# SQL Server Specialist Certificate Program

Maintaining SQL Server 2005

Week 8 – SSIS Packages: In Detail / Flat Files.
Creating Partitions.
More TSQL.

Steve Stedman - Instructor

# This Weeks Overview

- Review from Last Week
- Class Project
- SSIS
  - BCP – Bulk Copy Program
  - Bulk Insert
  - Import Export Wizard
- Partitioning
- More TSQL
- Review and Homework

# Topics from last week

- Mirroring
- Log Shipping
- Replication

# Class Project

- Presentations will be next week
- You will have 5 to 10 minutes to present (more if needed)
- After the presentation there will be a question and answer time 5 to 10 minutes
- Presentations should include a hand out or a presentation that can be added to blackboard

# Preparation

We will be using the AdventureWorks database.

Start by creating a directory C:\Data

We will be using this in some of the examples

# Preparation – Step 2

## Select Into – Create an empty Table

```
USE AdventureWorks;
GO
SELECT *
INTO AdventureWorks.Sales.Currency2
FROM AdventureWorks.Sales.Currency
WHERE 1=2
```

# 1. SSIS
# Working with Flat Files

- SQL Server Integration Services (SSIS)
- Flat Files
- BCP (Importing and Exporting)
- BULK INSERT task
- SSIS Import/Export Wizard

# SQL Server Integration Services (SSIS)

- Available in SQL Server 2005 and 2008
- Replaces DTS (Data Transformation Services) from older SQL Server
- Can be used on all versions of SQL Server 2005 except Express and Workgroup.
- Primary Use – Import, Export and Data Warehousing

# Flat Files

- Plain Text
- Mixed Text and Binary
- One Record per Line or Record
- Contains delimiters, or fixed fields
- Simply a list

# BCP – Bulk Copy Program

- A command line tool
- Copies between SQL Server and a data file
- Can be use for import or for export
- Simplified Syntax
  - bcp {[[database_name.][owner].]{table_name | view_name} | "query"}  {in | out } data_file

# BCP Export – Trusted Connection

- Copying table rows into a data file
- Using the out option we can copy from a table out to a data file
- Using a trusted connection (-T) you don't have to specify a login

```
bcp AdventureWorks.Sales.Currency out
Currency.dat -T -c
```

# BCP Export – Mixed Mode Authentication

- You must use the **-U** switch to specify your login ID
- Use the **-S** switch to specify the system name and, optionally, an instance name

bcp AdventureWorks.Sales.Currency out Currency.dat -c -U<login_id> -S<server_name\instance_name>

# BCP Import

- Copy rows from the data file into the table
- You can use a trusted connection or mixed mode authentication like export

bcp AdventureWorks.Sales.Currency2 **in** Currency.dat -T -c

- Verify the import succeeded in SSMS

# BCP Export – Using a Query

- Could be use to pull data from one database to another

```
bcp "SELECT E.EmployeeID, E.LoginID FROM
[AdventureWorks].HumanResources.Employee E"
QUERYOUT c:\Data\hrEmployees.txt -T -C -t , -r \n
```

# More BCP

- No knowledge of TSQL required
- Useful if you need to load or unload large amount of data
- Import data for temporary processing
- You can turn off constraint checking for BCP imports

# Lab Project

- Export and Import/Copy a table using BCP

# BULK INSERT task

- Used to load a data file into the SQL Server
- The table must exist

- Simplified Syntax

```
BULK INSERT [ table_name | view_name ] FROM 'data_file'
```

# How to import CSV file into SQL Server? **BULK INSERT**

- See the BulkInsert.txt file for source

- Create a table to insert to
- Create the CSV file to use for insert
- Run the BULK INSERT statement
- Examine the table to see your results
- Delete the table for cleanup

# SSIS Import/Export Wizard

- Another method to Import and Export
- Import/Export Additional Formats
  - Access 2003
  - Excel 2003
  - Flat Files
  - Others
- Right click on a database… select Tasks, then select Import or Export.

# SSIS Import/Export Wizard Demo

- Walk through exporting the output of the following query to a text file:

```
SELECT E.EmployeeID, E.LoginID
FROM HumanResources.Employee E
```

- Copy a table with the Import/Export wizard

# Lab Project

- Use the Import/Export wizard to export the output of a query of your choice
- Copy from one table to another with the Import Export Wizard

- End of this section. Any Questions?
- 10 minute break

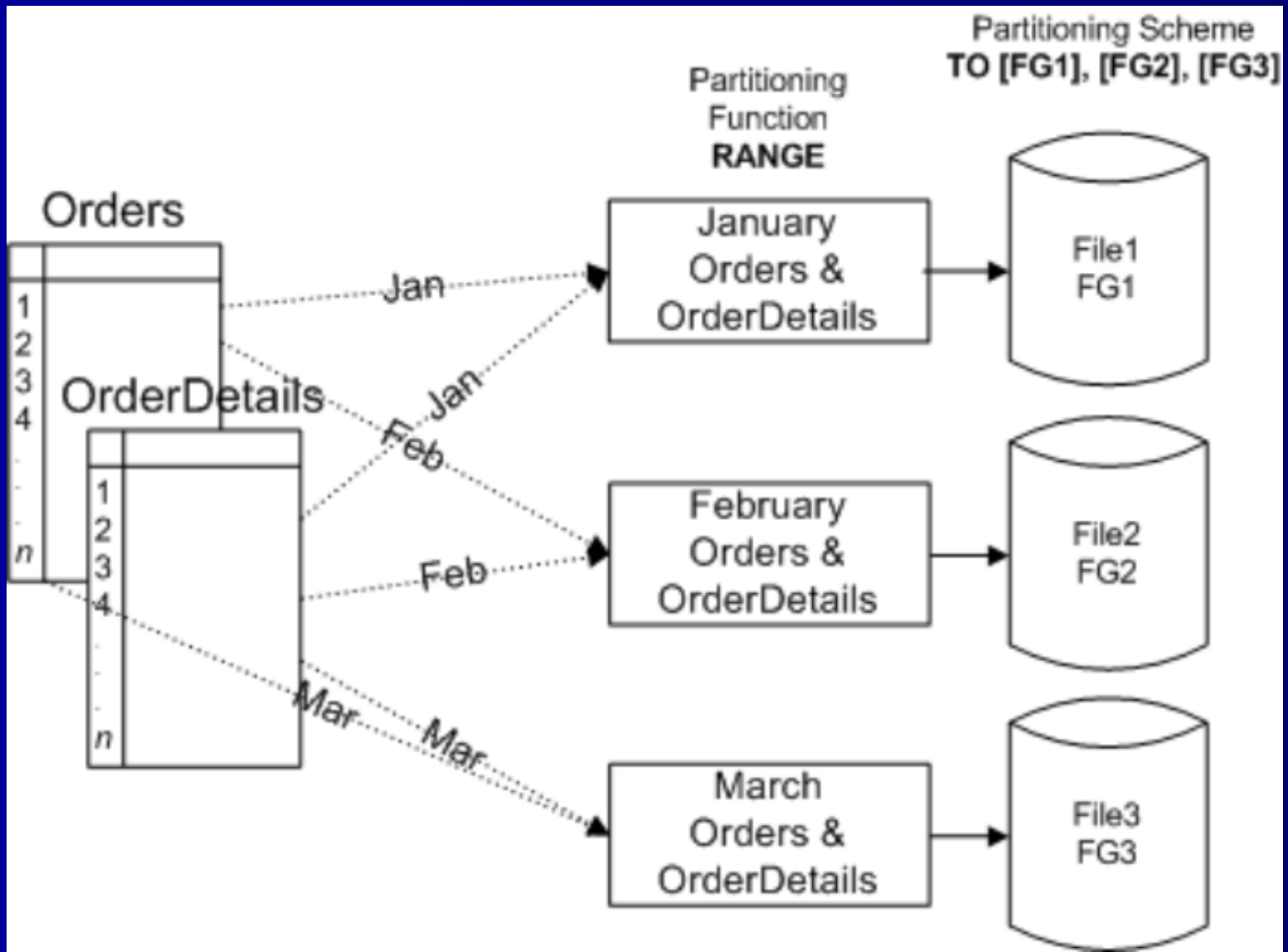
# 1. Partitions

- What are Partitions
- History
- Steps to Partition
- Managing Partitions

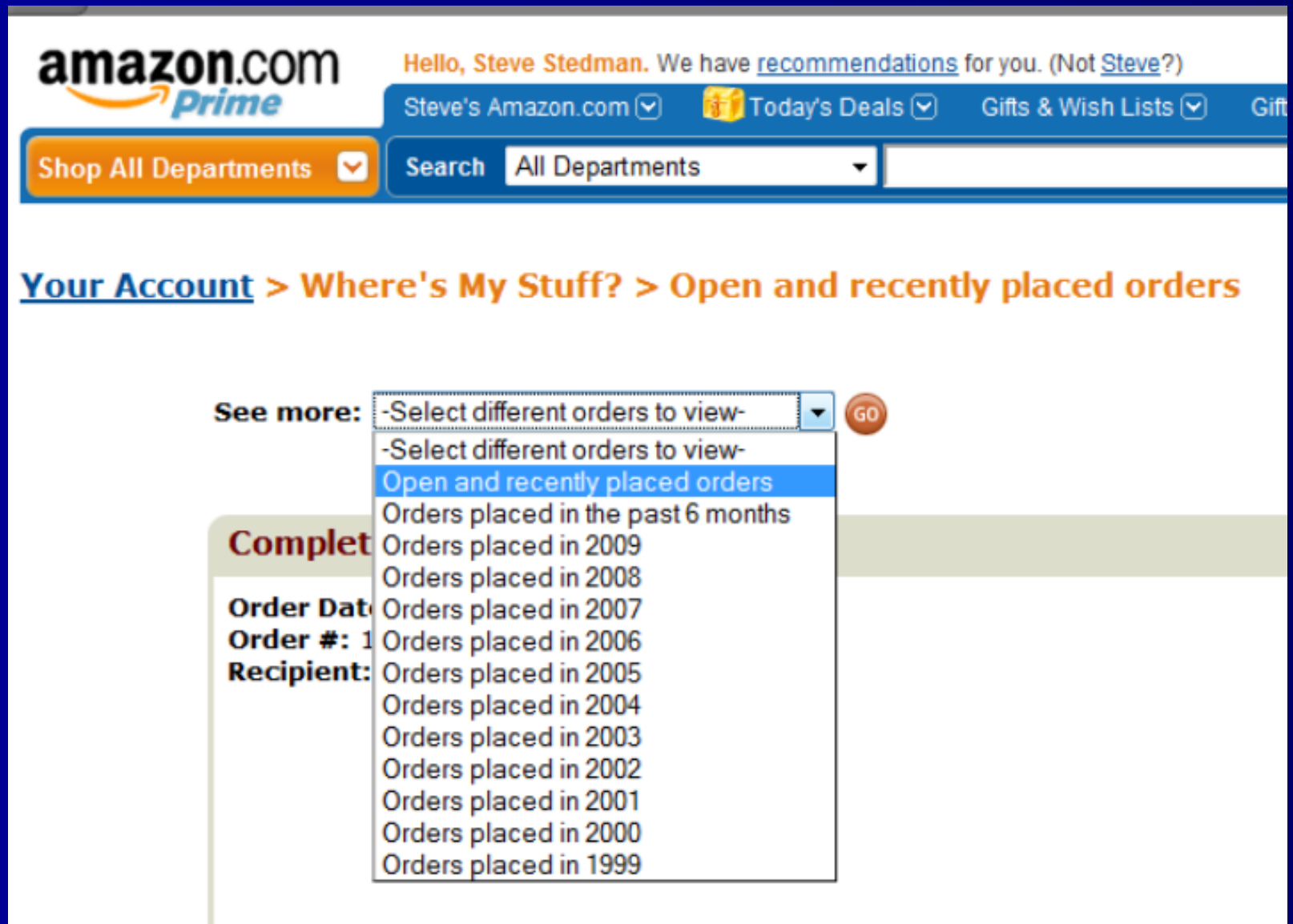
# What are Partitions

- Splitting tables across multiple file groups
- Used for massive tables to reduce the search time
- Allows for data segregation based on an data element

# Partitioning Example
# Partition orders by date



# Likely Partitioning at Amazon.com



# Partitions History

- New in SQL Server 2005 – Enterprise
- Also in SQL Server 2008
- Other databases have the same concept
  - Oracle – Partitioned Tables

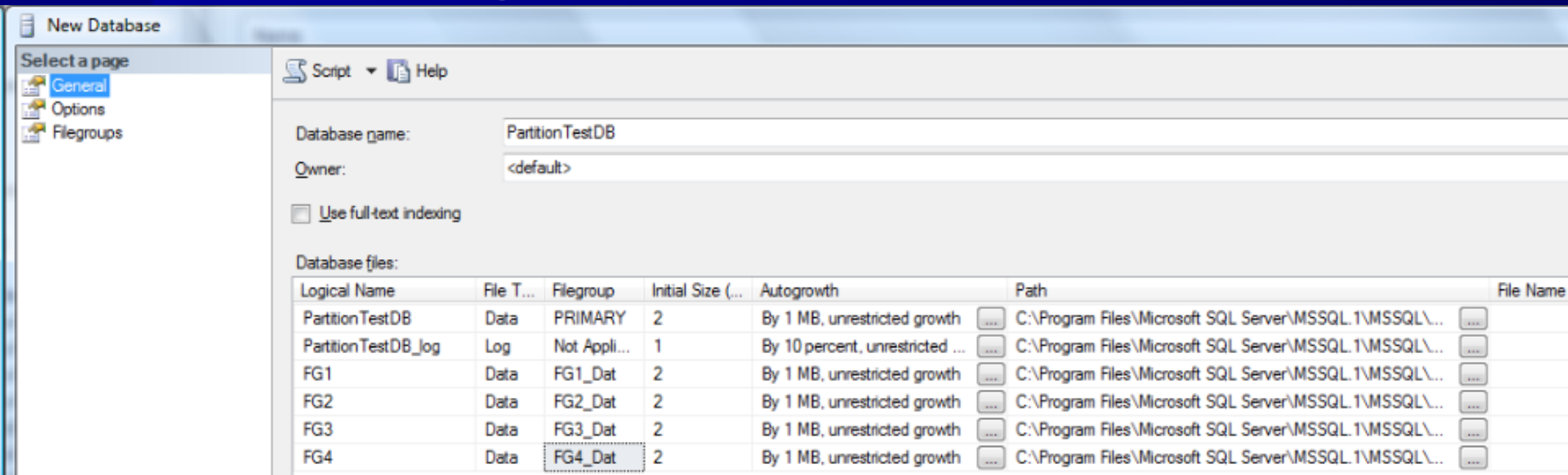
# Steps to Partition

- Create a database with multiple filegroups

- Create a partition function
- Create a partition scheme
- Create a table or index on the partition scheme

# Create a database with multiple filegroups

- You can use SSMS to create a partitioned database
- Set up multiple Filegroups when creating the database



# Create a partition function

CREATE PARTITION FUNCTION partition_function_name ( input_parameter_type ) AS RANGE [ LEFT | RIGHT ] FOR VALUES ( [ boundary_value [ ,...n ] ] ) [ ; ]

- LEFT or RIGHT ranges
  - Left is similar to less than (<)
  - Right is like greater than or equal to (>=)
- Simply a function to split up some values.
- No relation to a table, or to storage

# Sample: Create a partition function

- Assume that we will be partitioning on 5 digit zip codes stored in an int.

```
CREATE PARTITION FUNCTION partfunc (int) as
RANGE LEFT FOR VALUES (10000, 40000, 80000);

GO

SELECT * FROM sys.partition_range_values;
```

# Create a partition scheme

```
CREATE PARTITION SCHEME partition_scheme_name
AS PARTITION partition_function_name [ ALL ]
TO ( { file_group_name | [ PRIMARY ] } [ ,...n ] ) [ ; ]
```

- Specifies physical storage
- Just maps a function to a filegroup
- Not related to a table or index yet.

# Sample: Create a Partition Scheme

```
CREATE PARTITION SCHEME partscheme AS
PARTITION partfunc TO
([FG1_Dat], [FG2_Dat], [FG3_Dat], [FG4_Dat])
GO

SELECT * FROM sys.partition_schemes
GO
```

# Create a table on the partition scheme

CREATE TABLE [ database_name . [ schema_name ] . | schema_name . ] table_name

( { <column_definition> | <computed_column_definition> } [ <table_constraint> ] [ ,...n ] )

**[ ON { partition_scheme_name ( partition_column_name ) | filegroup | "default" } ]**

# Create an index on the partition scheme

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
ON <object> ( column [ ASC | DESC ] [ ,...n ] )
[ INCLUDE ( column_name [ ,...n ] ) ]
[ WITH ( <relational_index_option> [ ,...n ] ) ]
[ ON { partition_scheme_name ( column_name )
| filegroup_name
| default
}
]
[ ; ]
```

- Partitioning does not have to use the same columns in the index.

# Sample – Create the Partitioned Table

```
CREATE TABLE [dbo].[CustomerAddress](
[CustomerAddressID] [int] NOT NULL,
[City] [varchar](50) NULL,
[ZipCode] [int] NULL
) ON partscheme(ZipCode)
```

# Managing Partitions

- SPLIT
  - Splits a partition into 2
- MERGE
  - Combines 2 partitions into one
- SWITCH
  - Moves a partition from one table to another

# Partitioning Notes

- Developers and users don't need to know anything about the partitions to use them
- You can use the $partition variable to get specific partition information

```
select * from CustomerAddress
where $partition.partfunc(ZipCode) = 2

select *, $partition.partfunc(ZipCode)
from CustomerAddress
```

# Lab Project

- Create a new database, with multiple file groups
- Create a partitioning function
- Create a partitioning scheme
- Create a table using the partitioning scheme
- Insert into that table
- Display the rows in each partition

- End of this section. Any Questions?

# 1. Additional TSQL

- CASE statement
- Using sys.columns to find tables with a given column name
- Using self-join

# TSQL Case Statement

```
USE AdventureWorks;
GO
SELECT ProductNumber, Category =
CASE ProductLine
WHEN 'R' THEN 'Road'
WHEN 'M' THEN 'Mountain'
WHEN 'T' THEN 'Touring'
WHEN 'S' THEN 'Other sale items'
ELSE 'Not for sale'
END,
Name
FROM Production.Product
ORDER BY ProductNumber;
GO
```

# Using sys.columns to find tables with a given column name

```
USE AdventureWorks
GO
SELECT t.name AS table_name,
SCHEMA_NAME(schema_id) AS schema_name,
c.name AS column_name
FROM sys.tables AS t
INNER JOIN sys.columns c ON t.OBJECT_ID = c.
OBJECT_ID
WHERE c.name LIKE '%EmployeeID%'
ORDER BY schema_name, table_name
```

# Using self-join

- Self-join is a normal SQL join that joins one table to itself

```
USE AdventureWorks;
GO
SELECT DISTINCT pv1.ProductID, pv1.VendorID
FROM Purchasing.ProductVendor pv1
INNER JOIN Purchasing.ProductVendor pv2
ON pv1.ProductID = pv2.ProductID
AND pv1.VendorID = pv2.VendorID
ORDER BY pv1.ProductID
```

- End of this section. Any Questions?

# Homework

- Finish up your class projects

# Questions?