last name for the input parameter

```
- and saving the output value in the variable @SalesYTDBySalesPerson
```

```
EXECUTE Sales.uspGetEmployeeSalesYTD N'Blythe', @SalesYTD = @SalesYTDBySalesPerson OUTPUT;
```

- Display the value returned by the procedure.

```
PRINT 'Year-to-date sales for this employee is ' + convert(varchar(10),@SalesYTDBySalesPerson);
```

```
GO
```

## Returning a Value from Stored Procedures

➤ **Values can be returned from stored procedure using the following options**

- OUTPUT parameter
  - More than 1 parameter can be of type OUTPUT
- Return statement
  - Used to provide the execution status of the procedure to the calling program
  - Only one value can be returned
  - 0 to -99 are reserved for internal usage, one can return customized values also

➤ **Return value can be processed by the calling program as**

```
exec @return_value = <storedprocname>
```

June 22, 2015

Proprietary and Confidential

- 17 -

**IGATE**  
Speed. Agility. Imagination.

The above example checks the state for the ID of a specified contact. If the state is Washington (WA), a status of 1 is returned. Otherwise, 2 is returned for any other condition (a value other than WA for StateProvince or ContactID that did not match a row).

**Output parameters:** Scalar data can be returned from a stored procedure with output variables.

**RETURN:** A single integer value can be returned from a stored procedure with a RETURN statement.

**Result sets:** A stored procedure can return data via one or more SELECT statements.

**RAISERROR or THROW:** Informational or error messages can be returned to the calling application via RAISERROR or THROW.

**Example 1:**

```
CREATE Procedure usp_updateprodprice
    @_vcategory int,
    As
BEGIN
    if @_vcategory is NULL or @_vcategory <=0
        begin
            raiserror (50001, 1,1)
            return -1
        end
    Update Products set ProductPrice = ProductPrice*1.1
    WHERE CategoryID= @_vcategory
    return 0
END
```

```
DECLARE @return_value int
Exec @return_value = usp_updateprodprice 7
```

**Example 2**

```
USE AdventureWorks2012;
GO
CREATE PROCEDURE checkstate @param varchar(11)
AS
IF (SELECT StateProvince FROM Person.vAdditionalContactInfo WHERE ContactID =
    @param) = 'WA'
    RETURN 1
ELSE
    RETURN 2;
GO
DECLARE @return_status int;
EXEC @return_status = checkstate '2';
SELECT 'Return Status' = @return_status;
GO
```

## Recompiling Stored Procedures

- **Stored Procedures are recompiled to optimize the queries which makes up that Stored Procedure**
- **Stored Procedure needs recompilation when**
  - Data in underlying tables are changed
  - Indexes are added/removed in tables
- **Recompilation can be done by Using**
  - CREATE PROCEDURE [WITH RECOMPILE]
  - EXECUTE [procedure ]WITH RECOMPILE
  - sp\_recompile [procedure]

June 22, 2015

Proprietary and Confidential

- 18 -

**IGATE**  
Speed. Agility. Imagination

### Example of Procedure with Recompile

```
USE AdventureWorks2012;
CREATE PROCEDURE dbo.uspProductByVendor @Name varchar(30) = '%'
WITH RECOMPILE
AS
SET NOCOUNT ON;
SELECT v.Name AS 'Vendor name', p.Name AS 'Product name' FROM
Purchasing.Vendor AS v JOIN Purchasing.ProductVendor AS pv ON
v.BusinessEntityID = pv.BusinessEntityID JOIN Production.Product AS p ON
pv.ProductID = p.ProductID WHERE v.Name LIKE @Name;
```

### Example of Execute With Recompile

```
USE AdventureWorks2012;
GO
EXECUTE HumanResources.uspGetAllEmployees WITH RECOMPILE;
GO
```

### Example of sp\_recompile

```
USE AdventureWorks2012;
GO
EXEC sp_recompile N'HumanResources.uspGetAllEmployees';
GO
```

## To View the Definition of Stored Procedure

- **To view the definition of a procedure in Query Editor**
  - EXEC sp\_helptext N'AdventureWorks2012.dbo.usp.LogError';
- **To view the definition of a procedure with System Function: OBJECT\_DEFINITION**
  - SELECT OBJECT\_DEFINITION (OBJECT\_ID(N'AdventureWorks2012.dbo.usp.LogError'));
  - Change the database name and stored procedure name to reference the database and stored procedure that you want.

## Guidelines

- One Stored Procedure for One Task
- Create, Test, and Troubleshoot
- Avoid sp\_Prefix in Stored Procedure Names
- Use Same Connection Settings for All Stored Procedures
- Minimize Use of Temporary Stored Procedures

## Error Handling in Procedures

- SQL Server 2005 onwards error handling can be done with
  - TRY.. CATCH blocks
  - @@ERROR global variable
- If a statements inside a TRY block raises an exception then processing of TRY blocks stops and is then picked up in the CATCH block
- The syntax of the TRY CATCH is

```
BEGIN TRY  
    --statements  
END TRY  
BEGIN CATCH  
    --statements  
END CATCH
```

June 22, 2015

Proprietary and Confidential

- 21 -

**IGATE**  
Speed. Agility. Inspiration.

It is a known thing that during an application development one of the most common things we need to take care is Exception Handling . The same point holds good when we are building our databases also .

Typical scenarios where we need to handle errors in DB

1. When we perform some DML operations and need to check the output
2. When a transaction fails – we need to rollback or undo the changes done to the data

Error handling can be done in two ways in SQL Server

Using @@ERROR

Using TRY..CATCH BLOCK

Earlier version of SQL Server like (2000 or 7.x) did not support this TRY ..CATCH construct , therefore error handling was done only using @@ERROR global variable . Whenever an error occurs the variable automatically populates the error message , which needs to be traced in the next line for example

Insert into sometable values(....)

Select @@error

This will show the errorcode generated in the last SQL statement

## Error Handling

### Using @@Error

```

DECLARE @v_deptcode int
DECLARE @v_deptname varchar(10)
DECLARE @errorcode int

set @v_deptcode=10
set @v_deptname='Pre sales'

insert into dept values(@v_deptcode,'Pre sales')

set @errorcode = @@ERROR
if @errorcode > 0
begin
    print 'error'
    print @errorcode
end
else
    print 'added successfully'
  
```

### Using TRY ..CATCH

```

DECLARE @v_deptcode int
DECLARE @v_deptname varchar(10)
DECLARE @errorcode int

set @v_deptcode=10
set @v_deptname='Pre sales'

BEGIN TRY
    insert into dept values(@v_deptcode,'Pre sales')
END TRY

BEGIN CATCH
    PRINT 'An error occurred while inserting'
    PRINT ERROR_NUMBER()
END CATCH
  
```

June 22, 2015

Proprietary and Confidential

- 22 -

**IGATE**  
Speed. Agility. Imagination

TRY ..CATCH blocks are standard approach to exception handling in any modern language (Java, VB, C++, C# etc). The syntax of TRY ..CATCH block is almost similar to that of the programming languages . Nested Try catch is also possible

The general syntax of the TRY..CATCH is as follows

```

--sql statements
BEGIN TRY
    sql statements
    sql statements
END TRY
BEGIN CATCH
  
```

ERROR_MESSAGE()	Returns the complete description of the <b>error message</b>
ERROR_NUMBER()	Returns the number of the <b>error</b>
ERROR_SEVERITY()	Returns the number of the <b>Severity</b>
ERROR_STATE()	Returns the <b>error state</b> number
ERROR_PROCEDURE()	Returns the name of the stored procedure where the <b>error</b> occurred
ERROR_LINE()	Returns the line number that caused the <b>error</b>

## Error Handling using RAISEERROR

- **RAISERROR can be used to**
  - Return user defined or system messages back to the application
  - Assign a specific error number , severity and state to a message
- **Can be associated to a Query or a Procedure**
- **Has the following syntax**
- **RAISERROR (message ID | message str),severity,state**
- **Message ID has to be a number greater than 50,000**
- **Can be used along with TRY ..CATCH /other error handling mechanisms**

June 22, 2015 | Proprietary and Confidential | - 23 -

**IGATE**  
Speed. Agility. Imagination

Every stored procedure returns an integer return code to the caller. If the stored procedure does not explicitly set a value for the return code, the return code is 0.

**RAISERROR statement**  
**RAISERROR ( { msg\_id | msg\_str | @local\_variable } { ,severity  
,state } [ ,argument [ ,...n ] ] )**

**msg\_id**

Is a user-defined error message number stored in the **sys.messages** catalog view using **sp\_addmessage**. Error numbers for user-defined error messages should be greater than 50000. When **msg\_id** is not specified, RAISERROR raises an error message with an error number of 50000.

**msg\_str**

Is a user-defined message with formatting similar to the **printf** function in the C standard library. The error message can have a maximum of 2,047 characters. If the message contains 2,048 or more characters, only the first 2,044 are displayed and an ellipsis is added to indicate that the message has been truncated. Note that substitution parameters consume more characters than the output shows because of internal storage behavior. For example, the substitution parameter of %d with an assigned value of 2 actually produces one character in the message string but also internally takes up three additional characters of storage. This storage requirement decreases the number of available characters for message output.  
When **msg\_str** is specified, RAISERROR raises an error message with an error number of 5000.

**@local\_variable**

Is a variable of any valid character data type that contains a string formatted in the same manner as `msg_str`. `@local_variable` must be `char` or `varchar`, or be able to be implicitly converted to these data types.

**severity**

Is the user-defined severity level associated with this message. When using `msg_id` to raise a user-defined message created using `sp_addmessage`, the severity specified on `RAISERROR` overrides the severity specified in `sp_addmessage`.

Severity levels from 0 through 18 can be specified by any user. Severity levels from 19 through 25 can only be specified by members of the `sysadmin` fixed server role or users with `ALTER TRACE` permissions. For severity levels from 19 through 25, the `WITH LOG` option is required.

`RAISERROR` is used to return messages back to applications using the same format as a system error or warning message generated by the SQL Server Database Engine.

`RAISERROR` can return either:

- A user-defined error message that has been created using the `sp_addmessage` system stored procedure. These are messages with a message number greater than 50000 that can be viewed in the `sys.messages` catalog view.
- A message string specified in the `RAISERROR` statement.

A `RAISERROR` severity of 11 to 19 executed in the `TRY` block of a `TRY...CATCH` construct causes control to transfer to the associated `CATCH` block. Specify a severity of 10 or lower to return messages using `RAISERROR` without invoking a `CATCH` block. `PRINT` does not transfer control to a `CATCH` block.

When `RAISERROR` is used with the `msg_id` of a user-defined message in `sys.messages`, `msg_id` is returned as the SQL Server error number, or native error code. When `RAISERROR` is used with a `msg_str` instead of a `msg_id`, the SQL Server error number and native error number returned is 50000.

When you use `RAISERROR` to return a user-defined error message, use a different state number in each `RAISERROR` that references that error. This can help in diagnosing the errors when they are raised.

Use `RAISERROR` to:

- Help in troubleshooting Transact-SQL code.
- Check the values of data.
- Return messages that contain variable text.
- Cause execution to jump from a `TRY` block to the associated `CATCH` block.
- Return error information from the `CATCH` block to the calling batch or application

### Example of Raiserror with TRY ..CATCH

```
CREATE Procedure usp_updateprodprice
    @i_vcategory int,
    @i_vpriceinc money
As
BEGIN
    if @i_vcategory is NULL or @i_vcategory<=0
    begin
        raiserror(50001,1,1)
        return
    end
    if @i_vpriceinc <=0
    begin
        raiserror (50002,1,1)
        return
    end
```

June 22, 2015 | Proprietary and Confidential | - 25 -

IGATE  
Speed. Agility. Imagination

The structure of the new table is as follows

```
create table revised_product
    (ProductID int not null,
    ProductName nvarchar(80),
    unitPrice money,
    CategoryID int,
    revisedprice money
)
```

The procedure takes a CategoryID and price increase percentage and updates the revised product table with product details, old price and revised price . The procedure can be executed as follows

All the error codes and messages has been added in the sysmessages table using the procedure sp\_addmessage  
sp\_addmessage 50001,1,'invalid category'

When the procedure is executed , it will display the message associated with 50001

```
BEGIN TRY
    exec usp_updateprodprice -7,.2
END TRY
BEGIN CATCH
    select ERROR_MESSAGE()
END CATCH
```

### Example of Raiserror with TRY ..CATCH

```
if not exists(SELECT 'a' FROM Categories
              WHERE CategoryID = @_vcategory)
begin
    raiserror (50003,1,1)
    return
end
BEGIN TRY
    insert into revised_product
    select ProductID,ProductName,unitPrice,@_vcategory,unitPrice+unitPrice*@_vpriceinc
    FROM Products where CategoryID=@_vcategory

    return
END TRY

BEGIN CATCH
    raiserror (50004,1,1)
    rollback tran
    -- return -1
END CATCH
END
```

June 22, 2015

Proprietary and Confidential

- 26 -

IGATE  
Speed. Agility. Inspiration.

## THROW Statement

- Exception handling is now made easier with the introduction of the THROW statement in SQL Server 2012.
- In previous versions, RAISERROR was used to show an error message.

June 22, 2015

Proprietary and Confidential

- 27 -

IGATE  
Speed. Agility. Imagination

### Drawbacks of RAISERROR:

It requires a proper message number to be shown when raising any error.

The message number should exist in sys.messages.

RAISERROR cannot be used to re-throw an exception raised in a TRY..CATCH block.

RAISERROR has been considered as deprecated features

Unlike RAISERROR, THROW does not require that an error number to exist in sys.messages (although it has to be between 50000 and 2147483647).

You can throw an error using Throw as below:

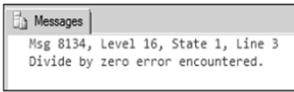
```
THROW 50001, 'Error message', 1;
```

This will return an error message:

```
Msg 50001, Level 16, State 1, Line 1 Error message
```

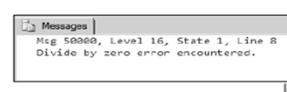
## Difference Between RaiseError and Throw

```
BEGIN TRY
DECLARE @MyInt int
SET @MyInt = 1/0
END TRY
BEGIN CATCH
DECLARE @ErrorMessage
nvarchar(4000), @ErrorSeverity int
SELECT @ErrorMessage =
ERROR_MESSAGE(),
@ErrorSeverity = ERROR_SEVERITY()
RAISERROR(@ErrorMessage,
@ErrorSeverity, 1)
END CATCH
```



June 22, 2015 | Proprietary and Confidential | - 28 -

```
BEGIN TRY
DECLARE @MyInt int
SET @MyInt = 1/0
END TRY
BEGIN CATCH
-- throw out the error
THROW
END CATCH
```



iGATE  
Speed. Agility. Imagination.

Note: All exceptions being raised by THROW have a severity of 16.

RAISERROR statement THROW statement  
If a msg\_id is passed to RAISERROR, the ID must be defined in sys.messages.  
The error\_number parameter does not have to be defined in sys.messages.

RAISERROR statement THROW statement  
The msg\_str parameter can contain printf formatting styles.  
The message parameter does not accept printf style formatting.

RAISERROR statement THROW statement  
The severity parameter specifies the severity of the exception.  
There is no severity parameter. The exception severity is always set to 16.

```
BEGIN TRY
SELECT 1/0
END TRY

BEGIN CATCH
THROW
END CATCH

USE tempdb;
GO
CREATE TABLE dbo.TestRethrow ( ID INT PRIMARY KEY );
BEGIN TRY
INSERT dbo.TestRethrow(ID) VALUES(1);
-- Force error 2627, Violation of PRIMARY KEY constraint to be raised.
INSERT dbo.TestRethrow(ID) VALUES(1);
END TRY
BEGIN CATCH
PRINT 'In catch block.';
THROW;
END CATCH;
```

### Advantages of THROW:

- THROW has now made the developer's life much easier and developers can now code independent of the Tester's input on the exception message.
- It can be used in a TRY..CATCH block.
- No restrictions on error message number to exist in sys.messages.

## Best Practices

- Verify Input Parameters
- Design Each Stored Procedure to Accomplish a Single Task
- Validate Data Before You Begin Transactions
- Use the Same Connection Settings for All Stored Procedures
- Use WITH ENCRYPTION to Hide Text of Stored Procedures

June 22, 2015 | Proprietary and Confidential | - 30 -

IGATE  
Speed. Agility. Imagination.

### Example of WITH ENCRYPTION

Use AdventureWorks2012

```
CREATE PROCEDURE HumanResources.uspEncryptThis
WITH ENCRYPTION
AS
SET NOCOUNT ON;
SELECT BusinessEntityID, JobTitle, NationalIDNumber,
VacationHours, SickLeaveHours FROM
HumanResources.Employee;
GO
```

The WITH ENCRYPTION option obfuscates the definition of the procedure when querying the system catalog or using metadata functions, as shown by the following examples.

Run sp\_helptext:

```
EXEC sp_helptext 'HumanResources.uspEncryptThis';
```

Here is the result set.

The text for object 'HumanResources.uspEncryptThis' is encrypted.

## Demo

### ➤ Stored Procedures



## Definition

- **Named Collections of Transact-SQL Statements**
- **Takes parameter and returns a single value**
- **Can be used as a part of expression**

Note : Table data types are also singular value

June 22, 2015

Proprietary and Confidential

| - 32 -

**IGATE**  
Speed. Agility. Imagination

### What is user defined function?

You've seen parameterized functions such as CONVERT, RTRIM, and ISNULL as well as parameterless function such as getdate(). SQL Server 2008 lets you create functions of your own. You should think of the built-in functions as almost like built-in commands in the SQL language; they are not listed in any system table, and there is no way for you to see how they are defined.

### Table Variables

To make full use of user-defined functions, it's useful to understand a special type of variable that SQL Server 2008 provides. You can declare a local variable as a table or use a table value as the result set of a user-defined function. You can think of table as a special kind of datatype. Note that you cannot use table variables as stored procedure or function parameters, nor can a column in a table be of type table.

Table variables have a well-defined scope, which is limited to the procedure, function, or batch in which they are declared. Here's a simple example:

```
DECLARE @pricelist TABLE(tid varchar(6), price money)
INSERT @pricelist SELECT title_id, price FROM titles
SELECT * FROM @pricelist
```

The definition of a table variable looks almost like the definition of a normal table, except that you use the word DECLARE instead of CREATE, and the name of the table variable comes before the word TABLE.

The definition of a table variable can include:

A column list defining the datatypes for each column and specifying the NULL or NOT NULL property

PRIMARY KEY, UNIQUE, CHECK, or DEFAULT constraints

The definition of a table variable cannot include:

Foreign key references to other tables

Columns referenced by a FOREIGN KEY constraint in another table

### Differences: Stored Procedure and Functions

Procedures	Function
Return single integer value represents return status	Return single value of any scalar data type supported by SQL server or Table type
Use execute statement to execute stored procedure	Can be called through select statement if it returns scalar value otherwise can be called through from statement if it returns table.
Use output parameter to pass values to caller	Use return statement to pass values to caller

## Creating a User-defined Function

```

CREATE Function udf_GetProductcategory(@i_prodID INT)
RETURNS nvarchar(40)
as
BEGIN
    declare @retvalue nvarchar(40)
    if @i_prodID is NULL or @i_prodID<=0
        return null
    SELECT@retvalue=CategoryName From
    PRODUCTS, CATEGORIES
    WHERE PRODUCTS.CategoryID = CATEGORIES.CategoryID
    AND PRODUCTS.ProductID=@i_prodID
    return@retvalue
END

```

June 22, 2015

Proprietary and Confidential

| - 34 -

**IGATE**  
Speed. Agility. Imagination

```

CREATE FUNCTION [ schema_name. ] function_name
[ [ { @parameter_name [ AS ] parameter_data_type
[ =default ] } [ ,...n ] ] ] RETURNS return_data_type
[ WITH <function_option> [ ,...n ] ]
[ AS ]
BEGIN
    function_body
    RETURN scalar_expression
END [ ; ]
schema_name
    Is the name of the schema to which the user-defined
    function belongs.
function_name
    Is the name of the user-defined function. Function names
    must comply with the rules for identifiers and must be
    unique within the database and to its schema.
@parameter_name
    Is a parameter in the user-defined function. One or more
    parameters can be declared.
A function can have a maximum of 1,024 parameters. The
value of each declared parameter must be supplied by
the user when the function is executed, unless a default
for the parameter is defined.
Specify a parameter name by using an at sign (@) as the
first character. The parameter name must comply with
the rules for identifiers. Parameters are local to the
function; the same parameter names can be used in
other functions. Parameters can take the place only of
constants; they cannot be used instead of table names,
column names, or the names of other database objects.

```

**return\_data\_type**

Is the return value of a scalar user-defined function. For Transact-SQL functions, all data types, including CLR user-defined types, are allowed except the **timestamp** data type. For CLR functions, all data types, including CLR user-defined types, are allowed except **text**, **ntext**, **image**, and **timestamp** data types. The nonscalar types **cursor** and **table** cannot be specified as a return data type in either Transact-SQL or CLR functions.

**Restrictions on function**

- A function can return only single value at a time
- The SQL statements within a function cannot include any nondeterministic system functions.
  - E.g `getdate()` function is nondeterministic hence cannot be used inside function but can be pass as argument.

## Altering and Dropping User-defined Functions

### ➤ Altering Functions

```
ALTER FUNCTION dbo.fn_NewRegion  
<Newfunction content>
```

- Retains assigned permissions
- Causes the new function definition to replace existing definition

### ➤ Dropping Functions

```
DROP FUNCTION dbo.fn_NewRegion
```

June 22, 2015

Proprietary and Confidential

| - 36 -

**IGATE**  
Speed. Agility. Imagination

## ALTER FUNCTION

Alters an existing Transact-SQL or CLR function that was previously created by executing the CREATE FUNCTION statement, without changing permissions and without affecting any dependent functions, stored procedures, or triggers.

### Syntax

#### Scalar Functions

```
ALTER FUNCTION [ schema_name. ] function_name  
( [ { @parameter_name [ AS ] parameter_data_type [= default] } [ ,...n ] ] ) RETURNS return_data_type  
[ AS ]  
BEGIN  
    function_body  
    RETURN scalar_expression  
END [ ; ]
```

All arguments have same meaning as it is in create function

## DROP FUNCTION

Removes one or more user-defined functions from the current database. User-defined functions are created by using create function and modified by using alter function.

## View Definition of a Function

- **SELECT definition, type**
- **FROM sys.sql\_modules AS m**
- **JOIN sys.objects AS o ON m.object\_id = o.object\_id**
- **AND type IN ('FN', 'IF', 'TF');**
- **GO**

## Demo

- Creating User-defined Functions



## Summary

- In this lesson, you have learnt:
- Creating, Executing, Modifying, and Dropping Stored Procedures
- Using Parameters in Stored Procedures
- Using User defined Functions
  - Scalar User-defined Function
  - Multi-Statement Table-valued Function
  - In-Line Table-valued Function



## Review Question

- **Question 1:** A stored procedure can return a single integer value
  - True
  - False
- **Question 2:** \_\_\_\_\_ stored procedures call subroutines written in languages like c, c++,NET
- **Question 3:** \_\_\_\_\_ function includes only one select statement



# MS SQL Server 2012 for BI – Database Development

## Lab Book

Copyright © 2011 IGATE Corporation. All rights reserved. No part of this publication shall be reproduced in any way, including but not limited to photocopy, photographic, magnetic, or other record, without the prior written permission of IGATE Corporation.

IGATE Corporation considers information included in this document to be Confidential and Proprietary.

## Document Revision History

Date	Revision No.	Author	Summary of Changes
25 <sup>th</sup> July 2011	2.0	Latha S.	Changes in Material made based on integration process
3 <sup>rd</sup> April, 2012	3.0	Shilpa Bhosle	Changes in Lab Book are made as an upgrade to SQL Server 2008
21 <sup>st</sup> Aug 2013	4.0	Shashank Saudagar	Changes in Lab Book are made as an upgrade to SQL Server 2012
14 <sup>th</sup> May 2015	5.0	Vaishali Kasture	Changes in Lab Book as per new curriculum of SQL Server 2012

# Table of Contents

<b>Getting Started .....</b>	<b>4</b>
Overview .....	4
Setup Checklist for SQL Server 2012 .....	4
<b>Lab 1. Getting connected to the SQL Server 2012 Server.....</b>	<b>5</b>
1.1: Steps to connect to the SQL Server 2012 Server .....	5
1.2: Getting Familiar with SQL Server .....	6
1.3: SQL Languages – DDL- Creating Tables, Alias Data Type and Constraints .....	7
1.4: Simple Queries & Merge Statement.....	9
1.5: Data Retrieval - Joins , Subqueries , SET Operators and DML.....	15
1.6: Indexes and Views .....	Error! Bookmark not defined.
1.7: Introduction to T-SQL blocks .....	Error! Bookmark not defined.
1.8: Procedures and Function and Exception Handling in SQL server .....	19
1.9: Triggers .....	Error! Bookmark not defined.
<b>Lab 2. SQL Server 2012 Language Enhancements .....</b>	<b>Error! Bookmark not defined.</b>
2.1 Creating and using Sequence Numbers .....	Error! Bookmark not defined.
2.2 Raising Exceptions using the THROW Statement .....	Error! Bookmark not defined.
2.3 ColumnStore Index .....	Error! Bookmark not defined.
2.4 With resultsets with Stored Procedure .....	Error! Bookmark not defined.
2.5 Selecting Results Using Paging .....	Error! Bookmark not defined.
<b>Summary .....</b>	<b>Error! Bookmark not defined.</b>
<b>Appendices.....</b>	<b>21</b>
Appendix A: Table Structure .....	21
Appendix B: Table of Figures.....	24
Appendix C: Table of Examples.....	Error! Bookmark not defined.

## Getting Started

### Overview

This Lab book is a guided tour for Learning SQL server 2008

Each section contains some examples and assignments. Follow the steps provided in the solved examples and then work out the Assignments given.

### Setup Checklist for SQL Server 2012

Here is what is expected on your machine in order for the lab to work.

#### **Minimum System Requirements**

##### **Processor, HDD & RAM**

- Processor - Minimum: AMD Opteron, AMD Athlon 64, Intel Xeon with Intel EM64T support, Intel Pentium IV with EM64T support
- Processor speed: Minimum: 1.4 GHz
- Recommended: 2.0 GHz or faster
- RAM - Minimum: 512 MB, Recommended: 2.048 GB or more
- HDD – 150 GB

##### **Operating System**

- Windows XP Professional x64
- Windows 7 Professional 64 bit

##### **SQL Server 2012 Developer Edition**

- SQL server 2012 client and a SQL server 2012 Server instance running on the Server.
- A database called training will be available; all objects for the lab session would be stored in that database alone

## Lab 1. Getting connected to the SQL Server 2012 Server

### 1.1: Steps to connect to the SQL Server 2012 Server

**Step 1:** Click Start, Programs, Microsoft SQL Server 2012, SQL Server Management Studio.

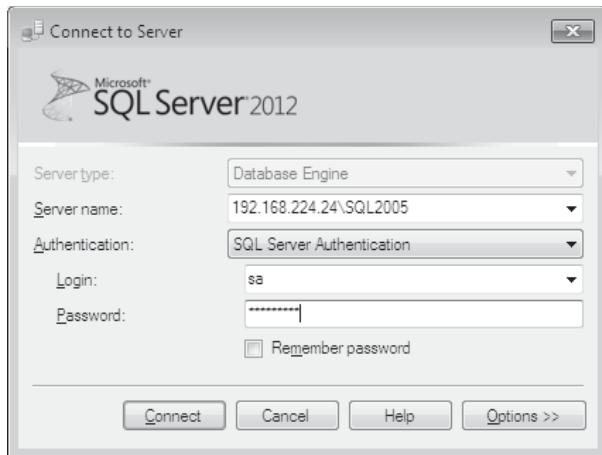


Figure 1: Connecting to SQL Server 2012

**Step 2:**

Enter the Login, Password and the Server name provided to you.

Login: <loginid>Passwd: <password>

**Step 3:** Click on New Query.

## 1.2: Getting Familiar with SQL Server

1. Identify all the system and user defined database in your system.
2. Make master database as your current database , by using the command

```
Use <databasename>
```

3. Find out if your active database is master ,by giving the command

```
Select db_name()  
go
```

4. Now make Training database as your active database
5. Find out the content of the database by giving the following command. Observe the output

```
Sp_help  
go
```

6. Repeat the above steps for master database and Northwind database
7. Find out the version of your SQL Server by giving the following command

```
Select @@version  
go
```

8. Find out the server date by giving the following commands

```
Select getdate()  
go
```

- 
9. Make Northwind as your current database , find out information about tables using the command - Categories ,Products , Orders, Order Details , Employees

```
Sp_help<tablename>
```

```
go
```

10. Make a note of all related tables and foreign key columns
11. Repeat the above operation of Training database tables as well

### 1.3: SQL Languages – DDL- Creating Tables, Alias Data Type and Constraints

The following questions will be solved using the Training database only

12. Create a Table called Customer\_<empid> with the following Columns

Customerid	Int	Unique NOT NULL
CustomerName	varchar(20)	Not Null
Address1	varchar(30)	
Address2	varchar(30)	
Contact Number	varchar(12)	Not Null
Postal Code	Varchar(10)	

1. Create a user defined data type called Region, which would store a character string of size 15.



**Hint:** Use Create Type Statement

2. Create a Default which would store the value 'NA' (North America)



**Hint:** create default

3. Bind the default to the Alias Data Type of Q13 i.e. region



**Hint:** use sp\_bindefault

**Syntax -** EXEC sp\_bindefault<DefaultName>, '<AliasName>'

4. Modify the table Customers to add the a column called Customer\_Region which would be of data type:  
Region
5. Add the column to the Customer Table.  
Gender char (1)
6. Using alter table statement add a constraint to the Gender column such that it would not accept any other values except 'M','F' and 'T'.
7. Create the Table Orders with the following Columns:

OrdersID	Int	NOT NULL IDENTITY with starting values 1000
Customerid	Int	Not Null
OrdersDate	Datetime	
Order_State	char(1)	can be only 'P' or 'C'

8. Add referential integrity constraint for Orders & Customer tables through CustomerId with the name fk\_CustOrders.

Using sp\_help check if the constraints have been added properly.

#### 1.4: Simple Queries & Merge Statement

For these questions, you will be using the University Schema , The table structure has been given in the appendix. These tables would be available in the Training database

1. List out Student\_Code, Student\_Name and Department\_Code of every Student
2. Do the same for all the staff's
3. Retrieve the details (Name, Salary and dept code) of the employees who are working in department 20, 30 and 40.
4. Display Student\_Code, Subjects and Total\_Marks for every student. Total\_Marks will calculate as Subject1 + Subject2 + Subject3 from Student\_Marks. The records should be displayed in the descending order of Total Score
5. List out all the books which starts with 'An', along with price
6. List out all the department codes in which students have joined this year
7. List out a report like this  
StaffCodeStaffName Dept Code Date of Joining No of years in the Company
8. List out all staff's who have joined before Jan 2000
9. Write a query which will display Student\_Name, Department\_Code and DOB of all students who born between January 1, 1981 and March 31, 1983.
10. List out all student codes who did not appear in the exam subject2
11. List the empno, name and Department No of the employees who have got experience of more than 18 years.
12. List out all the book code and library member codes whose return is still pending
13. Display the name and salary of the staff. Salary should be represented as X. Each X represents a 1000 in salary. It is assumed that a staff's salary to be multiples of 1000 , for example a salary of 5000 is represented as XXXXX

Sample Output

JOHN	10000	XXXXXXXXXX
ALLEN	12000	XXXXXXXXXXXX

14. Display name and date of birth of students where date of birth must be displayed in the format similar to “January, 12 1981” for those who were born on Saturday or Sunday.



**Hint:** Use datename or datepart function

15. List all the staff's whose birthday falls on the current month
16. How many books are stocked in the library?
17. How many books are there for topics Physics and Chemistry?
18. How many members are expected to return their books today?
19. Display the Highest, Lowest, Total & Average salary of all staff. Label the columns Maximum, Minimum, Total and Average respectively. Round the result to nearest whole number
20. How many staffs are managers”?

#### **Stretched Assignments**

- |     |   |
|-----|---|
| 21. | List out year wise total students passed. The report should be as given below. A student is considered to be passed only when he scores 60 and above in all 3 subjects individually |
|     | Year    No of students passed   |
| 22. | List out all the departments which is having a headcount of more than 10  |
| 23. | List the total cost of library inventory ( sum of prices of all books )   |
| 24. | List out category wise count of books costing more than Rs 1000/-   |
| 25. | How many students have joined in Physics dept (dept code is 10) last year?  |

### Working with Merge Statement

**Case Study** – The Countryside Confectioneries is one of the well-known names in the brands of confectioneries in Switzerland. The company Database Administrator John & his team, maintains the entire business data in SQL Server 2008. As a database administrator, John, need to perform the ETL (Extract, Transform & Load) on database quite often, wherein he needs to execute multiple INSERT, UPDATE & DELETE Operations on database target table by matching the records from the source table. For example, a products dimension table has information about the products; you need to sync-up this table with the latest information about the products from the source table.

To simplify above task John & his team uses SQL Server 2008's one of the remarkable programming enhancement called MERGE statement as MERGE SQL command to perform these operations in a single statement. He uses MERGE statement to so that he can eliminate the need of writing multiple and separate DML statements to refresh the target table with an updated product list or do lookups.

The following example demonstrates the use of MERGE statement in above given case study.

**A. Execute the following SQL queries in SQL Server 2008 SSMS as instructed & observe the output**

1. Create following tables in SQL Server 2008 – In this demo you will be creating Products table as Target table & UpdateProducts as Source Table. You will also populate these tables with some sample data

```
CREATE TABLE Products
(
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    Rate MONEY
)
--Insert records into target
table
```

```
CREATE TABLE UpdatedProducts
(
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    Rate MONEY
)
```

**MERGE Statement**

-Synchronize the target table with  
 -refreshed data from source table

```
MERGE Products AS TARGET
USING UpdatedProducts AS SOURCE
ON (TARGET.ProductID = SOURCE.ProductID)

-When records are matched, update
-the records if there is any change

WHEN MATCHED AND TARGET.ProductName<>SOURCE.ProductName
OR TARGET.Rate<>SOURCE.Rate THEN
UPDATE SET TARGET.ProductName = SOURCE.ProductName,
TARGET.Rate = SOURCE.Rate

-When no records are matched, insert
-the incoming records from source
-table to target table

WHEN NOT MATCHED BY TARGET THEN
INSERT (ProductID, ProductName, Rate)
VALUES (SOURCE.ProductID, SOURCE.ProductName, SOURCE.Rate)

-When there is a row that exists in target table and
-same record does not exist in source table
-then delete this record from target table

WHEN NOT MATCHED BY SOURCE THEN
DELETE

-$action specifies a column of type nvarchar(10)
-in the OUTPUT clause that returns one of three
-values for each row: 'INSERT', 'UPDATE', or 'DELETE',
-according to the action that was performed on that row

OUTPUT $action,
DELETED.ProductID AS TargetProductID,
DELETED.ProductName AS TargetProductName,
```

Create two tables (Target & Source) with a structure given below. Also, insert some sample data into both tables.

Target Table

```
CREATE TABLE EmployeeTarget
(
    EmpID INT NOT NULL,
    Designation VARCHAR (25)
    NOT NULL,
    Phone VARCHAR (20) NOT
```

Source Table

```
CREATE TABLE EmployeeSource
(
    EmpID INT NOT NULL,
    Designation VARCHAR (25)
    NOTNULL,
    Phone VARCHAR (20) NOT NULL,
```

**TODO** – Write a MERGE Statement, that matches the EmpID attribute in the Source with the EmpID attribute of target. When a match is found in the target, the target Employee's attributes should overwrite with the source Employee attributes. When a match is not found in the target, a new row should be inserted into the target, using the source Employee attributes. When a match is not found in the source, the target Employee row should be deleted:

#### Working with Grouping Set

- Create the following table & populate with some sample data.
- Write following query which uses Grouping Set in the query window.

Employee Table

```
CREATE TABLE Employee
(
    Employee_Number int NOT NULL
    PRIMARY KEY,
    Employee_Name varchar (30)
    NULL,
    Salary float NULL,
    Department_Number int NULL,
```

```
SELECT Region, Department_number, avg (salary)
Average_Salary
From Employee
Group BY GROUPING SETS
(
    (Region, Department_number),
    (Region),
    (Department_number)
)
```

- C. Execute above query & observe the output.
- D. The query performs following :
  - a. It generates result set grouped by each set mentioned in the Grouping Sets.
  - b. It also calculates average salary of every employee for each region and department.

**Optional** - One can get the same result achieved in SQL Server 2005 using the following

query:(**NOTE – This part of Lab is not compulsory to perform**)

```

SELECT Region, Department_number, avg (salary) Average_Salary
From Employee
Group BY (Region, Department_number)
UNION
SELECT Region, Department_number, avg (salary) Average_Salary
From Employee
Group BY (Region)
UNION
SELECT Region, Department_number, avg (salary) Average_Salary
From Employee
Group BY (Department_number)
  
```

## 1.5: Data Retrieval - Joins , Subqueries , SET Operators and DML

1. Write a query which displays Staff Name, Department Code, Department Name, and Salary for all staff who earns more than 20000.
2. Write a query to display Staff Name, Department Code, and Department Name for all staff who do not work in Department code 10
3. Print out a report like this
 

Book Name	No of times issued
Let us C	12
Linux Internals	9
4. List out number of students joined each department last year. The report should be displayed like this
 

Physics	12
Chemistry	40
5. Write a query that displays Staff Name, Salary, and Grade of all staff. Grade depends on the following table.

Salary	Grade
Salary >=50000	A
Salary >= 25000 < 50000	B
Salary>=10000 < 25000	C

6. Display the Staff Name, Hire date and day of the week on which staff was hired. Label the column as DAY. Order the result by the day of the week starting with Monday.
7. Generate a report which contains the following information.  
Staff Code, Staff Name, Designation, Department, Book Code, Book Name, Author, Fine  
For the staff who have not return the book. Fine will be calculated as Rs. 5 per day.  
 $\text{Fine} = 5 * (\text{No. of days} - \text{Current Date} - \text{Expected return date})$ , for others it should be displayed as –
8. List out a report like this  
Staff Code Staff Name Manager Code Manager Name



**Hint:** Use Self Join

9. Display Staff Code, Staff Name, and Department Name for those who have taken more than one book.
10. List out the names of all student code whose score in subject1 is equal to the highest score
11. Modify the above query to display student names along with the codes.

#### Stretched Assignments

12. List out all the staffs who are reporting to the same manager to whom staff 100060 reports to.
13. List out all the students along with the department who reads the same books which the professors read
14. List out all the authors who have written books on same category as written by Author David Gladstone.
15. Display the Student report Card for this year. The report Card should contain the following information.

Student Code Student Name Department Name Total Marks Grade
Grade is calculated as follows. If a student has scored < 60 or has not attempted an exam he is considered to an F
>80 - E
70-80 - A
60- 69 - B
<60 - F

**Stretched Assignments**

16. Display the Student Code, Student Name, and Department Name for that department in which maximum number of student are studying.
17. List out all the students along with the department who reads the same books which the professors of their respective department read.



**Hint:** Use co-related sub query

18. List out the names of all the books along with the author name, book code and category which have not been issued at all. Try solving this question using EXISTS.
19. List out the code and names of all staff and students belonging to department 20.



**Hint:** Use UNION

20. List out all the students who have not appeared for exams this year.

CustomerID	CustomerName	Address1	Address2	Contact	PostalCode	Region	Gender
ALFKI	AlfredsFutterkiste	Obere Str. 57	Berlin,Germany	030-0074321	12209	NULL	NULL
ANATR	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	México D.F.,Mexico	(5) 555-4729	5021	NA	NULL
ANTON	Antonio Moreno Taquería	Mataderos 2312	México D.F.,Mexico	(5) 555-3932	5023	NULL	NULL
AROUT	Around the Horn	120 Hanover Sq.	London,UK	(171) 555- 7788	WA1 1DP	NULL	NULL
BERGS	Berglundssnabbköp	Berguvsväge n 8	Luleå,Sweden	0921-12 34 65	S-958 22	NULL	NULL
BLAUS	Blauer See Delikatessen	Forsterstr. 57	Mannheim,Germany	0621-08460	68306	NA	NULL
BLONP	Blondesddslpère et fils	24, place Kléber	Strasbourg,France	88.60.15.31	67000	NULL	NULL
BOLID	BólidoComidaspreparadas	C/ Araquil, 67	Madrid,Spain	(91) 555 22 82	28023	EU	NULL

BONAP	Bon app'	12, rue des Bouchers	Marseille,France	91.24.45.40	13008	NULL	NULL
BOTTM	Bottom-Dollar Markets	23 Tsawassen Blvd.	Tsawassen,Canda	(604) 555-4729	T2F 8M4	BC	

21. List out all the student codes who have never taken books
22. Add the following records to the Customers Table , created in our earlier exercises
23. Replace the contact number of Customer id ANATR to (604) 3332345 .
24. Update the Address and Region of Customer BOTTM to the following  
19/2 12th Block , Spring Fields .  
Ireland - UK  
Region - EU
25. Insert the following records in the Orders table. The Order id should be automatically generated  
Save the commands in a script file (Script file has a .sql extension)

Customer ID	OrderDate	Order State
AROUT	4-Jul-96	P
ALFKI	5-Jul-96	C
BLONP	8-Jul-96	P
ANTON	8-Jul-96	P
ANTON	9-Jul-96	P
BOTTM	10-Jul-96	C
BONAP	11-Jul-96	P
ANATR	12-Jul-96	P
BLAUS	15-Jul-96	C
HILAA	16-Jul-96	P

26. Delete all the Customers whose Orders have been cleared .

- 
27. Remove all the records from the table using the truncate command. Rerun the script to populate the records once again
  28. Change the order status to C, for all orders before `15th July.

## 1.6: Procedures and Function and Exception Handling in SQL server

1. Write a function to compute age. The function should accept a date of birth and return age in years.
2. Write a function which will find if a book is available or not. The function should take a book code as a parameter and return one of the following values  
 “Not In stock” if the book does not exist in library.  
 “On Issue” if the book is available but is on issue (that means has not been returned yet)  
 “Available” if the book is available.
3. Write a procedure that accept Staff\_Code and updates the salary and store the old salary details in Staff\_Master\_Back (Staff\_Master\_Back has the same structure without any constraint) table. The procedure should return the updated salary as the return value  
 Exp < 2 then no Update  
 Exp >= 2 and <= 5 then 20% of salary  
 Exp > 5 then 25% of salary
4. Write a procedure to update the marks details in the Student\_marks table. The following is the logic.
  - The procedure should accept student code , and marks as input parameter
  - Year should be the current year.
  - Student code cannot be null, but marks can be null.
  - Student code should exist in the student master.
  - The entering record should be unique ,i.e no previous record should exist
  - Suitable exceptions should be raised and procedure should return -1.
  - IF the data is correct, it should be added in the Student marks table and a success value of 0 should be returned.
5. Create a procedure that accepts the book code as parameter from the user. Display the details of the students/staff that have borrowed that book and has not returned the same. The following details should be displayed

Student/Staff Code	Student/Staff Name	Issue Date	Designation	Expected Ret_Date
--------------------	--------------------	------------	-------------	-------------------

6. Write a procedure to insert details into Book\_Transaction table. Procedure should accept the book code and staff/student code. Date of issue is current date and the expected return date should be 10 days from the current date. If the expected return date falls on Saturday or Sunday, then it should be the next working day. Suitable exceptions should be handled.
  
7. Write a function to compute the following. Function should take Staff\_Code and return the cost to company.  

$$\text{DA} = 15\% \text{ Salary}, \text{HRA} = 20\% \text{ of Salary}, \text{TA} = 8\% \text{ of Salary}$$

Special Allowance will be decided based on the service in the company.

< 1 Year	Nil
$\geq 1 \text{ Year} < 2 \text{ Year}$	10% of Salary
$\geq 2 \text{ Year} < 4 \text{ Year}$	20% of Salary
$\geq 4 \text{ Year}$	30% of Salary
  
8. Write a T-SQL block, which will print this report for the entire Staff's.  

$$\text{Staff Code } \text{Staff Name } \text{department } \text{Date of Joining } \text{Cost to the Company}$$



**Hint:** Use the above written function

#### Stretched Assignment

9. Write a Procedure and a function which would do the following	<b>Function:</b> This function will return years of experience for a staff. This function will take the hiredate of the staff as an input parameter. The output will be rounded to the nearest year (1.4 year will be considered as 1 year and 1.5 year will be considered as 2 year).
<b>Procedure:</b> Use the function to calculate the additional allowance for the staff based on the experience.	$\text{Additional Allowance} = \text{Year of experience} \times 3000$ <p>Calculate the additional allowance and return the value as an output parameter.</p>

## Appendices

### Appendix A: Table Structure

Desig\_Master

Name	Null?	Type
Design_code	Not Null	int
Design_name		Varchar(50)

Department\_Master

Name	Null?	Type
Dept_Code	Not Null	int
Dept_name		Varchar(50)

Student\_Master Table

Name	Null?	Type	
Student_Code	Not Null	int	
Student_name	Not Null	Varchar(50)	
Dept_Code		int	FK ->Dept_Master
Student_dob		Datetime	
Student_Address		Varchar(240)	

Student\_Marks

Name	Null?	Type	
Student_Code		int	FK->Student_master
Student_Year	Not Null	int	
Subject1		int	

---

Subject2		int	
Subject3		int	

## Staff\_Master

Name	Null?	Type	
<u>Staff_code</u>	Not Null	int	
Staff_Name	Not Null	Varchar(50)	
Design_code		int	FK->Design_master
Dept_code		int	FK->Dept_Master
HireDate		Datetime	
Staff_dob		Datetime	
Staff_address		Varchar(240)	
Mgr_code		int	
Staff_sal		decimal (10,2)	

## Book\_Master

Name	Null?	Type
<u>Book_Code</u>	Not Null	int
Book_Name	Not Null	Varchar(50)
Book_pub_year		int
Book_pub_author	Not Null	Varchar(50)
Book_category	Not null	Varchar(10)

## Book\_Transaction

Name	Null?	Type	
<u>Book_Code</u>		int	Fk->Book_master
<u>Student_code</u>	Null	int	FK->Student_master
<u>Staff_code</u>	Null	int	FK->Staff_master
<u>Book_Issue_date</u>	Not Null	Datetime	
<u>Book_expected_return_date</u>	Not Null	Datetime	
<u>Book_actual_return_date</u>	Null	Datetime	

## Appendix B: Table of Figures

Figure 1: Connecting to SQL Server 2012 .....	5
---	---