Upgrading Skills

Primavera ACADEMY

SQL – Foundations to Advanced

Curso RE_START

2020-v0.0-EN

Primavera ACADEMY

Introducing T-SQL

Module 1

Summary

- Introducing T-SQL
- Creating Database Objects
- T-SQL Select
- Select Data with Multiple Tables
- Insert, Update and Delete Data

T-SQL

- T-SQL (Transact-SQL) is an extension of SQL (Structured Query Language)
- T-SQL is central to using SQL Server, since all applications that communicate with an instance of SQL Server do so by sending T-SQL statements to the server, regardless of the user interface of the application
- T-SQL scripts are written in a text editor, preferably in SQL Server Query Editor
- These T-SQL scripts can be saved in .sql files
- The scripts are structured in statements organized by categories
- T-SQL language is divided into 4 subsets:
  - Data Manipulation Language (DML)
  - Data Definition Language (DDL)
  - Data Control Language (DCL)
  - Transaction Control Language (TCL)

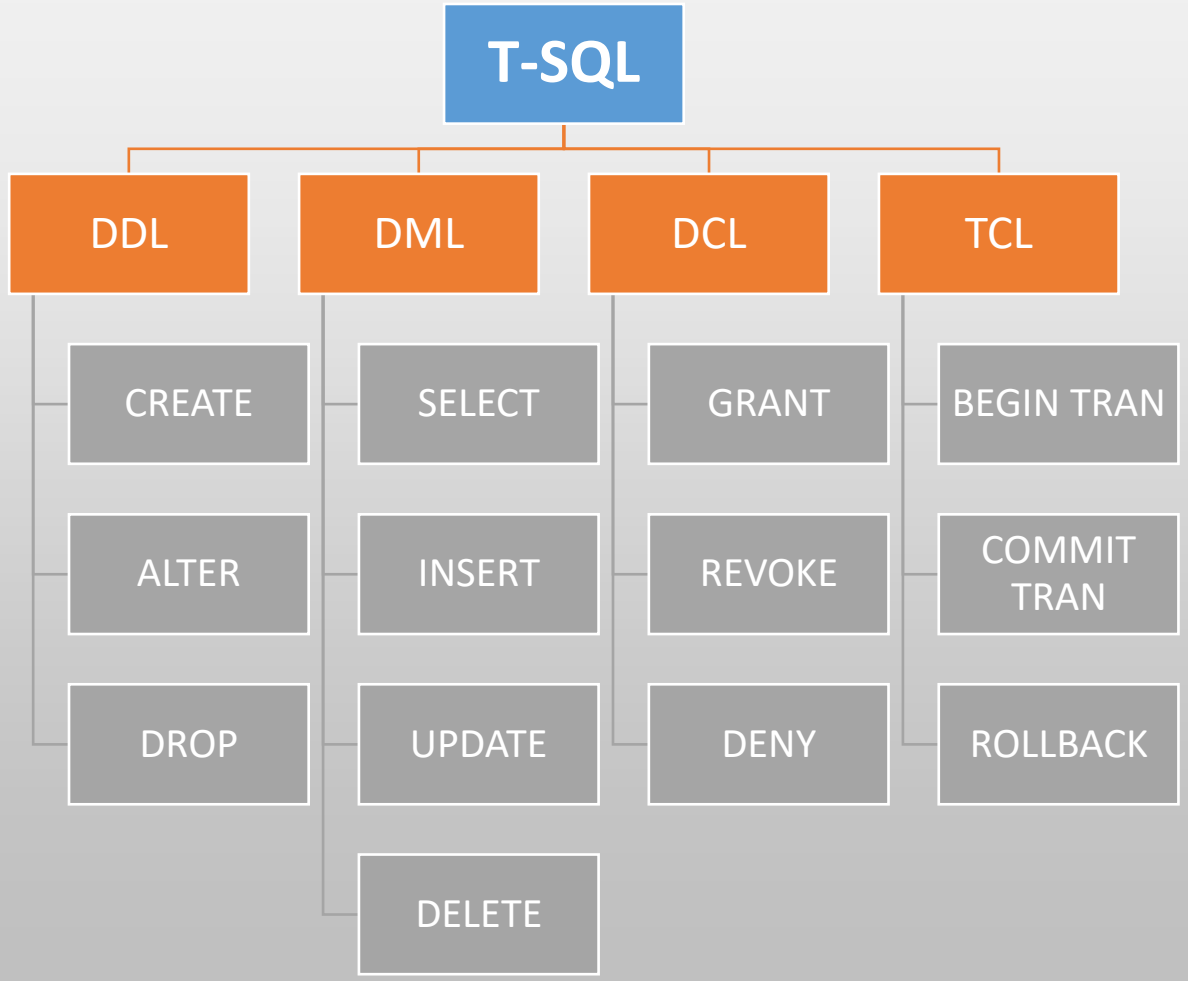
T-SQL

DDL
- Data Definition Language
- Definir objetos

DML
- Data Manipulation Language
- Manipular dados

DCL
- Data Control Language
- Gerir permissões

TCL
- Transaction Control Language
- Gerir transações



Data Definition Language (DDL)

- DDL deals with creating and manipulating database objects like tables, constraints, views and stored procedures
- Core statements:

CREATE	Creates a SQL Server database object (table, view or stored procedure)
ALTER	Changes an existing object
DROP	Removes an object from the database

```sql
-- CREATE
CREATE TABLE Person(
    PersonID INT IDENTITY (1,1) CONSTRAINT PK_PersonID PRIMARY KEY,
    FirstName NVARCHAR(20),
    LastName NVARCHAR(25)
);

-- ALTER
ALTER TABLE Person
ADD BirthDate DATETIME

-- DROP
DROP TABLE Person
```

Data Manipulation Language (DML)

- DML is used for manipulating data
- Core statements:

SELECT	Select/read records from database tables
INSERT	Insert new records in table
UPDATE	Update existing records
DELETE	Delete existing records

```sql
-- SELECT
SELECT *
FROM employee

-- INSERT
INSERT INTO employee(emp_id, fname, minit, lname,
                     job_id, job_lvl, pub_id, hire_date)
            VALUES('0000000', 'Almir', 'M', 'Vuk',
                     7, 12, 1207, 2009-05-09)

-- UPDATE
UPDATE employee
SET fname = 'ALMIR'
WHERE emp_id = '0000000'

-- DELETE
DELETE
FROM employee
WHERE emp_id = '0000000'
```

Data Control Language (DCL)

- DCL is used for securing data (authorization and permissions)
- Core statements:

GRANT	To give permissions on database objetcs to users
DENY	To deny permissions to users
REVOKE	To take back permission from users

```sql

-- GRANT
GRANT SELECT, INSERT, UPDATE, DELETE ON Employees TO almir

-- REVOKE
REVOKE INSERT ON Employees TO almir

-- DENY
DENY UPDATE ON Employees TO almir

```

Transaction Control Language (TCL)

- TCL is used for manipulating transactions
- Core statements:

BEGIN TRAN	Begin of transaction
COMMIT TRAN	Commit for completed transaction
ROLLBACK	Go back to beginning if something went wrong in transaction

```sql
BEGIN TRANSACTION
UPDATE dbo.authors
SET au_fname = 'Almir'
WHERE au_id = '172-32-1176'

UPDATE authors
SET au_fname = 'Almir'
WHERE city = 'Mostar'

IF @@ROWCOUNT = 5
    COMMIT TRANSACTION
ELSE
    ROLLBACK TRANSACTION
```

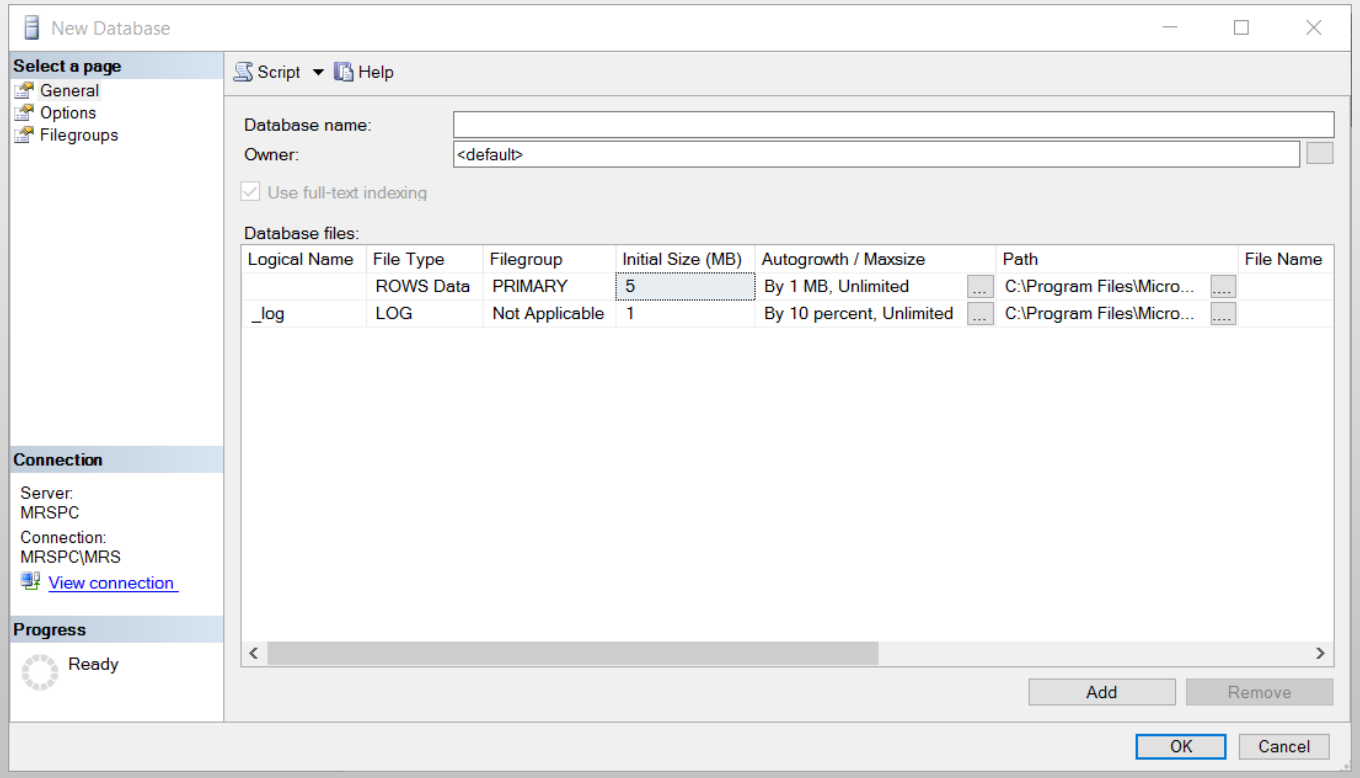
Primavera ACADEMY

Creating Database Objects

Module 2

Creating Databases using SSMS

- SSMS helps in the creation of databases
- Steps: SSMS > Object Explorer > Databases right click > New Database



Creating Databases using SSMS

- General page:
  - Database name: will be used as the name for the data and log files
  - Owner: the database owner (dbo) has full administration rights on the database
  - Database files:

Logical Name	It's a friendly name for the database
File Type	The type of files that will be generated: Rows Data for the data file (MDF) and Log for the log file (LDF)
Filegroup	It groups database objects and files together for allocation and administration purposes
Path	The physical files location
Initial Size (MB)	Database's initial size in MB
Autogrowth / Maxsize	Database's options for autogrowth and maxsize
Path	Physical path for saving database's files
File Name	The physical name of the database files

Creating Databases using T-SQL

- CREATE DATABASE statement
- Syntax and example :

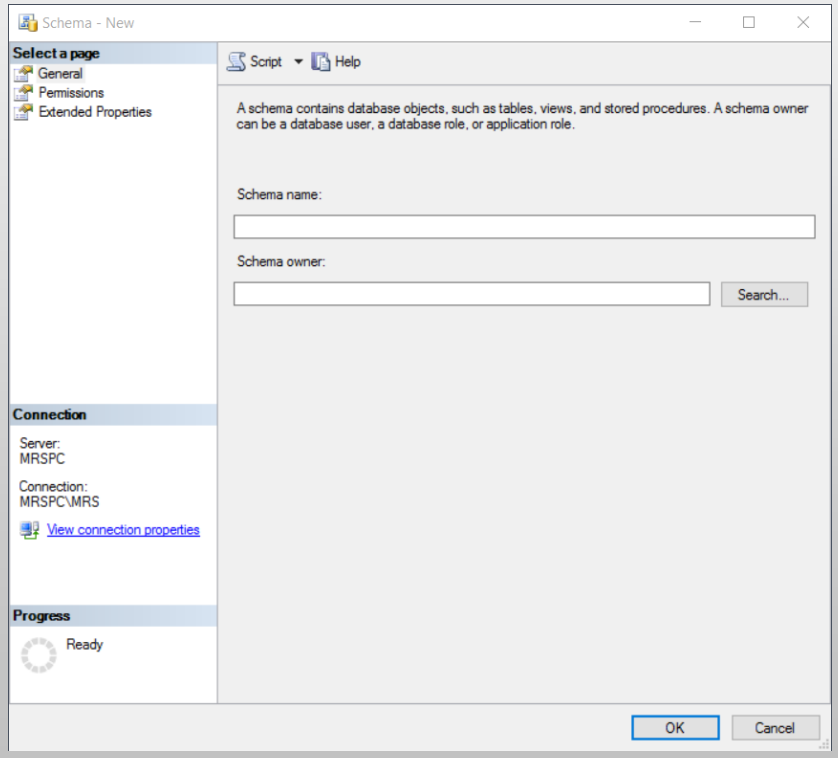
```
CREATE DATABASE database_name
[ CONTAINMENT = { NONE | PARTIAL } ]
[ ON
      [ PRIMARY ] <filespec> [ ,...n ]
      [ , <filegroup> [ ,...n ] ]
      [ LOG ON <filespec> [ ,...n ] ]
]
[ COLLATE collation_name ]
[ WITH  <option> [,...n ] ]
[;]
```

```sql
CREATE DATABASE TesteDB
ON  PRIMARY
(
  NAME = N'TesteDB',
  FILENAME = N'C:\Data\TesteDB.mdf',
  SIZE = 5120KB,
  MAXSIZE = UNLIMITED,
  FILEGROWTH = 1024KB
)
LOG ON
(
  NAME = N'TesteDB_log',
  FILENAME = N'C:\Log\TesteDB_log.ldf',
  SIZE = 1024KB,
  MAXSIZE = 2048GB,
  FILEGROWTH = 10%
);
GO
```

Database Schemas

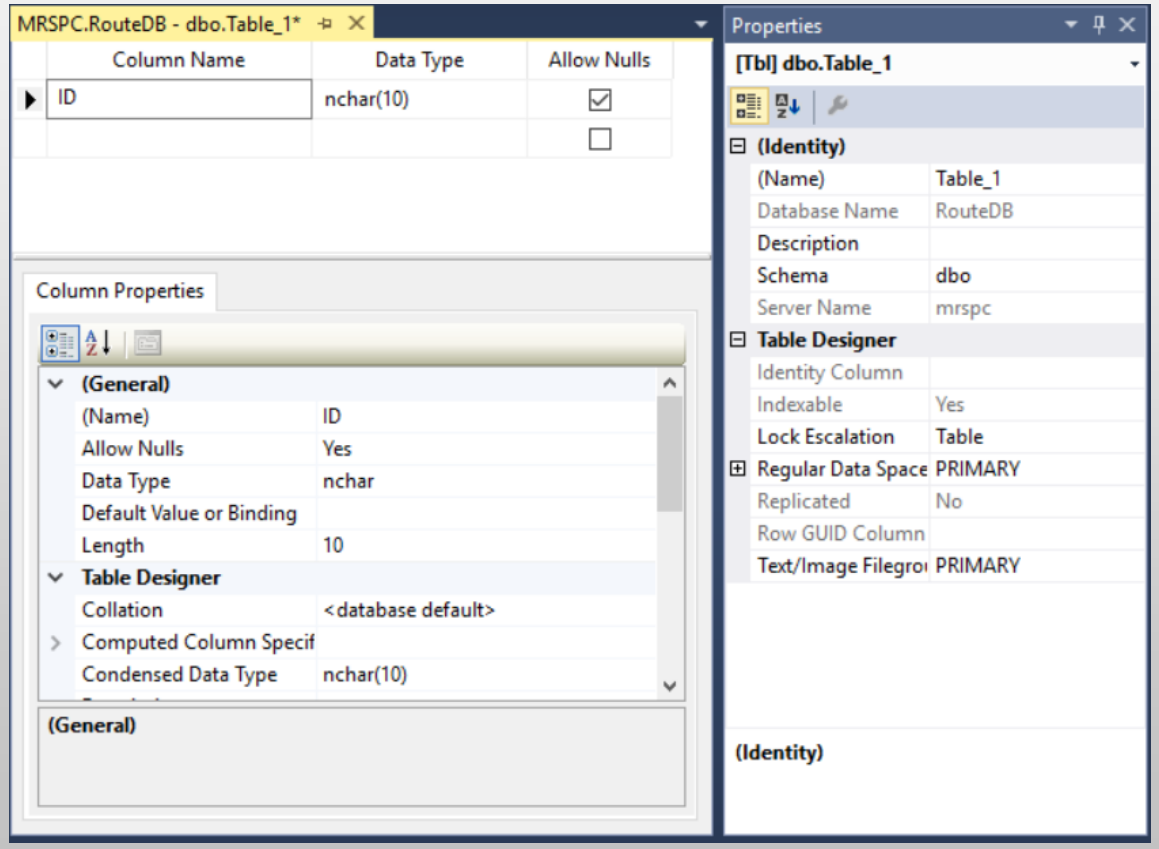
- Schema is a database object that allows you to logically group objects, manage and define ownership, independently of the individual users in the database
- The schemas are located in: Database > Security > Schemas
- Example:

```
USE TesteDB;
GO
CREATE SCHEMA Controlo AUTHORIZATION MRS;
GO
```



Creating Tables Using SSMS

- Steps: SSMS > Object Explorer > Database > Tables > Right-click > New Table > Define table columns



Creating Tables Using T-SQL

- CREATE TABLE statement

```
      --Simple CREATE TABLE Syntax (common if not using options)
CREATE TABLE
    [ database_name . [ schema_name ] . | schema_name . ] table_name
    ( { <column_definition> } [ ,...n ] )
[ ; ]
```

```
USE TesteDB;
GO

CREATE TABLE dbo.Person
(
     ID int IDENTITY (1,1) NOT NULL,
     Name nvarchar(70) NOT NULL,
     BirthDate date,
     CONSTRAINT PK_Person_ID PRIMARY KEY CLUSTERED (ID ASC)
);
GO
```

Manipulating Tables

- Alter:
  - Using SSMS: Object Explorer > Database > Table > Right-click > Modify > Design > …
  - Using T-SQL:

```
ALTER TABLE dbo.Person
  ADD CONSTRAINT PK_Person_ID
  PRIMARY KEY CLUSTERED(ID ASC);
GO
```

- Drop:
  - Using SSMS: Object Explorer > Database > Table > Right-click > Delete
  - Using T-SQL:

```
DROP TABLE dbo.Person;
GO
```

Views

- A view is simply a virtual table consisting of different columns from one or more tables
- It´s an abstraction layer for tables relationship complexity
- Unlike a table, a view is stored in the database as a query object; therefore, a view is an object that obtains its data from one or more tables
- Database > Views

```
USE AdventureWorks2016;
GO

SELECT * FROM HumanResources.vEmployee;
GO
```

Stored Procedures

- A stored procedure is a an written T-SQL statement which has been saved into the database
- One of the things that will save you time when running the same query over and over again is to create a stored procedure, which you can then execute from within the database's command environment
- Database > Programmability > Stored Procedures

```sql
USE AdventureWorks2016;
GO

EXEC uspGetOrderTrackingByTrackingNumber 'EE33-45E8-9F';
GO
```

Functions

- A function is a an written T-SQL statement which has been saved into the database
- It can be used in queries or within the programming environment, such as stored procedures, functions, triggers, etc.
- Returns a value
- Database > Programmability > Functions

```sql
-- Criar a função
CREATE FUNCTION Sales.udfNetSale(
    @quantity INT,
    @listPrice DEC(10,2),
    @discount DEC(4,2)
)
RETURNS DEC(10,2)
AS
BEGIN
    RETURN @quantity * @listPrice * (1 - @discount);
END;


-- Usar a função
SELECT
    Sales.udfNetSale(10, 100, 0.1) AS [Net Sale];
```

Primavera ACADEMY

T-SQL Select

Module 3

Data Types

- In SQL Server, each column, local variable, expression, and parameter has a related data type
- A data type is an attribute that specifies the type of data that the object can hold
- Data types categories:
  - Exact numerics
  - Approximate numerics
  - Date and time
  - Character strings
  - Unicode character strings
  - Binary strings
  - Other data types

See resume table

Data Conversion

- Need for data types conversion:
  - When data from one object is moved to, compared with, or combined with data from another object, the data may have to be converted from the data type of one object to the data type of the other
  - When data from a Transact-SQL result column, return code, or output parameter is moved into a program variable, the data must be converted from the SQL Server system data type to the data type of the variable
- Types of conversions: implicitly or explicitly
  - Implicit conversions are not visible to the user, since SQL Server automatically converts the data from one data type to another
  - Explicit conversions use the CAST or CONVERT functions

Select Data

- SELECT Statement
- Syntax
- T-SQL Operators
- SELECT…FROM
- SELECT…INTO
- SELECT…WHERE
- SELECT…GROUP BY
- SELECT…HAVING
- SELECT…ORDER BY

SELECT Statement

- The SELECT statement is used to extract data from tables or views
- A simple SELECT uses one table but is possible to combine several tables using JOINs and INTERSECTs
- To build a correct SELECT you must provide:
  - Tables to retrieve data from (FROM)
  - Columns to show
  - Conditions that data must comply (optional WHERE)
  - Output order (optional ORDER BY)

Logical Processing Order

```
5SELECT select_list [INTO new_table]
1[FROM table_source]
2[WHERE search_condition]
3[GROUP BY group_by_expression]
4[HAVING search_condition]
6[ORDER BY order_expression [ASC|DESC]]
```

- The clauses are logically processed in the following order:
  1. FROM
  2. WHERE
  3. GROUP BY
  4. HAVING
  5. SELECT
  6. ORDER BY

T-SQL Operators

- Some operators used in T-SQL expressions:

Arithmetic Operators
Logical Operators
Comparison Operators

- Arithmetic Operators

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Module

T-SQL Operators

- Logical Operators

Operator	Meaning
AND	TRUE if both Boolean expressions are TRUE
BETWEEN	TRUE if the operand is within a range
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Reverses the value of any other Boolean operator
OR	TRUE if either Boolean expression is TRUE

- Comparison Operators

Operator	Meaning
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to
!=	Not equal to
!<	Not less than
!>	Not greater than

SELECT…FROM

SELECT select_list
FROM table_source

```sql
-- SELECT simples
SELECT [DepartmentID], [Name], [GroupName]
FROM [AdventureWorks2012].[HumanResources].[Department];
```

100 %

Results | Messages

	DepartmentID	Name	GroupName
1	1	Engineering	Research and Development
2	2	Tool Design	Research and Development
3	3	Sales	Sales and Marketing
4	4	Marketing	Sales and Marketing
5	5	Purchasing	Inventory Management
6	6	Research and Development	Research and Development
7	7	Production	Manufacturing
8	8	Production Control	Manufacturing
9	9	Human Resources	Executive General and Administration
10	10	Finance	Executive General and Administration
11	11	Information Services	Executive General and Administration
12	12	Document Control	Quality Assurance
13	13	Quality Assurance	Quality Assurance
14	14	Facilities and Maintenance	Executive General and Administration
15	15	Shipping and Receiving	Inventory Management
16	16	Executive	Executive General and Administration

SELECT…INTO

SELECT select_list
INTO new_table
FROM table_source

```
5  -- SELECT + INTO
6  SELECT [DepartmentID], [Name], [GroupName] INTO [AdventureWorks2012].[HumanResources].[Department_backup]
7  FROM [AdventureWorks2012].[HumanResources].[Department];
8
```

100 %

Messages

```
(16 row(s) affected)
```

- Tables
  - System Tables
  - FileTables
  - dbo.AWBuildVersion
  - dbo.DatabaseLog
  - dbo.ErrorLog
  - HumanResources.Department
  - HumanResources.Department_backup

SELECT…WHERE

SELECT select_list
FROM table_source
WHERE search_condition

Predicate LIKE

Find strings similar or not exactly equal to
Uses wildcards: % and _

Wildcards

% more than 1 character
_ 1 character in that position

```
-- SELECT ... WHERE 1
SELECT [DepartmentID], [Name], [GroupName]
FROM [AdventureWorks2012].[HumanResources].[Department]
WHERE Name = 'Production';

-- SELECT ... WHERE 2
SELECT [DepartmentID], [Name], [GroupName]
FROM [AdventureWorks2012].[HumanResources].[Department]
WHERE Name LIKE 'Production%';
```

100 %

Results | Messages

	Departmen...	Name	GroupName
1	7	Production	Manufacturing

	Departmen...	Name	GroupName
1	7	Production	Manufacturing
2	8	Production Control	Manufacturing

SELECT...GROUP BY

SELECT select_list
FROM table_source
GROUP BY group_by_expression

```
19  -- SELECT ... GROUP BY 1 --> ERRO PORQUE QUANDO SE AGRUPA TEM DE SE AGREGAR VALORES
20  SELECT [DepartmentID], [Name], [GroupName]
21  FROM [AdventureWorks2012].[HumanResources].[Department]
22  GROUP BY [GroupName];
23
```

100 %

Messages

```
Msg 8120, Level 16, State 1, Line 2
Column 'AdventureWorks2012.HumanResources.Department.DepartmentID' is invalid in the select list becaus
```

```
24  -- SELECT ... GROUP BY 2
25  SELECT [GroupName], [Name]
26  FROM [AdventureWorks2012].[HumanResources].[Department]
27  GROUP BY [GroupName], [Name];
28
```

100 %

Results | Messages

	GroupName	Name
1	Research and Development	Engineering
2	Research and Development	Tool Design
3	Sales and Marketing	Sales
4	Sales and Marketing	Marketing
5	Inventory Management	Purchasing
6	Research and Development	Research and Development

```
29  -- SELECT ... GROUP BY 3
30  SELECT [GroupName], COUNT([GroupName]) AS Contagem
31  FROM [AdventureWorks2012].[HumanResources].[Department]
32  GROUP BY [GroupName];
33
```

100 %

Results | Messages

	GroupName	Contagem
1	Executive General and Administration	5
2	Inventory Management	2
3	Manufacturing	2
4	Quality Assurance	2
5	Research and Development	3
6	Sales and Marketing	2

SELECT…HAVING

SELECT select_list
FROM table_source
GROUP BY group_by_expression
HAVING search_condition

```sql
-- SELECT ... GROUP BY ... HAVING
SELECT [GroupName], COUNT([GroupName]) AS Contagem
FROM [AdventureWorks2012].[HumanResources].[Department]
GROUP BY [GroupName]
HAVING COUNT([GroupName]) > 2;
```

100 %

Results | Messages

	GroupName	Contagem
1	Executive General and Administration	5
2	Research and Development	3

SELECT...ORDER BY

SELECT select_list
FROM table_source
ORDER BY order_expression [ASC|DESC]

```sql
-- SELECT ... ORDER BY v1
SELECT [DepartmentID], [Name], [GroupName]
FROM [AdventureWorks2012].[HumanResources].[Depart
ORDER BY [Name];

-- SELECT ... ORDER BY v2
SELECT [DepartmentID], [Name], [GroupName]
FROM [AdventureWorks2012].[HumanResources].[Depart
ORDER BY [Name] DESC;
```

100 %

Results | Messages

	DepartmentID	Name	GroupName
1	12	Document Control	Quality Assurance
2	1	Engineering	Research and Developmen
3	16	Executive	Executive General and Adm

	DepartmentID	Name	GroupName
1	2	Tool Design	Research and Developmen
2	15	Shipping and Receiving	Inventory Management
3	3	Sales	Sales and Marketing

```sql
-- SELECT ... ORDER BY v3
SELECT [GroupName], COUNT([GroupName]) AS Contagem
FROM [AdventureWorks2012].[HumanResources].[Departm
GROUP BY [GroupName]
ORDER BY COUNT([GroupName]) DESC;

-- SELECT ... ORDER BY v4
SELECT [GroupName], COUNT([GroupName]) AS Contagem
FROM [AdventureWorks2012].[HumanResources].[Departm
GROUP BY [GroupName]
ORDER BY COUNT([GroupName]) DESC, [GroupName] ASC;
```

100 %

Results | Messages

	GroupName	Contagem
1	Executive General and Administration	5
2	Research and Development	3
3	Sales and Marketing	2
4	Inventory Management	2
5	Manufacturing	2
6	Quality Assurance	2

	GroupName	Contagem
1	Executive General and Administration	5
2	Research and Development	3
3	Inventory Management	2
4	Manufacturing	2
5	Quality Assurance	2
6	Sales and Marketing	2

Primavera ACADEMY

Select Data with Multiple Tables

Module 4

Join Types

- The JOIN clause allows you to combine related data from multiple table sources
- Types of JOIN statements:

Join Type	Description
INNER	Starts with Cartesian product and then applies filter to match rows between tables based on predicate
CROSS	Combines all rows in both tables, aka, Cartesian Product
OUTER [LEFT | RIGHT]	Starts with Cartesian Product and all rows from designated table preserved, matching rows from other table retrieved. Additional NULLs inserted as placeholders

INNER JOIN

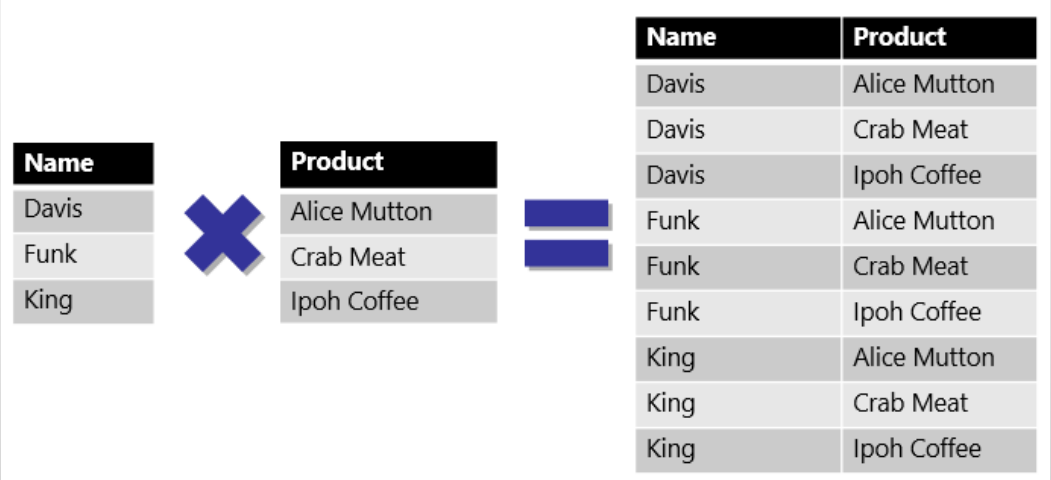
- List tables in FROM Clause separated by JOIN operator
- Table aliases preferred
- Table order does not matter

```
SELECT c.CategoryName, p.ProductName
FROM dbo.Categories AS c
INNER JOIN dbo.Products AS p
ON c.CategoryID = p.CategoryID


-- ON tabela1.pk = tabela2.fk
```

CROSS JOIN

- Combine all possible combinations of two sets, aka, Cartesian Product




```
SELECT c.CategoryName, p.ProductName
FROM dbo.Categories AS c
CROSS JOIN dbo.Products AS p
```

OUTER JOIN

- LEFT OUTER JOIN
  - Return all rows from first table and only matches from second

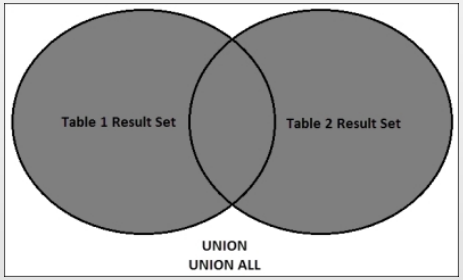
```
SELECT c.ContactName, o.OrderID
FROM dbo.Customers AS c
LEFT OUTER JOIN dbo.Orders AS o
ON c.CustomerID = o.CustomerID
```

- RIGHT OUTER JOIN
  - Return all rows from second table and only matches from first

```
SELECT c.ContactName, o.OrderID
FROM dbo.Customers AS c
RIGHT OUTER JOIN dbo.Orders AS o
ON c.CustomerID = o.CustomerID
```

UNION

- Used to combine sets from the same table or different tables
- All selected columns need to be of the same data type



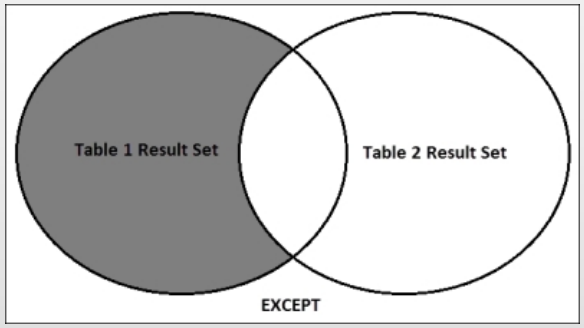

```
-- List the countries where we have customers OR suppliers

-- Without union
SELECT Country FROM dbo.Suppliers
SELECT Country FROM dbo.Customers

-- With union
SELECT Country FROM dbo.Suppliers
UNION
SELECT Country FROM dbo.Customers
```

EXCEPT

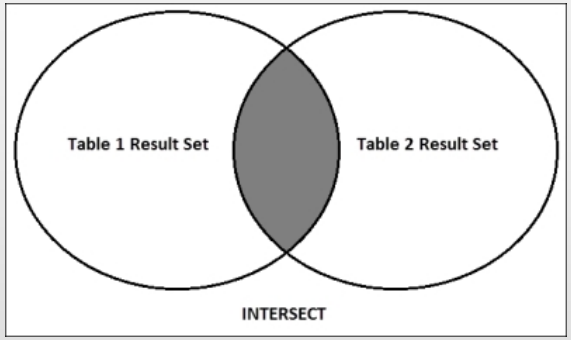
- Used to find what records exists in one set and not in the other set




```
-- Return a list of customers countries where there are no suppliers
SELECT DISTINCT Country FROM dbo.Customers
EXCEPT
SELECT DISTINCT Country FROM dbo.Suppliers
```

INTERSECT

- Used to find common values between sets from different tables




```
-- Create a list with countries from both our customers and suppliers
SELECT DISTINCT Country FROM dbo.Suppliers
INTERSECT
SELECT DISTINCT  Country FROM dbo.Customers
```

Primavera ACADEMY

Insert, Update and Delete Data

Module 5

INSERT Statement

- This statement inserts a single row by default

```sql
-- Omitting the PK
INSERT INTO dbo.Categories
(
  CategoryName,
  [Description]
)
VALUES
(
  'New Category',
  'With nothing...'
);
```

```sql
INSERT INTO dbo.Categories
(
  CategoryID,
  CategoryName,
  [Description]
)
VALUES
(
  , -- ERRO! OMITIR A COLUNA
  'New Category',
  'With nothing...'
);
```

```sql
-- Insert value into PK
INSERT INTO dbo.Region
(
  RegionID,
  RegionDescription
)
VALUES
(
  (SELECT MAX(RegionID) FROM dbo.Region ) + 1,
  'New Region'
);
```

UPDATE Statement

- UPDATE statement is used to modify data
  - Updates rows in a table or view filtered with a WHERE clause
  - Only columns specified in the SET clause are modified

```
UPDATE dbo.Region
SET RegionDescription = 'Old Region'
WHERE RegionDescription =  'New Region';
```

```
UPDATE dbo.Products
SET UnitPrice = (UnitPrice * 1.10), Discontinued = 0
WHERE ProductName =  'Guaraná Fantástica';
```

```
-- Suggestion: add a table Notes field and set it’s value to ‘changed’ to rollback
UPDATE dbo.Products
SET Discontinued = 0
FROM dbo.Categories AS c
INNER JOIN dbo.Products AS p
ON c.CategoryID = p.CategoryID
WHERE p.Discontinued = 1
```

DELETE Statement

- DELETE without a WHERE clause deletes all rows
  - In this case is better to use TRUNCATE TABLE
    - Minimally logged
    - It will fail if the table is referenced by a foreign key constraint in another table
- Use a WHERE clause to delete specific rows

```sql
DELETE FROM dbo.Region;
```

```sql
DELETE FROM dbo.Region
WHERE RegionDescription = 'Old Region';
```

```sql
DELETE dbo.Products
FROM dbo.Categories AS c
INNER JOIN dbo.Products AS p
ON c.CategoryID = p.CategoryID
WHERE c.CategoryName = 'New Category';
```

Upgrading Skills

Primavera ACADEMY