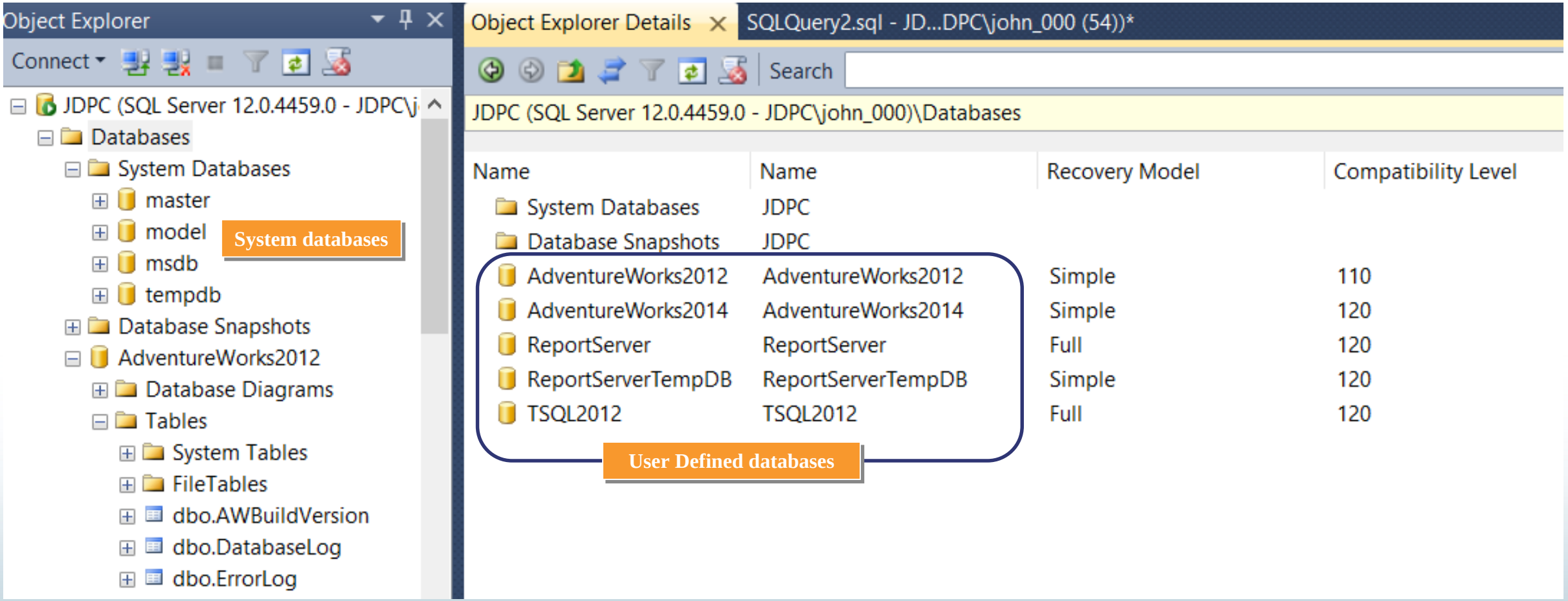
Welcome to New Horizons Computer Learning Center

John Deardurff
Senior Technical Instructor
MCSA, MCSD, MCSE, MCDBA, MCITP, MCT, MVP

Transact-SQL Study Guide

SQL Server Management Studio



System Databases	Description
master	Stores all system-level configuration
msdb	Holds SQL Server Agent configuration including job, backup and restore history
model	Is the template for new databases
tempdb	Holds temporary data like temporary tables, table variables, hash tables and the row version store
resource	Hidden read-only database that contains system objects that are mapped to the sys schema of databases

Using the SELECT statement

```
--Use the * to SELECT all columns from a table
SELECT *
FROM HR.Employees
--It is best practice to choose specific columns
SELECT firstname, lastname
FROM HR.Employees
--You can also sort your results
SELECT firstname, lastname
FROM HR.Employees
ORDER BY lastname DESC, firstname ASC
```

Filtering Records with the WHERE statement

```
--Filtering for a single row
SELECT FirstName, LastName FROM HR.Employees
WHERE empid = 2

--Filtering for multiple rows
SELECT FirstName, LastName FROM HR.Employees
WHERE empid = 2 or empid = 4

--Filtering using the IN predicate
SELECT FirstName, LastName FROM HR.Employees
WHERE empid IN(2, 4)
```

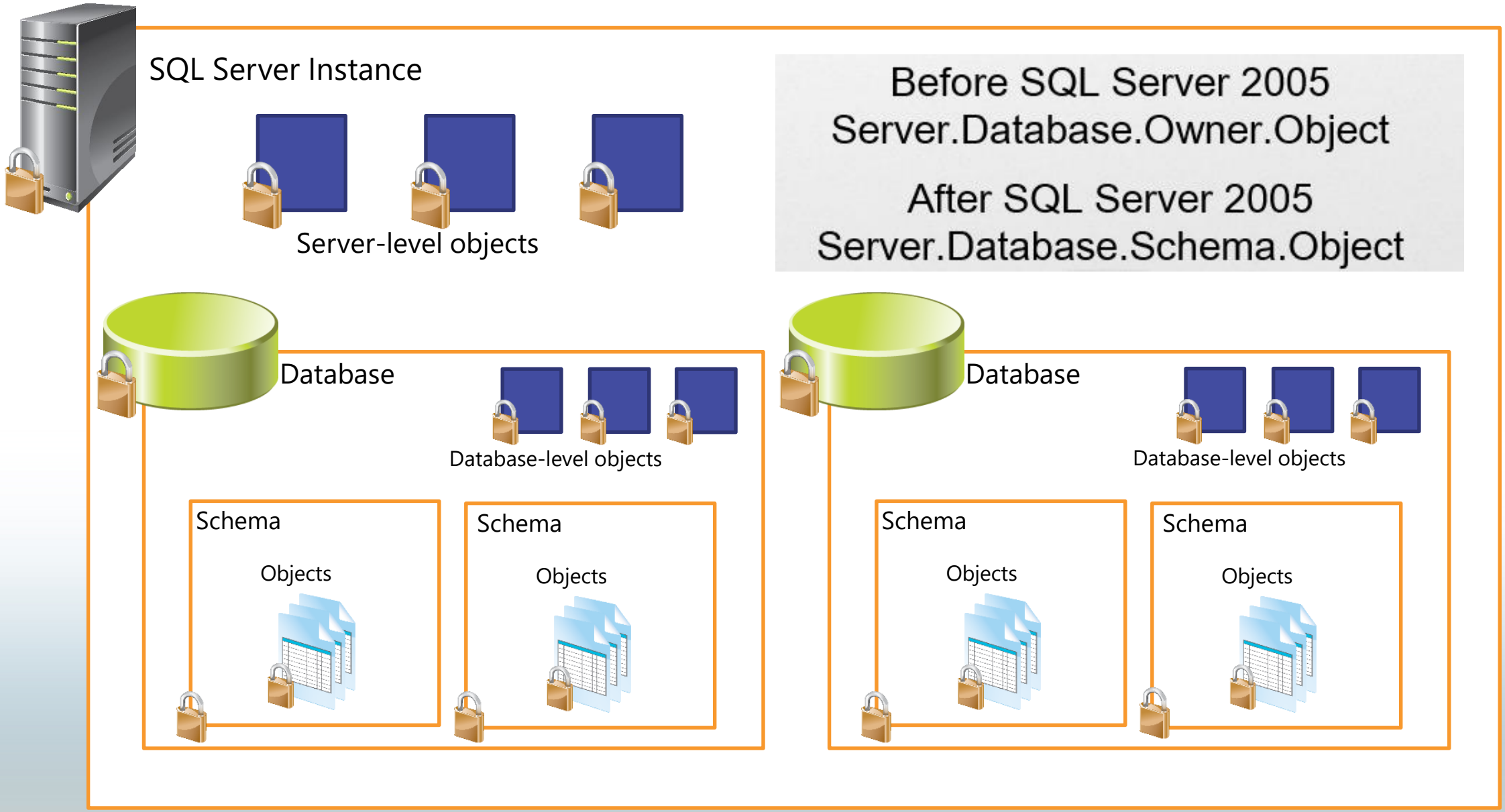
Filtering Records with the WHERE statement

```
--Filtering for a single row
SELECT FirstName, LastName FROM HR.Employees
WHERE LastName = 'Davis'

--Filtering using the Like statement and %
SELECT FirstName, LastName FROM HR.Employees
WHERE LastName LIKE 'D%'

--Filtering using the Like statement and _
SELECT FirstName, LastName FROM HR.Employees
WHERE LastName LIKE '_a%'
```

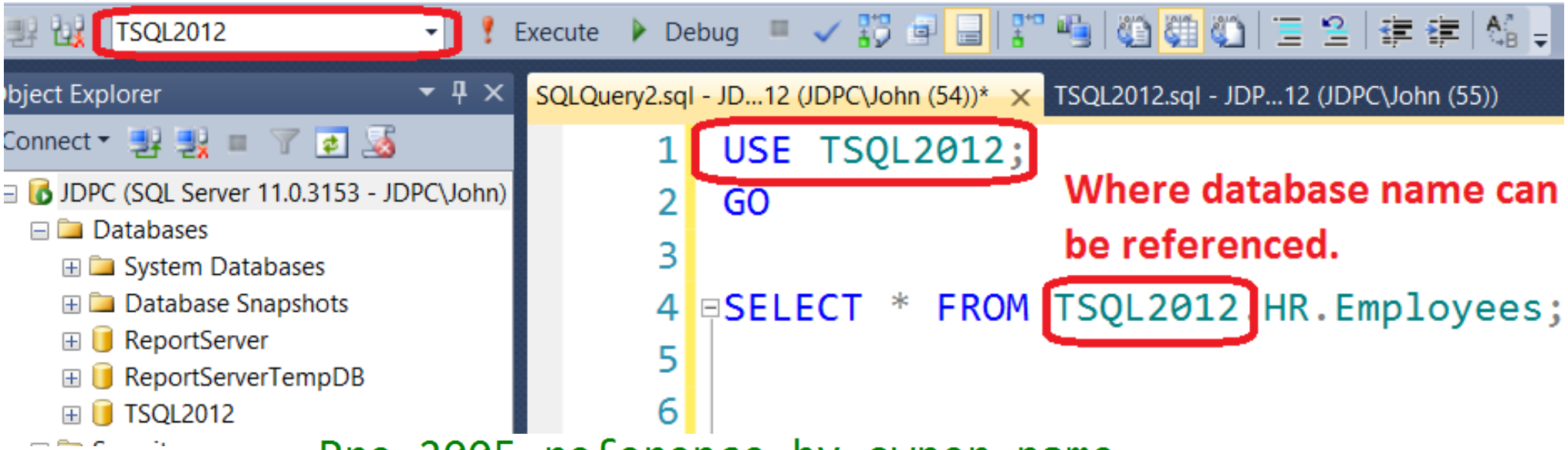
Filtering Records with the WHERE statement

```sql
--Filtering for NULL values
SELECT * FROM HR.Employees
WHERE region IS NULL

--Filtering for NOT NULL values
SELECT * FROM HR.Employees
WHERE region IS NOT NULL

--Filtering for Dates using comparison operators
SELECT * FROM HR.Employees
WHERE HireDate >= '04/01/2002' AND HireDate <= '08/14/2002'

--Filtering for Dates using the BETWEEN statement
SELECT * FROM HR.Employees
WHERE HireDate BETWEEN '04/01/2002' AND '08/14/2002'
```

Four Part Name

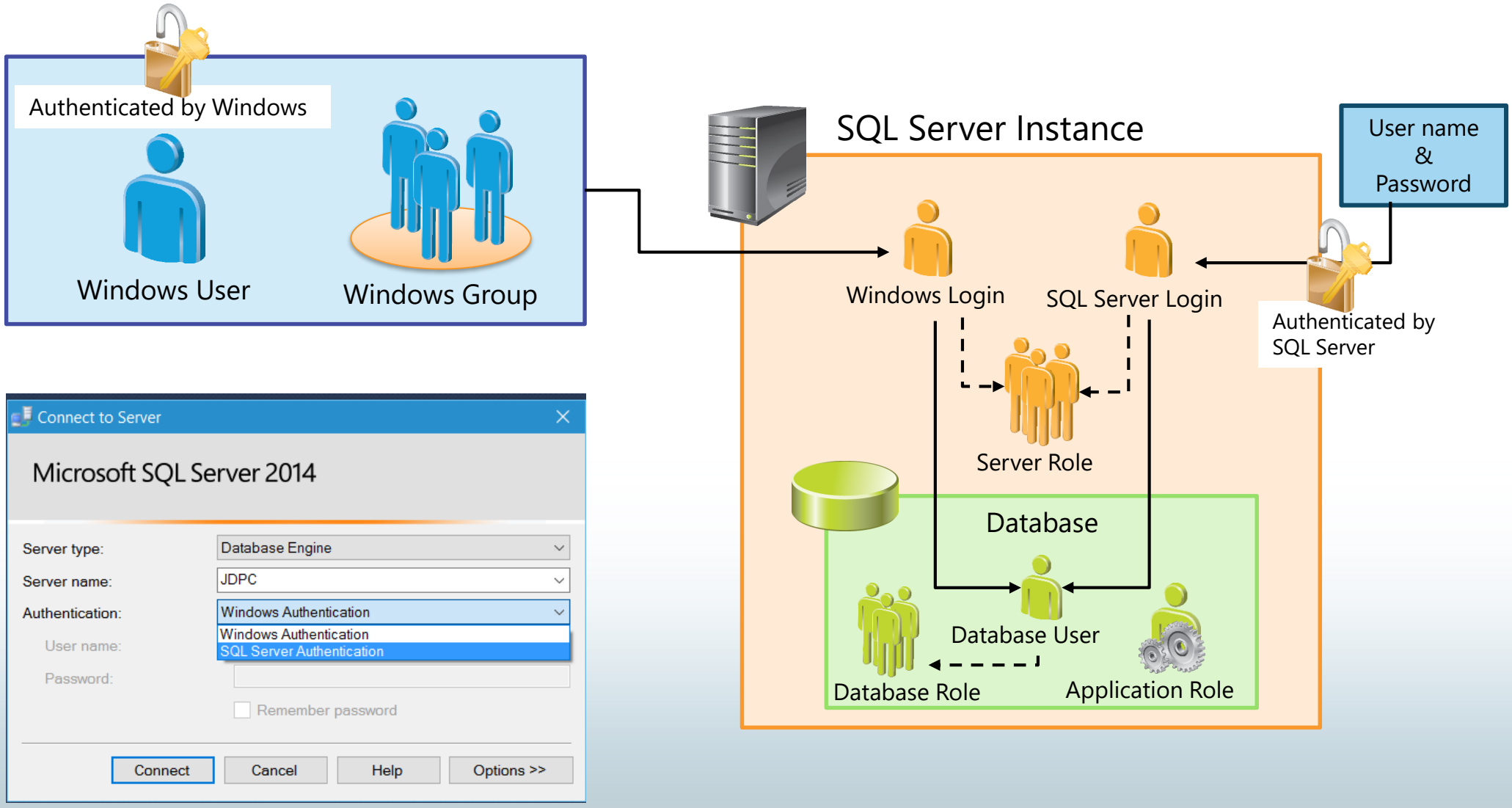


Four Part Name




```
--Pre 2005 reference by owner name
SELECT * FROM John.Employees;

--Using dbo to reference an object
SELECT * FROM dbo.Employees;

--Post 2005 reference by schema name
SELECT * FROM HR.Employees;
```

Security Principals



Logical Query Processing

The order in which a query is written is not the order in which it is evaluated

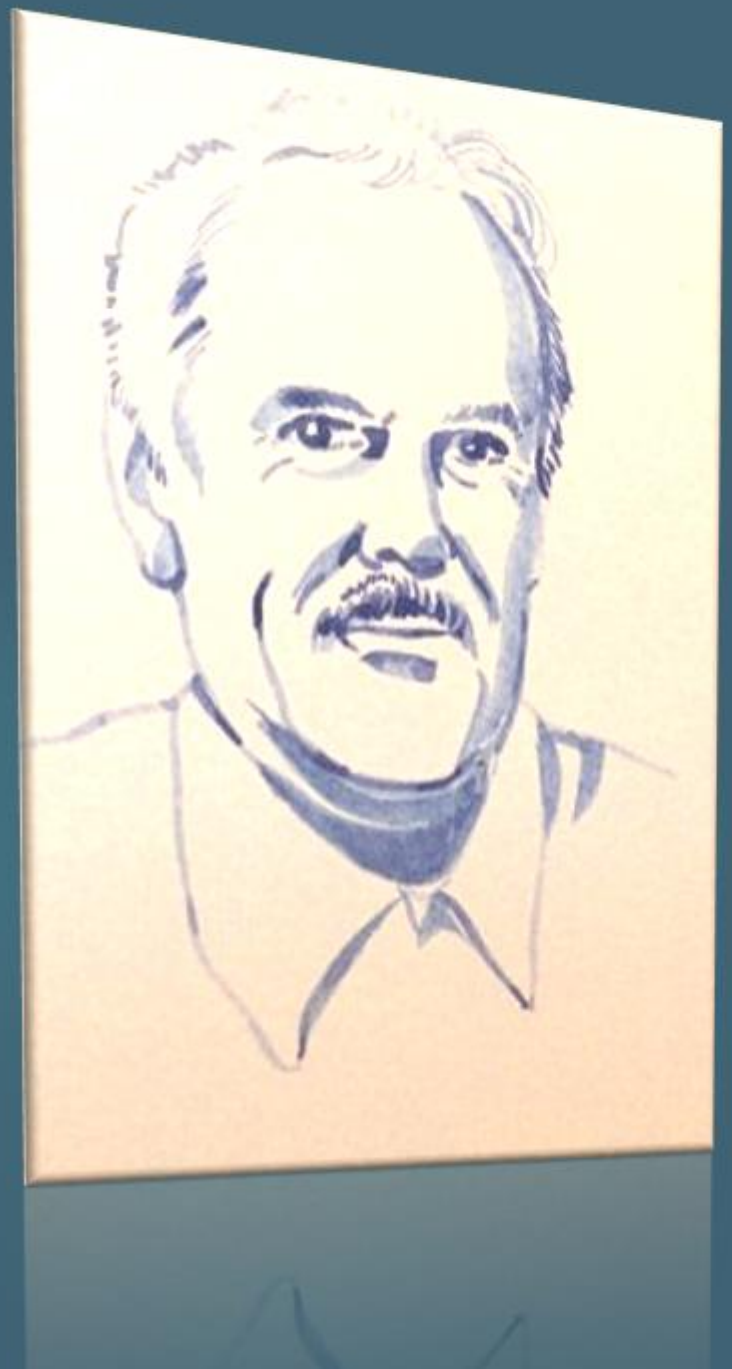
Element	Expression	Role
5: SELECT	<select list>	Defines which columns to return
1: FROM	<table source>	Defines table(s) to query
2: WHERE	<search condition>	Filters rows using a predicate
3: GROUP BY	<group by list>	Arranges rows by groups
4: HAVING	<search condition>	Filters groups using a predicate
6: ORDER BY	<order by list>	Sorts the output

SQL Statement Categories

DML (Manipulation)	DCL (Control)	DDL (Definition)	TCL (Transactional)
INSERT	GRANT	CREATE	BEGIN
UPDATE	DENY	ALTER	COMMIT
DELETE	REVOKE	DROP	ROLLBACK
SELECT (DQL)		TRUNCATE	SAVE

Ground rules for database design

- Identify Database Purpose
- Review Existing Data
- Make a Preliminary list of fields
- Group fields into tables
- Review for Maintenance problems
- Designate Primary Key



Codd's Law

A non-key field must provide a fact about

NF1 – the *key*
NF2 – the *whole* key
NF3 – and *nothing but* the key

So help me Codd.

Why Normalize Data?

- Save the typing of repetitive data
- Save storage space
- Avoid frequent restructuring of tables an other objects to accommodate new data
- Increase flexibility to query, sort, summarize, and group data.

Normalization Guidelines

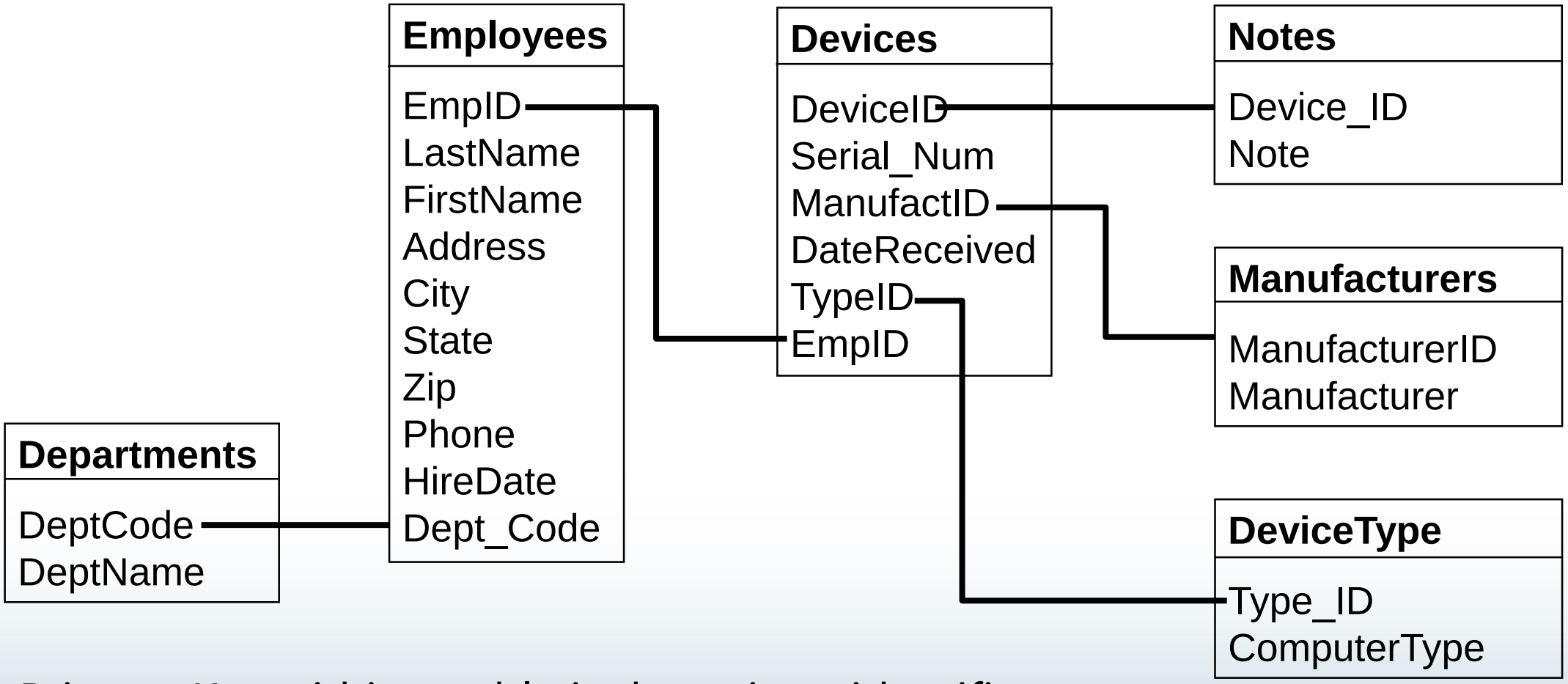
- Locate or create a unique field to become a Primary Key
- Each field should contain smallest meaningful value. (First Name, Last Name)
- There should not be repeated groups of fields within a table. (Computer 1, Computer 2, Computer 3)
- There should not be unnecessarily repeated data within a field. (HiTech, HiTech, HiTech)
- All fields should pertain to every record within the table.

Data before Normalization

Employees						
EmpID	Name	Address	Dept	Device1	Device2	Device3
1	Sarah Connor	123 Highway Road	Administration			
2	Bjorn Wilde	456 Easy Street	Marketing	5001	5003	5005
3	Gwen Stacy	789 Lois Lane	Administration	5002	5007	
4	Peter Gibbons	806 Sunday Drive	Sales	5004		
5	Winston Smith	1984 Vicorty Apt.	Administration	5006		

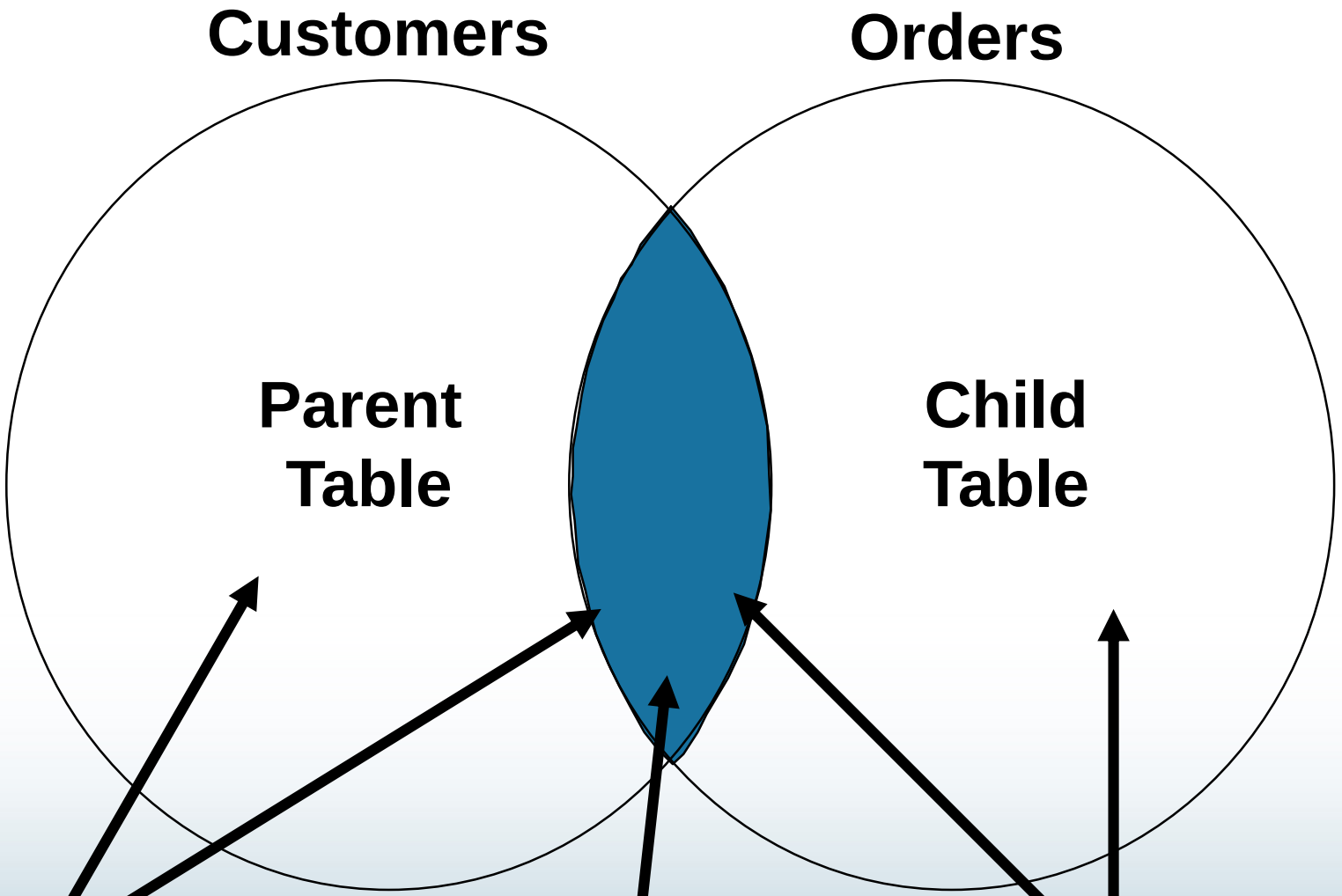
Devices						
DeviceID	Serial_Num	Manufacturer	Date Received	Type	EmpID	Notes
5001	MK5001DT	DigiTech	10/19/2008	Desktop	2	Noisy
5002	AD5002HT	HiTech	2/16/2011	Laptop	3	
5003	MK5003DT	DigiTech	10/19/2012	Tablet	2	Warranty
5004	SL5004DT	DigiTech	4/1/2012	Laptop	4	Cracked
5005	MK5005DT	DigiTech	6/7/2011	Laptop	2	
5006	AD5006HT	HiTech	5/5/2013	Desktop	5	Noisy
5007	AD5007HT	HiTech	6/10/2010	Tablet	3	

Finding Relationships



Primary Key within a table is the unique identifier
Primary Key within a relationship is the best describer
Foreign Key within a relationship connects Child table to Parent table

Join Types



Left Outer
All **Customers**
& related **Orders**

Inner
Customers who
placed **Orders**

Right Outer
All **Orders**
& related **Customers**

Inner Joins

CUSTOMERS

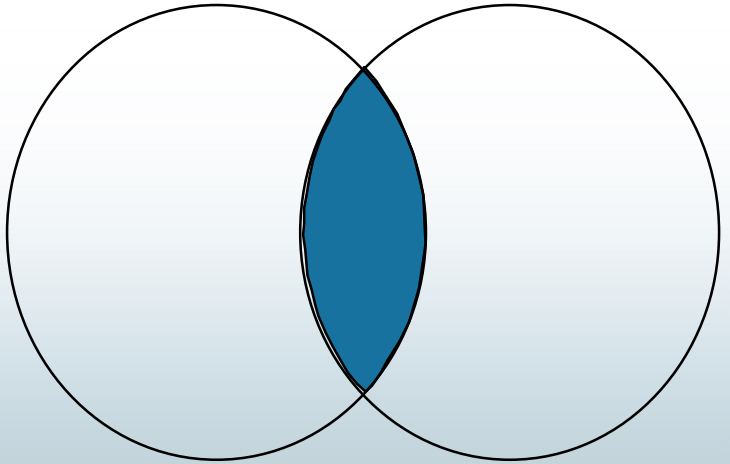
CustomerID	First_Name	Last_Name	Club
250	Andy	Anderson	1
255	Jeff	Rollins	0
267	Bob	Smith	1
278	Jenny	Jefferson	0
388	Cindy	Samuels	2

ORDERS

OrderID	CustID	ProductID	Price	Qty
1	250	23569	5.60	3
2	250	32575	2.45	3
3	250	32600	3.20	5
4	255	30550	3.15	4
5	278	33002	3.75	6
6	290	32667	2.50	9
7	388	32600	3.20	7
8	388	29058	4.25	5
9	402	32660	3.75	2

```
SELECT CustomerID, Last_Name, Qty
FROM Customers AS C
INNER JOIN Orders AS O
ON C.CustomerID = O.CustID
```

CustomerID	Last_Name	Qty
250	Anderson	3
250	Anderson	3
250	Anderson	5
255	Rollins	4
278	Jefferson	6
388	Samuels	7
388	Samuels	5



Left Outer Joins

CUSTOMERS

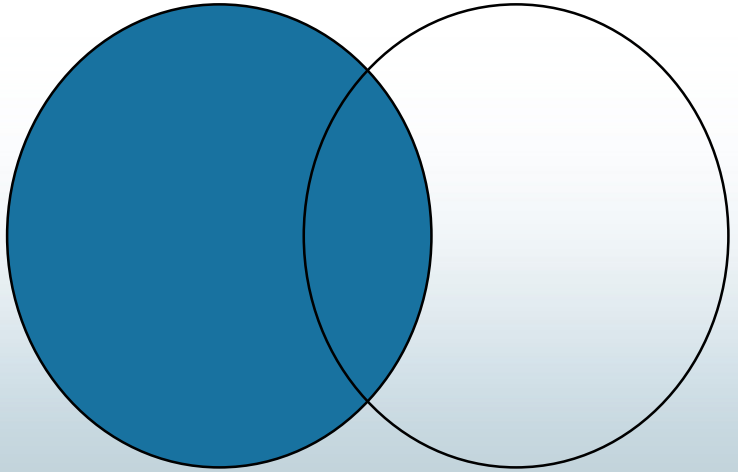
CustomerID	First_Name	Last_Name	Club
250	Andy	Anderson	1
255	Jeff	Rollins	0
267	Bob	Smith	1
278	Jenny	Jefferson	0
388	Cindy	Samuels	2

ORDERS

OrderID	CustID	ProductID	Price	Qty
1	250	23569	5.60	3
2	250	32575	2.45	3
3	250	32600	3.20	5
4	255	30550	3.15	4
5	278	33002	3.75	6
6	290	32667	2.50	9
7	388	32600	3.20	7
8	388	29058	4.25	5
9	402	32660	3.75	2

```
SELECT CustomerID, Last_Name, Qty
FROM Customers AS C
LEFT OUTER JOIN Orders AS O
ON C.CustomerID = O.CustID
```

CustomerID	Last_Name	Qty
250	Anderson	3
250	Anderson	3
250	Anderson	5
255	Rollins	4
267	Smith	NULL
278	Jefferson	6
388	Samuels	7
388	Samuels	5



Right Outer Joins

CUSTOMERS

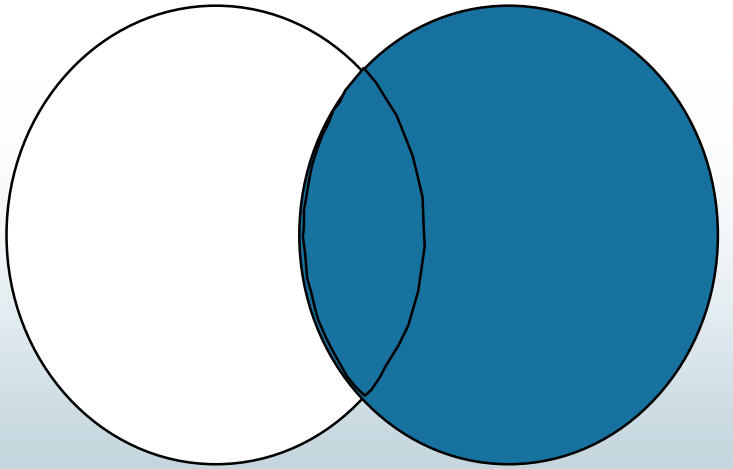
CustomerID	First_Name	Last_Name	Club
250	Andy	Anderson	1
255	Jeff	Rollins	0
267	Bob	Smith	1
278	Jenny	Jefferson	0
388	Cindy	Samuels	2

ORDERS

OrderID	CustID	ProductID	Price	Qty
1	250	23569	5.60	3
2	250	32575	2.45	3
3	250	32600	3.20	5
4	255	30550	3.15	4
5	278	33002	3.75	6
6	290	32667	2.50	9
7	388	32600	3.20	7
8	388	29058	4.25	5
9	402	32660	3.75	2

```
SELECT CustomerID, Last_Name, Qty
FROM Customers AS C
RIGHT OUTER JOIN Orders AS O
ON C.CustomerID = O.CustID
```

CustID	Last_Name	Qty
250	Anderson	3
250	Anderson	3
250	Anderson	5
255	Rollins	4
278	Jefferson	6
290	NULL	9
388	Samuels	7
388	Samuels	5
402	NULL	2



Full Outer Joins

CUSTOMERS

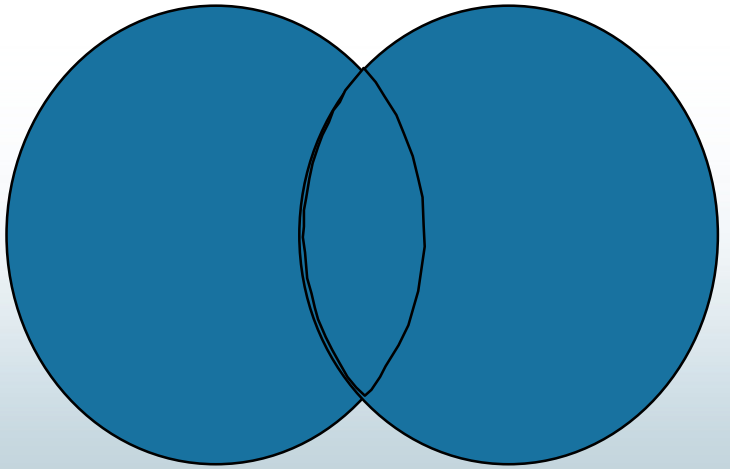
CustomerID	First_Name	Last_Name	Club
250	Andy	Anderson	1
255	Jeff	Rollins	0
267	Bob	Smith	1
278	Jenny	Jefferson	0
388	Cindy	Samuels	2

ORDERS

OrderID	CustID	ProductID	Price	Qty
1	250	23569	5.60	3
2	250	32575	2.45	3
3	250	32600	3.20	5
4	255	30550	3.15	4
5	278	33002	3.75	6
6	290	32667	2.50	9
7	388	32600	3.20	7
8	388	29058	4.25	5
9	402	32660	3.75	2

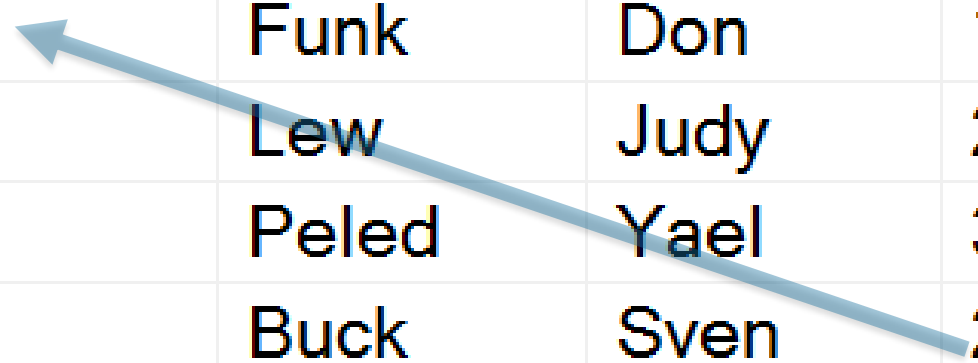
```
SELECT C.CustomerID, O.Custid,
       Last_Name, Qty
FROM Customers AS C
FULL OUTER JOIN Orders AS O
ON C.CustomerID = O.CustID
```

CustomerID	Custid	Last_Name	Qty
250	250	Anderson	3
250	250	Anderson	3
250	250	Anderson	5
255	255	Rollins	4
267	NULL	Smith	NULL
278	278	Jefferson	6
388	388	Samuels	7
388	388	Samuels	5
NULL	290	NULL	9
NULL	402	NULL	2



Self Joins: Comparing data within a table

empid	lastname	firstname	mgrid
1	Davis	Sara	NULL
2	Funk	Don	1
3	Lew	Judy	2
4	Peled	Yael	3
5	Buck	Sven	2
6	Suurs	Paul	5
7	King	Russell	5



- Start with employee whose manager you want to find.
- Observe the value in the mgrid column.
- Locate the matching value in the empid column.

Writing a Self Join

```
SELECT E.empid, e.lastname, e.firstname,
        m.lastname AS Manager
FROM HR.Employees AS E
INNER JOIN HR.Employees AS M
ON E.mgrid = M.empid
```

empid	lastname	firstname	Manager
2	Funk	Don	Davis
3	Lew	Judy	Funk
4	Peled	Yael	Lew
5	Buck	Sven	Funk
6	Suurs	Paul	Buck
7	King	Russell	Buck
8	Cameron	Maria	Lew

Constraints

- **PRIMARY KEY** – Ensures that a column has a unique value for each record.
- **NOT NULL** – Enforces that every record has a value for the column
- **DEFAULT** – Specifies a value for the column when a value is not included
- **UNIQUE** – Enforces that each column uses distinct and unique values
- **CHECK** – Enforces specific rules that a column must follow
- **FOREIGN KEY** – Enforces the child relationship with a parent table

```
CREATE TABLE HR.Employees
(EmpID int IDENTITY PRIMARY KEY,
 FirstName varchar(15) NOT NULL,
 LastName varchar(20) NOT NULL,
 JobTitle varchar(20) NOT NULL,
 HireDate date DEFAULT GETDATE(),
 BirthDate date NULL,
 PhoneNumber varchar(15) UNIQUE,
 DeptCode tinyint)
```

Adding a CHECK constraint

```
--Insert Record to test constraints
INSERT INTO HR.Employees
VALUES ('John', 'Deardurff', 'MCT',
        DEFAULT, NULL, '555-867-5309',2)

--Add Check Constraint
ALTER TABLE HR.Employees
ADD CONSTRAINT CK_HireDate18
CHECK(HireDate > DATEADD(YY, 18, Birthdate))
```

Foreign Key Constraints

```
--Create Table to join to HR.Employees
CREATE TABLE HR.Departments
(DeptID tinyint PRIMARY KEY,
 DeptName varchar(15))

--Insert values into new table
INSERT INTO HR.Departments
VALUES (1, 'Accounting'), (2, 'Training'),
       (3, 'Sales'), (4, 'Marketing')

--Add Foreign Key Constraint
ALTER TABLE HR.Employees
ADD CONSTRAINT FK_Emp_Dept FOREIGN KEY (DeptCode)
REFERENCES HR.Departments(DeptID)
ON UPDATE CASCADE ON DELETE SET NULL
```

Altering Tables

```
--Adding Columns to a Table
ALTER TABLE HR.Departments
ADD ManagerID tinyint NULL,
    Location varchar(10)
GO
--Modifying a Column in a Table
ALTER TABLE HR.Departments
ALTER COLUMN DeptName varchar(10) NULL
GO
--Removing a Column from a Table
ALTER TABLE HR.Departments
DROP COLUMN Location
GO
```

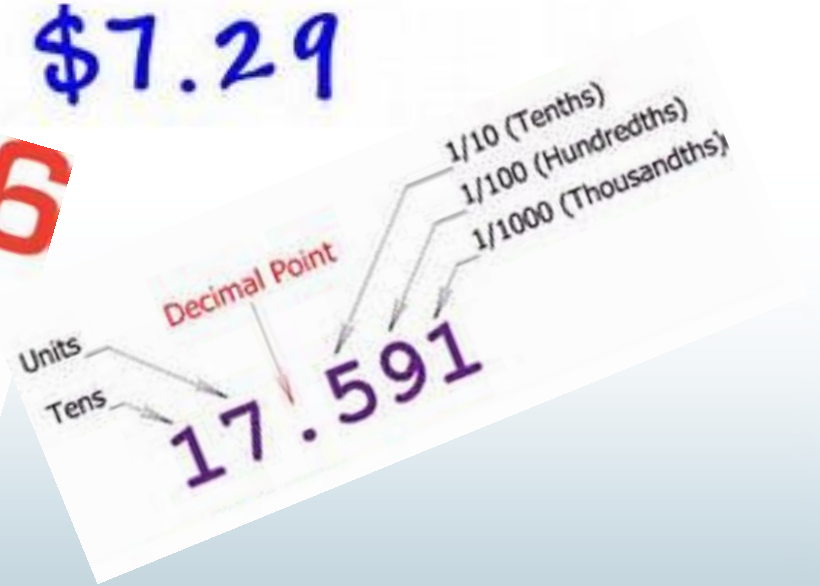
Why Data Types Matter

- Used for Columns, Variables, Expressions, & Parameters
- Fundamental to writing queries in T-SQL
- Fundamental to designing tables.
- Choosing the right data type ensures data integrity and accuracy by creating a constraint on data input
- Choosing the right data type can save hard drive space, space in memory, and bandwidth on your network

SQL Server Data Types

Data types determine what kind of data can be held:

Integers, characters, dates, money, decimals, etc.



Data Type Associations

Columns

```
EmpID int IDENTITY
```

Expressions

```
SELECT 42 + 11 + 'Total'
```

Variables

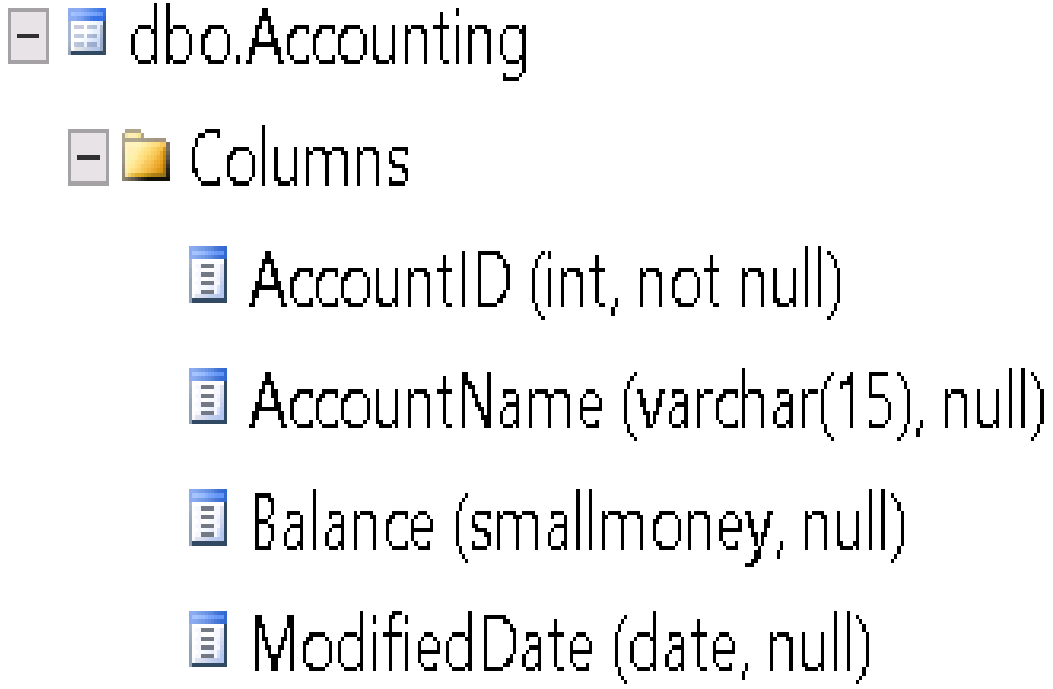
```
DECLARE @String1 as char(10)
```

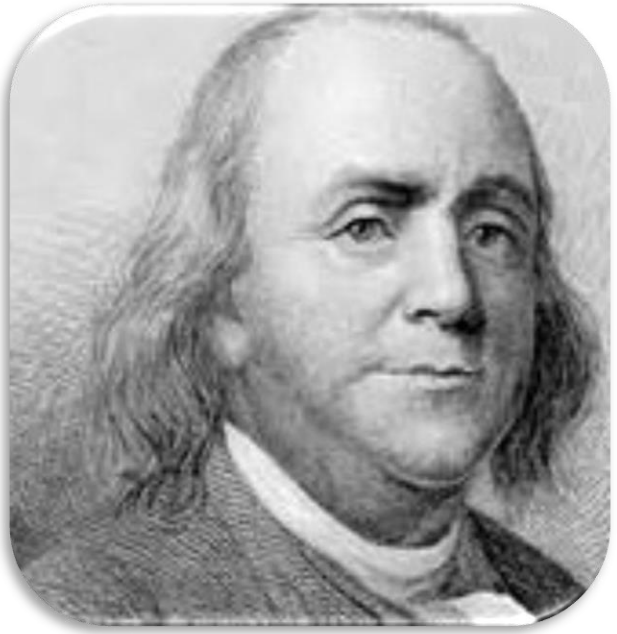
Parameters

```
SELECT LEN(@String1)
```

Using Data Type with Columns

```
1  CREATE TABLE Accounting
2  (AccountID int IDENTITY,
3   AccountName varchar(15),
4   Balance smallmoney,
5   ModifiedDate date)
6   GO
7
```





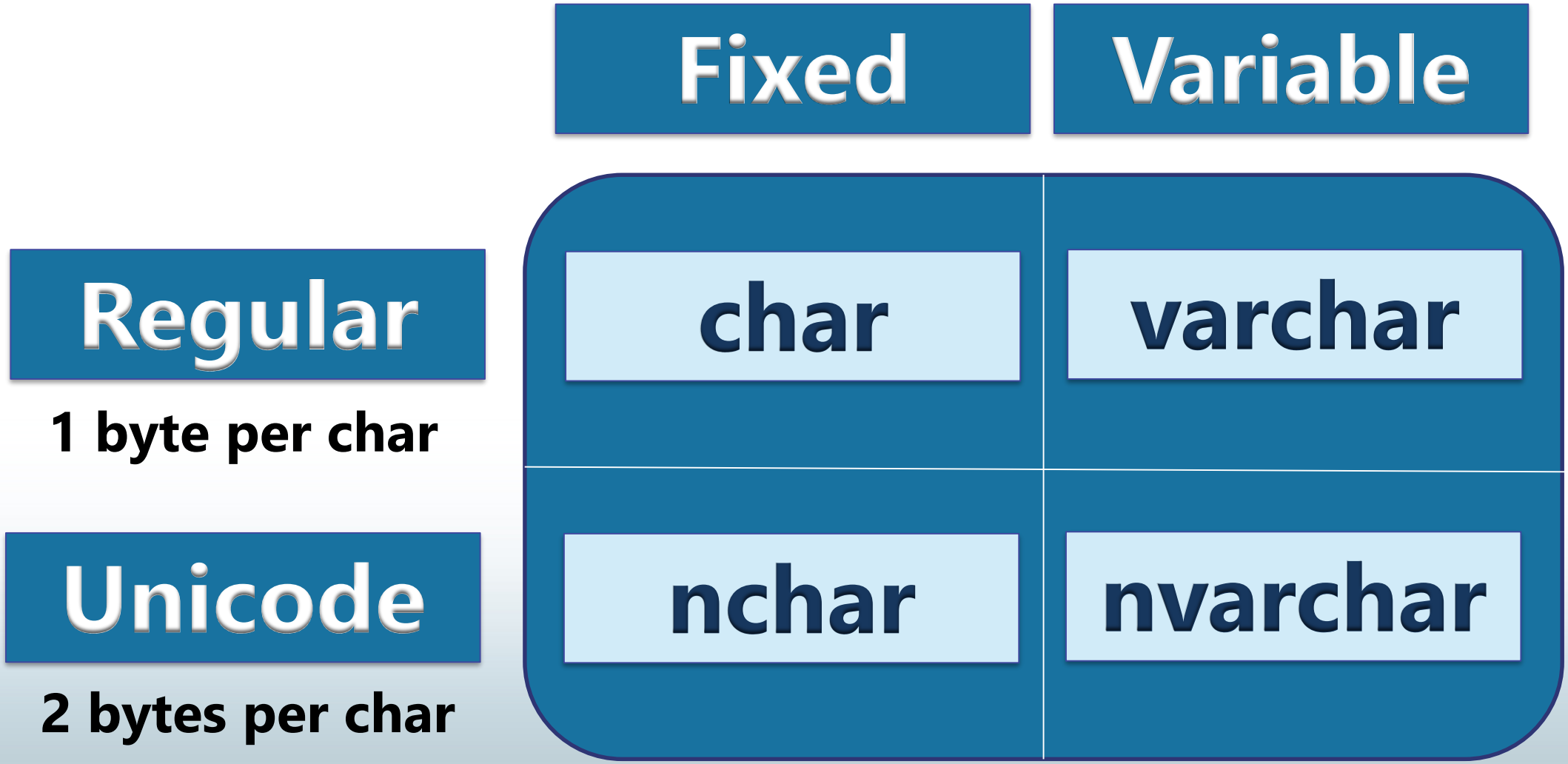
A Byte saved is a Byte earned.
~ Benjamin Franklin

The problem with quotes from the internet is it is hard to verify their authenticity.
~ Abraham Lincoln

Character Data Types

Data Type	Range	Storage
CHAR(n), NCHAR(n)	1-8000 characters 1-4000 characters	bytes 2*n bytes
VARCHAR(n), NVARCHAR(n)	1-8000 characters 1-4000 characters	n+2 bytes (2*n) +2 bytes
VARCHAR(MAX), NVARCHAR(MAX)	1-2^31-1 characters	Actual length + 2

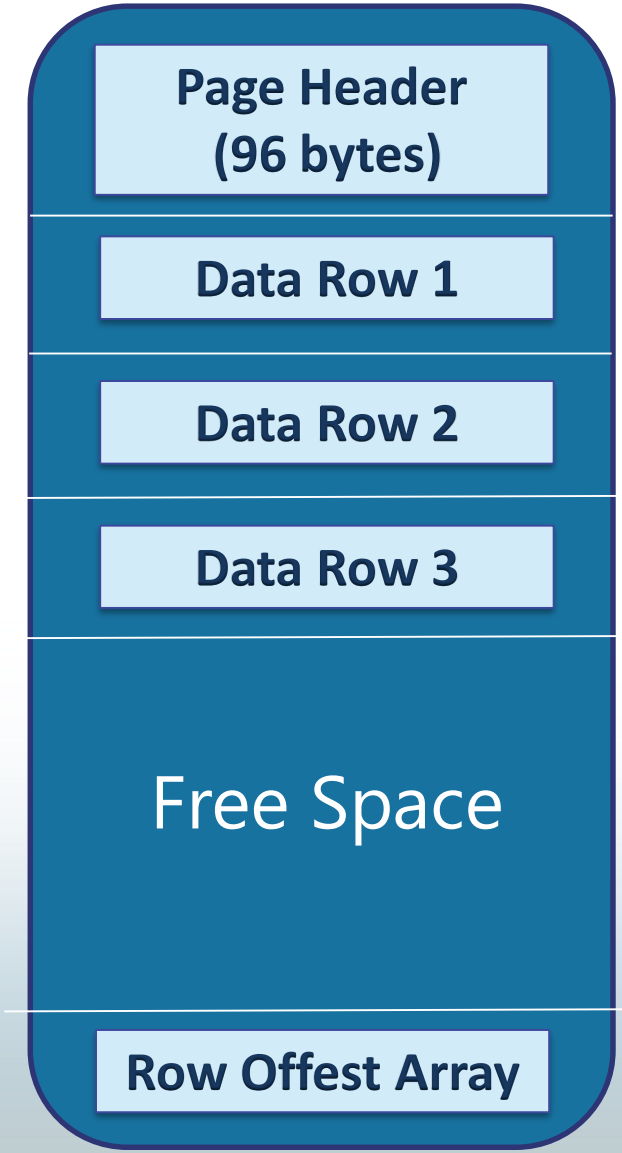
Character Data Types



Character Data Code Example

```
DECLARE @String1 AS char(10)
DECLARE @String2 varchar(10)

SET @String1 = 'Hello'
SET @String2 = 'World'

SELECT @String1 + @String2 AS ConcatStrings --Concatenation
SELECT LEN(@String1), LEN(@String2) --How many characters?
SELECT DATALENGTH(@String1), DATALENGTH(@String2) --How many bytes?
```

Rows stored in Data Pages

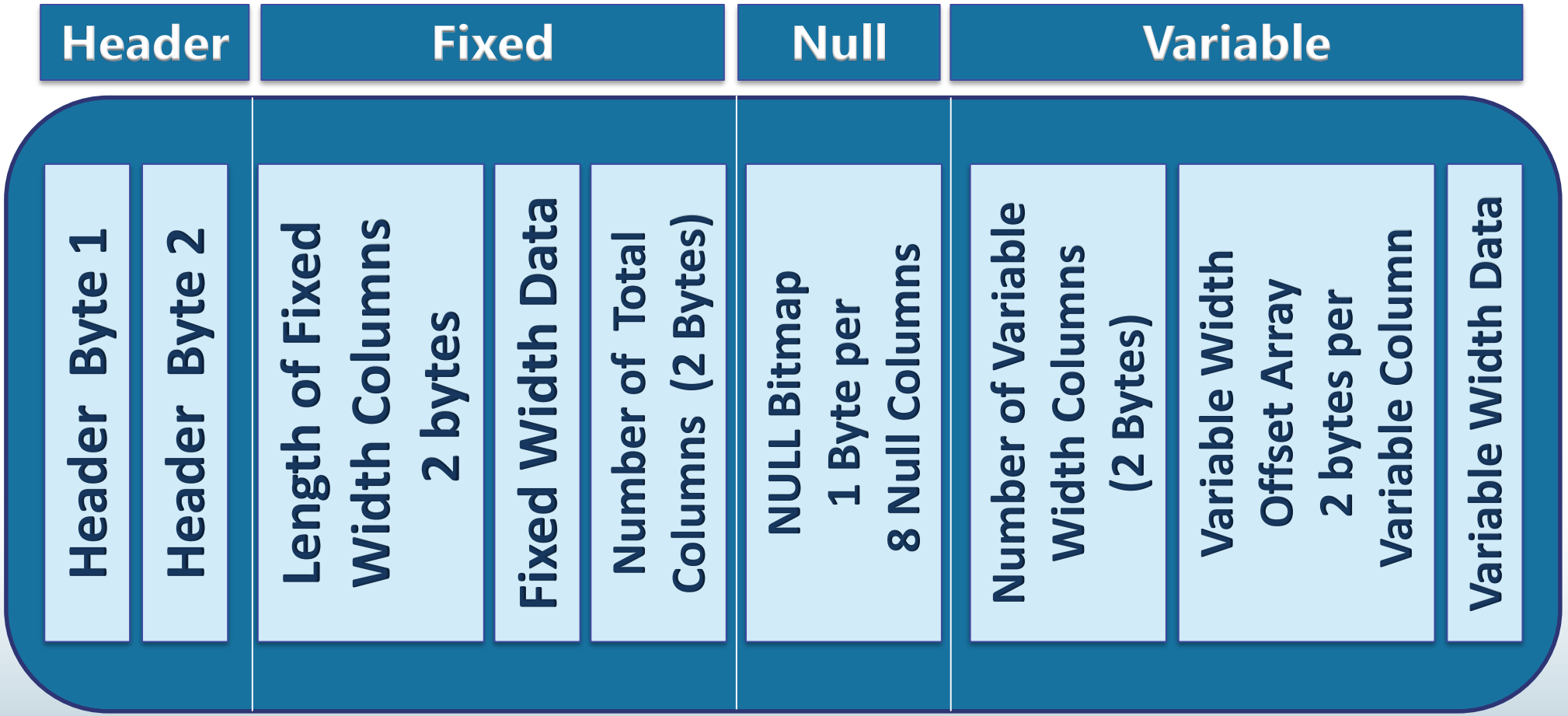


Header contains Page Information such a Page Number and Page Type

8000 bytes for Data Rows or Free Space

Displays how far from beginning of page each row is located.

FixVar Row Format



Integer Data Types

Data type	Range	Storage (bytes)
tinyint	0 to 255	1
smallint	-32,768 to 32,767	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

Integer Data Type Coding Example

```
DECLARE @tiny as tinyint = 255
DECLARE @small as smallint = 32767
DECLARE @integer as int = 2147483647
DECLARE @BIG as bigint = 9223372036854775807
```

TinyInteger	SmallInteger	Integer	BigInteger
255	32767	2147483647	9223372036854775807

Other Numeric Data Types

Data type	Range	Storage (bytes)
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
float /real	Approximate data type for floating point numeric data	4-8

Date and Time Data Types

Data Type	Storage (bytes)	Date Range	Accuracy
DATETIME	8	January 1, 1753 to December 31, 9999	3-1/3 milliseconds
SMALLDATETIME	4	January 1, 1900 to June 6, 2079	1 minute
DATETIME2	6 to 8	January 1, 0001 to December 31, 9999	100 nanoseconds
DATE	3	January 1, 0001 to December 31, 9999	1 day
TIME	3 to 5		100 nanoseconds
DATETIMEOFFSET	8 to 10	January 1, 0001 to December 31, 9999	100 nanoseconds

Date and Time Data Types

- Older versions of SQL Server supported only DATETIME and SMALLDATETIME
- DATE, TIME, DATETIME2, and DATETIMEOFFSET introduced in SQL Server 2008
- SQL Server doesn't offer an option for entering a date or time value explicitly
  - Dates and times are entered as character literals and converted explicitly or implicitly
    - For example, CHAR converted to DATETIME due to precedence

Working with Date and Time Separately

- DATETIME, SMALLDATETIME, DATETIME2, and DATETIMEOFFSET include both date and time data
- If only date is specified, time set to midnight (all zeroes)
- If only time is specified, date set to base date (January 1, 1900)

```
DECLARE @DateOnly DATETIME = '20140214'
DECLARE @TimeOnly DATETIME = '15:30:45'

SELECT @DateOnly AS DateOnly, @TimeOnly AS TimeOnly
```

DateOnly	TimeOnly
2014-02-14 00:00:00.000	1900-01-01 15:30:45.000

Working with Time Accuracy

```
DECLARE @Time1 DATETIME = '15:30:45'
DECLARE @Time2 DATETIME2 = '15:30:45'
DECLARE @Time3 TIME = '15:30:45'
DECLARE @Time4 TIME(0) = '15:30:45'
DECLARE @Time7 TIME(7) = '15:30:45'

SELECT @Time1 AS DTexample,
       @Time2 AS DTexample2,
       @Time3 AS TimeOnly,
       @Time4 AS TimeZero,
       @Time7 AS TimeSeven
```

DTexample	DTexample2	TimeOnly	TimeZero	TimeSeven
1900-01-01 15:30:45.000	1900-01-01 15:30:45.0000000	15:30:45.0000000	15:30:45	15:30:45.0000000

Data Type Precedence

- Data type precedence determines which data type will be chosen when expressions of different types are combined
- Data type with the lower precedence is implicitly converted to the data type with the higher precedence
- Conversion to type of lower precedence must be made explicitly (with CAST or CONVERT function)

CHAR -> VARCHAR -> NCHAR -> NVARCHAR -> BIT -> TINYINT -> SMALLINT -> INT -> BIGINT -> MONEY -> DECIMAL -> TIME -> DATE -> DATETIME2 -> XML

https://msdn.microsoft.com/en-us/library/ms190309.aspx

Data Type Conversion Functions

```
--Implicit Conversion
SELECT 42 + 11 + '25'

--Conversion Error
SELECT 42 + 11 + ' Total'

--Explicit Conversion using CAST
SELECT CAST(42 + 11 as char(2)) + ' Total'

--Explicit Conversion using CONVERT
SELECT CONVERT(char(2), 42 + 11) + ' Total'
```

Data Type Conversion Functions (continued)

```
--Parse can convert character data to Dates or Money
SELECT PARSE ('7 February 2015 3:30pm' as datetime2 USING 'en-US') as Result
SELECT PARSE ('$85.22' as money USING 'en-US') AS Result

--If PARSE can not convert it will give an ERROR
SELECT PARSE ('20 dollars' as money USING 'en-US') AS Result --ERROR

--The Try_Parse and Try_Convert Functions will return NULL if an Error
SELECT TRY_PARSE ('20 dollars' as money USING 'en-US') AS Result

SELECT
    CASE WHEN TRY_CONVERT(int, 'test') IS NULL
    THEN 'Cast Failed'
    ELSE 'Convert Succeeded'
END AS Result
```

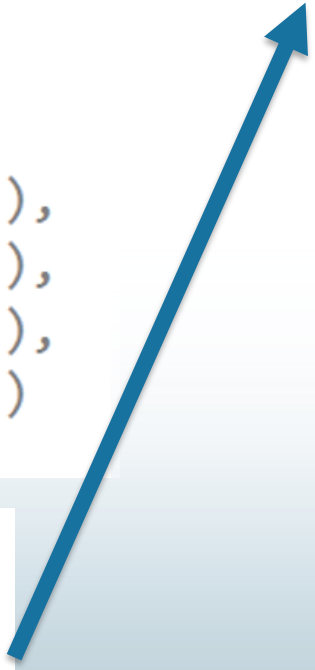
Additional Data Types

Data type	Range	Storage (bytes)
rowversion	Auto-Generated	8
uniqueidentifier	Auto-Generated	16
xml	0-2 GB	0-2 GB
cursor	NA	NA
hierarchyid	NA	Depends on content
sql_variant	0-8000 bytes	Depends on content
table	NA	NA

Data Manipulation Language (DML)

```
CREATE TABLE dbo.TestTable
(TestID int IDENTITY,
 TestCode char(2),
 TestName varchar(30),
 TestDate date)
GO

INSERT dbo.TestTable
VALUES ('R1', 'Row 1', '20080612'),
       ('R2', 'Row 2', '20090713'),
       ('R3', 'Row 3', '20100814'),
       ('R4', 'Row 4', '20110915')
```

TestID	TestCode	TestName	TestDate
1	R1	Row 1	2008-06-12
2	R2	Row 2	2009-07-13
3	R3	Row 3	2010-08-14
4	R4	Row 4	2011-09-15



```
UPDATE dbo.TestTable
SET TestDate = GETDATE()
WHERE TestID = 3

DELETE dbo.TestTable
WHERE TestCode IN('R2', 'R4')
```

TestID	TestCode	TestName	TestDate
1	R1	Row 1	2008-06-12
3	R3	Row 3	2012-08-16

Inserting Values into IDENTITY Columns

The column_list must be used to insert values into an identity column, and the SET IDENTITY_INSERT option must be ON for the table

```
CREATE TABLE dbo.TestTable(TestID int IDENTITY,
    TestName char(15) NOT NULL)
GO

INSERT INTO dbo.TestTable VALUES ('First Row')
INSERT INTO dbo.TestTable (TestName)
    VALUES ('Second Row')
INSERT INTO dbo.TestTable VALUES
    ('Third Row'), ('Fourth Row')

SET IDENTITY_INSERT dbo.TestTable ON
INSERT INTO dbo.TestTable (TestID, TestName)
    VALUES (-99, 'Explicit Row')
SET IDENTITY_INSERT dbo.TestTable OFF

SELECT * FROM dbo.TestTable
```

TestID	TestName
1	First Row
2	Second Row
3	Third Row
4	Fourth Row
-99	Explicit Row

SEQUENCES

```
CREATE SEQUENCE dbo.SeqOrders
AS int START with 100 INCREMENT BY 100
GO

CREATE TABLE North_Orders
(OrderID int Primary Key,
 Store char(5))
CREATE TABLE South_Orders
(OrderID int Primary Key,
 Store char(5))
GO

---Run Code three times
INSERT INTO North_Orders
VALUES (NEXT VALUE FOR dbo.SeqOrders, 'North')
INSERT INTO South_Orders
VALUES (NEXT VALUE FOR dbo.SeqOrders, 'South')
```

```
SELECT * FROM North_Orders
UNION
SELECT * FROM South_Orders
```

OrderID	Store
100	North
200	South
300	North
400	South
500	North
600	South

Sequences allow the ability to increment numbers across more than one table.

Using the OUTPUT Clause

Using OUTPUT in a DML statement returns information from each row affected by the DML statement

```
1  CREATE TABLE dbo.TestTable
2  (TestID int IDENTITY,
3    TestName char(30) NOT NULL)
4  GO
5  ----------------------------------
6  INSERT INTO dbo.TestTable
7       OUTPUT inserted.*
8  VALUES('First Row'), ('Second Row'),
9       ('Third Row'),('Fourth Row')
10 ----------------------------------
11 DELETE dbo.TestTable
12      OUTPUT deleted.*
13 WHERE TestID = 4
14 ----------------------------------
15 UPDATE dbo.TestTable
16 SET TestName = 'Another Row'
17      OUTPUT inserted.*, deleted.*
18 WHERE TestID = 3
```

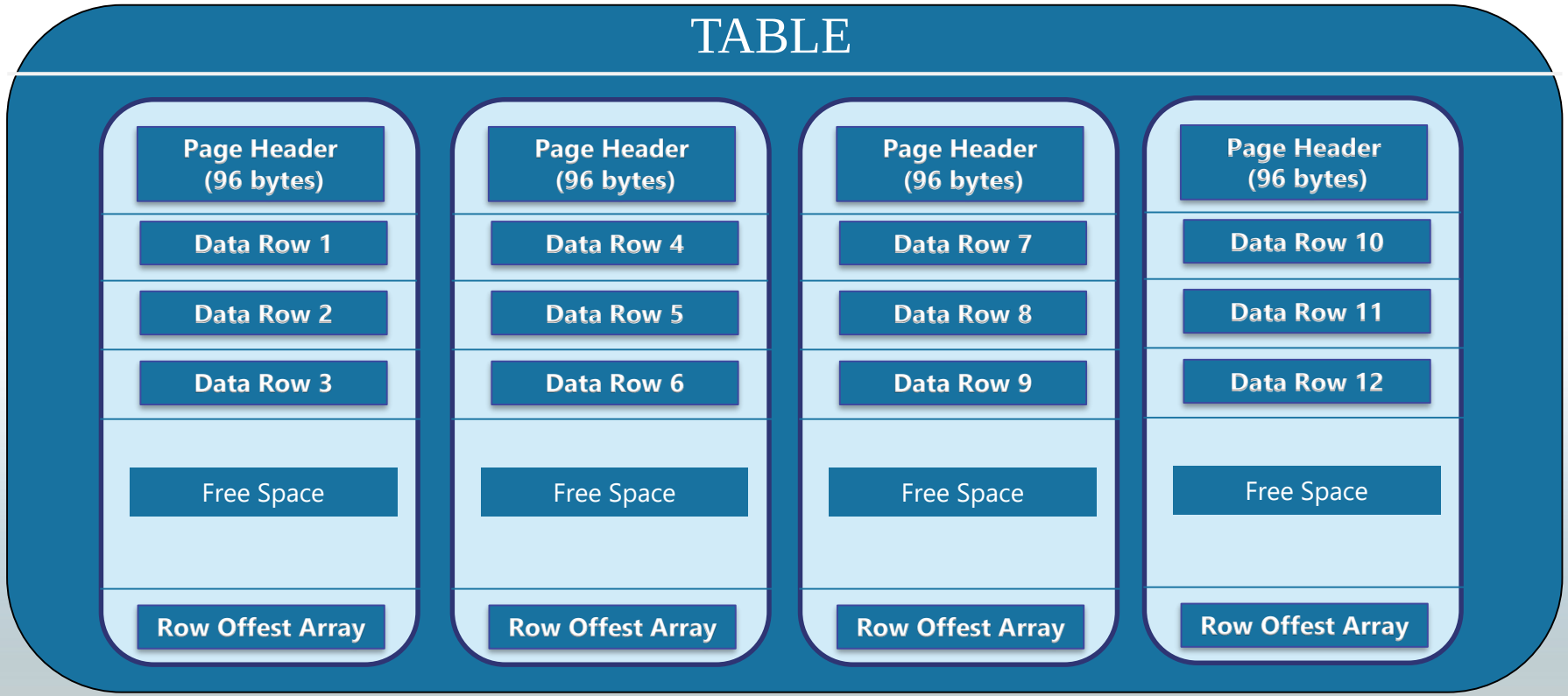
Virtual Tables

- Inserted and Deleted Virtual Tables allow the ability to access data before and after data modification.
- Available in AFTER and INSTEAD of Triggers.
- Created at the beginning of a statement.

Statement	Inserted	Deleted
INSERT	Rows just inserted	
DELETE		Rows just deleted
UPDATE	Modified row contents	Original row contents

Delete vs Truncate vs Drop

```
--Removes Records from the Table
DELETE Accounting.BankAccounts
--Removes Data Pages from Table
TRUNCATE TABLE Acccounting.BankAccounts
--Removes Entire Table
DROP TABLE Accounting.BankAccounts
```



COALESCE Function

```
CREATE TABLE dbo.Wages
(EmpID int,
 hourly_wage int,
 salary int,
 commission int,
 num_sales int)
GO

INSERT INTO dbo.Wages
VALUES (1,10,NULL, NULL, NULL),
(2, NULL, 41600, NULL, NULL),
(3, 12, NULL, NULL, NULL),
(4, NULL, NULL, 10000, 5)

SELECT *,
COALESCE(hourly_wage*40*52, salary,
        commission * num_sales)
FROM dbo.Wages
```

EmpID	hourly_wage	salary	commission	num_sales	(No column name)
1	10	NULL	NULL	NULL	20800
2	NULL	41600	NULL	NULL	41600
3	12	NULL	NULL	NULL	24960
4	NULL	NULL	10000	5	50000

FORMAT Function

```
DECLARE @date DATETIME = GETDATE()
SELECT @date as 'Unformatted_Date',
    FORMAT(@date, 'd', 'en-US') AS 'US English Dates'
```

Unformatted_Date	US English Dates
2012-08-16 00:00:00.000	8/16/2012

```
DECLARE @numbers SMALLMONEY = 5.4321
SELECT @numbers as'Unformatted_Number',
    FORMAT(@numbers, 'N', 'en-US') AS 'Number_Format',
    FORMAT(@numbers, 'G', 'en-US') AS 'General_Format',
    FORMAT(@numbers, 'C', 'en-US') AS 'Currency_Format'
```

Unformatted_Number	Number_Format	General_Format	Currency_Format
5.4321	5.43	5.4321	$5.43

Aggregate Functions and Group By Examples

```
SELECT COUNT(orderid) as CountOrders,
       MAX(orderdate) as MaxOrderDate,
       AVG(freight) as AvgFreight
FROM Sales.Orders
```

Aggregate Functions

```
SELECT orderid,
    SUM(unitprice * qty) as OrderTotal
FROM Sales.OrderDetails
WHERE orderid < 10260
GROUP BY orderid
HAVING SUM(unitprice * qty) > 1000
ORDER BY OrderTotal
```

Group By and Having

Using Aggregate Functions with NULL values

```
CREATE TABLE CountTable
(Numbers tinyint)

INSERT CountTable
VALUES (25), (10), (NULL),
       (30), (45), (NULL)
```

CountNoNulls	CountWithNulls	AvgNumber	SumDivCount	AvgIsNull
4	6	27	18	18

```
SELECT COUNT(Numbers) AS CountNoNulls,
       COUNT(*) AS CountWithNulls,
       AVG(Numbers) AS AvgNumber,
       SUM(Numbers)/Count(*) AS SumDivCount,
       AVG(ISNULL(Numbers,0)) AS AvgIsNull
FROM CountTable
```

Self-Contained Subqueries

```
SELECT orderid, productid, unitprice, qty
FROM Sales.OrderDetails
WHERE orderid =
    (SELECT MAX(orderid) as lastorder
     FROM Sales.Orders)
```

Scalar Subquery

```
SELECT custid, empid, orderid, orderdate
FROM Sales.Orders
WHERE custid IN
    (SELECT custid FROM Sales.Customers
        WHERE country = 'Mexico')
```

Multi-Value Subquery

Correlated Subqueries

```
SELECT custid, empid, orderid, orderdate
FROM Sales.Orders as O1
WHERE orderdate =
    (SELECT MAX(orderdate)
        FROM Sales.Orders AS O2
        WHERE O2.empid = O1.empid)
ORDER BY empid, orderdate
```

custid	empid	orderid	orderdate
65	1	11077	2008-05-06 00:00:00.000
58	2	11073	2008-05-05 00:00:00.000
44	2	11070	2008-05-05 00:00:00.000
37	3	11063	2008-04-30 00:00:00.000
9	4	11076	2008-05-06 00:00:00.000
74	5	11043	2008-04-22 00:00:00.000
10	6	11045	2008-04-23 00:00:00.000
73	7	11074	2008-05-06 00:00:00.000

Creating a Scalar Function

```
CREATE FUNCTION EndofMonth
(@StartDate Date)
RETURNS DATE
AS
BEGIN
RETURN
    DATEADD(D, -1,
    DATEADD(M, 1,
    CAST(
    CAST(DATEPART(M,@StartDate) as varchar(2))
     + '/1/' +
    CAST(DATEPART(YYYY,@StartDate) AS varchar(4))
    AS DATE)))
END
```

Creating Views and Table-Valued Functions

```
CREATE VIEW jd_vwPhoneList
AS
SELECT lastname + ', ' + firstname as FullName,
       phone as Phone
FROM HR.Employees
GO
```

User-Defined View

```
CREATE FUNCTION jd_funPhoneList
(@empid as int)
RETURNS table
AS
RETURN
SELECT lastname + ', ' + firstname as FullName,
       phone as Phone
FROM HR.Employees
WHERE empid = @empid
GO
```

Table-Valued Function

Derived Tables and Common Table Expressions

```
SELECT FullName, Phone
FROM (SELECT lastname + ', ' + firstname as FullName,
       phone as Phone FROM HR.Employees) as DerivedTable
WHERE FullName like 'd%'
GO
```

Derived Table

```
WITH CTE_PhoneList AS
(SELECT lastname + ', ' + firstname as FullName,
       phone as Phone FROM HR.Employees)
SELECT FullName, Phone
FROM CTE_PhoneList
WHERE FullName like 'd%'
GO
```

Common Table Expression

SET Operators

- **UNION** returns results from both sets without duplicates
- **UNION ALL** returns results from both sets with duplicates
- **INTERSECT** only returns results that appear in both sets
- **EXCEPT** returns rows from first set unless it is in second set

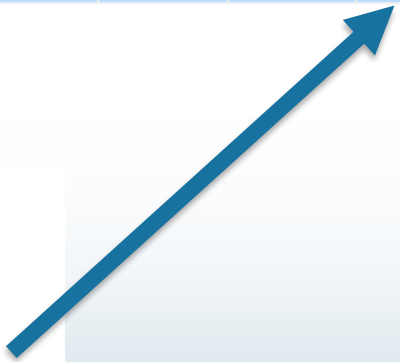
```
1  --Four Londons, Two Seattles, Tacoma, Kirkland, Redmond
2  SELECT City, Country FROM HR.Employees -- 9 employees
3  --UNION (ALL), INTERSECT, EXCEPT
4  SELECT City, Country FROM Sales.Customers-- 91 customers
5  --6 Londons, Seattle, Kirkland, No Tacoma, No Redmond
```

Ranking Functions Example

```
CREATE TABLE ProductsSold
(ID tinyint IDENTITY,
 Name varchar(10),
 Sold smallint)
GO
INSERT INTO ProductsSold
VALUES ('John', 2200), ('Deidre', 2100),
       ('Greg', 2000), ('Elizabeth', 1800),
       ('Breanna',1800), ('Jeff', 1500),
       ('Ken', 1500), ('Cassie', 1300)
```

```
SELECT Name, Sold,
RANK() OVER(Order By Sold DESC) as 'Rank',
DENSE_RANK() OVER(Order By Sold DESC) as 'Dense',
ROW_NUMBER() OVER(Order By Sold DESC) as 'RowNum',
NTile(2) OVER(Order By Sold DESC) as 'NTile(2)'
FROM ProductsSold
```

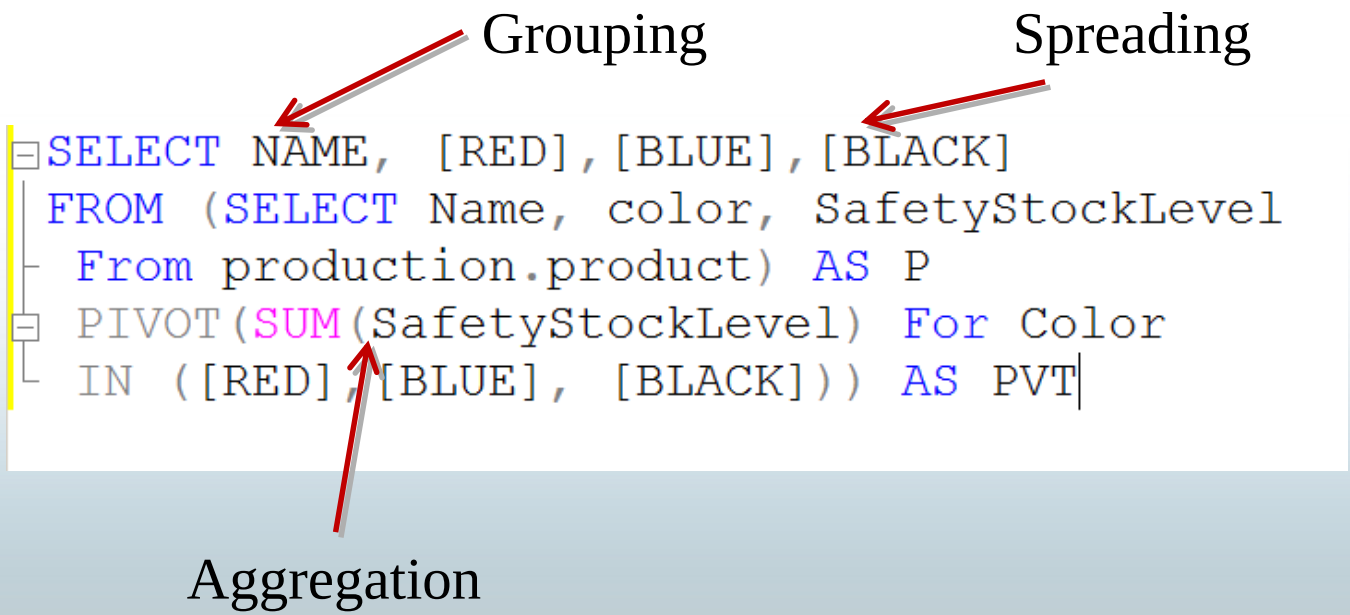
Name	Sold	Rank	Dense	RowNum	NTile(2)
John	2200	1	1	1	1
Deidre	2100	2	2	2	1
Greg	2000	3	3	3	1
Elizabeth	1800	4	4	4	1
Breanna	1800	4	4	5	2
Jeff	1500	6	5	6	2
Ken	1500	6	5	7	2
Cassie	1300	8	6	8	2



Elements of PIVOT

Pivoting includes three phases:

1. Grouping determines which elements in the FROM clause gets a row in the result set
2. Spreading provides the distinct values to be pivoted across
3. Aggregation performs an aggregation function (such as SUM)

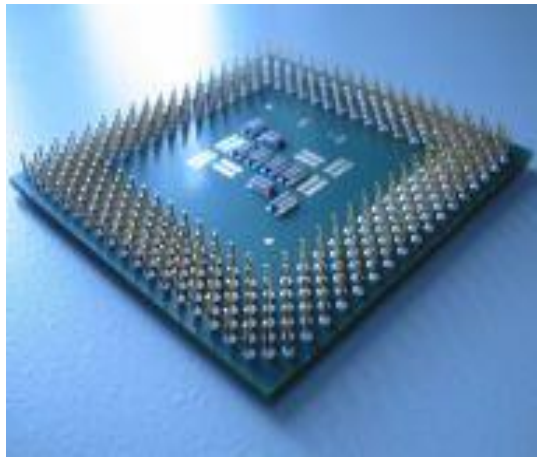


ROLLLUP and CUBE example

```
Select ProductID, Shelf, SUM(Quantity)
 From Production.ProductInventory
 Where ProductID < 6
 Group By Rollup(ProductID, Shelf)

Select ProductID, Shelf, SUM(Quantity)
 From Production.ProductInventory
 Where ProductID < 6
 Group By Cube(ProductID, Shelf)
```

Batches vs Transactions

Batches



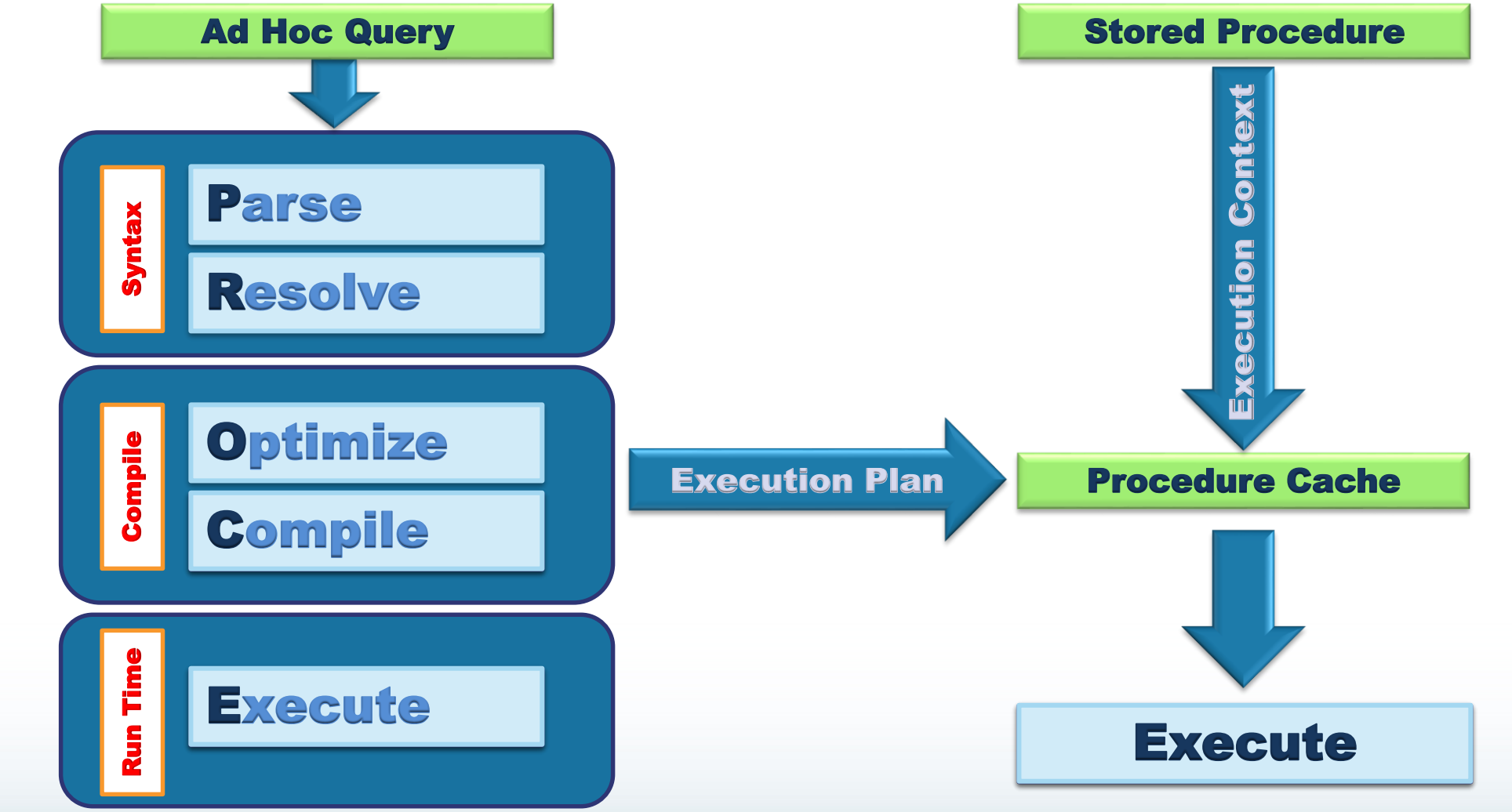
Sends Code to the Processor

VS

Transactions

TSQL2012.mdf

Modifies Data in the Database



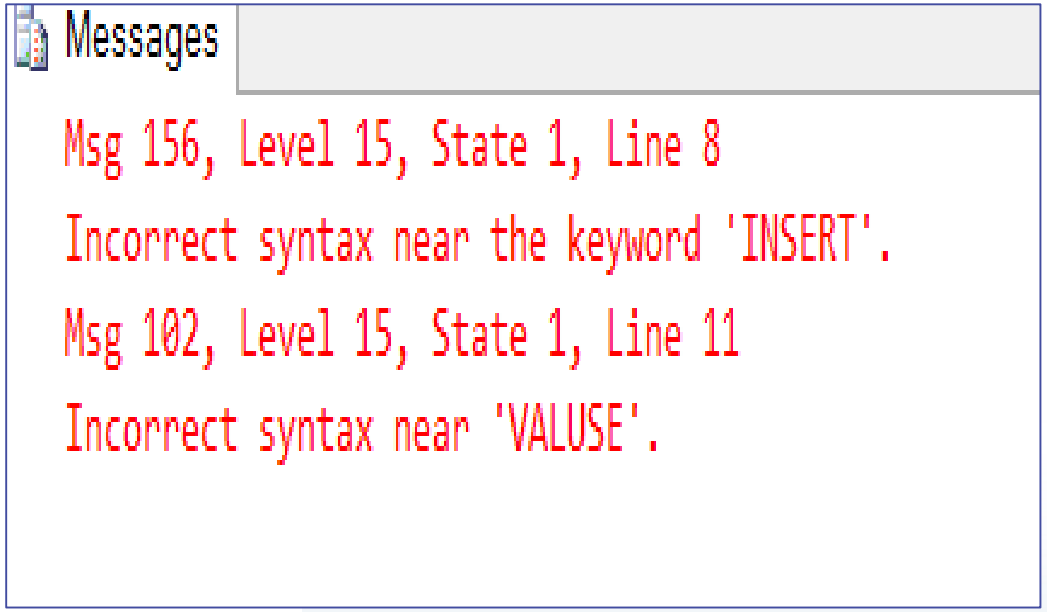
SQL

Sets

empid	lastname	firstna...	title	titleofcourt...	birthdate
1	Davis	Sara	CEO	Ms.	1958-12-08 00:00:00.000
2	Funk	Don	Vice President, Sales	Dr.	1962-02-19 00:00:00.000
3	Lew	Judy	Sales Manager	Ms.	1973-08-30 00:00:00.000
4	Peled	Yael	Sales Representative	Mrs.	1947-09-19 00:00:00.000
5	Buck	Sven	Sales Manager	Mr.	1965-03-04 00:00:00.000

Working with Batches

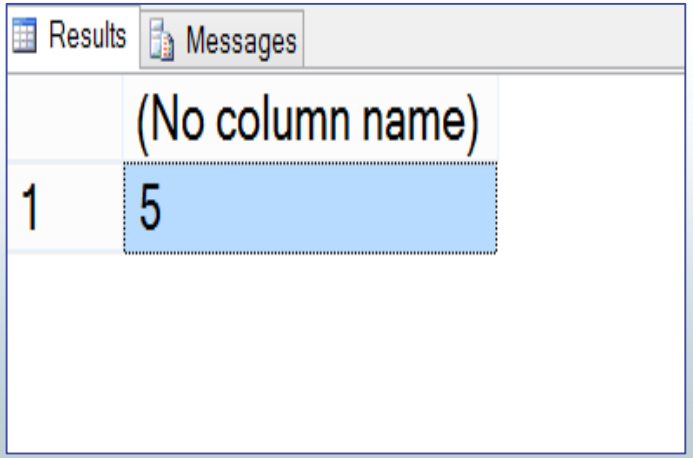
```
CREATE SCHEMA Accounting Authorization dbo
CREATE TABLE BankAccounts
     (AcctID tinyint IDENTITY,
      AcctName varchar(15),
      Balance smallmoney,
      ModifiedDate date)

INSERT INTO Accounting.BankAccounts
VALUES ('John', 500, GETDATE())
INSERT INTO Accounting.BankAccounts
VALUSE ('Armando', 750, GETDATE())
GO
```



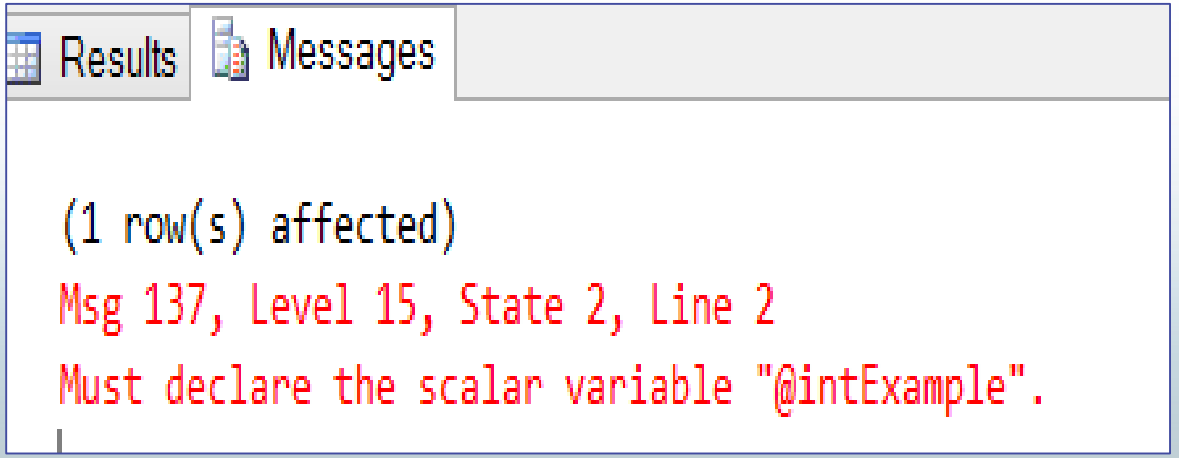
Batches and Variable Scope

```
DECLARE @intExample as int = 5
SELECT @intExample
GO

SELECT @intExample
```

First Batch Successful



Second Batch Fails



Creating a Synonym

```
CREATE SYNONYM dbo.AcctBal
FOR Accounting.BankAccounts

SELECT * FROM Accounting.BankAccounts
SELECT * FROM dbo.AcctBal
SELECT * FROM AcctBal
```

Inserting Records into an IDENTITY Field

```
SET IDENTITY_INSERT Accounting.BankAccounts ON
BEGIN TRAN
    INSERT INTO Accounting.BankAccounts
    (AcctID, AcctName, Balance, ModifiedDate)
    VALUES (29,'Kelli', 1250, GETDATE()),
           (27,'Jessica', 1005, GETDATE()),
           (18,'Maddison', 745, GETDATE()),
           (31,'Alicen', 555, GETDATE()),
           (15,'Molly', 790, GETDATE()),
           (34,'Amy', 650, GETDATE()),
           (32,'Logan', 1050, GETDATE()),
           (33,'Tommy', 450, GETDATE()),
           (36,'David', 850, GETDATE()),
           (22,'Reagan', 630, GETDATE())
COMMIT TRAN
SET IDENTITY_INSERT Accounting.BankAccounts OFF
```

Adding Records using a WHILE Loop

```
DECLARE @AcctID AS tinyint = 1

WHILE @AcctID < 10
    BEGIN
        INSERT INTO Accounting.BankAccounts
            (AcctName, Balance, ModifiedDate)
        SELECT lastname, 100 * @AcctID, GETDATE()
            FROM HR.Employees
            WHERE empid = @AcctID
        SET @AcctID += 1
    END
```

Altering a Stored Procedure

```
ALTER PROCEDURE spAccountTransfer
 (@Amount smallmoney, @a1 tinyint, @a2 tinyint)
AS
SET NOCOUNT ON

UPDATE Accounting.BankAccounts
SET Balance -= @Amount
WHERE AcctID = @a1

UPDATE Accounting.BankAccounts
SET Balance += @Amount
WHERE AcctID = @a2

PRINT 'Transfer Complete'
GO
```

Transactions must pass the ACID test

Atomicity – **All or Nothing**

Consistent – **Only valid data**

Isolated – **No interference**

Durable – **Data is recoverable**

Database Recovery

Restore

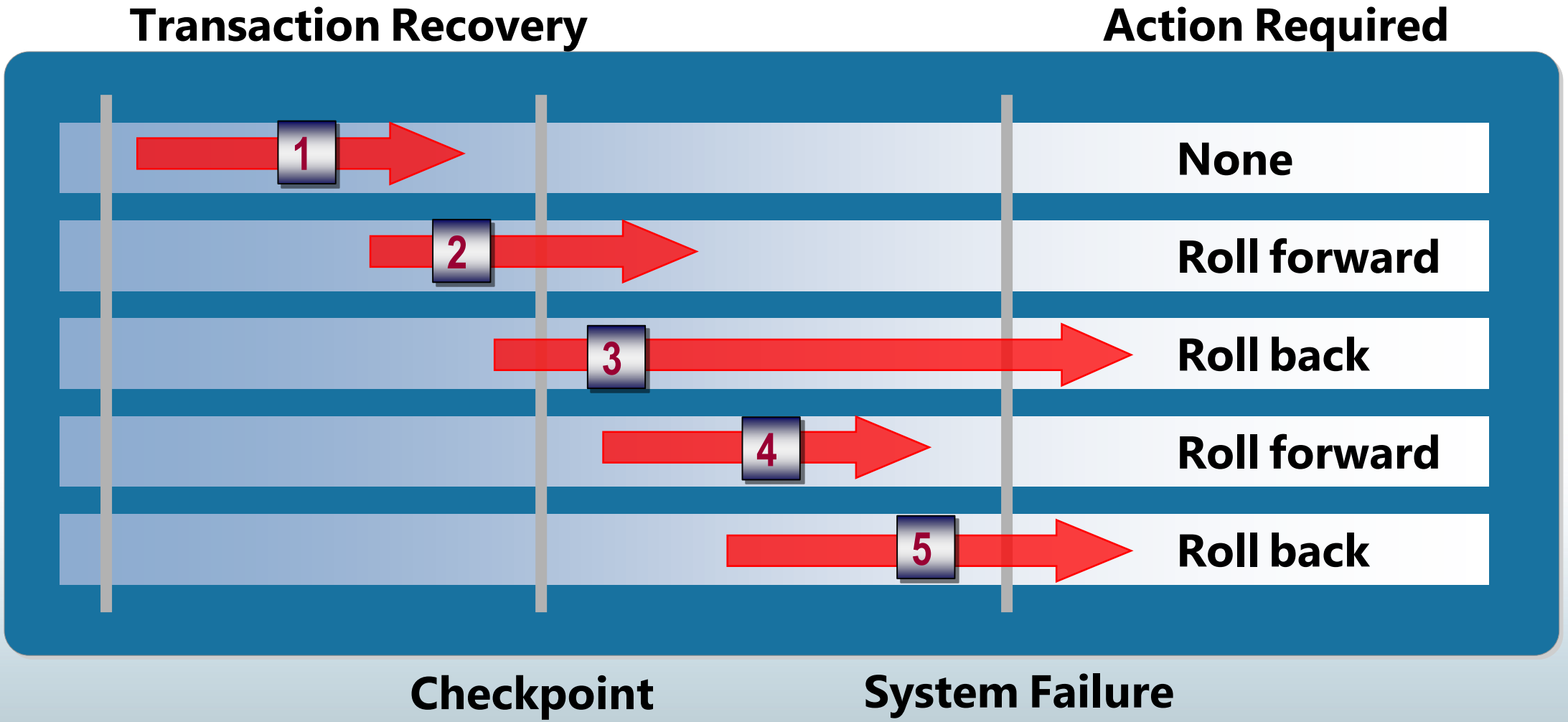
Data Copy – Create files and copies data

Recovery

Redo – Applies committed Transactions from log.

Undo – Rolls back uncommitted transactions

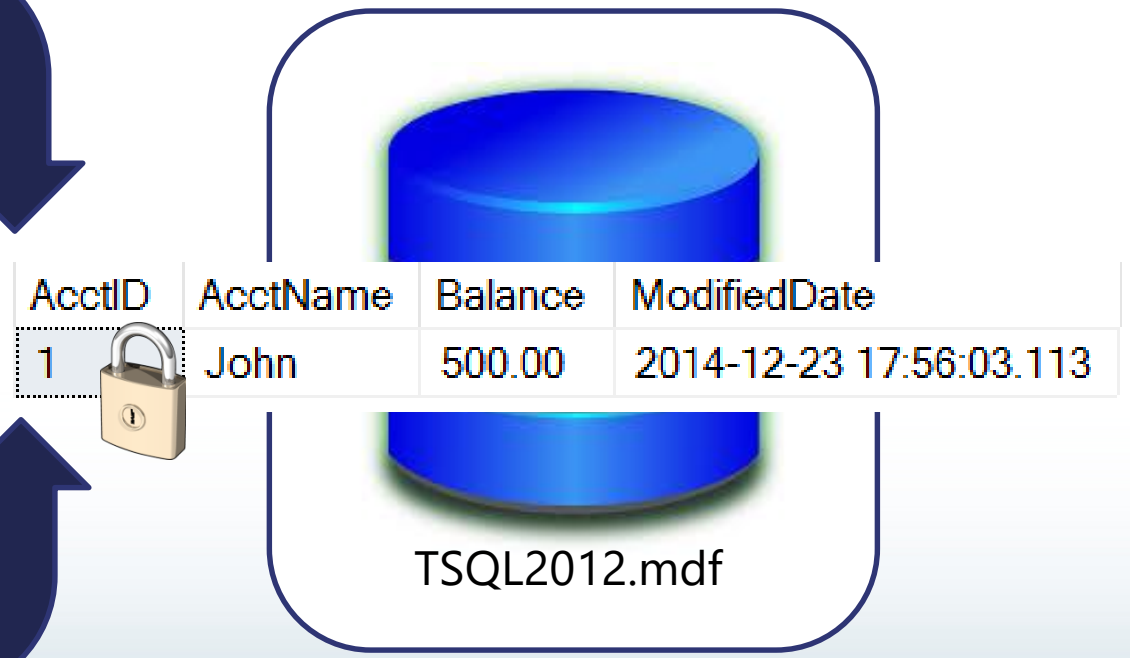
Transaction Log Recovery



What is a Lock?

```
UPDATE Accounting.BankAccounts
SET Balance -= 300
WHERE AcctID = 1
```

Transaction 1



Transaction 2

```
UPDATE Accounting.BankAccounts
SET Balance -= 400
WHERE AcctID = 1
```

Auto Commit Transactions without Error Handling

```
UPDATE Accounting.BankAccounts
SET Balance -= 200
WHERE AcctID = 1

UPDATE Accounting.BankAccounts
SET Balance += 200
WHERE AcctID = 2
```

TSQL2012.ldf

Checkpoint

TSQL2012.mdf

Explicit Transactions without Error Handling

```
BEGIN TRANSACTION BankTransfer
    UPDATE Accounting.BankAccounts
    SET Balance -= 2/0
    WHERE AcctID = 1

    UPDATE Accounting.BankAccounts
    SET Balance += 200
    WHERE AcctID = 2
COMMIT TRANSACTION
```

TSQL2012.ldf

Checkpoint

TSQL2012.mdf

Explicit Transactions with Error Handling

```
BEGIN TRY
    BEGIN TRANSACTION BankTransfer
        UPDATE Accounting.BankAccounts
        SET Balance -= 2/0
        WHERE AcctID = 1

        UPDATE Accounting.BankAccounts
        SET Balance += 200
        WHERE AcctID = 2
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
IF ERROR_NUMBER() = 8134
    BEGIN
        ROLLBACK TRANSACTION
        PRINT 'Divide by Zero ' +
        CAST(ERROR_NUMBER() as char(4))
    END
END CATCH
```

TSQL2012.ldf

Checkpoint

TSQL2012.mdf