SQL SERVER

YOUR Data, Any Place, Any Time

Course Outline

Installation

- Writing Queries Using Microsoft SQL Server T-SQL
- Implementing a Microsoft SQL Server Database
- Maintain Microsoft SQL Server Database
- SQL Server Business Intelligence

Installation Requirements

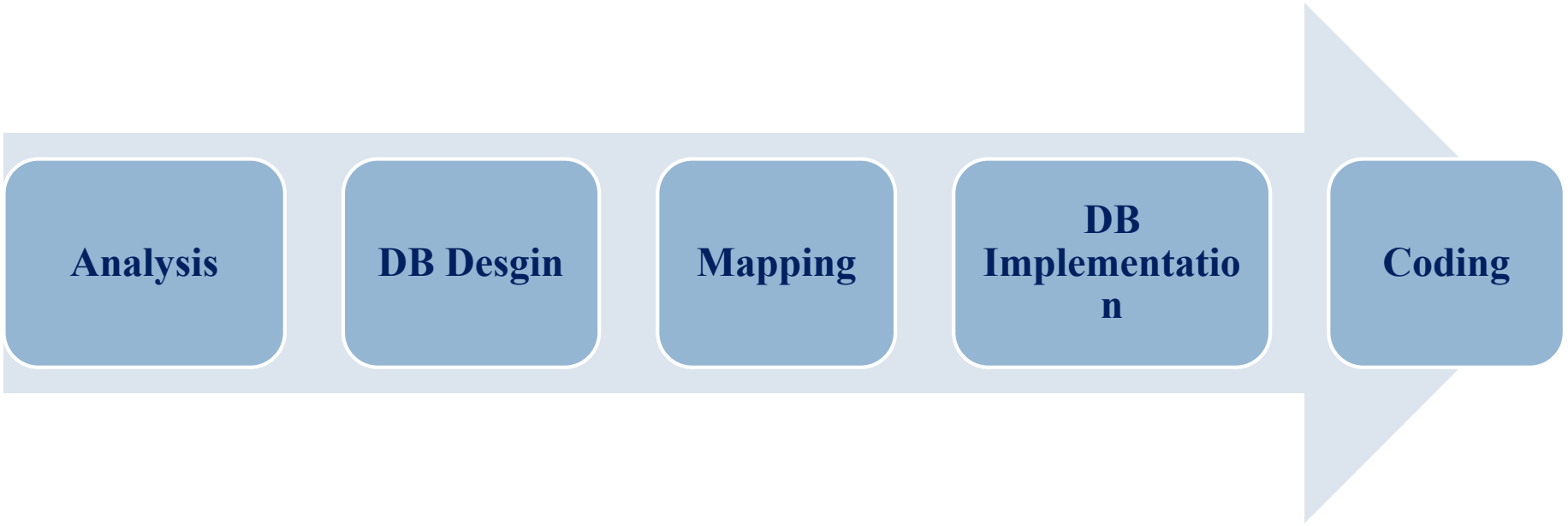
- Hardware
  - CPU 1.4 GHz processor
  - 1 GB memory
  - 4 GB disk space for Complete installation

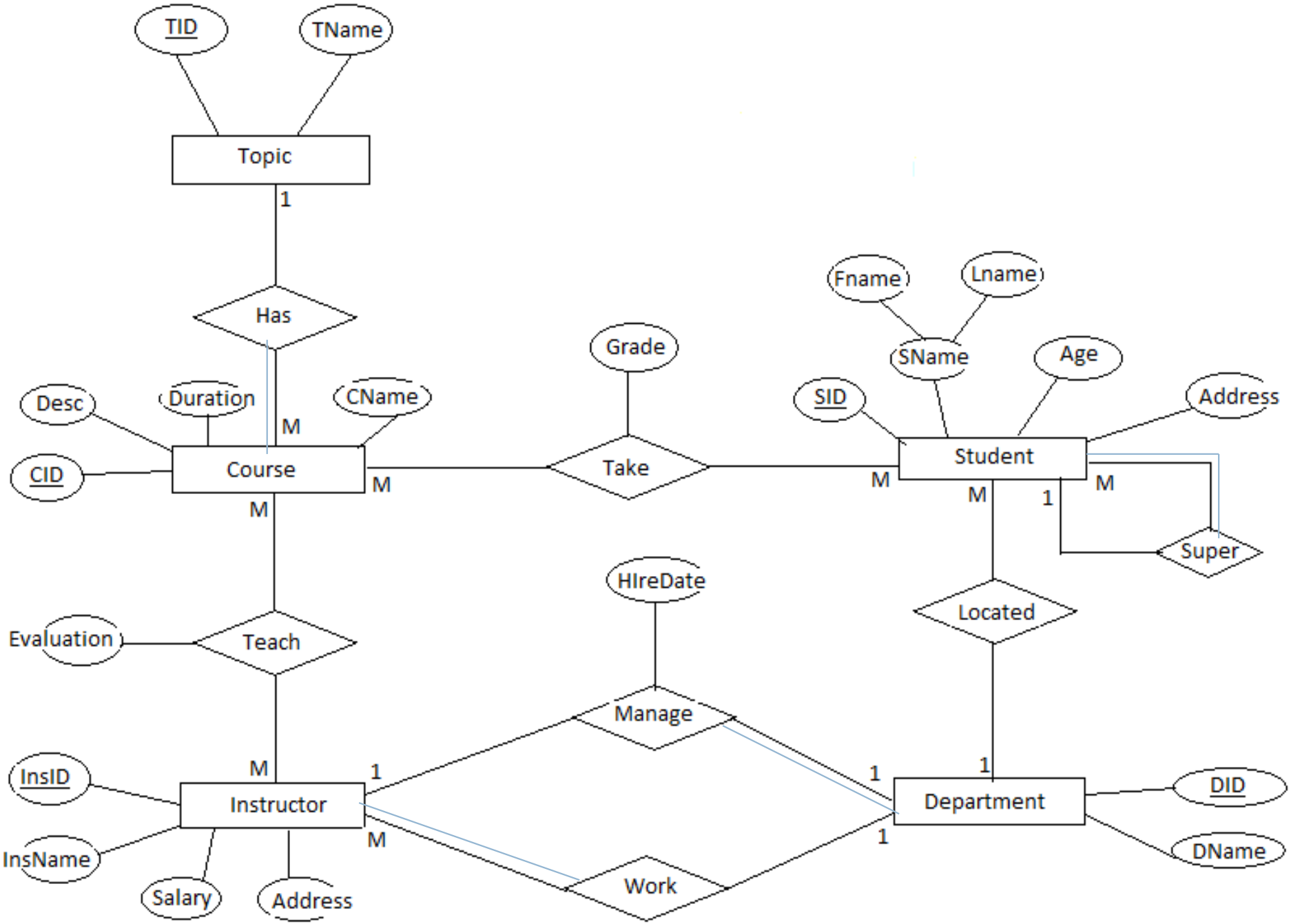
- Software
  - Windows Vista SP2, Windows Server 2008 SP1, Windows7 SP1
  - .Net 3.5 framework
  - Internet Explorer 7

Revision

- DB LifeCycle
- Overview of Relational Databases
- What Is Normalization?

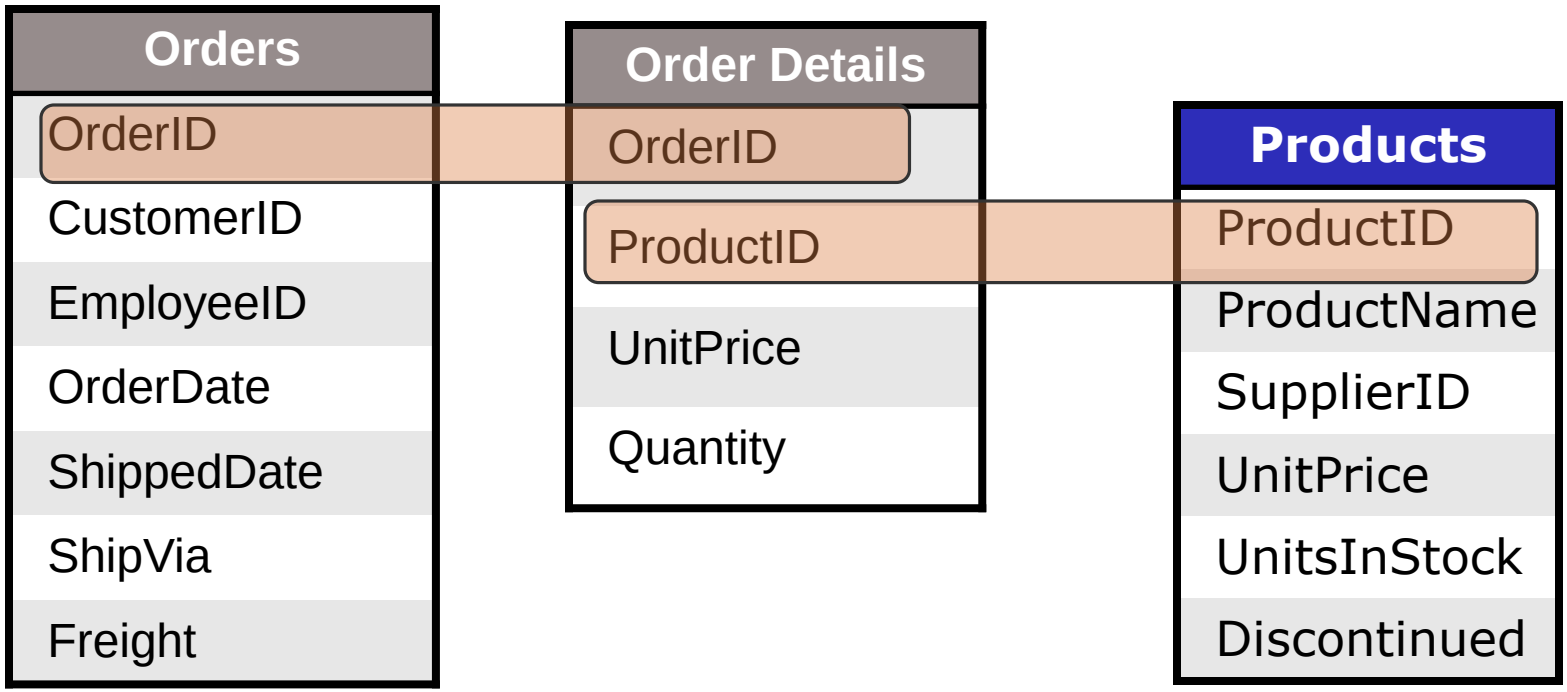
DB Life Cycle

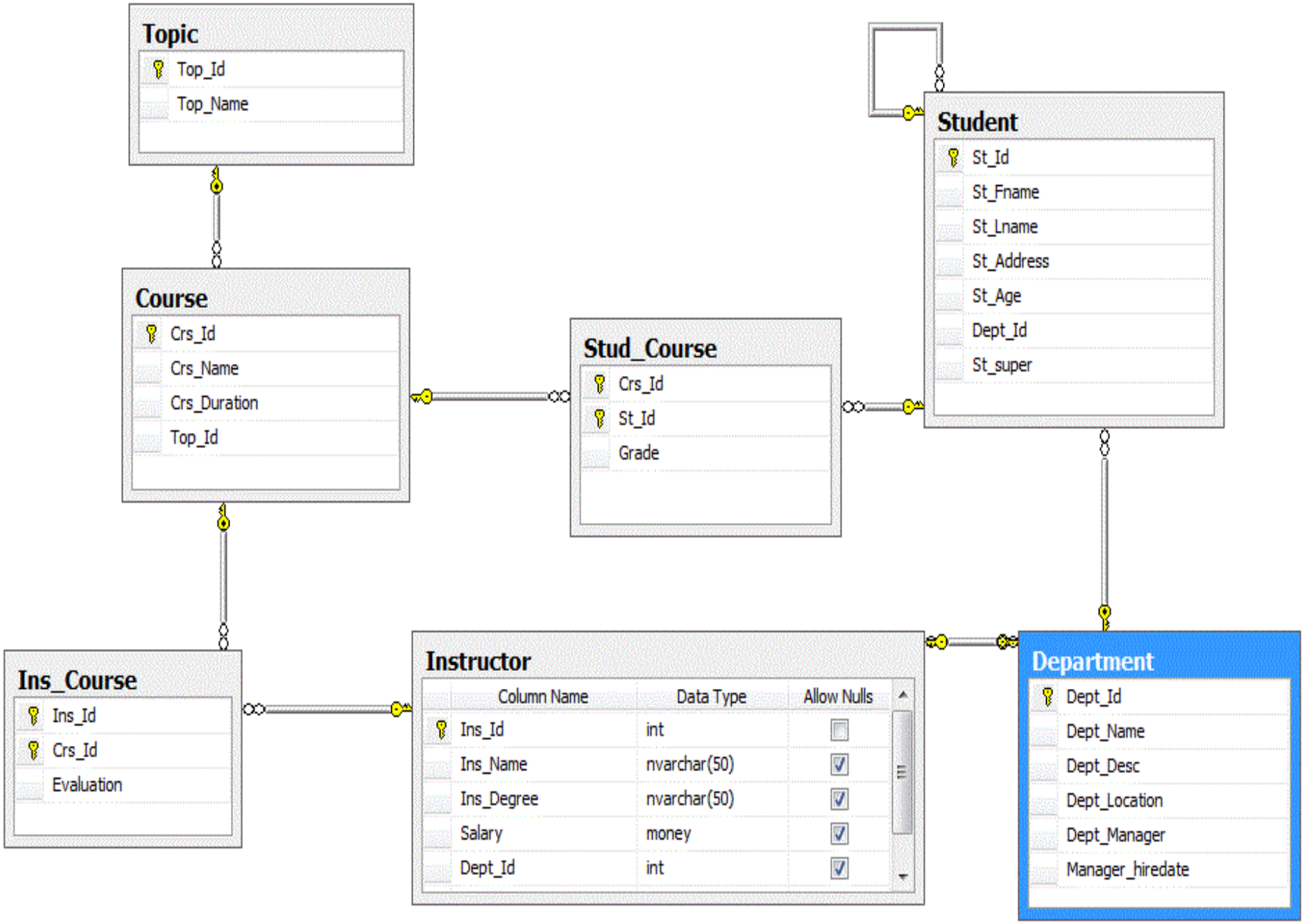



TID
TName
Topic
1
Has
M
Desc
Duration
CName
CID
Course
M
M
Grade
Take
M
Fname
Lname
SName
Age
SID
Address
Student
M
1
M
Super
HIreDate
Located
Evaluation
Teach
M
Manage
1
1
InsID
Instructor
M
1
Department
DID
InsName
Salary
Address
Work
DName

Overview of Relational Databases

- SQL Server is a Fully RDBMS
- The tables have one-to-many relationships




Topic
Top_Id
Top_Name
Course
Crs_Id
Crs_Name
Crs_Duration
Top_Id
Stud_Course
Crs_Id
St_Id
Grade
Student
St_Id
St_Fname
St_Lname
St_Address
St_Age
Dept_Id
St_super
Ins_Course
Ins_Id
Crs_Id
Evaluation
Instructor
Column Name
Data Type
Allow Nulls
Ins_Id
int
Ins_Name
nvarchar(50)
Ins_Degree
nvarchar(50)
Salary
money
Dept_Id
int
Department
Dept_Id
Dept_Name
Dept_Desc
Dept_Location
Dept_Manager
Manager_hiredate

What Is Normalization?

The process for removing redundant data from a database

Benefits

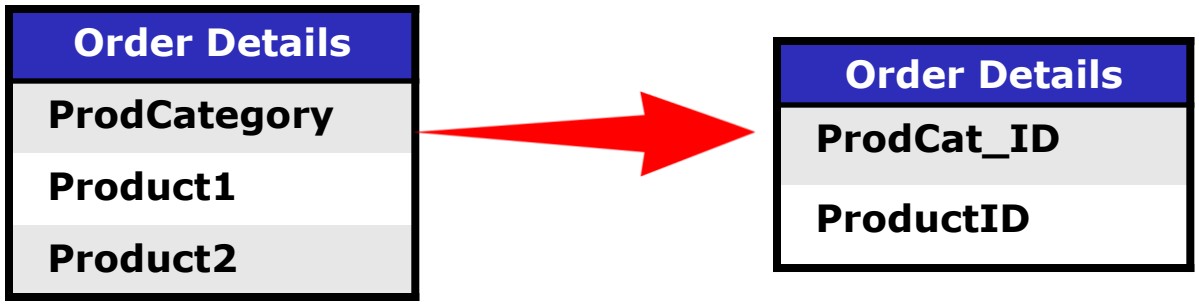
- ✓ Accelerates sorting and indexing
- ✓ Allows more clustered indexes
- ✓ Helps UPDATE performance
- ✓ More compact databases

Disadvantages

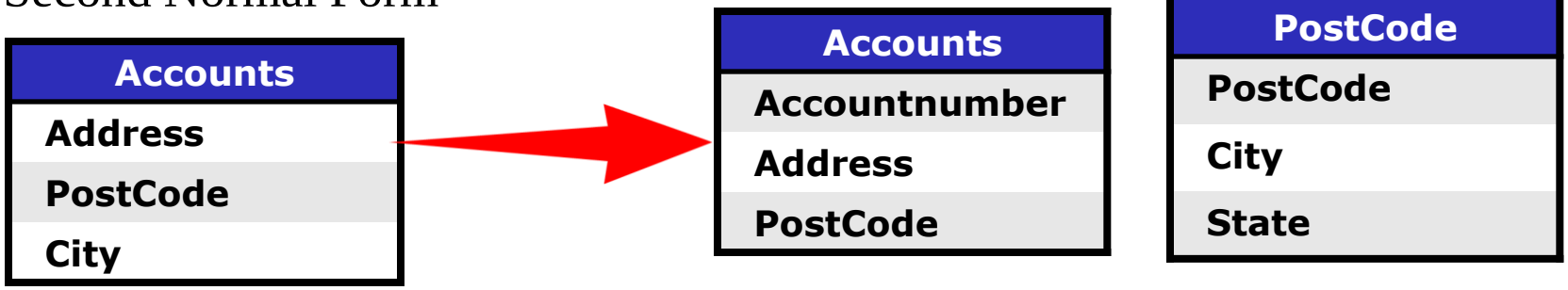
- Increase in tables to join
- Slower data retrieval
- Insertion of code in tables
- Difficulty in data model query

The Normalization Process

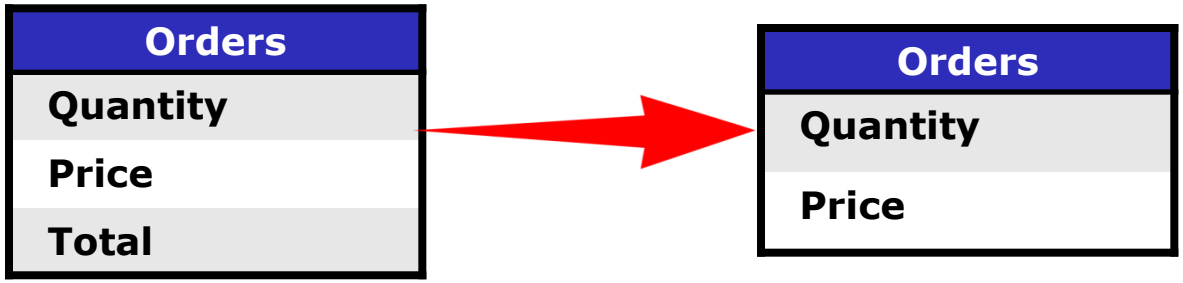
- First Normal Form



- Second Normal Form



- Third Normal Form



Normalization

- **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 with a primary key.

- **Second Normal Form**
- • Create separate tables for sets of values that apply to multiple records.
- • Relate these tables with a foreign key.

- **Third Normal Form**
- • Eliminate fields that do not depend on the key. X is fully dependent on Y “Primary key”
- • Transitive dependencies must be eliminated, so all records must rely only on the primary key.

Overview

- Overview of SQL Server
- Using Querying Tools
- SQL Server Databases
- SQL Server Database Objects
- Overview T-SQL
- Overview of Data Types

SQL Server Versions History

1st Generation	
SQL Server Version	Features
SQL 6.0/6.5 (1995)	First version designed specifically for Windows NT Replication
SQL Server 4.2 (1992)	Developed for Windows NT 3.1
SQL Server 1.0 (1989)	Developed by Microsoft, Sybase, and Ashton-Tate for OS/2

2nd Generation	
SQL Server Version	Features
SQL2000	Focus on Performance and Scalability XML support Data Mining Reporting Services
SQL Server 7.0 (1999)	Restructure of Relational Server Data Transformation Services Online Analytical Processing

SQL Server Versions History

3rd Generation	
SQL Server Version	Features
SQL 2014,2016,2017,2019	Security&Performance
SQL 2012	Always On Power View File Table Sequence Data Quality Service
SQL2008/SQL2008 R2	Power Pivot Enhance SharePoint Integration T-SQL (Ranking, Merge, Output) Improve and enhance for BI Tools
SQL2005	High Availability(includes DB Mirroring) Security Enhancements (DB Schema) Integration Services SQLCLR XML and Web services supports

SQL Server Editions

Edition	Description
Enterprise	For large scale, business-critical applications
Standard-Developer	For small/medium, departmental applications
BI Edition	For BI Services
Express	Entry level/learning edition
Azure	For Cloud

SQL Server Components

Server Component	Description
SQL Server Database Engine	Core service for storing and processing data
Analysis Services	Tools for creating and managing analytical processing
Reporting Services	Components for creating and deploying reports
Integration Services	Tools for moving, copying, and transforming data

Data Quality Service & Master Data Service

SQL Server Management Tools

Management tools	Description
SQL Server Management Studio	An environment to access, configure, manage, and administer SQL components
SQL Server Configuration Manager	An interface to provide management for SQL services, protocols, and client aliases
SQL Server Profiler	A GUI tool to profile and trace the Database Engine and Analysis Services
Database Engine Tuning Advisor	An application to create an optimal sets of indexes, indexed views, and partitions
BI Development Studio (SQL Server Data Tool)	An IDE for creating Analysis Services, Reporting Services, and Integration Services

SQL Server Database Engine Components

Components	Description
• Protocols	Ways to implement the external interface to the SQL Server
• Relational Engine	Interface into the storage engine, composed of services to interact with the underlying database storage components and features
• Storage Engine	Core of SQL Server, a highly scalable and available service for data storage, processing, and security

Tools for Querying SQL Server Databases

Tool	Description
SQL Server Management Studio	• Used for interactive creation of T-SQL scripts • To access, configure, manage, and create many other SQL Server Objects
Microsoft Office Excel (Reporting tools)	• A spreadsheet used by financial and business professional to retrieve data
SQLCMD	• A command used by administrators for command line and batch files processing `SQLCMD -S server\instance -i C:\script`
PowerShell	• An environment used by administrators for command line and batch processing

Database Objectives

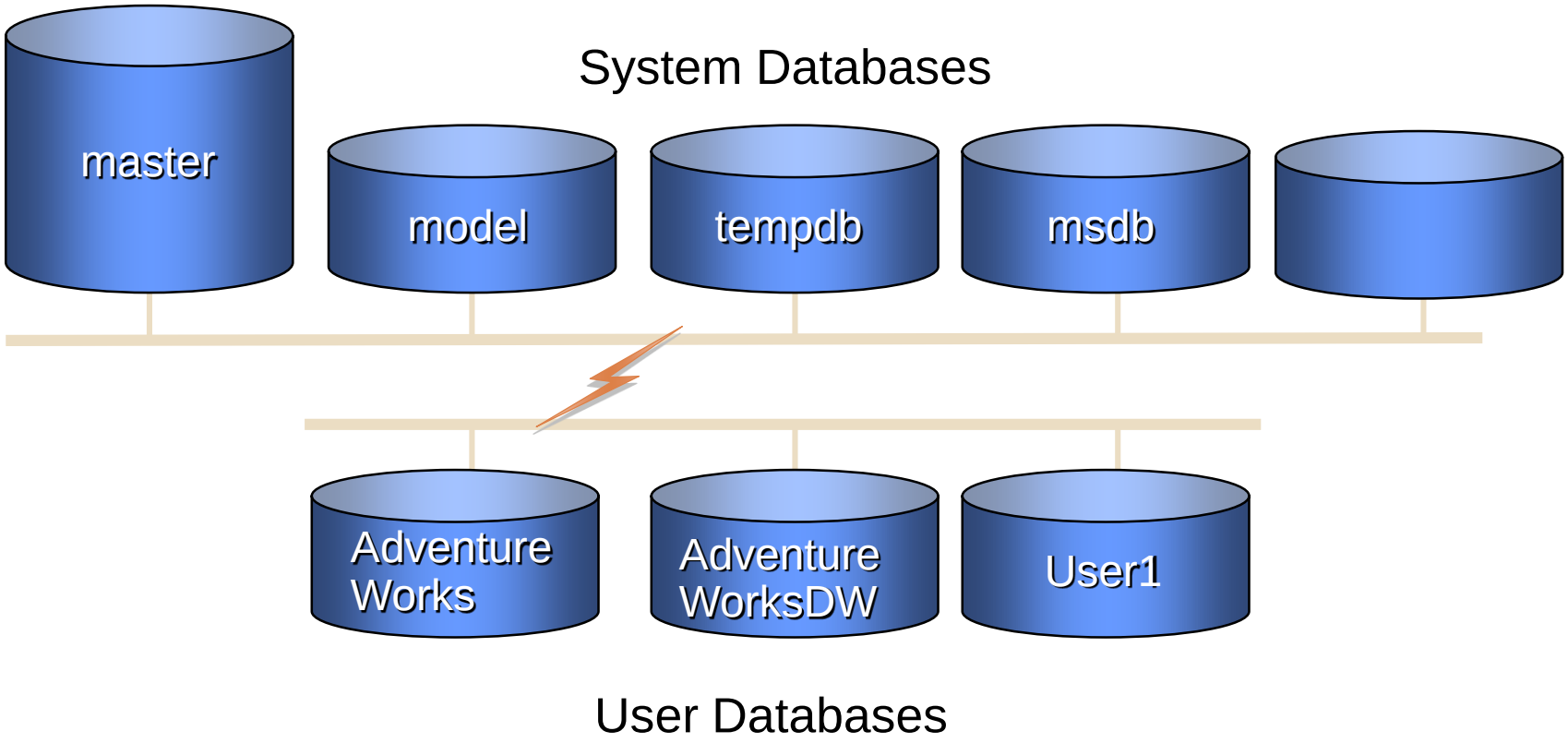
Objects	Notes
Tables	Contain all the data in SQL Server databases
Views	Act like a virtual table or a stored query
Indexes	Enable fast retrieval, built from one or more columns in table or view
Triggers	Execute a batch of SQL code when an insert, update or delete command is executed against a specific table
Procedures	Accept parameters, contain statements, and return values.
Constraints	Prevent inconsistent data from being placed in a column
Rules	Specify acceptable values that can be inserted in column

Authentication Modes

SQL Server Authentication Mode

- Windows Authentication
- Mixed (windows and SQL authentication)

SQL Server Databases



T-SQL History

- Developed in the early 1970
- ANSI-SQL defined by the American National Standards Institute
- Microsoft implementation is T-SQL, or Transact SQL
- Other implementations include PL/SQL and IBM's SQL Procedural Language.

Categories of T-SQL Statements

- DML – Data Manipulation Language
- DCL – Data Control Language
- DDL – Data Definition Language
- TCL - Transactional Control Language
- DQL - SQL Select Statements

Executing Queries

Executing queries occurs when in a query session by:

- Selecting the Execute Icon
- Pressing the F5 key

Note:

Select the Database Before Executing Query or write

Use Keyword + DB Name on top of the Query

Commenting T-SQL Code

- Comments are statements about the meaning of the code
- When used, there is no execution performed on the text

There are two ways to comment code using T-SQL:

- The use of a beginning /* and ending */ creates comments

```
/*
This is a comment
*/
```

- The double dash comments to the end of line

```
--This is a comment
```

Batch

- Recall that a batch is a series of one or more statements submitted and executed at the same time
- Example:

```
delete sales
     where stor_id = "5023"
     and ord_num = "AB-123-DEF-425-1Z3"
delete salesdetail
     where stor_id = "5023"
     and ord_num = "AB-123-DEF-425-1Z3"
select * from sales
     where stor_id = "5023"
select * from salesdetail
     where stor_id = "5023"
go
```

Batch Restrictions

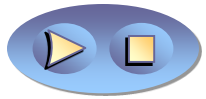
- These statements must be in their own batch:
  - **create default**
  - **create rule**
  - **create procedure**
  - **create trigger**
  - **declare cursor**
- **use** must be the last (or only) statement in a batch
- You cannot drop and recreate an object in the same batch
- You cannot bind a rule or default to a column and insert values into that column in the same batch

Introduction to Basic T-SQL Syntax

There are four primary properties to the SELECT statement

1. The number and attributes of the columns in the result set
2. The tables from which the result set data is retrieved
3. The conditions the rows in the source tables must meet
4. The sequence which the rows of the result set are ordered

```
SELECT ProductID, Name, ListPrice
FROM Production.Product
WHERE ListPrice > $40
ORDER BY ListPrice ASC
```



Querying and Filtering Data

Querying and Filtering Data

- Using the SELECT Statement
- Filtering Data
- Working with NULL Values
- Formatting Result Sets
- Performance Considerations for Writing Queries

SELECT Statement

- Elements of the SELECT Statement
- Retrieving Columns in a Table

Elements of the SELECT Statement

```
SELECT select_list
[INTO new_table_name]
 FROM table_source
[ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

Retrieving Columns in a Table

Displays All Columns in the Employee Table

```
USE AdventureWorks ;
GO
SELECT *
FROM HumanResources.Employee
```

Displays Only FirstName, LastName and JobTitle Columns

```
USE AdventureWorks ;
GO
SELECT FirstName, LastName, JobTitle
FROM HumanResources.Employee
```

Filtering Data

- Retrieving Specific Rows in a Table
- Filtering Data by Using Comparison Operators
- Filtering Data by Using String Comparisons
- Filtering Data by Using Logical Operators
- Retrieving a Range of Values
- Retrieving a List of Values

Retrieving Specific Rows in a Table

Simple WHERE clause

```
USE ITI;
GO
SELECT  *
FROM Student
WHERE Age >20
```

WHERE Clause Using a Predicate

```
USE ITI;
GO
SELECT *
FROM Instructor
WHERE Salary IS NULL;
```

Types of T-SQL Operators

Type	Operators
• Arithmetic operators	• +, -, *, /, % *Vacation + SickLeave AS 'Total PTO'*
• Assignment operator	• = *SET @MyCounter = 1*
• Comparison operators	• =, <, >, <>, !, >=, <= *IF (@MyProduct <> 0) …*
• Logical operators	• AND, OR, NOT *WHERE Department = 'Sales' AND (Shift = 'Evening' OR Shift = 'Night')*
• String concatenation operator	• + *SELECT LastName + ', ' + FirstName AS Moniker*

Using Comparison Operators

- Comparison operators test whether two expressions are the same.
- Comparison operators return a Boolean value of TRUE, FALSE, or UNKNOWN.

Scalar Comparison Operators

=, <>, >, >=, <, <=, !=

```
USE AdventureWorks ;
GO
SELECT FirstName, LastName, MiddleName
FROM Person.Person
WHERE ModifiedDate >= '01/01/2004'
```

Using String Comparisons

- String Comparisons are used for data types of text, ntext, char, nchar, varchar, and nvarchar
- Predicates are available for full or partial match comparisons

```
WHERE LastName = 'Johnson'
```

```
WHERE LastName LIKE 'Johns%n'
```

```
WHERE CONTAINS(LastName, 'Johnson')
```

```
WHERE FREETEXT(Description, 'Johnson')
```

Using Logical Operators

- Logical operators are used to combine conditions in a statement

Returns only rows with first name of ‘John’ and last name of ‘Smith’

```
WHERE FirstName = ‘John’ AND LastName = ‘Smith’
```

Returns all rows with first name of ‘John’ and all rows with last name of ‘Smith’

```
WHERE FirstName = ‘John’ OR LastName = ‘Smith’
```

Returns all rows with first name of ‘John’ and last name not equal to ‘Smith’

```
WHERE FirstName = ‘John’ AND NOT LastName = ‘Smith’
```

Operator Precedence

~ (Bitwise Not)

*(Multiply), /(Division), %(Modulo)

+(Positive), -(Negative), +(Add), (+Concatenate),
-(Subtract), ^(Bitwise Exclusive OR), |(Bitwise OR)

Comparisons
=, >, <, >=, <=, <>, !=, !>, !<

NOT

AND

ALL, ANY, BETWEEN, IN, LIKE, OR, SOME

=(Assignment)

Retrieving a Range of Values

- BETWEEN tests for data values within a range of values.

```
SELECT *
FROM Student
WHERE age BETWEEN 25 AND 30
```

- BETWEEN uses the same logic as >= AND <=

```
SELECT *
FROM Student
WHERE age>=25 AND age<=30
```

Retrieving a List of Values

- IN tests a column's values against a list of possible values.

```
SELECT SalesOrderID, OrderQty, ProductID, UnitPrice
FROM Sales.SalesOrderDetail
WHERE ProductID IN (750, 753, 765, 770)
```

- IN uses the same logic as multiple comparisons with the OR predicate between them

```
SELECT SalesOrderID, OrderQty, ProductID, UnitPrice
FROM Sales.SalesOrderDetail
WHERE ProductID = 750 OR ProductID = 753
      OR ProductID = 765 OR ProductID = 770
```

Working with NULL Values

- NULL is an UNKNOWN value
- NULL is not a zero (0) value or an empty string
- NULL values are not equal
- Comparing NULL to any other value returns UNKNOWN
- A NULL value cannot be included in a calculation.
- The special SPARSE keyword can be used to conserve space in columns that allow NULL values.
- Use IS NULL to test for NULL values in an argument.

Work with NULL Values

ISNULL() returns a given value if the column value is NULL

```
SELECT ISNULL(st_fname,’ ’)
FROM Student;
```

NULLIF() returns NULL if both specified expressions are equal

```
select Nullif(st_age,dept_id) from Student
where St_Id=7
```

COALESCE() returns the first non NULL expression among its arguments, similar to a CASE statement

```
SELECT CAST(COALESCE(hourly_wage * 40 * 52,  salary,
   commission * num_sales) AS money) AS 'Total Salary'
FROM wages
```

Formatting Result Sets

- Sorting Data
- Eliminating Duplicate Rows
- Labeling Columns in Result Sets
- Using String Literals
- Using Expressions

Sorting Data

```
SELECT LastName, FirstName, MiddleName
FROM Person.Person
ORDER BY LastName, FirstName
```

Eliminating Duplicate Rows

```
SELECT DISTINCT LastName, FirstName, MiddleName
FROM Person.Person
ORDER BY LastName, FirstName
```

Labeling Columns in Result Sets

- Aliases are used to create custom column headers in the result set display.
- You can rename actual or derived columns
- The optional **AS** clause can be added to make the statement more readable.
- Both statements below are equivalent

```
SELECT salary*12 as [annual salary]
From instructor
```

Using String Literals

- String Literals:
- Are constant values.
- Can be inserted into derived columns to format data.
- Can be used as alternate values in functions, such as the ISNULL() function.

```
SELECT (LastName + ', ' + FirstName + ' ' +
ISNULL(SUBSTRING(MiddleName, 1, 1), ' ')) AS Name
FROM Person.Person
ORDER BY LastName, FirstName, MiddleName
```

Using Expressions

- Using mathematical expressions in SELECT and WHERE clauses
- Using functions in expressions

```
SELECT Name, ProductNumber, ListPrice AS OldPrice, (ListPrice *
   1.1) AS NewPrice
FROM Production.Product
WHERE ListPrice > 0 AND (ListPrice/StandardCost) > .8
```

```
SELECT Name, ProductNumber, ListPrice AS OldPrice, (ListPrice *
   1.1) AS NewPrice
FROM Production.Product
WHERE SellEndDate < GETDATE()
```

SQLCMD

The **sqlcmd** command line utility enables us to interact with SQL Server from the command line. It can:

- Define connection setting information
- Execute Transact-SQL (T-SQL) statements
- Call external scripts
- Use environment variables
- Store the output results of executed queries in a specified text file

Learn SQLCMD

Queries -2

Grouping and Summarizing Data

Grouping and Summarizing Data

- Summarizing Data by Using Aggregate Functions
- Summarizing Grouped Data
- Ranking Grouped Data
- Creating Crosstab Queries

Summarizing Data by Using Aggregate Functions

- Aggregate Functions Native to SQL Server
- Using Aggregate Functions with NULL Values
- CLR Integration, Assemblies
- Implementing Custom Aggregate Functions

Aggregate Functions Locations

- The select list of a SELECT statement
- A COMPUTE or COMPUTE BY clause
- A HAVING clause

```
USE ITI

SELECT MAX(Age)

FROM student

GROUP BY dept_id;
```

Using Aggregate Functions With NULL Values

- Most aggregate functions ignore NULL values
- NULL values may produce unexpected or incorrect results
- Use the ISNULL function to correct this issue

```
USE ITI

SELECT AVG(ISNULL(age,0)) AS ‘AVG’

FROM student
```

- The COUNT(*) function is an exception and returns the total number of records in a table

Summarizing Grouped Data

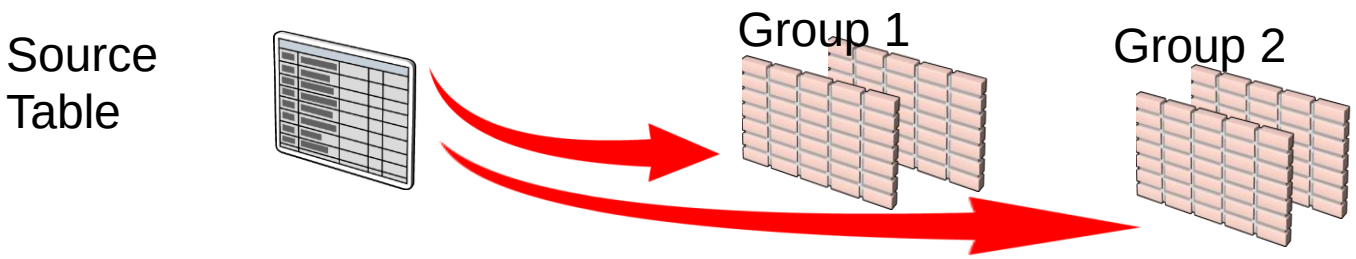
- Using the GROUP BY clause
- Filtering Grouped Data by Using the HAVING Clause
- Building a Query for Summarizing Grouped Data – GROUP BY
- Examining How the ROLLUP and CUBE Operators Work
- Using the ROLLUP and CUBE Operators
- Using the COMPUTE and COMPUTE BY Clauses
- Building a Query for Summarizing Grouped Data - COMPUTE
- Using GROUPING SETS

Using the GROUP BY Clause

- Specifies the groups into which the output rows must be placed
- Calculates a summary value for aggregate functions

```
SELECT SalesOrderID, SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
ORDER BY SalesOrderID
```

SalesOrderID	SubTotal
1	23761
2	45791
3	75909
4	19900



Filtering Grouped Data by Using the HAVING Clause

- Specifies a search condition for a group
- Can be used only with the SELECT statement

```
SELECT dept_id, count(dept_id) AS #ofstudents
FROM student
GROUP BY dept_id
HAVING count(dept_id) > 5
ORDER BY dept_id
```

SalesOrderID	SubTotal
43875	121761.939600
43884	115696.331324
44518	126198.336168
44528	108783.587200
44530	104958.806836
44795	104111.515642
46066	100378.907800
...	

Examining How the ROLLUP and CUBE Operators Work

- ROLLUP and CUBE generate summary information in a query
- ROLLUP generates a result set showing the aggregates for a hierarchy of values in selected columns

```
SELECT a, b, c, SUM ( <expression> )

FROM T

GROUP BY ROLLUP (a,b,c)
```

- CUBE generates a result set that shows the aggregates for all combination of values in selected columns

```
SELECT a, b, c, SUM (<expression>)

FROM T

GROUP BY CUBE (a,b,c)
```

Using the ROLLUP and CUBE Operators

```
SELECT ProductID,Shelf,SUM(Quantity) AS QtySum
FROM Production.ProductInventory
WHERE ProductID<6
GROUP BY ROLLUP(ProductID,Shelf)
```

ProductID	Shelf	QtySum
1	A	761
1	B	324
1	NULL	1085
2	A	791
2	B	318
2	NULL	1109
3	A	909
3	B	443
3	NULL	1352
4	A	900

```
SELECT ProductID,Shelf,SUM(Quantity) AS QtySum
FROM Production.ProductInventory
WHERE ProductID<6
GROUP BY CUBE(ProductID, Shelf)
```

ProductID	Shelf	QtySum
1	A	761
2	A	791
3	A	909
4	A	900
NULL	A	3361
1	B	324
2	B	318
3	B	443
4	B	442
NULL	B	1507

Using GROUPING SETS

- New GROUP BY operator that aggregates several groupings in one query
  - Eliminates multiple GROUP BY queries

```
SELECT T.[Group] AS 'Region', T.CountryRegionCode AS 'Country'
    ,S.Name AS 'Store', H.SalesPersonID
    ,SUM(TotalDue) AS 'Total Sales'
FROM Sales.Customer C
    INNER JOIN Sales.Store S
        ON C.StoreID  = S.BusinessEntityID
    INNER JOIN Sales.SalesTerritory T
        ON C.TerritoryID  = T.TerritoryID
    INNER JOIN Sales.SalesOrderHeader H
        ON C.CustomerID = H.CustomerID
WHERE T.[Group] = 'Europe'
    AND T.CountryRegionCode IN('DE', 'FR')
    AND SUBSTRING(S.Name,1,4)IN('Vers', 'Spa ')
GROUP BY GROUPING SETS
    (T.[Group], T.CountryRegionCode, S.Name, H.SalesPersonID)
ORDER BY T.[Group], T.CountryRegionCode, S.Name, H.SalesPersonID;
```

Region	Country	Store
NULL	NULL	NULL
NULL	NULL	NULL
NULL	NULL	NULL
NULL	NULL	Spa and Exercise Outfitters
NULL	NULL	Versatile Sporting Good Company
NULL	DE	NULL
NULL	FR	NULL
Europe	NULL	NULL

Supports additional options as ability to use with ROLLUP and CUBE operators

Queries -3

Joining Data from Multiple Tables

Joining Data from Multiple Tables

- Querying Multiple Tables by Using Joins
- Applying Joins for Typical Reporting Needs
- Combining and Limiting Result Sets

Querying Multiple Tables by Using Joins

- Fundamentals of Joins
- Categorizing Statements by Types of Joins
- Joining Data Using Inner Joins
- Joining Data Using Outer Joins
- Joining Data Using Cross Joins
- Identifying the Potential Impact of a Cartesian Product

Fundamentals of Joins

Joins:

- Select Specific Columns from Multiple Tables
  - JOIN keyword specifies that tables are joined and how to join them
  - ON keyword specifies join condition
- Query Two or More Tables to Produce a Result Set
  - Use Primary and Foreign Keys as join conditions
  - Use columns common to specified tables to join tables

Simplified JOIN Syntax:

```
FROM first_table join_type second_table
[ON (join_condition)]
```

Categorizing Statements by Types of Joins

- Inner Join
  - Includes equi-joins and natural joins
  - Use comparison operators to match rows
- Outer Join
  - Includes left, right, or full outer joins
- Cross Join
  - Also called Cartesian products
- Self Join
  - Refers to any join used to join a table to itself

Identifying the Potential Impact of a Cartesian Product

A Cartesian Product:

- ✓ Is defined as all possible combinations of rows in all tables
- ✓ Results in a rowset containing the number of rows in the first table times the number of rows in the second
- ✓ Can result in huge result sets that take several hours to complete!

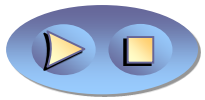
Joining Tables by Using Non-Equi Joins

- The same Operators and Predicates used for Inner Joins can be used for Not-Equal Joins

```
SELECT DISTINCT p1.ProductSubcategoryID, p1.ListPrice
FROM Production.Product p1
    INNER JOIN Production.Product p2
    ON p1.ProductSubcateogoryID = p2.ProductSubcategoryID
    AND p1.ListPrice <> p2.ListPrice
WHERE p1.ListPrice < $15 AND p2.ListPrice < $15
ORDER BY ProductSubcategoryID
```

Result Set:

```
ProductSubcateogoryID    ListPrice
-------------------------------------
23                       8.99
23                       9.50
...
(8 row(s) affected)
```



Combining and Limiting Result Sets

- Combining Result Sets by Using the UNION Operator
- Limiting Result Sets by Using the EXCEPT and INTERSECT Operators
- Identifying the Order of Precedence of UNION, EXCEPT, and INTERSECT
- Limiting Result Sets by Using the TOP and TABLESAMPLE Operators
- Categorizing Statements that Limit Result Sets

Combining Result Sets by Using the UNION Operator

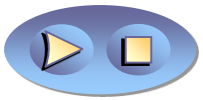
- UNION combines the results of two or more queries into a single result set that includes all the rows that belong to all queries in the union

```
SELECT * FROM testa
UNION ALL
SELECT * FROM testb;
```

Result Set:

```
columna    columnb
-------------------
100        test
100        test
...
(8 row(s) affected)
```

The number and order of columns must be the same in all queries and all data types must be compatible



Limiting Result Sets by Using the EXCEPT and INTERSECT Operators

- EXCEPT returns any distinct values from the query to the left of the EXCEPT operand that are not also returned from the right query
- INTERSECT returns any distinct values that are returned by both the query on the left and right sides of the INTERSECT operand

EXCEPT Example:

```
SELECT ProductID
FROM Production.Product
EXCEPT
SELECT ProductID
FROM Production.WorkOrder
```

INTERSECT Example:

```
SELECT ProductID
FROM Production.Product
INTERSECT
SELECT ProductID
FROM Production.WorkOrder
```

Order of Precedence of UNION, EXCEPT, and INTERSECT

EXCEPT, INTERSECT, and UNION are evaluated in the context of the following precedence:

1. Expressions in parentheses
2. The INTERSECT operand
3. EXCEPT and UNION evaluated from Left to Right based on their position in the expression

Limiting Result Sets by Using the TOP and TABLESAMPLE Operators

- TOP and TABLESAMPLE limit the number of rows returned in a result set

TOP Example:

```
SELECT TOP (15)
   FirstName, LastName
FROM Person.Person
```

Result Sets

```
FirstName   LastName
--------------------
Syed        Abbas
Catherine   Abel
...
(15 row(s) affected)
```

TABLESAMPLE Example:

```
SELECT
   FirstName, LastName
FROM Person.Person
TABLESAMPLE (1 PERCENT)
```

```
FirstName   LastName
--------------------
Eduardo     Barnes
Edward      Barnes
...
(199 row(s) affected)
```

Categorizing Statements That Limit Result Sets

- UNION
  - Combines the results of two or more SELECT statements into a single result set
- EXCEPT and INTERSECT
  - Compares the results of two or more SELECT statements and return distinct values
- TOP
  - Specifies that only the first set of rows will be returned from the query result
- TABLESAMPLE
  - Limits the number of rows returned from a table in the FROM clause to a sample number or PERCENT of rows

Queries -4

Working with Subqueries

Working with Subqueries

- Writing Basic Subqueries
- Writing Correlated Subqueries
- Comparing Subqueries with Joins and Temporary Tables
- Using Common Table Expressions

Writing Basic Subqueries

- What Are Subqueries?
- Using Subqueries as Expressions
- Using the ANY, ALL, and SOME Operators
- Scalar versus Tabular Subqueries
- Rules for Writing Subqueries

What Are Subqueries?

- Queries nested inside a SELECT, INSERT, UPDATE, or DELETE statement
- Can be used anywhere an Expression is allowed

```
SELECT ProductID, Name
FROM Production.Product
WHERE Color NOT IN
    (SELECT Color
    FROM Production.Product
    WHERE ProductID = 5)
```

Result Set:

```
ProductID   Name
-----------------------------------
1           Adjustable Race
2           Bearing Ball
...
(504 row(s) affected)
```

Using Subqueries as Expressions

A Subquery can be substituted anywhere an expression can be used in the following statements, except in an ORDER BY list:

- ✓ SELECT
- ✓ INSERT
- ✓ UPDATE
- ✓ DELETE

```
SELECT  Name, ListPrice,
(SELECT AVG(ListPrice) FROM Production.Product)          AS
Average,
ListPrice-(SELECT AVG(ListPrice) FROM Production.Product)
As Diff
FROM Production.Product
WHERE ProductSubcategoryID = 1
```

Result Set:

```
Name                          ListPrice  Average    Difference
----------------------------------------------------------
Mountain-100 Silver, 38   3399.99    438.6662   2961.3238
Mountain-100 Silver, 42   3399.99    438.6662   2961.3238
...
(32 row(s) affected)
```

Using the ANY, ALL, and SOME Operators

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

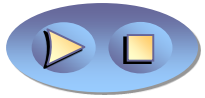
```
SELECT Name
FROM Production.Product
WHERE ListPrice >= ANY
    (SELECT MAX (ListPrice)
     FROM Production.Product
     GROUP BY ProductSubcategoryID)
```

```
Name
----------------------
LL Mountain Seat
ML Mountain Seat ...
(304 row(s) affected)
```

ALL Example:

```
SELECT Name
FROM Production.Product
WHERE ListPrice >= ALL
    (SELECT MAX (ListPrice)
     FROM Production.Product
     GROUP BY ProductSubcategoryID)
```

```
Name
--------------------
Road-150 Red, 62
Road-150 Red, 44...
(5 row(s) affected)
```



Scalar versus Tabular Subqueries

- A scalar subquery returns a single row of data, while a tabular subquery returns multiple rows of data

Scalar Subquery:

```
create table T1 (a int, b int)
create table T2 (a int, b int)
select *
from T1
where T1.a > (select max(T2.a)
from T2 where T2.b < T1.b)
```

Result Sets

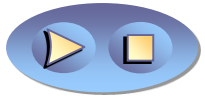
```
a           b
-----------------------

...
(0 row(s) affected)
```

Tabular Subquery:

```
SELECT Name
FROM Production.Product
WHERE ListPrice in
    (SELECT ListPrice
     FROM Production.Product
     WHERE Name = 'Chainring Bolts' )
```

```
Name
------------------
Adjustable Race
Bearing Ball ...
(200 row(s) affected)
```



Rules for Writing Subqueries

Subquery Allowances

- Subqueries can be specified in many places
- A subquery can itself include one or more subqueries

Subquery Restrictions

- SELECT list of a subquery introduced with a comparison operator can include only one expression
- WHERE clauses must be join-compatible
- ntext, text, and image data types cannot be used
- Column names in a statement are implicitly qualified by the table referenced in the FROM clause

Subqueries versus Joins

- Joins can yield better performance in some cases where existence must be checked
- Joins are performed faster by SQL Server than subqueries
- Subqueries can often be rewritten as joins
- SQL Server query optimizer is intelligent enough to covert a subquery into a join if it can be done
- Subqueries are useful for answering questions that are too complex to answer with joins

Understanding Set-Based Logic

Set-based logic

- SQL Server iterates through data
- Deals with results as a set instead of row-by-row

```
SELECT ProductID,
Purchasing.Vendor.VendorID, Name
FROM Purchasing.ProductVendor JOIN
Purchasing.Vendor
    ON (Purchasing.ProductVendor.VendorID
= Purchasing.Vendor.VendorID)
WHERE StandardPrice > $10
  AND Name LIKE N'F%'
GO
```

Queries - 5

Modifying Data in Tables

Modifying Data in Tables

- Inserting Data into Tables
- Deleting Data from Tables
- Updating Data in Tables
- Overview of Transactions

Inserting Data into Tables

- INSERT Fundamentals
- INSERT Statement Definitions
- INSERT Statement Examples
- Inserting Values into Identity Columns

INSERT Fundamentals

- The INSERT statement adds one or more new rows to a table
- INSERT inserts *data_values* as one or more rows into the specified *table_or_view*
- *column_list* is a list of column names used to specify the columns for which data is supplied

INSERT Syntax:

```
INSERT [INTO] table_or_view [(column_list)] data_values
```

INSERT Statement Definitions

INSERT using SELECT

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

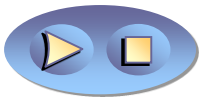
INSERT using EXECUTE

```
INSERT dbo.SomeTable
EXECUTE SomeProcedure    ”predefined procedure”
```

INSERT using TOP

```
INSERT TOP (#) INTO SomeTableA
    SELECT SomeColumnX, SomeColumnY
    FROM SomeTableB
```

```
INSERT INTO SomeTableA
    SELECT TOP (#) SomeColumnX, SomeColumnY
    FROM SomeTableB
```



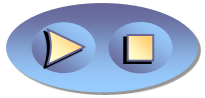
INSERT Statement Examples

Using a Simple INSERT Statement

```
INSERT INTO Production.UnitMeasure
VALUES (N'F2', N'Square Feet', GETDATE());
```

Inserting Multiple Rows of Data (Row Constructor)

```
INSERT INTO Production.UnitMeasure
VALUES (N'F2', N'Square Feet', GETDATE()),
(N'Y2', N'Square Yards', GETDATE());
```



Inserting Values into Identity Columns

- *column_list* and VALUES 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.T1 ( column_1 int
IDENTITY, column_2 VARCHAR(30));
GO
INSERT T1 VALUES ('Row #1');
INSERT T1 (column_2) VALUES ('Row #2');
GO
SET IDENTITY_INSERT T1 ON;
GO
INSERT INTO T1 (column_1,column_2)
    VALUES (-99, 'Explicit identity
value');
GO
SELECT column_1, column_2
FROM T1;
```

To see Identity
select AddressID from Person.Address
select Address.$IDENTITY from person.Address

Deleting Data from Tables

- DELETE Fundamentals
- DELETE Statement Definitions
- Defining and Using the TRUNCATE Statement
- TRUNCATE versus DELETE

DELETE Fundamentals

- The DELETE statement removes one or more rows in a table or view
- DELETE removes rows from the *table_or_view* parameter that meet the *search condition*
- *table_sources* can be used to specify additional tables or views that can be used by the WHERE clause

DELETE Syntax:

```
DELETE table_or_view

FROM table_sources

WHERE search_condition
```

DELETE Statement Definitions

DELETE with no WHERE clause

```
DELETE FROM SomeTable;
```

```
DELETE FROM Sales.SalesPerson;
```

DELETE using a Subquery

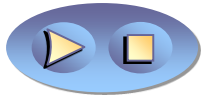
```
DELETE FROM SomeTable
WHERE SomeColumn IN
    (Subquery Definition);
```

```
DELETE FROM
Sales.SalesPersonQuotaHistory
WHERE SalesPersonID IN
     (SELECT SalesPersonID
      FROM Sales.SalesPerson
      WHERE SalesYTD >
2500000.00);
```

DELETE using TOP

```
DELETE TOP (#) PERCENT
FROM SomeTable;
```

```
DELETE TOP (2.5) PERCENT
FROM
Production.ProductInventory;
```



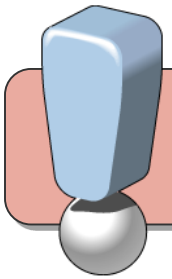
Defining and Using the TRUNCATE Statement

TRUNCATE TABLE Syntax

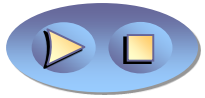
```
TRUNCATE TABLE
    [ { database_name.[ schema_name ]. | schema_name . } ]
    table_name
[ ; ]
```

TRUNCATE TABLE Example

```
TRUNCATE TABLE HumanResources.JobCandidate;
```



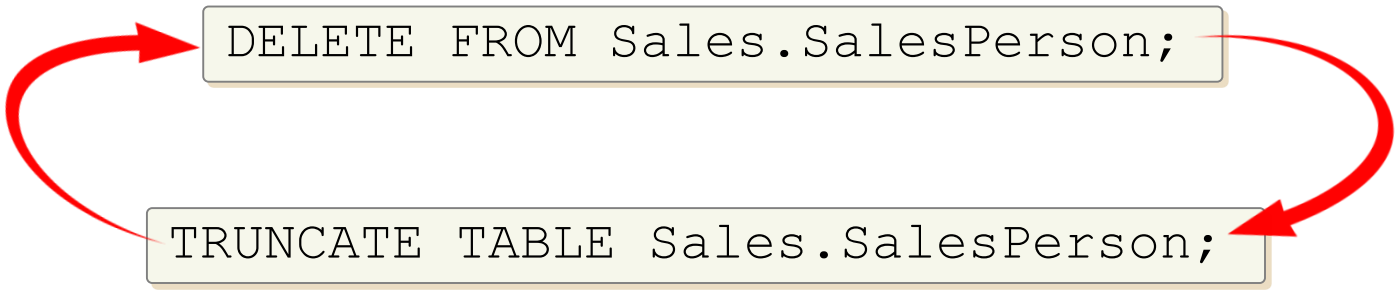
You cannot use TRUNCATE TABLE on tables that are referenced by a FOREIGN KEY constraint



TRUNCATE versus DELETE

TRUNCATE TABLE has the following advantages over DELETE:

- Less transaction log space is used
- Fewer locks are typically used
- Zero pages are left in the table



Updating Data in Tables

- UPDATE Fundamentals
- UPDATE Statement Definitions
- Updating with Information from another Table
- UPDATE and the OUTPUT Clause

UPDATE Fundamentals

- The UPDATE statement changes data values in one, many, or all rows of a table
- An UPDATE statement referencing a table or view can change the data in only one base table at a time
- UPDATE has three major clauses:
  - SET – comma-separated list of columns to be updated
  - FROM – supplies values for the SET clause
  - WHERE – specifies a search condition for the SET clause

UPDATE Syntax:

```
UPDATE table_or_view

SET column_name = expression

FROM table_sources

WHERE search_condition
```

UPDATE Statement Definitions

Simple UPDATE Statement

```
UPDATE SomeTable
SET Column = Value
```

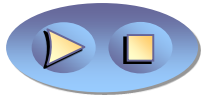
```
UPDATE Sales.SalesPerson
SET Bonus = 6000;
```

```
UPDATE Sales.SalesPerson
SET Bonus = Bonus * 2;
```

UPDATE with a WHERE clause

```
UPDATE SomeTable
SET Column = Value
WHERE SearchExpression
```

```
UPDATE Production.Product
SET Color = N'Metallic Red'
WHERE Name LIKE N'Road-250%'
    AND Color = N'Red';
```



Updating with Information from Another Table

UPDATE using a Subquery

```
UPDATE SomeTable
SET Column = Value
FROM SomeSubquery
```

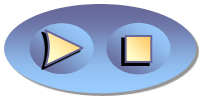
```
UPDATE Sales.SalesPerson
SET SalesYTD = SalesYTD + SubTotal
FROM Sales.SalesPerson AS sp
JOIN Sales.SalesOrderHeader AS so
    ON sp.BusinessEntityID = so.SalesPersonID
    AND so.OrderDate = (SELECT MAX(OrderDate)
                        FROM Sales.SalesOrderHeader
                        WHERE SalesPersonID =
                              sp.BusinessEntityID);
```

Before

```
SalesYTD
--------------
677558.4653
4557045.0459
```

After

```
SalesYTD
--------------
721382.488
4593234.5123
```



Lab

Help Index