SQL Screwups

Abstract

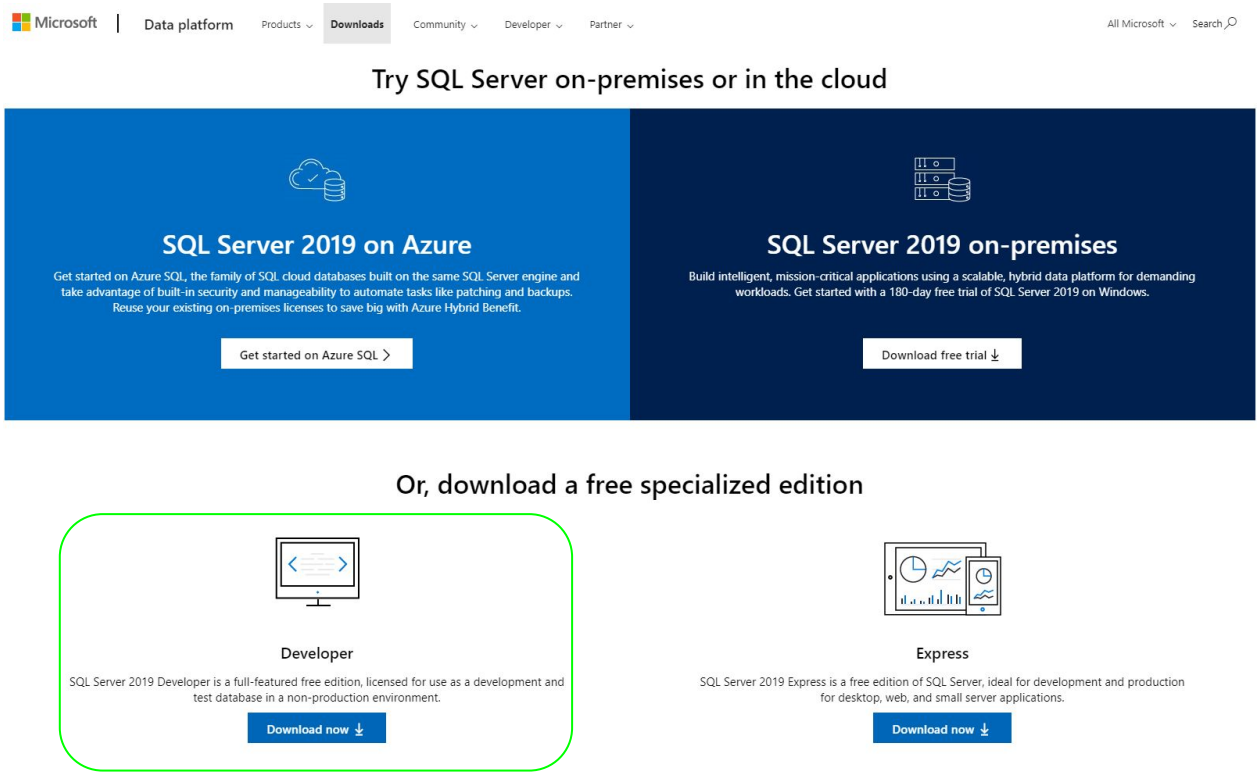
Basic features of Microsoft SQL Server can cause headaches if not used as intended. We'll explore several of the most basic SQL Server mistakes of my career so you can avoid them and write better SQL.

keep in touch

evansmith.dev

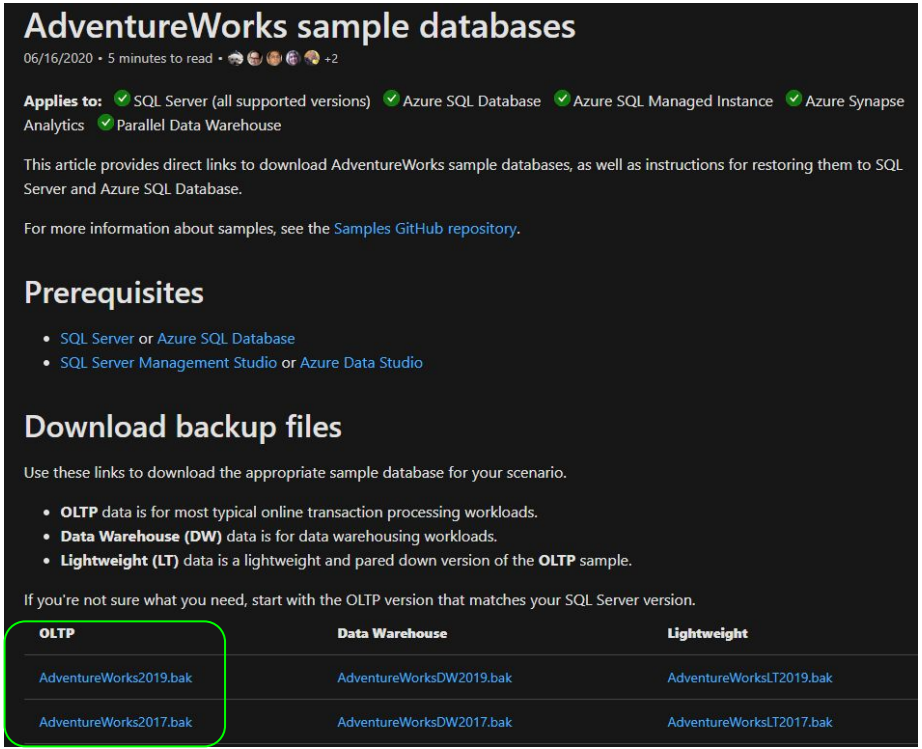
@evansmithdev

SQL Server Developer Edition



https://www.microsoft.com/en-us/sql-server/sql-server-downloads

Database Structure



AdventureWorks sample databases

06/16/2020 • 5 minutes to read • +2

Applies to: SQL Server (all supported versions) Azure SQL Database Azure SQL Managed Instance Azure Synapse Analytics Parallel Data Warehouse

This article provides direct links to download AdventureWorks sample databases, as well as instructions for restoring them to SQL Server and Azure SQL Database.

For more information about samples, see the Samples GitHub repository.

Prerequisites

- SQL Server or Azure SQL Database
- SQL Server Management Studio or Azure Data Studio

Download backup files

Use these links to download the appropriate sample database for your scenario.

- **OLTP** data is for most typical online transaction processing workloads.
- **Data Warehouse (DW)** data is for data warehousing workloads.
- **Lightweight (LT)** data is a lightweight and pared down version of the **OLTP** sample.

If you're not sure what you need, start with the OLTP version that matches your SQL Server version.

OLTP	Data Warehouse	Lightweight
AdventureWorks2019.bak	AdventureWorksDW2019.bak	AdventureWorksLT2019.bak
AdventureWorks2017.bak	AdventureWorksDW2017.bak	AdventureWorksLT2017.bak

https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15&tabs=ssms

Generate Sample Data

≡ Eric L. Anderson

Getting a Sample SQL Server Database

September 30, 2018 / SQL Server / Databases, SQL Server, SQL Server Management Studio, SSMS, Wide World Importers

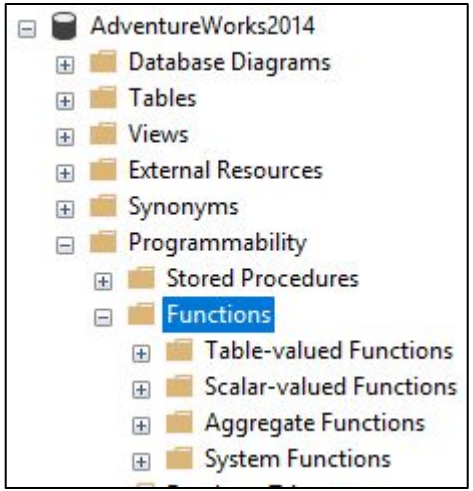
As tends to happen this isn't the post I set out to write today. We hit an issue at work the other day that I want to write about, but to do so I need a sample database (I can't use the data from work). I wanted something that is more fleshed out that my normal single table contact database. I google for AdventureWorks, since that is the sample database I have always seen in examples.

https://elanderson.net/2018/09/getting-a-sample-sql-server-database/

user-defined functions

- like functions in programming languages
- routines that accept parameters
- perform an action
- return the result of that action as a value
- return value can either be a single scalar value or a result set

*** not allowed to cause side effects (modify data)



https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/user-defined-functions?view=sql-server-ver15

01 scalar-valued functions

- user-defined function
- return value is a single scalar value
- examples:
    - CalculateExtendedPrice(quantity, unitPrice)
    - ConvertWaterFromGallonsToPounds(gallons)

Order Line Extended: Inline Calculation

```
SELECT SalesOrderDetailID
      ,OrderQty
      ,ProductID
      ,UnitPrice
      ,ROUND(OrderQty * UnitPrice, 2) as LineExtended
FROM Sales.SalesOrderDetail
WHERE SalesOrderID = 43659
ORDER BY SalesOrderID
        ,SalesOrderDetailID
```

Order Line Extended: Refactor As A Function

```
CREATE FUNCTION Sales.GetLineExtended
(
     @Quantity int
    ,@UnitPrice decimal(12,6)
)
RETURNS money
AS
BEGIN
    DECLARE @Extended money;
    SELECT @Extended = ROUND(@Quantity * @UnitPrice, 2);
    RETURN @Extended;
END
```

Order Line Extended: Call The Function

```
SELECT SalesOrderDetailID
      ,OrderQty
      ,ProductID
      ,UnitPrice
      ,Sales.GetLineExtended(OrderQty, UnitPrice) as LineExtended
FROM Sales.SalesOrderDetail
WHERE SalesOrderID = 43659
ORDER BY SalesOrderID
        ,SalesOrderDetailID;
```

Order SubTotal: Inline Calculation

```
SELECT Orders.Id
      ,Orders.Created
      ,Orders.Customer
      ,Customers.FirstName
      ,Customers.LastName
      ,SUM(ROUND(OrderLines.Quantity * Items.Price, 2)) as SubTotal
FROM Orders
INNER JOIN Customers ON
     Customers.Id = Orders.Customer
LEFT OUTER JOIN OrderLines ON
     OrderLines.OrderId = Orders.Id
LEFT OUTER JOIN Items ON
     Items.Sku = OrderLines.Sku
GROUP BY Orders.Id
     ,Orders.Created
     ,Orders,Customer
     ,Orders.FirstName
     ,Orders.LastName
```

Order SubTotal: Refactor As A Function

```
CREATE FUNCTION GetSubTotal(@orderId bigint)
    RETURNS decimal(17,2)
    AS
    BEGIN
        DECLARE @subtotal decimal(17,2)
        SELECT @subtotal = SUM(ROUND(OrderLines.Quantity * Items.Price, 2))
        FROM Orders
        LEFT OUTER JOIN OrderLines ON
                OrderLines.OrderId = Orders.Id
        LEFT OUTER JOIN Items ON
                Items.Sku = OrderLines.Sku
        WHERE Orders.Id = @orderId
        GROUP BY Orders.Id
        RETURN @subtotal
    END
```

Order SubTotal: Call The Function

```
SELECT Orders.Id
      ,Orders.Created
      ,Orders.Customer
      ,Customers.FirstName
      ,Customers.LastName
      ,GetSubTotal(Orders.Id) as SubTotal
FROM Orders
INNER JOIN Customers ON
       Customers.Id = Orders.Customer
```

NOTICE

- Simplified query
- Removed joins
- Removed group by

Order SubTotal: Call The Function

NOTICE

- Subquery
- Runs Per Row

```
SELECT Orders.Id
      ,Orders.Created
      ,Orders.Customer
      ,Customers.FirstName
      ,Customers.LastName
      ,(SELECT @subtotal = SUM(ROUND(OrderLines.Quantity * Items.Price, 2))
        FROM Orders
        LEFT OUTER JOIN OrderLines ON
                OrderLines.OrderId = Orders.Id
        LEFT OUTER JOIN Items ON
                Items.Sku = OrderLines.Sku
        WHERE Orders.Id = @orderId
        GROUP BY Orders.Id) as SubTotal
FROM Orders
INNER JOIN Customers ON
    Customers.Id = Orders.Customer
```

01 scalar-valued functions

- ➔ avoid querying additional data
- ➔ query plan estimator lies
- ➔ newer versions are improving

Google: slow scalar function

02 stored procedures

“a group of one or more Transact-SQL statements”

similar to a user-defined function

*** allowed to cause side effects (modify data)

https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver15

Get Sales: Inline

```
SELECT *
FROM Sales.SalesOrderHeader
WHERE OrderDate >= '2013-07-01'
  AND OrderDate <  '2014-07-01'
  AND OnlineOrderFlag = 1
  AND CustomerID = 11091
ORDER BY SalesOrderID;
```

Get Sales: Refactor As A Stored Procedure

NAME

```
CREATE PROCEDURE Sales.GetOrdersForCustomerInDateRange
```

INPUTS

```
    @OrderDateRangeBegin datetime2(7) = NULL
   ,@OrderDateRangeEnd   datetime2(7) = NULL
   ,@OnlineOrderFlag     bit          = 0
   ,@CustomerID          int          = NULL
AS
BEGIN
```

GUTS

```
    SELECT *
    FROM Sales.SalesOrderHeader
    WHERE OrderDate >= @OrderDateRangeBegin
      AND OrderDate <  @OrderDateRangeEnd
      AND OnlineOrderFlag = @OnlineOrderFlag
      AND CustomerID = @CustomerID
    ORDER BY SalesOrderID;
END
```

Get Sales: Call the Stored Procedure

```
EXEC Sales.GetOrdersForCustomerInDateRange
     @OrderDateRangeBegin = '2013-07-01'
    ,@OrderDateRangeEnd   = '2014-07-01'
    ,@OnlineOrderFlag     = 1
    ,@CustomerID          = 11091
```

Get Sales: Handle Defaults

```
CREATE PROCEDURE Sales.GetOrders
    @OrderDateRangeBegin  datetime2(7) = NULL
   ,@OrderDateRangeEnd    datetime2(7) = NULL
   ,@OnlineOrderFlag      bit          = NULL
   ,@CustomerID           int          = NULL
AS
BEGIN
    SELECT *
    FROM Sales.SalesOrderHeader
    WHERE OrderDate >= ISNULL(@OrderDateRangeBegin, '0001-01-01')
      AND OrderDate <  ISNULL(@OrderDateRangeEnd  , '9999-12-31')
      AND OnlineOrderFlag = ISNULL(@OnlineOrderFlag, OnlineOrderFlag)
      AND CustomerID = ISNULL(@CustomerID, CustomerID)
    ORDER BY SalesOrderID;
END
```

Get Sales: Call the Stored Procedure

```
-- get my online orders when I was 24
EXEC Sales.GetOrdersForCustomerInDateRange
     @OrderDateRangeBegin = '2013-07-01'
    ,@OrderDateRangeEnd   = '2014-07-01'
    ,@OnlineOrderFlag     = 1
    ,@CustomerID          = 11091

-- get all my orders when I was 24
EXEC Sales.GetOrdersForCustomerInDateRange
     @OrderDateRangeBegin = '2013-07-01'
    ,@OrderDateRangeEnd   = '2014-07-01'
    ,@CustomerID          = 11091
```

Get Sales: Call the Stored Procedure

```
-- get all online orders
EXEC Sales.GetOrdersForCustomerInDateRange @OnlineOrderFlag = 1
```

story time

- Greenfield (ish) development
- Notoriously archaic lead
- Prove we're modern
- Arbitrary decision to use SPs
- Problems ensued
  - Version control (alters, rather than drop/create)
  - Shared SQL repositories (repository pattern)
- Lead reassigned
- Evan takes over
- Rip out SPs

Simple Case: Get Order Data

```
SELECT *
FROM Orders
WHERE Orders.Id = '101'
```

Id	Created	Customer
101	2018-07-04	1

```
EXEC Sales.GetOrdersDynamic
'Sales.SalesOrderHeader.CustomerID IN
(SELECT Sales.Customer.CustomerID
 FROM Sales.Customer
 WHERE Sales.Customer.TerritoryID = 10)'
```

Needed to inject dynamic predicates provided by other areas of our code

```
CREATE PROCEDURE OrderDetailReport_Get
    @sqlFilter varchar(max)
    AS
    BEGIN
        DECLARE @query nvarchar(max)
        SET @query = 'SELECT * FROM Orders WHERE ' + @sqlFilter;
        execute sp_executesql @query
    END
GO

exec dbo.OrderDetailReport_Get 'Orders.Id = ''101'''
```

Id	Created	Customer
101	2018-07-04	1

Get Sales: More Realistic Simplified Example

```
CREATE PROCEDURE Sales.GetOrdersDynamic
    @OrderDateRangeBegin datetime2(7)   = NULL
   ,@OrderDateRangeEnd   datetime2(7)   = NULL
   ,@OnlineOrderFlag     bit            = NULL
   ,@CustomerID          int            = NULL
   ,@WildWestSql         nvarchar(max)  = NULL
AS
BEGIN
  DECLARE @query nvarchar(max) = N'
    SELECT *
    FROM Sales.SalesOrderHeader
    WHERE OrderDate >= ''' + CONVERT(nvarchar, ISNULL(@OrderDateRangeBegin, N'1973-01-01'), 120) + N'''
      AND OrderDate <  ''' + CONVERT(nvarchar, ISNULL(@OrderDateRangeEnd  , N'9999-12-31'), 120) + N'''
      AND OnlineOrderFlag = ISNULL('+ CASE WHEN @OnlineOrderFlag IS NULL THEN N'NULL' WHEN @OnlineOrderFlag = 0 THEN
N'0' ELSE N'1' END + N', OnlineOrderFlag)
      AND CustomerID = ISNULL('+ CASE WHEN @CustomerID IS NULL THEN N'NULL' ELSE CONVERT(nvarchar, @CustomerID) END +
N', CustomerID)';

  IF @WildWestSql IS NOT NULL
    SET @query = @query + N' AND ' + @WildWestSql + N' ';

  SET @query = @query + N'ORDER BY SalesOrderID;'
  exec(@query);
END
```

Execution Plans: Dynamic Predicates

```
CREATE PROCEDURE Sales.GetOrdersDynamic
    @OrderDateRangeBegin datetime2(7)   = NULL
   ,@OrderDateRangeEnd   datetime2(7)   = NULL
   ,@OnlineOrderFlag       bit           = NULL
   ,@CustomerID            int           = NULL
   ,@WildWestSql           nvarchar(max) = NULL
AS
BEGIN
  DECLARE @query nvarchar(max) = N'
    SELECT *
    FROM Sales.SalesOrderHeader
    WHERE 1=1'
  IF @OrderDateRangeBegin IS NOT NULL
    @query = @query + N'
      AND OrderDate >= ''' + CONVERT(nvarchar, ISNULL(@OrderDateRangeBegin, N'1973-01-01'), 120) + N''''

  IF @OrderDateRangeEnd IS NOT NULL
    @query = @query + N'
      AND OrderDate <  ''' + CONVERT(nvarchar, ISNULL(@OrderDateRangeEnd  , N'9999-12-31'), 120) + N''''

  IF @OnlineOrderFlag IS NOT NULL
    @query = @query + N'
      AND OnlineOrderFlag = ISNULL(' + CASE WHEN @OnlineOrderFlag IS NULL THEN N'NULL' WHEN @OnlineOrderFlag = 0 THEN N'0' ELSE N'1'
END + N', OnlineOrderFlag)'

  IF @CustomerID IS NOT NULL
    @query = @query + N'
      AND CustomerID = ISNULL(' + CASE WHEN @CustomerID IS NULL THEN N'NULL' ELSE CONVERT(nvarchar, @CustomerID) END + N',
CustomerID)';

  IF @WildWestSql IS NOT NULL
    @query = @query + N' AND ' + @WildWestSql + N' ';

  @query = @query + N'ORDER BY SalesOrderID;'
  exec(@query);
END
```

Just build the query that you need based on your BL and move on with your life.

```
SELECT *
FROM Orders
WHERE Orders.Id = '101'
```

Id	Created	Customer
101	2018-07-04	1

02 stored procedures

- ➔ not a silver bullet
- ➔ have a reason for your decision
- ➔ evaluate the impact
- ➔ use stored procedures where appropriate

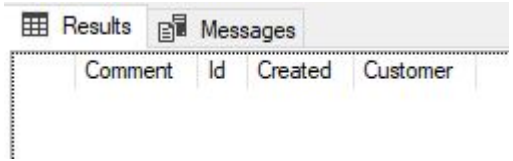
Google: when should I use stored procedures

03 null

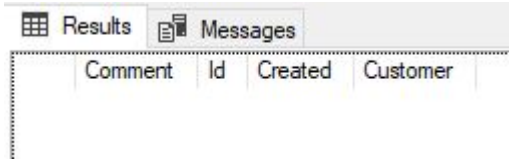
null is not equal to anything

null is not even equal to null

```sql
-- get orders with no comment
SELECT Comment, *
FROM Sales.SalesOrderHeader
WHERE Comment = NULL
```



```
-- get orders with no comment
SELECT Comment, *
FROM Sales.SalesOrderHeader
WHERE Comment <> NULL
```



```
-- get orders with no comment
SELECT Comment, *
FROM Sales.SalesOrderHeader
WHERE Comment IS NOT NULL
```

Id	Created	Customer
100	2018-07-01	1
101	2018-07-04	1
102	2018-07-05	1
103	2018-07-04	2
104	2018-07-01	3
105	2018-07-04	4
106	2018-07-06	4
107	2018-07-12	4

1

```sql
-- null is not same as ''
SELECT Comment, *
FROM Sales.SalesOrderHeader
WHERE Comment = ''
```

2

```sql
-- handle both
SELECT Comment, *
FROM Sales.SalesOrderHeader
WHERE ISNULL(Comment, '') <> ''
```

3

```sql
SELECT Comment, *
FROM Sales.SalesOrderHeader
WHERE NULLIF(Comment, '') IS NOT NULL
```

be mindful

```
-- Syntax for SQL Server

SET ANSI_NULLS { ON | OFF }
```

```
-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse

SET ANSI_NULLS ON
```

https://docs.microsoft.com/en-us/sql/t-sql/statements/set-ansi-nulls-transact-sql?view=sql-server-ver15

04 **filter join columns**

outer join

filter based on a column from the joined table

any column from a joined table can be null, even if it is defined as not null

filter on joined columns

```
-- report customers available for salesperson 281 to engage (cold calls,
follow-ups, etc)
SELECT *
FROM Sales.Customer
LEFT OUTER JOIN Sales.Store ON
        Store.BusinessEntityID = Customer.StoreID
WHERE Store.SalesPersonID = 281

-- report customers and total amount of orders before 2014
SELECT Customer.CustomerID, SUM(SalesOrderHeader.TotalDue) as Total
FROM Sales.Customer
LEFT OUTER JOIN Sales.SalesOrderHeader ON
        SalesOrderHeader.CustomerID = Customer.CustomerID
WHERE OrderDate < '2014'
GROUP BY Customer.CustomerID
```

filter on joined columns

```
-- salesperson 281 can also engage customers who do not belong to a store,
though
SELECT *
FROM Sales.Customer
LEFT OUTER JOIN Sales.Store ON
        Store.BusinessEntityID = Customer.StoreID
WHERE ISNULL(Store.SalesPersonID, 281) = 281

-- report customers and total amount of orders before 2014
SELECT Customer.CustomerID, SUM(SalesOrderHeader.TotalDue) as Total
FROM Sales.Customer
LEFT OUTER JOIN Sales.SalesOrderHeader ON
        SalesOrderHeader.CustomerID = Customer.CustomerID
    AND OrderDate < '2014'
GROUP BY Customer.CustomerID
```

05 type mismatches

```
SELECT *
FROM Person.Address
WHERE City = 'Nashville'
```

get help

```
exec sp_help 'Person.Address'
```

	Name	Owner	Type	Created_datetime
1	Address	dbo	user table	2017-10-27 14:33:01.697

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	AddressID	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2	AddressLine1	nvarchar	no	120			no	(n/a)	(n/a)	SQL_Latin1_General_CP1_CI_AS
3	AddressLine2	nvarchar	no	120			yes	(n/a)	(n/a)	SQL_Latin1_General_CP1_CI_AS
4	City	nvarchar	no	60			no	(n/a)	(n/a)	SQL_Latin1_General_CP1_CI_AS
5	StateProvinceID	int	no	4	10	0	no	(n/a)	(n/a)	NULL
6	PostalCode	nvarchar	no	30			no	(n/a)	(n/a)	SQL_Latin1_General_CP1_CI_AS
7	SpatialLocation	geogra...	no	-1			yes	(n/a)	yes	NULL
8	rowguid	uniquei...	no	16			no	(n/a)	(n/a)	NULL

	Identity	Seed	Increment	Not For Replication
1	AddressID	1	1	1

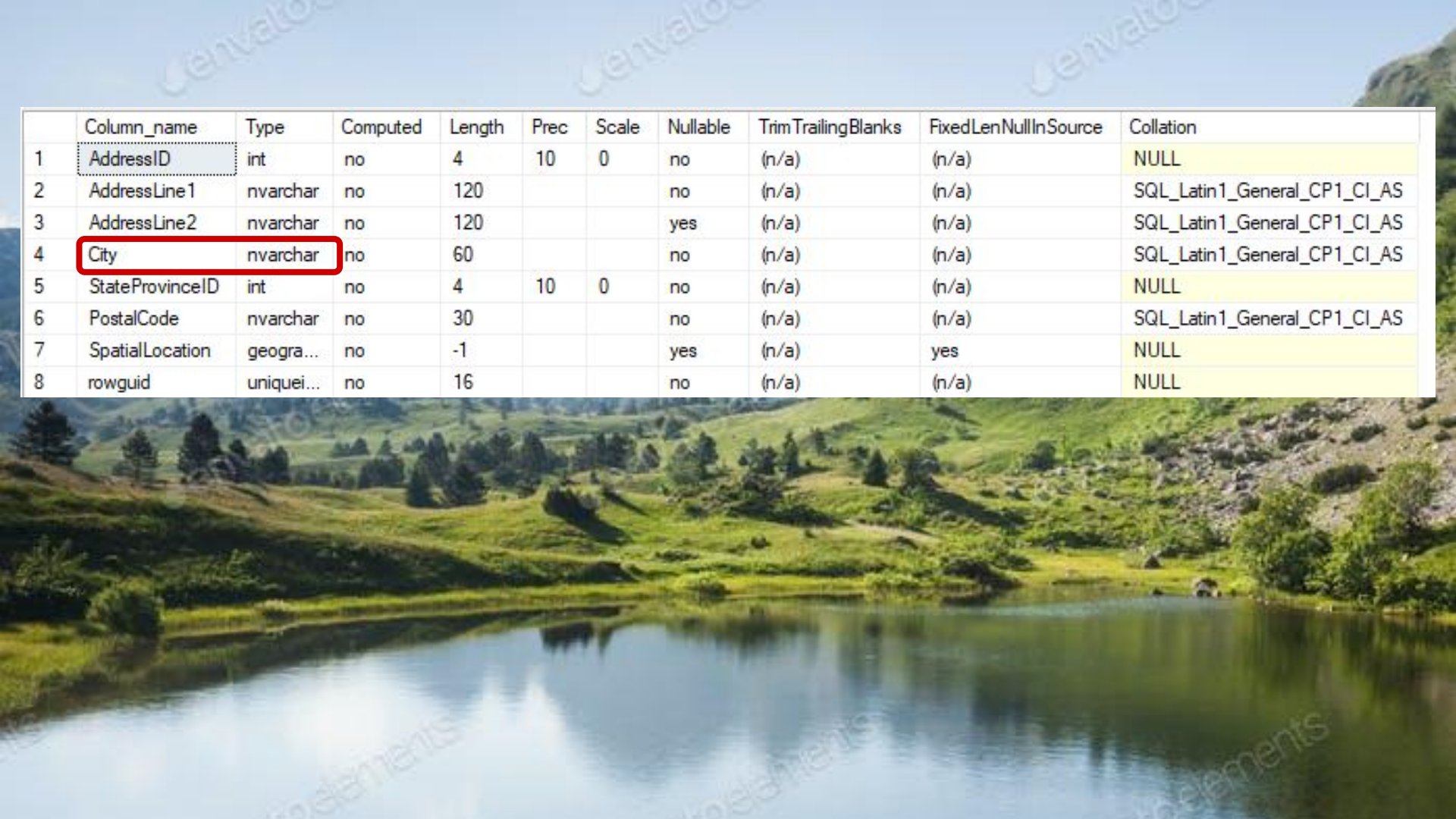
	RowGuidCol
1	rowguid

	Data_located_on_filegroup
1	PRIMARY

	index_name	index_description	index_keys
1	AK_Address_rowguid	nonclustered, unique located on PRIMARY	rowguid
2	IX_Address_AddressLine1_AddressLine2_City_StateP...	nonclustered, unique located on PRIMARY	AddressLine1, AddressLine2, City, StateProvinceI...
3	IX_Address_StateProvinceID	nonclustered located on PRIMARY	StateProvinceID
4	PK_Address_AddressID	clustered, unique, primary key located on ...	AddressID

	constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_keys
1	DEFAULT on column ModifiedDate	DF_Address_ModifiedDate	(n/a)	(n/a)	(n/a)	(n/a)	(getdate())
2	DEFAULT on column rowguid	DF_Address_rowguid	(n/a)	(n/a)	(n/a)	(n/a)	(newid())
3	FOREIGN KEY	FK_Address_StateProvin...	No Action	No Action	Enabled	Is_For_Replication	StateProvinc...
4							REFERENCE...
5	PRIMARY KEY (clustered)	PK_Address_AddressID	(n/a)	(n/a)	(n/a)	(n/a)	AddressID

	Table is referenced by foreign key
1	AdventureWorks2019.Person.BusinessEntityAddress: F...
2	AdventureWorks2019.Sales.SalesOrderHeader: FK_Sa...
3	AdventureWorks2019.Sales.SalesOrderHeader: FK_Sa...



	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	AddressID	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2	AddressLine1	nvarchar	no	120			no	(n/a)	(n/a)	SQL_Latin1_General_CP1_CI_AS
3	AddressLine2	nvarchar	no	120			yes	(n/a)	(n/a)	SQL_Latin1_General_CP1_CI_AS
4	City	nvarchar	no	60			no	(n/a)	(n/a)	SQL_Latin1_General_CP1_CI_AS
5	StateProvinceID	int	no	4	10	0	no	(n/a)	(n/a)	NULL
6	PostalCode	nvarchar	no	30			no	(n/a)	(n/a)	SQL_Latin1_General_CP1_CI_AS
7	SpatialLocation	geogra...	no	-1			yes	(n/a)	yes	NULL
8	rowguid	uniquei...	no	16			no	(n/a)	(n/a)	NULL

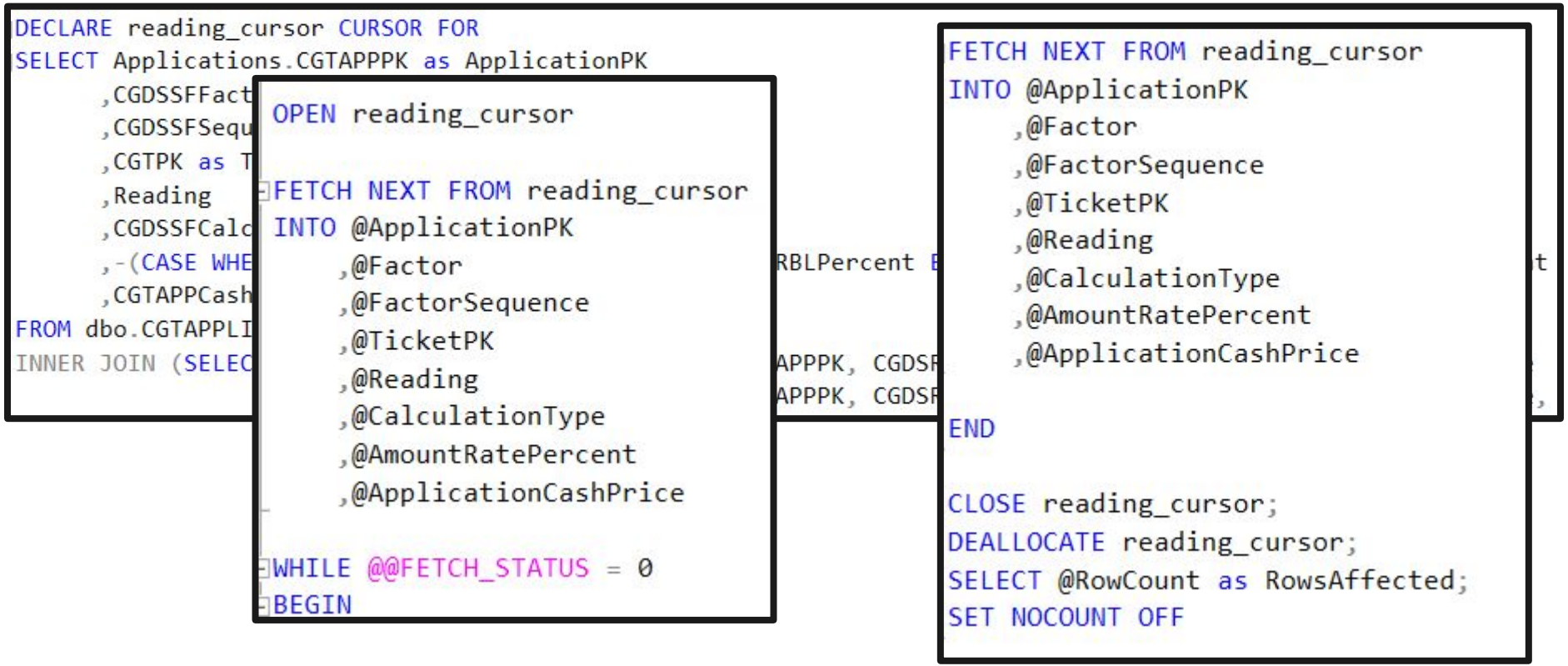
cursor and while
06

necessary in some scenarios

work with datasets, not individual rows

avoid if possible

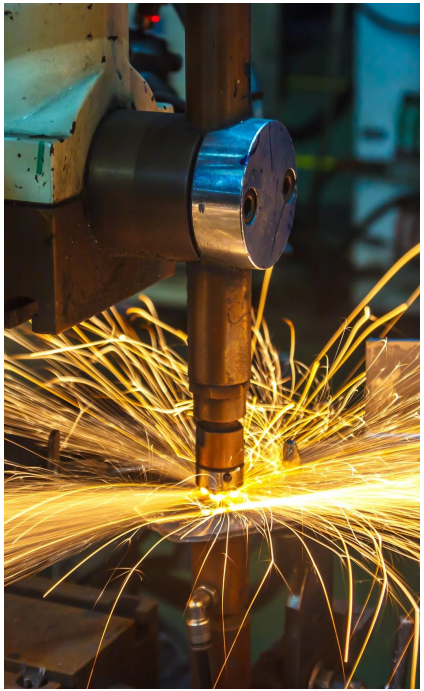
What in the world?!



07 sargable predicates

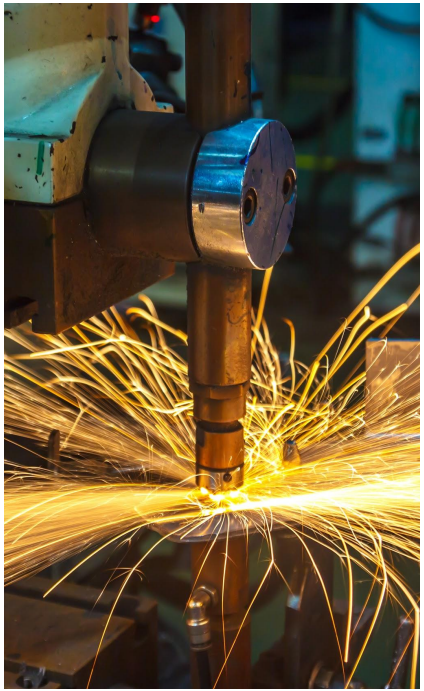
```sql
-- in your rolodex app, you click the S tab
SELECT *
FROM Person.Person
WHERE LEFT(LastName, 1) = 'S'
  AND DATEPART(year, ModifiedDate) > 2011
```

```sql
-- don't modify your searchable arguments
SELECT *
FROM Person.Person
WHERE LastName LIKE 'S%'
  AND ModifiedDate > 2011
```



advantages

- ➔ consuming less system resources
- ➔ speeding up query performance
- ➔ using indexes more effectively



do better, get a coach

evansmith.dev