Waters

THE SCIENCE OF WHAT'S POSSIBLE.™

Introduction to Microsoft SQL Server Databases

July 2019

Course outline

Module 1

Introduction to databases

Module 2

Introduction to Data Manipulation Language (DML)

Module 3

Designing and Implementing Tables (DDL)

Module 4

Querying Multiple Tables

Module 5

Other Database Objects (Views, Stored Procedures, Functions)

Module 6

Database Performance (indexes)

Module 1

Introduction to Databases

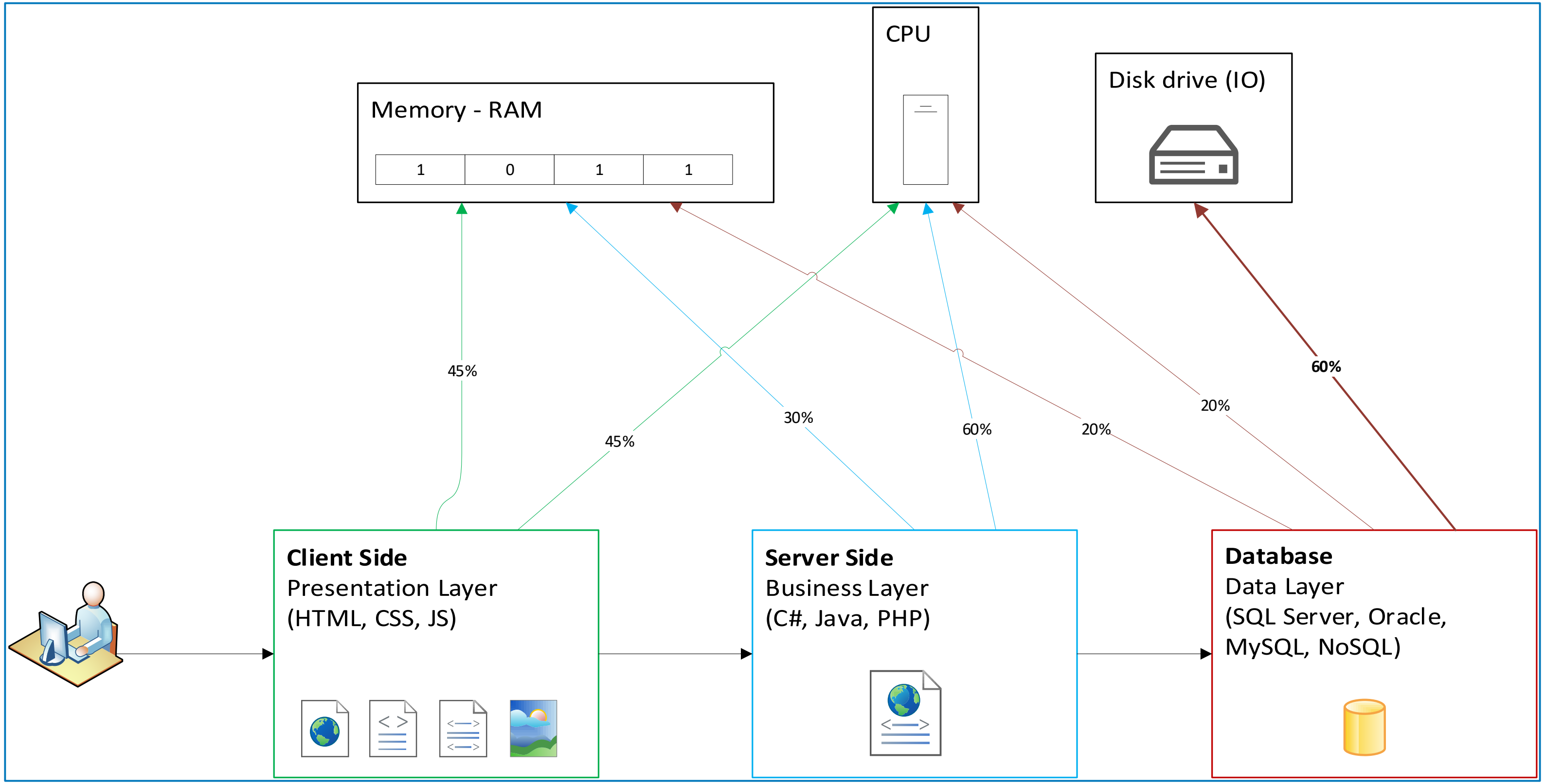
Module Overview

- Introduction to Relational Database
- Other Types of Databases and Storage
- Database Languages in SQL Server

Lesson 1: Introduction to Relational Databases

- Application Architecture Overview
- Relational Database Fundamentals
- Tables in Relational Databases
- Introduction to Normalization

Application Architecture Overview



Relational Database Fundamentals

- SQL Server is a relational database management system
- Databases contain objects and data
- Each database has multiple tables
- Tables are joined together to extract meaningful information

Tables in Relational Databases

Table Relationships

Orders table

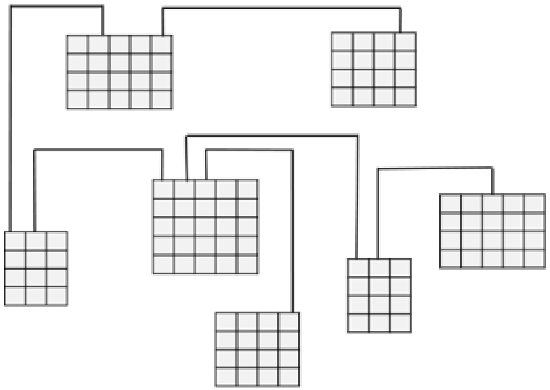
Order ID	Customer ID	Order Date	Shipper ID
503	1	03/22/2016	2
504	2	03/22/2016	3
505	2	03/23/2016	2
506	3	03/23/2016	4

Customers table

Customer ID	First Name	Last Name	Street Address	City
1	Latasha	Navarro	6954 Ranch Rd	Denver
2	Abby	Sai	7074 Spoonwood Court	Seattle
3	Julia	Nelson	2196 Coat Court	Chicago
4	Adam	Ross	4378 Westminster Place	New York

Introduction to Normalization

- OLTP databases typically perform better when redundant data is minimized
- Normalization is the process of reducing redundant data in a database
  - First normal form
  - Second normal form
  - Third normal form

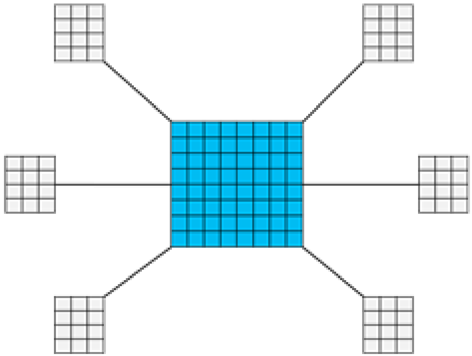


Lesson 2: Other Types of Databases and Storage

- Data Warehouses
- NoSQL Databases

Data Warehouses

- Organizations use data warehouses to store data for analysis
- Denormalization is a common practice in data warehouse design
- Data warehouses are populated by ETL processes
- A common data warehouse design approach is the dimensional model



NoSQL Databases

- NoSQL databases:
  - Do not use the relational data model
  - Offer better performance than relational databases for very large volumes of complex data
  - Some NoSQL databases lack key relational features
- Increasing prevalence of NoSQL databases:
  - The need to store and manage big data
  - Greater availability of scale-out technologies
- Types of NoSQL databases include:
  - Document-oriented databases
  - Object-oriented databases

Lesson 3: Database Languages in SQL Server

- Structured Query Language
- Transact-SQL Queries
- Categories of T-SQL Statements

Structured Query Language

- SQL is a standard language for use with relational databases
- SQL standards are maintained by ANSI and ISO
- Proprietary RDBMS systems have their own extensions of SQL, such as Transact-SQL
- Subsets of SQL include:
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
  - Data Control Language (DCL)

Transact-SQL Queries

- T-SQL is a set-based language
- T-SQL is written in scripts with .sql extension
- GO keyword separates batches

```
CREATE TABLE dbo.Employees
(
		EmployeeID int PRIMARY KEY,
		LastName nvarchar(25),
		FirstName nvarchar(25)
);
GO

INSERT INTO dbo.Employees
		(EmployeeID, LastName, FirstName)
VALUES
		(121, N'O''Neil', N'Carlene');
GO
```

Transact-SQL Queries

SELECT list identifies the columns to select data from

FROM clause states the table or tables that contain the columns

```
SELECT SalesOrderID, OrderDate
FROM Sales.SalesOrderHeader
WHERE SalespersonID = 282;
GO
```

WHERE clause filters the rows returned

Categories of T-SQL Statements

Waters
THE SCIENCE OF WHAT'S POSSIBLE.™

DML*	DDL	DCL
• Data Manipulation Language • Used to query and manipulate data • SELECT, INSERT, UPDATE, DELETE	• Data Definition Language • Used to define database objects • CREATE, ALTER, DROP	• Data Control Language • Used to manage security permissions • GRANT, REVOKE, DENY

*DML with SELECT is the focus of this course

Module 2

Introduction to Data Manipulation Language (DML)

Module Overview

- Understanding the Logical Order of Operations in SELECT Statements
- Using Column and Table Aliases
- Adding Data to Tables
- Modifying and Removing Data

Lesson 1: Understanding the Logical Order of Operations in SELECT Statements

- Elements of a SELECT Statement
- T-SQL Language Elements: Predicates and Operators

Elements of a SELECT Statement

Element	Expression	Role
SELECT	<select list>	Defines which columns to return
FROM	<table source>	Defines table(s) to query
WHERE	<search condition>	Filters returned data using a predicate
GROUP BY	<group by list>	Arranges rows by groups
HAVING	<search condition>	Filters groups by a predicate
ORDER BY	<order by list>	Sorts the results

T-SQL Language Elements: Predicates and Operators

Elements:	Predicates and Operators:
Predicates	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
Comparison Operators	=, >, <, >=, <=, <>, !=, !>, !<
Logical Operators	AND, OR, NOT
Arithmetic Operators	*, /, %, +, -,
Concatenation	+

Lesson 2: Using Column and Table Aliases

- Use Aliases to Refer to Columns
- Use Aliases to Refer to Tables
- Logical Query Processing
- The Impact of Logical Processing Order on Aliases
- Applying the Logical Order of Operations to Writing SELECT Statements
- Demonstration: Using Column and Table Aliases

Use Aliases to Refer to Columns

- Column aliases using AS

```
SELECT orderid, unitprice, qty AS quantity
FROM Sales.OrderDetails;
```

- Column aliases using =

```
SELECT orderid, unitprice, quantity = qty
FROM Sales.OrderDetails;
```

- Accidental column aliases

```
SELECT orderid, unitprice quantity
FROM Sales.OrderDetails;
```

Use Aliases to Refer to Tables

- Create table aliases in the FROM clause
- Create table aliases with AS

```
SELECT custid, orderdate
FROM SalesOrders AS SO;
```

- Create table aliases without AS

```
SELECT custid, orderdate
FROM SalesOrders SO;
```

- Using table aliases in the SELECT clause

```
SELECT SO.custid, SO.orderdate
FROM SalesOrders AS SO
```

Logical Query Processing

1. SELECT	<select list>
2. FROM	<table source>
3. WHERE	<search condition>
4. GROUP BY	<group by list>
5. HAVING	<search condition>
6. ORDER BY	<order by list>

1. FROM	<table source>
2. WHERE	<search condition>
3. GROUP BY	<group by list>
4. HAVING	<search condition>
5. SELECT	<select list>
6. ORDER BY	<order by list>

The order how the query is logically evaluated is different from the order on which is written.

The Impact of Logical Processing Order on Aliases

- FROM, WHERE, and HAVING clauses processed before SELECT
- Aliases created in SELECT clause only visible to ORDER BY
- Expressions aliased in FROM clause may be repeated elsewhere in query

Example 1 (not working):

```
SELECT orderid, unitprice, qty AS quantity
FROM Sales.OrderDetails;
WHERE quantity = 5  -- error, alias unknown because order of execution
```

Example 2 (working):

```
SELECT orderdate, SalesOrders.custid, SO.orderdate
FROM SalesOrders AS SO;
```

Applying the Logical Order of Operations to Writing SELECT Statements

```
USE TSQL;

SELECT EmployeeId, YEAR(OrderDate) AS OrderYear
FROM Sales.Orders
WHERE CustomerId = 71
GROUP BY EmployeeId, YEAR(OrderDate)
HAVING COUNT(*) > 1
ORDER BY EmployeeId, OrderYear;
```

Demonstration: Querying a SQL Server Database

In this demonstration, you will see how to:

- Use a SELECT statement to return all rows and columns from a table
- Use a SELECT statement to return all rows and specific columns from a table
- Use a WHERE clause to filter the rows that a SELECT statement returns
- Use different operators in a WHERE clause
- Use column and table aliases

Lesson 3: Adding Data to Tables

- Using INSERT to Add Data
- Using INSERT with Data Providers
- Using SELECT INTO
- Demonstration: Adding Data to Tables

Using INSERT to Add Data

- The INSERT … VALUES statement inserts a new row

```
INSERT INTO Sales.OrderDetails (orderid, productid, unitprice, qty, discount)
VALUES    (10255,39,18,2,0.05);
```

- Table and row constructors add multirow capability to INSERT … VALUES

```
INSERT INTO Sales.OrderDetails (orderid, productid, unitprice, qty, discount)
VALUES
(10256,39,18,2,0.05),
(10258,39,18,5,0.10);
```

Using INSERT with Data Providers

- INSERT … SELECT to insert rows from another table:

```
INSERT INTO Sales.OrderDetails (orderid, productid, unitprice, qty, discount)
SELECT * FROM NewOrderDetails
```

Demonstration: Adding Data to Tables

In this demonstration, you will see how to:

- Add data to a table using the INSERT statement
- Use the OUTPUT keyword with INSERT
- Use stored procedure output to insert data into a table
- Use SELECT INTO for populating a table with data and create the table structure at the same time

Lesson 4: Modifying and Removing Data

- Using UPDATE to Modify Data

- Demonstration: Manipulating Data Using the UPDATE and DELETE Statements and MERGING Data Using Conditional DML

Using UPDATE to Modify Data

- UPDATE changes all rows in a table or view
- Unless rows are filtered with a WHERE clause or constrained with a JOIN clause
- Column values are changed with the SET clause

```
UPDATE Production.Products
   SET   unitprice = (unitprice * 1.04)
WHERE   categoryid =  1 AND discontinued = 0
;
```

```
UPDATE Production.Products
   SET     unitprice *= 1.04
                            -- Using compound
                            -- assignment operators
WHERE   categoryid =  1 AND discontinued = 0;
```

Demonstration: Manipulating Data Using the UPDATE and DELETE Statements and MERGING Data Using Conditional DML

In this demonstration, you will see how to:

- UPDATE row, column intersections within tables
- DELETE complete rows from within tables
- Apply multiple data manipulation language (DML) operations by using the MERGE statement
- Understand how to use the OUTPUT clause to monitor data changes during DML operations
- Understand how to access prior and current data elements, in addition to showing the DML operation performed

Lab: Introduction to T-SQL Querying

- Exercise 1: Working with SELECT
- Exercise 2: Working with INSERT
- Exercise 3: Working with UPDATE
- Exercise 4: Working with DELETE

Download the **Exercises.sql** file from **Lab02\Starter** folder from GitHub

Module Review and Takeaways

- Review Question(s)

Module 3

Designing and Implementing Tables (DDL)

Module Overview

- Designing Tables
- Entity Relationship Modelling
- Data Types
- Working with Schemas
- Creating and Altering Tables

Lesson 1: Designing Tables

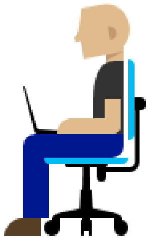
- What Is a Table?
- Normalizing Data
- Common Normalization Forms
- Demonstration: Working with Normalization

What Is a Table?

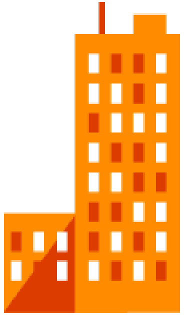
- Relational databases store data in tables (relations)
  - Defined by a collection of columns (identified by name)
  - Contain zero or more rows
- Tables typically represent a type of object or entity
  - Employees, purchase orders, customers, and sales orders are examples of entities
  - Consistent naming convention for tables is important
- Tables are a security boundary
- Each row usually represents a single instance of the object or entity
  - One employee, or one purchase order, for example
  - Rows of tables have no order

Entities

- Types of Entities

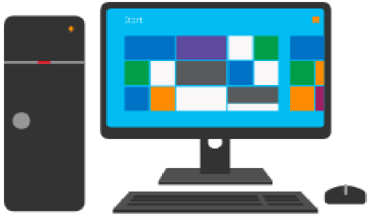


Person



Place

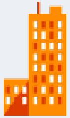
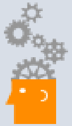
Event



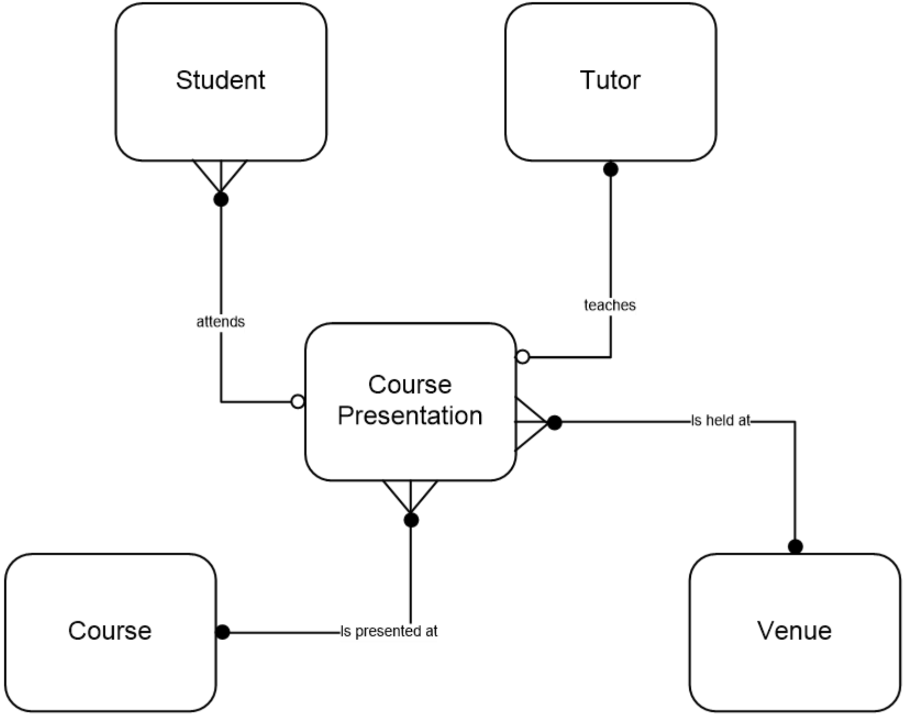
Thing

Attributes and Keys

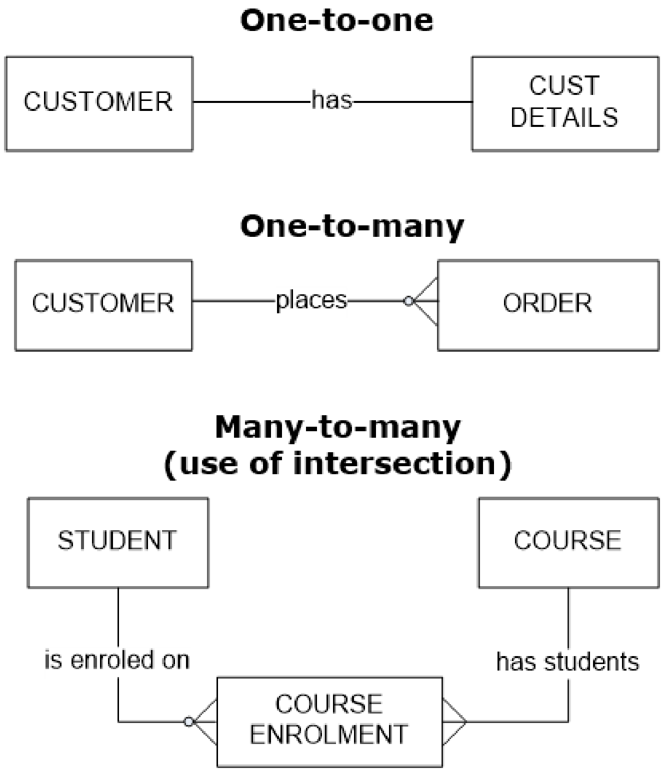
- Typical attributes for entities

Entity	Typical Attributes
 **Person**	• First name • Last name • Gender • Date of birth
 **Place**	• Type of place • Street address • City • Telephone number
 **Event**	• Type of event • Date of event • Time of event • Contact person
 **Thing**	• Type of thing • Manufacturer • Serial number • Location

Relationships



Types of Relationships



Primary Keys

- The primary key uniquely identifies each row within a table
- Candidate key could be used to uniquely identify a row
  - Must be unique and cannot be NULL (unknown)
  - Can involve multiple columns
  - Should not change
  - Primary key is one candidate key
  - Most tables will only have a single candidate key
- Debate surrounding natural vs. surrogate keys
  - Natural key: formed from data related to the entity
  - Surrogate key: usually codes or numbers

Foreign Keys

- Foreign keys are references between tables:
  - Foreign key in one table holds the primary key from another table
  - Self-references are permitted
- Rows that do not exist in the referenced table cannot be inserted in a referencing table
- Rows cannot be deleted or updated without cascading options
- Multiple foreign keys can exist in one table

Creating Tables (DDL)

- Tables are created using the CREATE TABLE statement
- Specify column names and data types
- Specify NULL or NOT NULL
- Specify the primary key

```
CREATE TABLE LineItems(
       OrderID int NOT NULL,
       ProductID int NOT NULL,
       UnitPrice money NOT NULL,
       Quantity smallint NOT NULL,
 CONSTRAINT PK_LineItems PRIMARY KEY
([OrderID], [ProductID]));
GO
```

Dropping Tables (DDL)

- Tables are removed by using the DROP TABLE statement
- Reference tables (via foreign keys) cannot be dropped
- All permissions, constraints, indexes, and triggers are also dropped
- Code that references the table, such as a stored procedure, is not dropped

DROP TABLE dbo.Persons

Altering Tables (DDL)

- Use the ALTER TABLE statement to modify tables
- ALTER TABLE retains permissions to the table
- ALTER TABLE retains the data in the table
- ALTER TABLE is used to:
  - Add or drop columns and constraints
  - Enable or disable constraints and triggers

Introduction to Data Types

OrderID and **ProductID** columns will contain integer data

UnitPrice column will contain money data

```
CREATE TABLE LineItems(
       OrderID int NOT NULL,
       ProductID int NOT NULL,
       UnitPrice money NOT NULL,
       Quantity smallint NOT NULL,
 CONSTRAINT PK_LineItems PRIMARY KEY
([OrderID], [ProductID]));
GO
```

Quantity column will contain small integer data

Introduction to Data Types

- Data types determine what can be stored
  - Constrain the type of data that an object can hold and the permitted operations
  - Provide limits on the range of values
- Data types apply to database columns, variables, expressions, and parameters
- Critical to choose appropriate data types
  - Assist with query optimization
  - Provide a level of self-documentation
- Three basic sets of data types
  - System data types
  - Alias data types
  - User-defined data types

Exact Numeric Data Types

Data Type	Range	Storage (bytes)
tinyint	0 to 255	1
smallint	-32,768 to 32,768	2
int	2^{31} (-2,147,483,648) to 2^{31}-1 (2,147,483,647)	4
bigint	-2^{63} - 2^{63}-1 (+/- 9 quintillion)	8
bit	1, 0 or NULL	1
decimal/numeric	-10^{38} +1 through 10^{38} – 1 when maximum precision is used	5-17
money	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	8
smallmoney	-214,748.3648 to 214,748.3647	4

Date and Time Data Types

- Rich set of options is available for storing date and time data
- ISO standard date formats remove ambiguity in date formats, for example 2016-06-01
- Large set of functions available for processing date and time data types

Data Type	Storage (bytes)	Date Range (Gregorian Calendar)	Accuracy	Recommended Entry Format
datetime	8	January 1, 1753 to December 31, 9999	Rounded to increments of .000, .003, or .007 seconds	YYYYMMDD hh:mm:ss[.mmm]
smalldatetime	4	January 1, 1900 to June 6, 2079	1 minute	YYYYMMDD hh:mm:ss[.mmm]
datetime2	6 to 8	January 1, 0001 to December 31, 9999	100 nanoseconds	YYYYMMDD hh:mm:ss[.nnnnnnn]
date	3	January 1, 0001 to December 31, 9999	1 day	YYYY-MM-DD
time	3 to 5	n/a – time only	100 nanoseconds	hh:mm:ss[.nnnnnnn]

Character String Data Types

- Non-Unicode character string data types:
  - char (n)
  - varchar (n)
  - varchar (max)
- Unicode character string types:
  - nchar (n)
  - nvarchar (n)

Other Data Types

Data Type	Range	Storage (bytes)	Remarks
xml	0-2 GB	0-2 GB	Stores XML in native hierarchical structure
uniqueidentifier	Auto-generated	16	Globally unique identifier (GUID)
hierarchyid	n/a	Depends on content	Represents position in a hierarchy
rowversion	Auto-generated	8	Previously called timestamp
geometry	0-2 GB	0-2 GB	Shape definitions in Euclidian geometry
geography	0-2 GB	0-2 GB	Shape definitions in round-earth geometry
sql_variant	0-8000 bytes	Depends on content	Can store data of various other data types in the same column
cursor	n/a	n/a	Not a storage datatype—used for cursor operations
table	n/a	n/a	Not a storage data type—used for query operations

T-SQL Language Elements: Functions

String Functions	Date and Time Functions	Aggregate Functions
• SUBSTRING • LEFT, RIGHT • LEN • REPLACE • REPLICATE • UPPER, LOWER • LTRIM, RTRIM • STUFF • SOUNDEX	• GETDATE • SYSDATETIME • GETUTCDATE • DATEADD • DATEDIFF • YEAR • MONTH • DAY • DATENAME • DATEPART • ISDATE	• SUM • MIN • MAX • AVG • COUNT • COUNT_BIG • STDEV • STDEVP • VAR

Keys and Constraints

- PRIMARY KEY constraints ensure uniqueness
- FOREIGN KEY constraints maintain referential integrity
- UNIQUE constraints ensure uniqueness for non-primary key columns
- DEFAULT constraints provide a default value
- CHECK constraints check values against defined criteria

Introduction to Data Types

OrderID and **ProductID** columns will contain integer data

UnitPrice column will contain money data

```
CREATE TABLE LineItems(
        OrderID int NOT NULL,
        ProductID int NOT NULL,
        UnitPrice money NOT NULL,
        Quantity smallint NOT NULL,
 CONSTRAINT PK_LineItems PRIMARY KEY
([OrderID], [ProductID]));
GO
```

Quantity column will contain small integer data

Demonstration: Creating Databases and Tables

In this demonstration, you will see how to:

- Create a database
- Create tables
- Create a CHECK constraint
- Create a DEFAULT constraint
- Create a UNIQUE constraint

Normalizing Data

- Normalization is a process
  - Ensures that database structures are appropriate
  - Ensures that poor design characteristics are avoided
- Edgar F. Codd invented the relational model
  - Introduced the concept of normalization
  - Referred to the degrees of normalization as forms
- Database designs should initially be normalized
  - Denormalization might be applied later to improve performance or to make analysis of data more straightforward

Common Normalization Forms

- First Normal Form
  - Eliminate repeating groups in individual tables
  - Create a separate table for each set of related data
  - Identify each set of related data by using a primary key
- Second Normal Form
  - Non-key columns should not be dependent on only part of a primary key
  - These columns should be in a separate table and related by using a foreign key
- Third Normal Form
  - Eliminate fields that do not depend on the key

Demonstration: Working with Normalization

- In this demonstration, you will see how to alter a table to conform to third normal form

Options for Enforcing Data Integrity

- Data type: defines the type of data that can be stored in a column
- Nullability: determines whether a value must be present in a column
- Constraints: defines rules that limit the values that can be stored in a column, or how values in different columns must be related; also default values
- Triggers: define code that is executed automatically when data in a table is modified

Module Review and Takeaways

- Review Question(s)
- Real-world Issues and Scenarios
- Tools
- Best Practice
- Common Issues and Troubleshooting Tips

Lab: Creating Databases and Tables

In this demonstration, you will see how to:

- Create a database
- Create tables
- Create a CHECK constraint
- Create a DEFAULT constraint
- Create a UNIQUE constraint

Module 4

Querying Multiple Tables

Module Overview

- Understanding Joins
- Querying with Inner Joins
- Querying with Outer Joins
- Querying with Cross Joins and Self Joins

Lesson 1: Understanding Joins

- The FROM Clause and Virtual Tables
- Join Terminology: Cartesian Product
- Overview of Join Types
- T-SQL Syntax Choices
- Demonstration: Understanding Joins

The FROM Clause and Virtual Tables

- FROM clause determines source tables to be used in SELECT statement
- FROM clause can contain tables and operators
- Result set of FROM clause is virtual table
  - Subsequent logical operations in SELECT statement consume this virtual table
- FROM clause can establish table aliases for use by subsequent phases of query

Join Terminology: Cartesian Product

- Characteristics of a Cartesian product
  - Output or intermediate result of FROM clause
  - Combine all possible combinations of two sets
  - In T-SQL queries, usually not desired
    - Special case: table of numbers

Name
Davis
Funk
King

×

Product
Alice Mutton
Crab Meat
Ipoh Coffee

=

Name	Product
Davis	Alice Mutton
Davis	Crab Meat
Davis	Ipoh Coffee
Funk	Alice Mutton
Funk	Crab Meat
Funk	Ipoh Coffee
King	Alice Mutton
King	Crab Meat
King	Ipoh Coffee

Overview of Join Types

- Join types in FROM clauses specify the operations performed on the virtual table:

Join Type	Description
Cross	Combines all rows in both tables (creates Cartesian product)
Inner	Starts with Cartesian product; applies filter to match rows between tables based on predicate
Outer	Starts with Cartesian product; all rows from designated table preserved, matching rows from other table retrieved. Additional NULLs inserted as placeholders

T-SQL Syntax Choices

- ANSI SQL-92
  - Tables joined by JOIN operator in FROM Clause

```
SELECT ...
FROM   Table1 JOIN Table2
       ON <on_predicate>
```

- ANSI SQL-89
  - Tables joined by commas in FROM Clause
    - Not recommended: accidental Cartesian products!

```
SELECT ...
FROM   Table1, Table2
WHERE  <where_predicate>
```

Lesson 2: Querying with Inner Joins

- Understanding Inner Joins
- Inner Join Syntax
- Inner Join Examples
- Demonstration: Querying with Inner Joins

Understanding Inner Joins

- Returns only rows where a match is found in both input tables
- Matches rows based on attributes supplied in predicate
  - ON clause in SQL-92 syntax (preferred)
  - WHERE clause in SQL-89 syntax
- Why filter in ON clause?
  - Logical separation between filtering for purposes of join and filtering results in WHERE
  - Typically no difference to query optimizer

Inner Join Syntax

- List tables in FROM Clause separated by JOIN operator
- Table aliases preferred
- Table order does not matter

```
FROM table1 AS t1 JOIN table2 AS t2
    ON t1.column = t2.column
```

```
SELECT o.orderid,
          o.orderdate,
          od.productid,
          od.unitprice,
          od.qty
FROM Sales.Orders AS o
        JOIN Sales.OrderDetails AS od
        ON o.orderid = od.orderid;
```

Inner Join Examples

- Join based on single matching attribute

```
SELECT ...
FROM Production.Categories AS C
    JOIN Production.Products AS P
        ON C.categoryid = P.categoryid;
```

- Join based on multiple matching attributes (composite join)

```
-- List cities and countries where both --
customers and employees live
SELECT DISTINCT e.city, e.country
FROM Sales.Customers AS c
JOIN     HR.Employees AS e
    ON c.city = e.city AND
    c.country = e.country;
```

Demonstration: Understanding Joins

In this demonstration, you will see how to:

- Use joins

Lab: Understanding Joins

In this demonstration, you will see how to:

- Use joins

Module 5

Other Database Objects

Module Overview

- Views
- Stored Procedures
- Triggers
- Functions

Lesson 2: Views

- What Is a View?
- Views and Security

What Is a View?

- A view is a stored SELECT statement
  - Views mask database complexity for end users
- Use the CREATE VIEW Transact-SQL statement to create views

```
CREATE VIEW VW_CustomerOrders
AS
SELECT C.CustomerID, Name, DateOFBirth, OrderID, OrderDate
FROM Person.Customer AS C
JOIN Sales.[Order] AS O
ON C.CustomerID = O.CustomerID;
GO
```

Views and Security

- End users only require permission on the view, not on the underlying tables
- Benefits of this approach include:
  - Users can only access columns that the view references
  - Reduces the risk of users exploring data that they should not access
  - Simplifies permission management for database administrators

Lesson 3: Stored Procedures, Triggers, and Functions

- Stored Procedures
- Benefits of Stored Procedures
- Input Parameters
- Demonstration: Creating and Using a Stored Procedure
- Triggers
- Functions

Stored Procedures

- Stored procedures encapsulate Transact-SQL statements such as INSERT, UPDATE, DELETE, and SELECT statements
- Use the CREATE PROCEDURE Transact-SQL statement to create stored procedures

```
CREATE PROCEDURE USP_Orders
AS
SELECT OrderID, OrderDate FROM Sales.[Order];
GO

EXEC USP_Orders;
GO
```

- System stored procedures enable the management of database objects and tasks

Benefits of Stored Procedures

- Modularization
- Performance
- Security
- Standardization and code reuse

Input Parameters

- Input parameters enable users to pass values to a stored procedure when they execute it
- Parameters have a data type

```
CREATE PROCEDURE USP_InsertCustomer
@CustomerID int
,@name varchar (50)
,@DateofBirth datetime
,@Address varchar (50)
AS
INSERT INTO Person.Customer
VALUES
(@CustomerID
,@name
,@DateOfBirth
,@Address);
GO
```

Demonstration: Creating and Using a Stored Procedure

In this demonstration, you will see how to:

- Create a parameterized stored procedure
- Execute a stored procedure by passing input parameter values to it
- Verify the actions performed by the stored procedure

Triggers

Waters
THE SCIENCE OF WHAT'S POSSIBLE.™

- DML Triggers
  - AFTER triggers
  - INSTEAD OF triggers
- DDL triggers
- Logon triggers
- Use a CREATE TRIGGER Transact-SQL statement to create triggers

Functions

- Performs a specific task
- Built-in function types:
  - Scalar functions – operate on a single value and return a single scalar result
  - Aggregate functions – operate on a range of values and return a single result
- User-defined function types:
  - Inline table-valued functions – return a table populated by a single SELECT statement
  - Multi-statement table-valued functions – return a table populated by the results of multiple statements

Module Review and Takeaways

- Review Question(s)

Module 6

Performance

Module Overview

- Indexing
- Query Performance
- Concurrency

Lesson 1: Indexing

- Clustered Index
- Nonclustered Index
- Distribution Statistics
- Demonstration: Testing Index Performance

Clustered Index

- Clustered indexes determine the logical order of rows within a table
  - Conceptually, a table with a clustered index is like a dictionary, whose terms are the index key
- Characteristics of clustered indexes:
  - A clustered index on a column causes a table to be stored with rows logically organized by that column's values
  - A clustered index is not a separate physical structure from the table—index data is stored with the table
  - One clustered index per table

Nonclustered Index

- Nonclustered indexes are separate structures with pointers back to the location of the data
  - Conceptually similar to a subject index printed at the back of a book
- Nonclustered indexes provide alternate ways to rapidly locate data
  - If a table's clustered index is on empid, a nonclustered index on last name may be useful for queries that use lastname in the predicate
- A table may have multiple nonclustered indexes
  - Adding nonclustered indexes adds to storage requirements for a database, and adds to processing time when data is updated

Demonstration: Testing Index Performance

- In this demonstration, you will see how to evaluate the performance impact of indexes

Module Review and Takeaways

- Best Practice