Introduction to Database

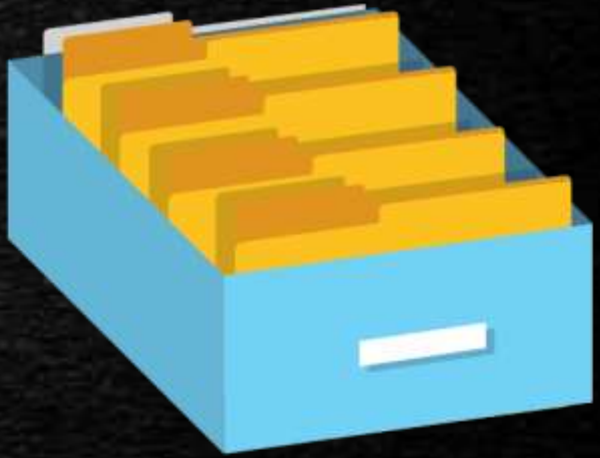
Introduction to DB, Why it's needed

Here is A Problem...

- A store deals with the products of many companies.
- Whenever there is a shortage of material the store owner places the order for the material to the concerned vendor.
- How would he manage the data for his store?

Solution 1 : Using Paper Files

- Paper files
  - A register/ file for one main item
  - Product register, Vendor List, Billing, Accounts



- Drawback:
  - Difficult to maintain and search
  - Repetition of data becomes necessary to make work easy

Solution 2 : Using Custom Software

- Data Management Software

  - Advantage
    - Easy to search
    - Easy to manipulate data
    - Easy to filter data according to a criteria

  - Drawback:
    - Application bound to the data storage
    - Modification is difficult

The Ultimate Solution : Use a Database

- Why Database

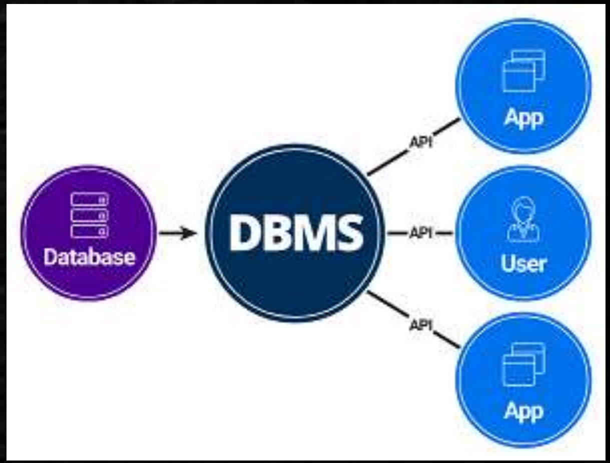
- Data represented in the form of logical tables.
- Modification of the structure of data is easy
- The programmer does not have to worry about the 'How' part
- Simple language to communicate with the database (SQL)
- Searching for data and sorting are easy

What is database and DBMS?

- Database is an organized collection of interrelated data

- The data is stored together without harmful or unnecessary redundancy

- A Database Management System (DBMS) is software designed to store, retrieve, define, and manage data in a database.



Characteristics of a good database:

- **Performance**:
  - Facility for the retrieval and manipulation of data irrespective of the number of tables with minimum time
- **Minimal redundancy**:
  - The database system should support minimal redundancy of data.

- **MultiUser**:
  - The db should provide multiuser support.

Characteristics of a good database:

- **Integrity**:
  - When multiple users uses the db the data items and the associations b/w the data should not be destroyed.

- **Privacy and security**:
  - The data should be protected against accidental or intentional access by unauthorized persons.

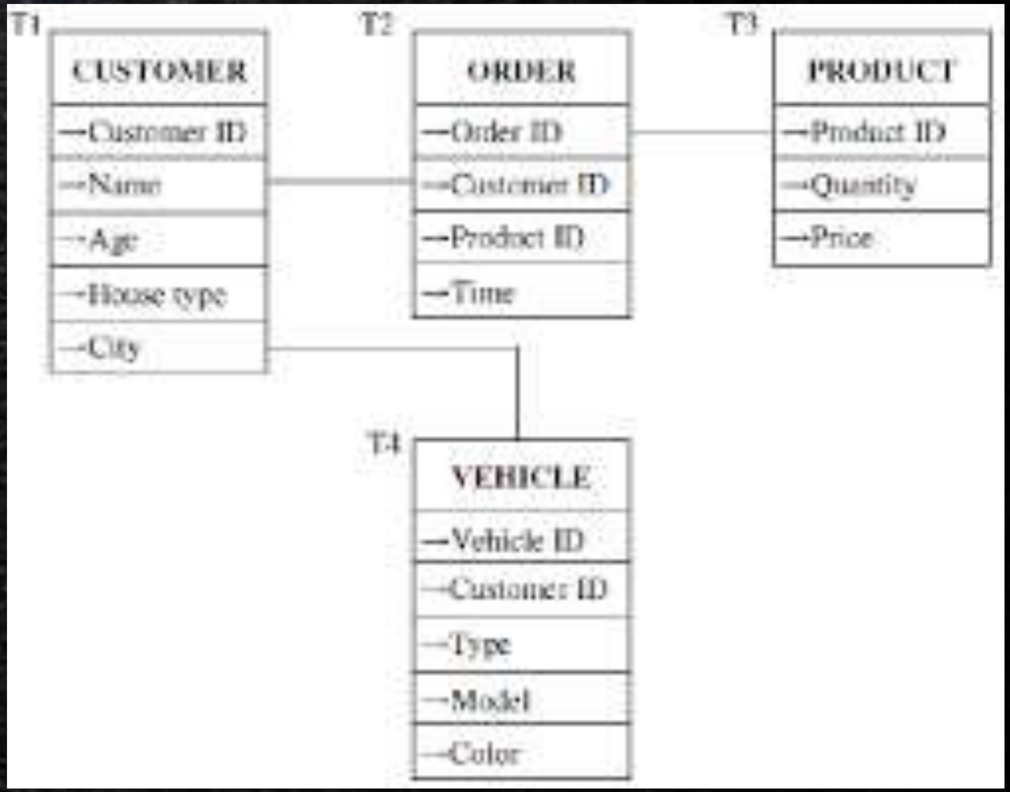
- **DB Language**:
  - The Db language used should be easy and powerful .
  - The most popular Db language is SQL

SQL

Introduction to RDBMS Concepts

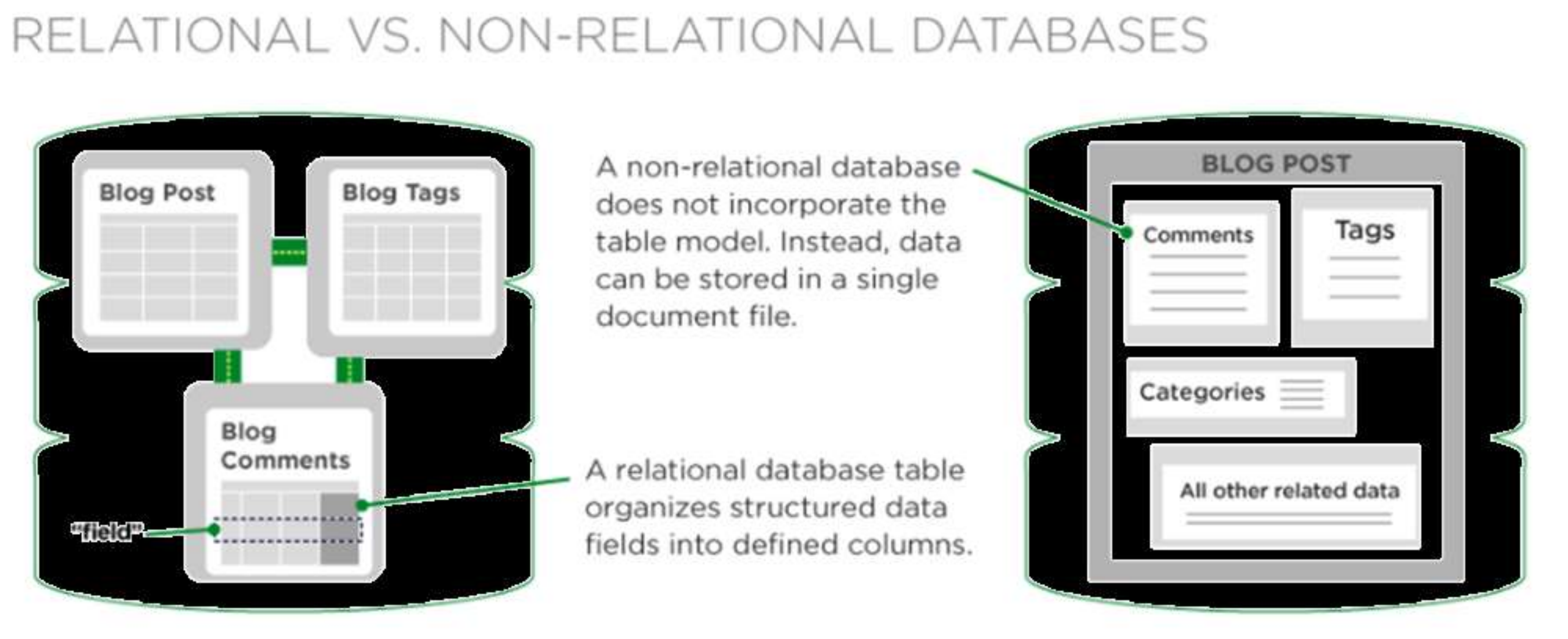
What is a Relational Database

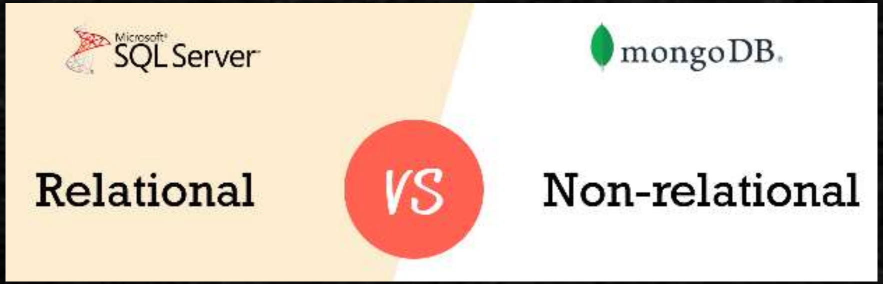
- A relational database is a collection of data items with pre-defined relationships between them.
- These items are organized as a set of tables with columns and rows. ...
- Each row in a table could be marked with a unique identifier called a primary key, and rows among multiple tables can be made related using foreign keys.



Eg: All modern database management systems like SQL, MS SQL Server, IBM DB2, ORACLE, My-SQL and Microsoft Access are based on RDBMS.

Relational Database (SQL) vs Non-Relational (NoSQL)





SQL Model to represent Employee Data in Table

COLUMN

ROW

EmpID	EmpName	DepartmentID	Salary
ED101	Kannan	101	8000
ED102	Daasan	100	9000
ED103	Sathru	102	10000

PK

FK

Relational Database vs Non-Relational

- This is how data is represented in a non-relational DB

Key	Document

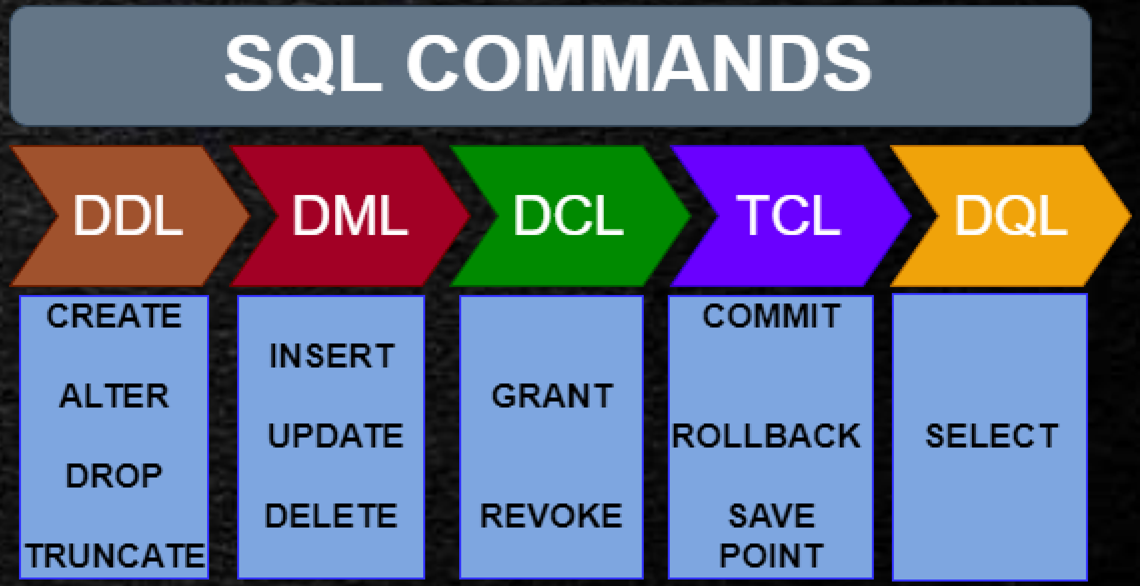
1001	{
	"CustomerID": 99,
	"OrderItems": [
	{ "ProductID": 2010,
	"Quantity": 2,
	"Cost": 520
	},
	{ "ProductID": 4365,
	"Quantity": 1,
	"Cost": 18
	}],
	"OrderDate": "04/01/2017"
	}

1002	{
	"CustomerID": 220,
	"OrderItems": [
	{ "ProductID": 1285,
	"Quantity": 1,
	"Cost": 120
	}],
	"OrderDate": "05/08/2017"
	}

SQL - Structured Query Language

- **SQL** is the common language to communicate with Database

- Parts of SQL
  - DDL-Data Definition Language
  - DML-Data Manipulation Language
  - TCL-Transaction Control Language
  - DCL-Data Control Language
  - DQL-Data Query Language

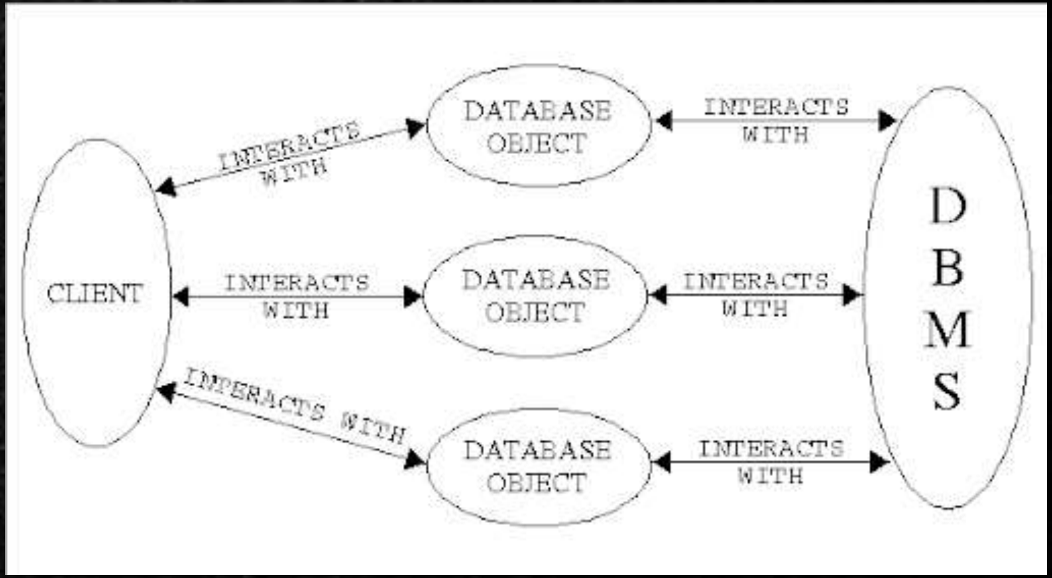


SQL Database Objects

- Table – Basic Unit of Storage
- View – Logical Subset of data from one or more tables
- Index – Improves the performance of some queries
- Synonym – Gives alternative names to objects

Object Naming Rules

- Must begin with a letter
- Must be 1 –30 chars long
- Must contain only A-Z, a-z, 0-9, $ and #
- Must not duplicate the name of another object owned by the same user
- Must not be database reserved word
    - A table can have upto 1000 columns



SQL Server

Basics of MS SQL Server

Microsoft SQL Server

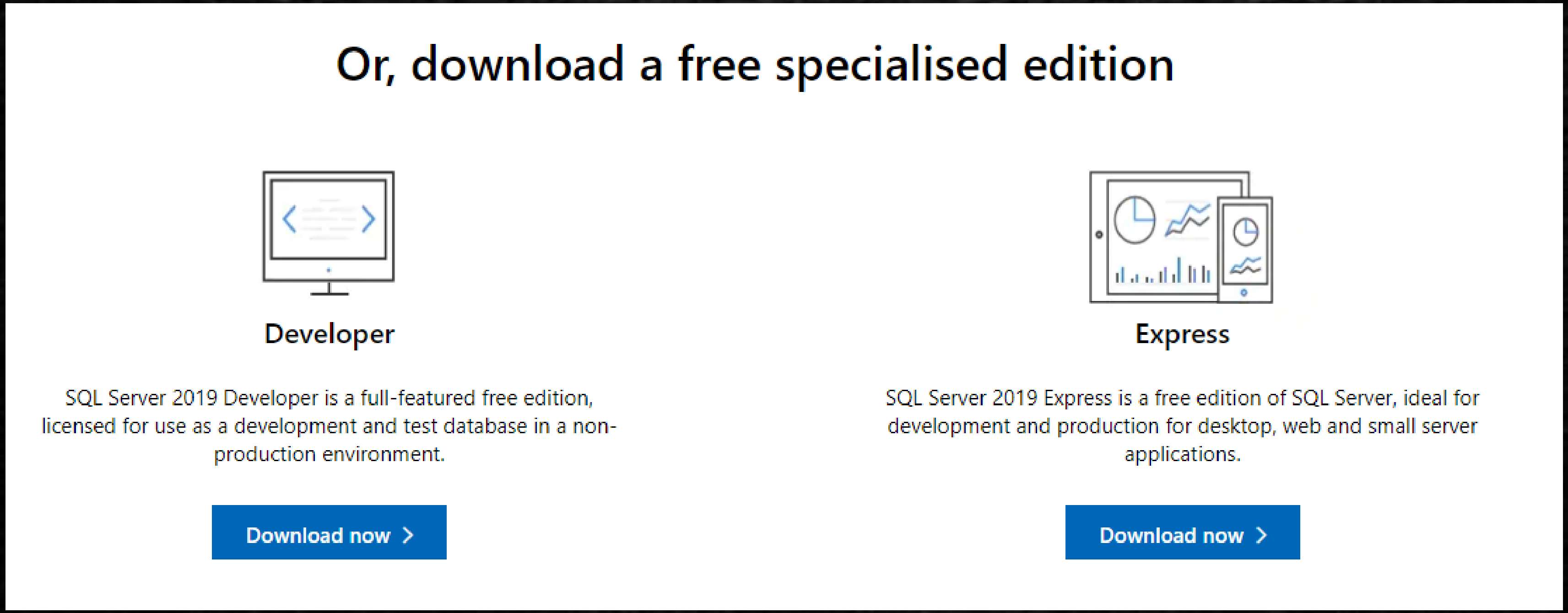
- SQL Server is software (A Relational Database Management System) developed by Microsoft.
- It is also called MS SQL Server. It is implemented from the specification of RDBMS
- The interface tool for SQL Server is SQL Server Management Studio (SSMS)

Usage of Microsoft SQL Server

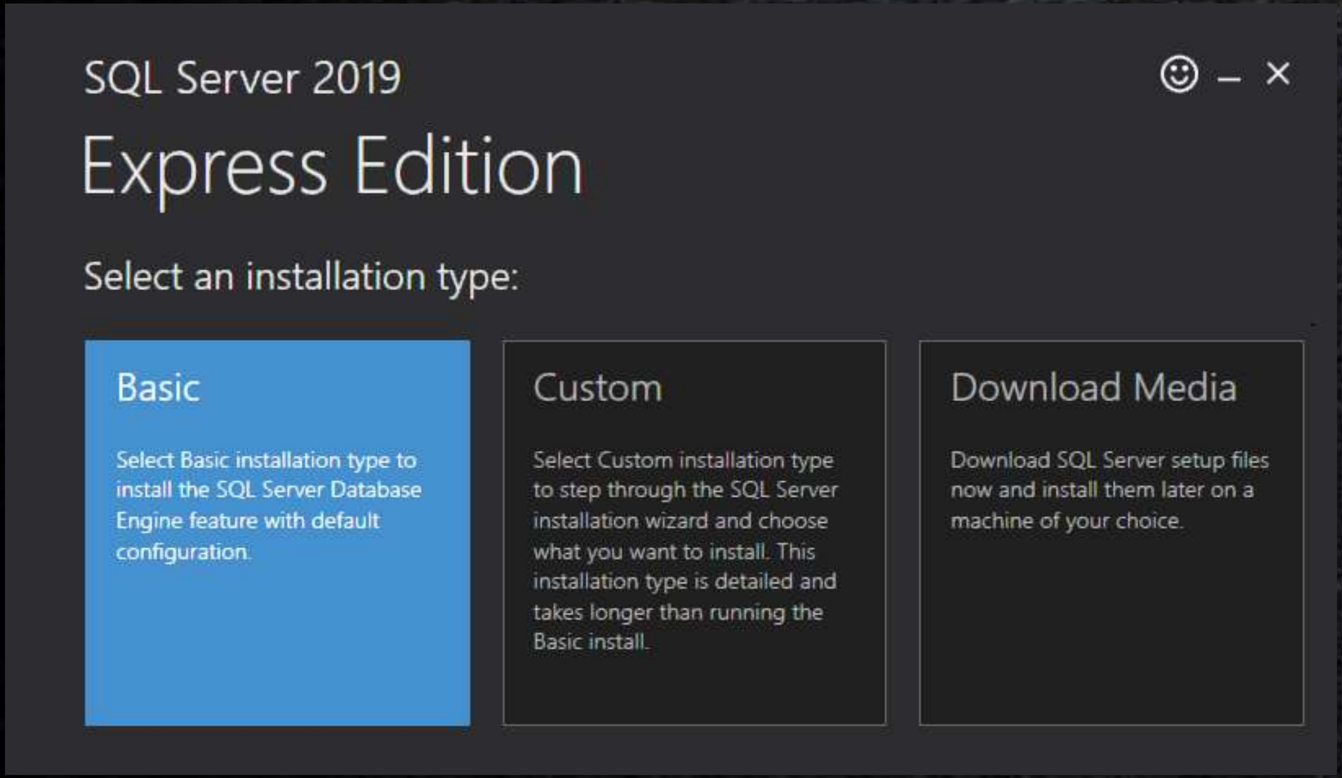
- To build and maintain databases.
- To analyze the data using SQL Server Analysis Services (SSAS).
- To generate reports using SQL Server Reporting Services (SSRS).
- To perform Extract Transform Load operations using SQL Server Integration Services (SSIS).

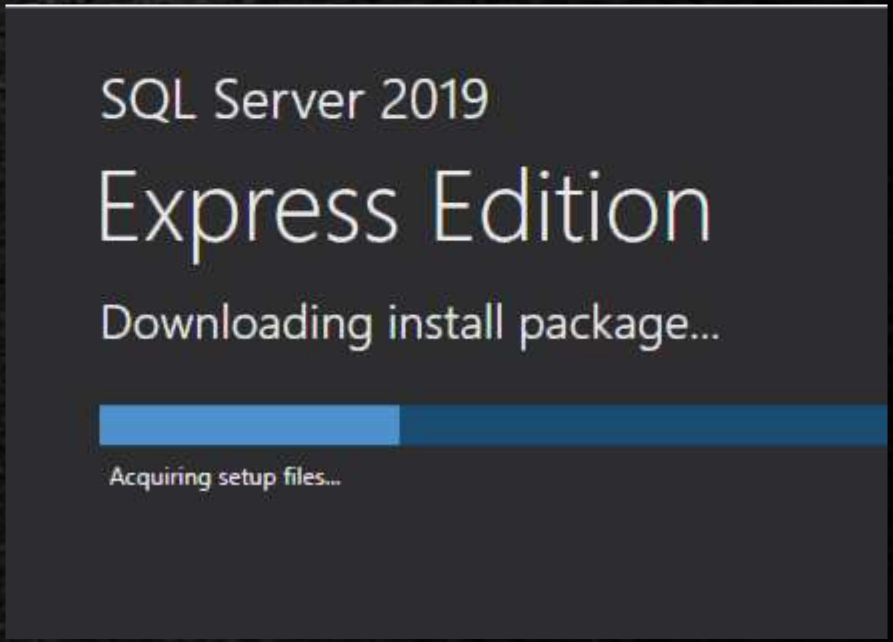
Installing Microsoft SQL Server

- Go to https://www.microsoft.com/en-in/sql-server/sql-server-downloads
- Download the Express Edition

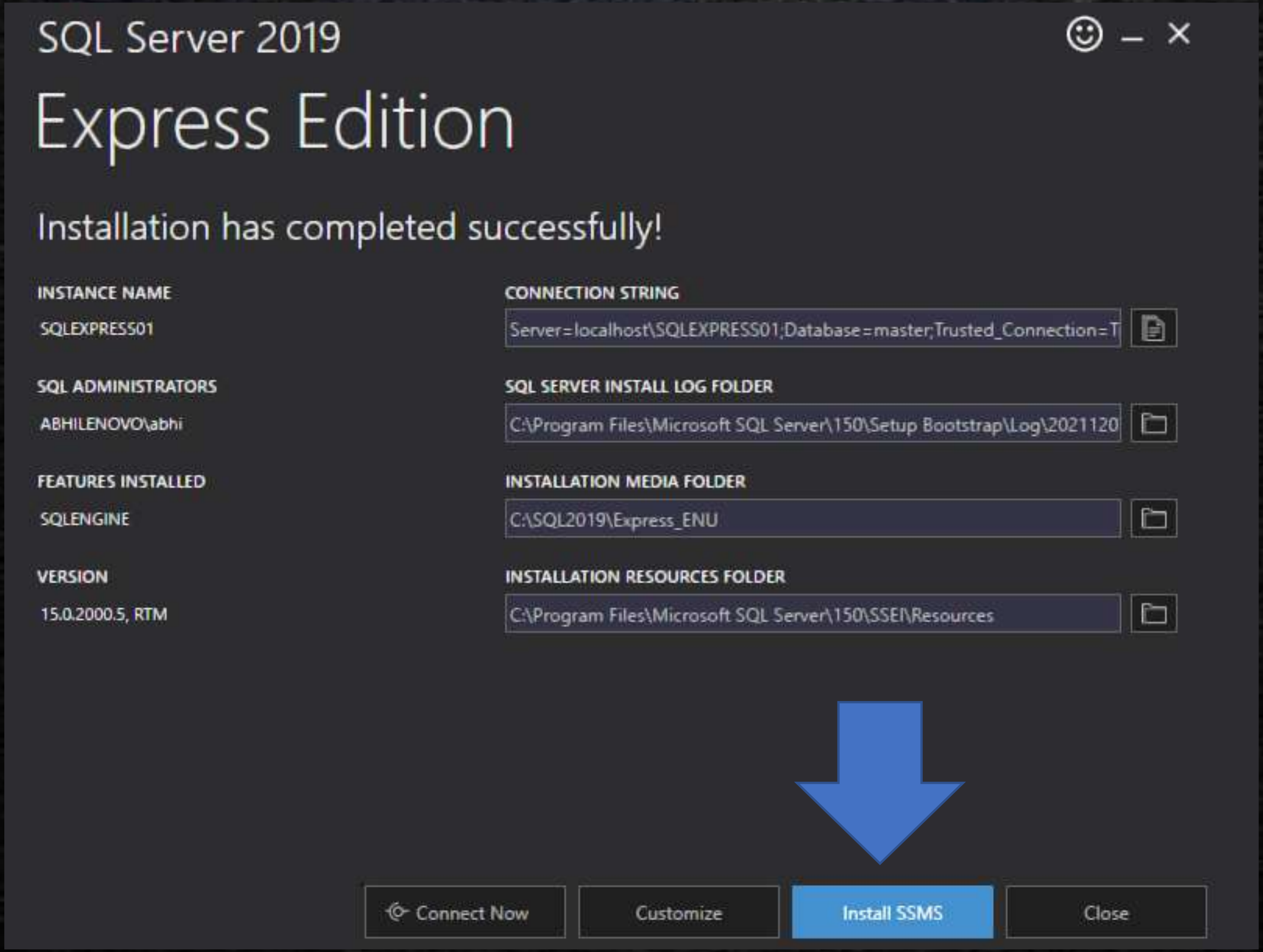


Installing Microsoft SQL Server





Installing Microsoft SQL Server



Installing Microsoft SQL Server Management Studio

Download SQL Server Management Studio (SSMS)

Article • 12/04/2021 • 7 minutes to read • +34

Is this page helpful?

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

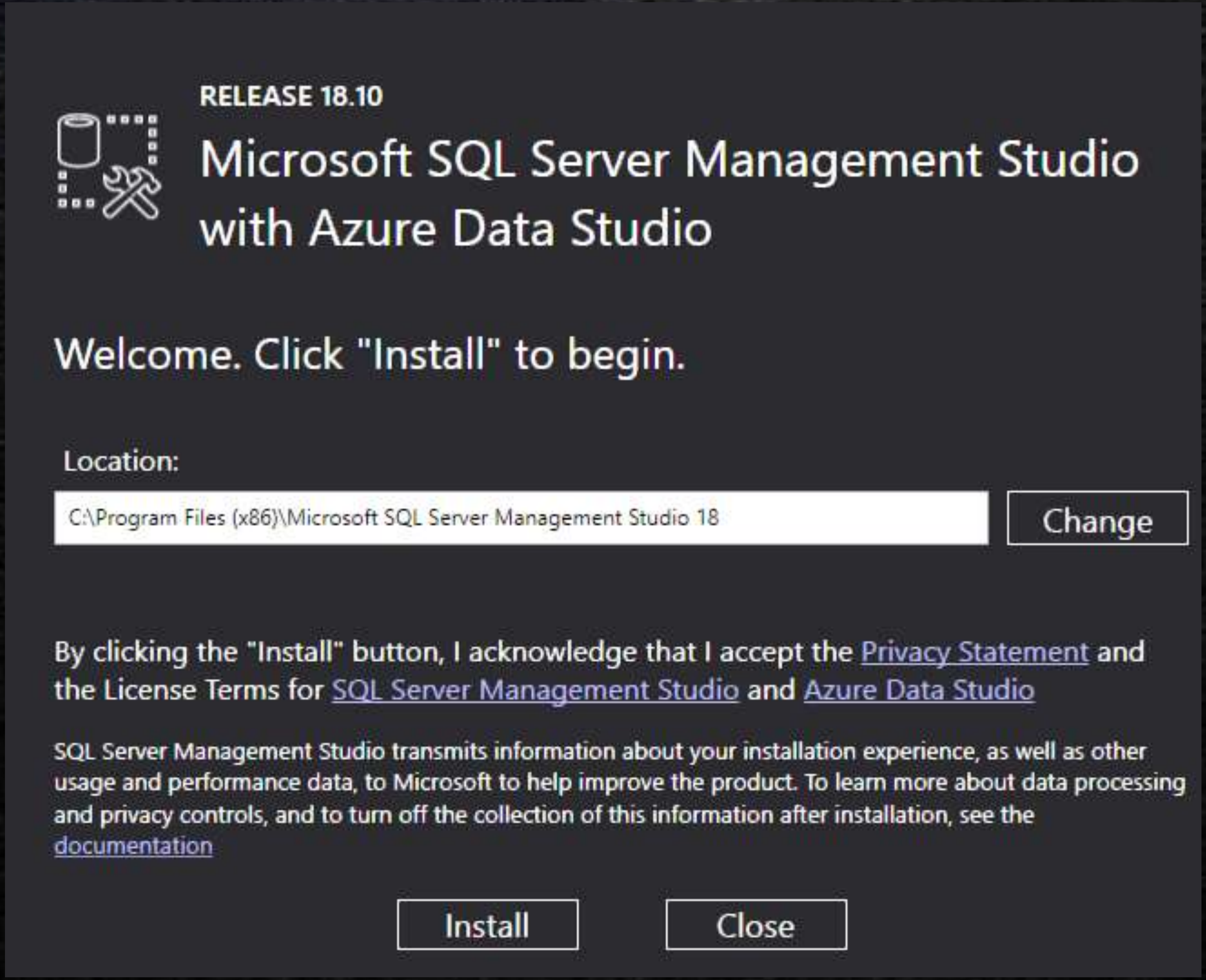
SQL Server Management Studio (SSMS) is an integrated environment for managing any SQL infrastructure, from SQL Server to Azure SQL Database. SSMS provides tools to configure, monitor, and administer instances of SQL Server and databases. Use SSMS to deploy, monitor, and upgrade the data-tier components used by your applications, and build queries and scripts.

Use SSMS to query, design, and manage your databases and data warehouses, wherever they are - on your local computer, or in the cloud.

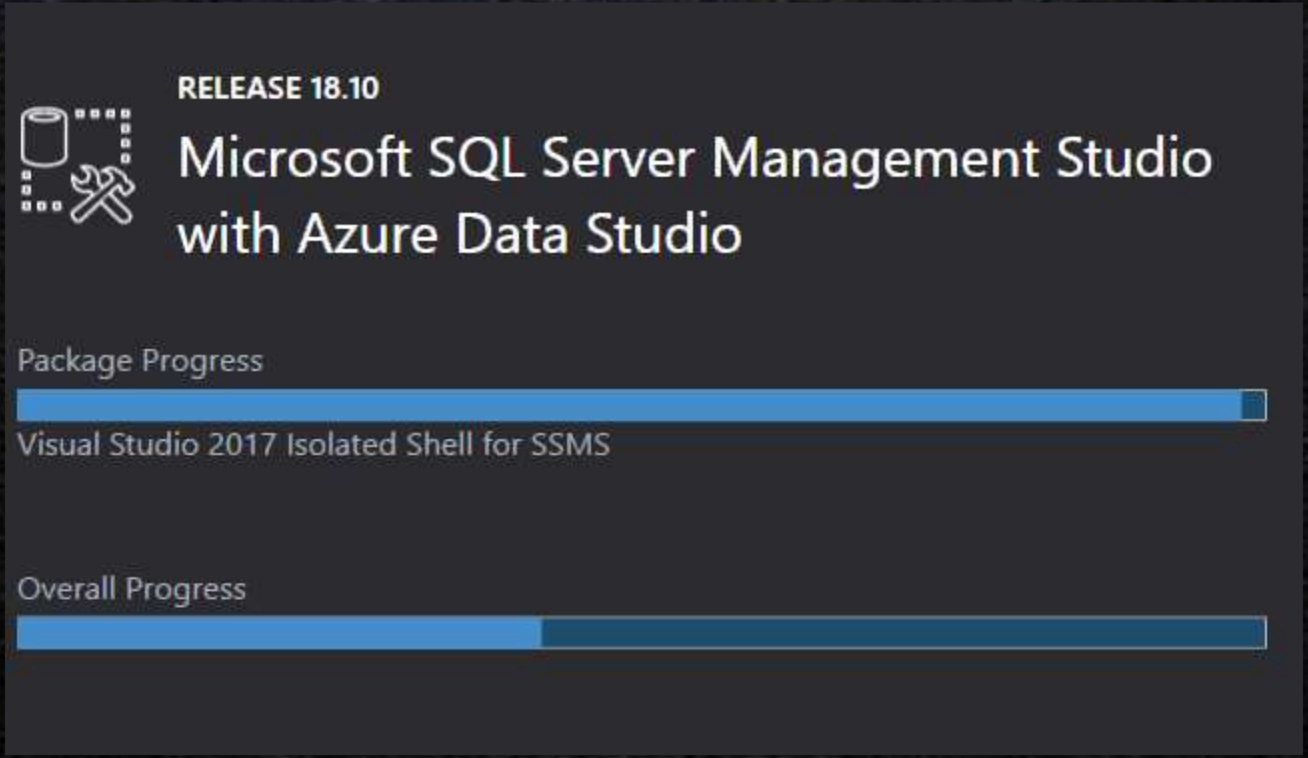
Download SSMS

Free Download for SQL Server Management Studio (SSMS) 18.10

Installing Microsoft SQL Server Management Studio

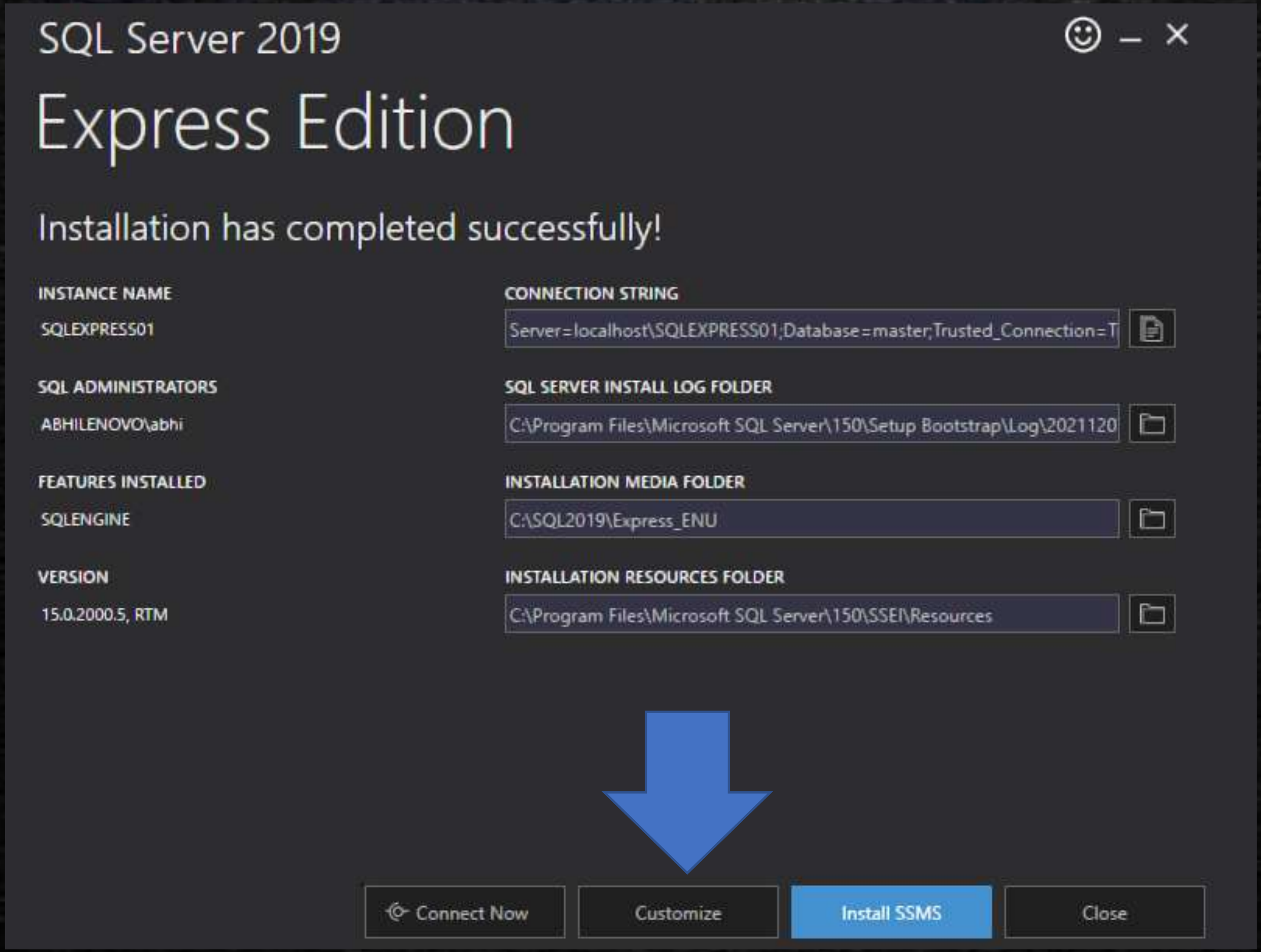


Installing Microsoft SQL Server Management Studio

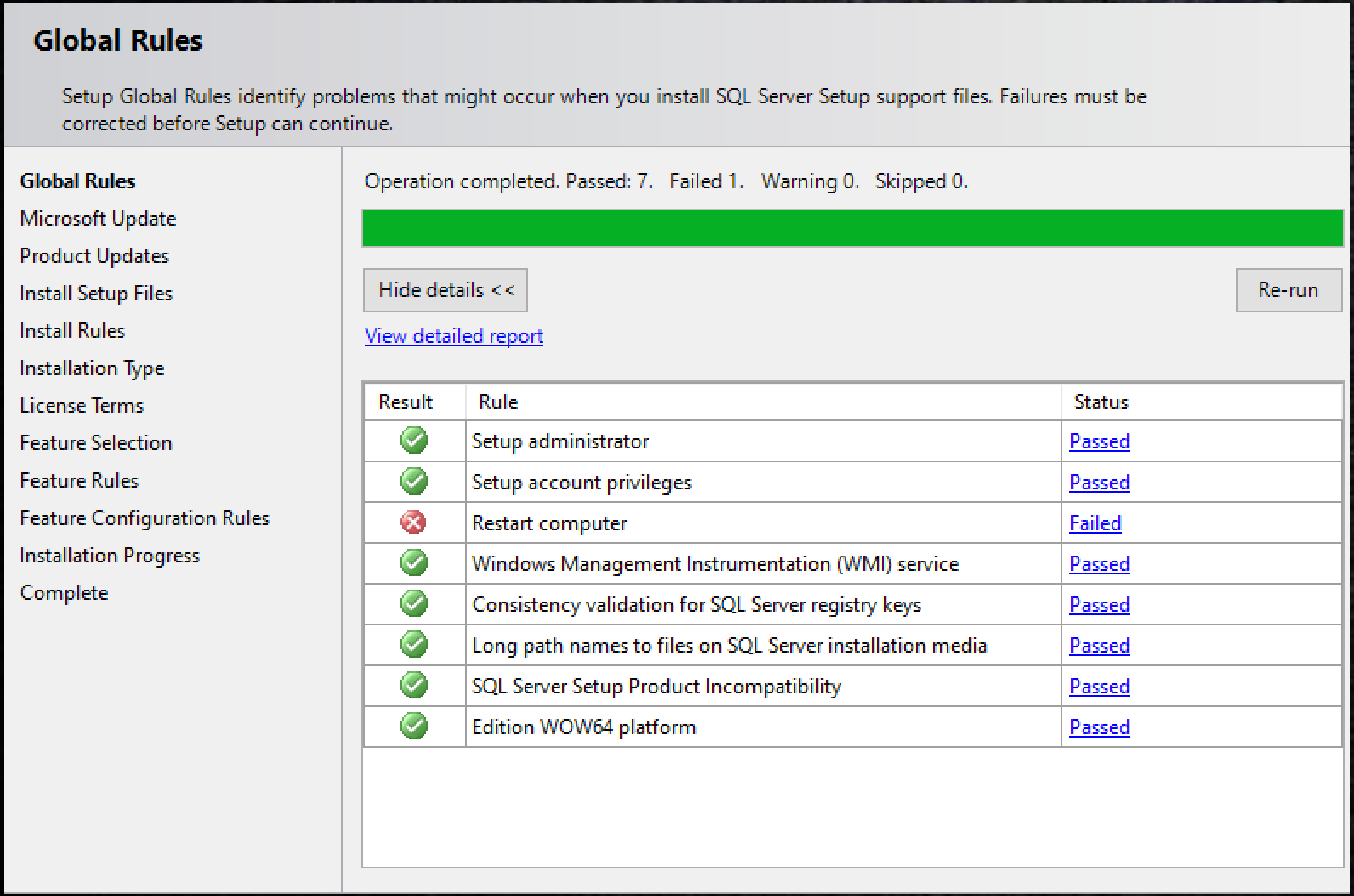


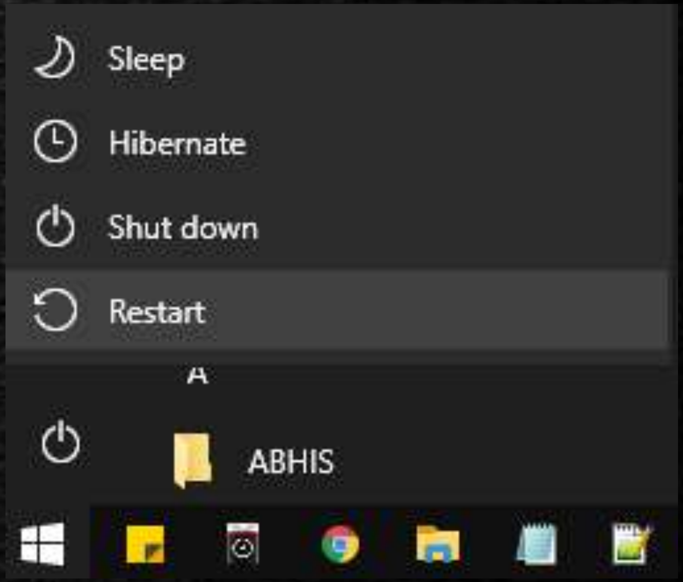
Installing Microsoft SQL Server Management Studio

Installing Microsoft SQL Server



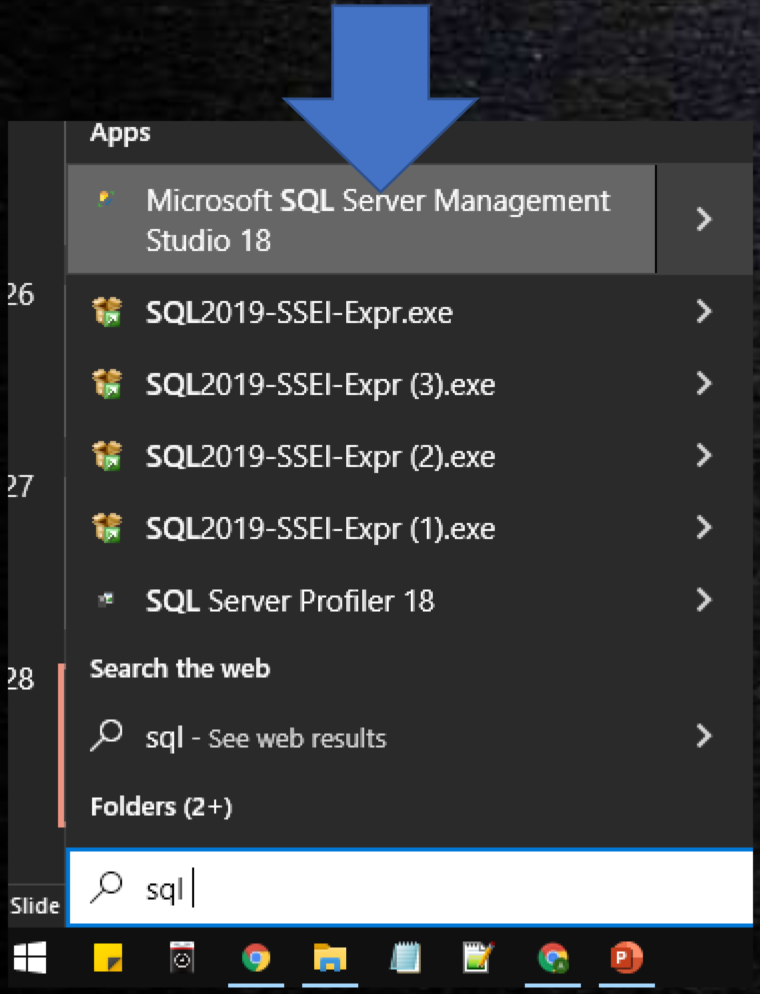
Installing Microsoft SQL Server

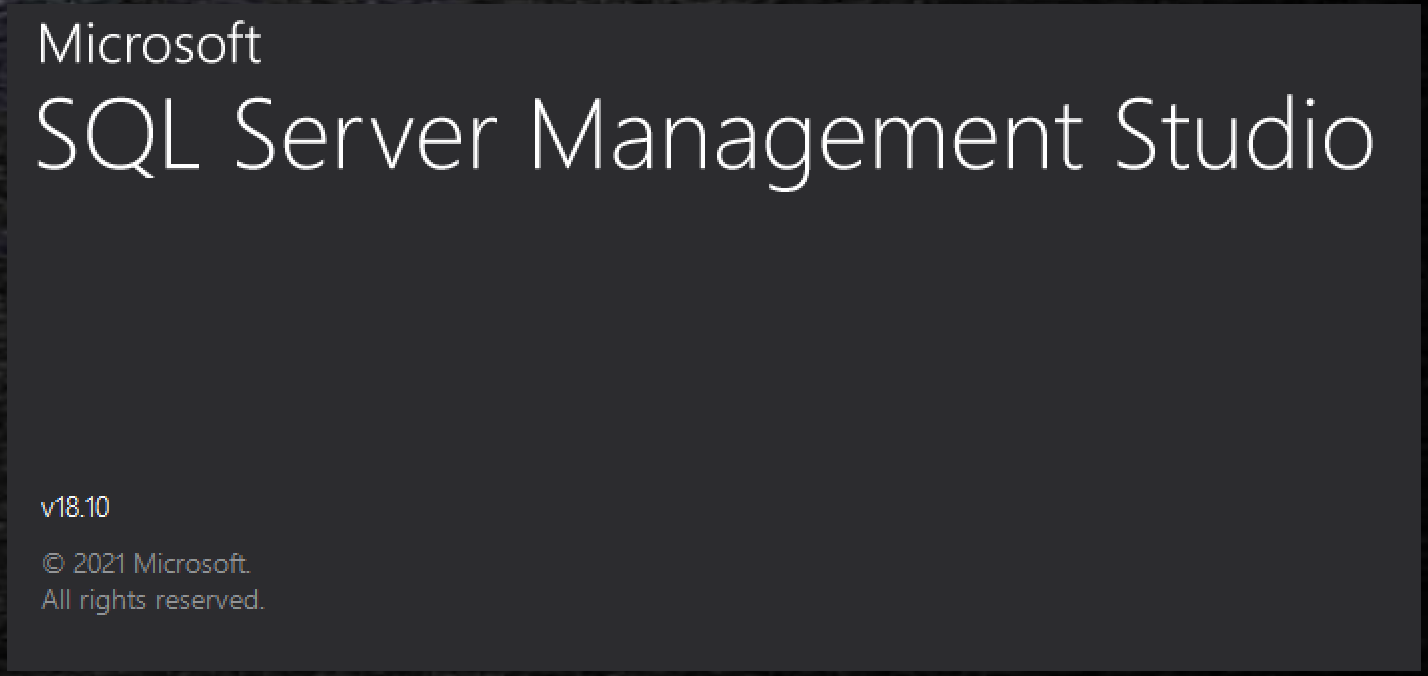




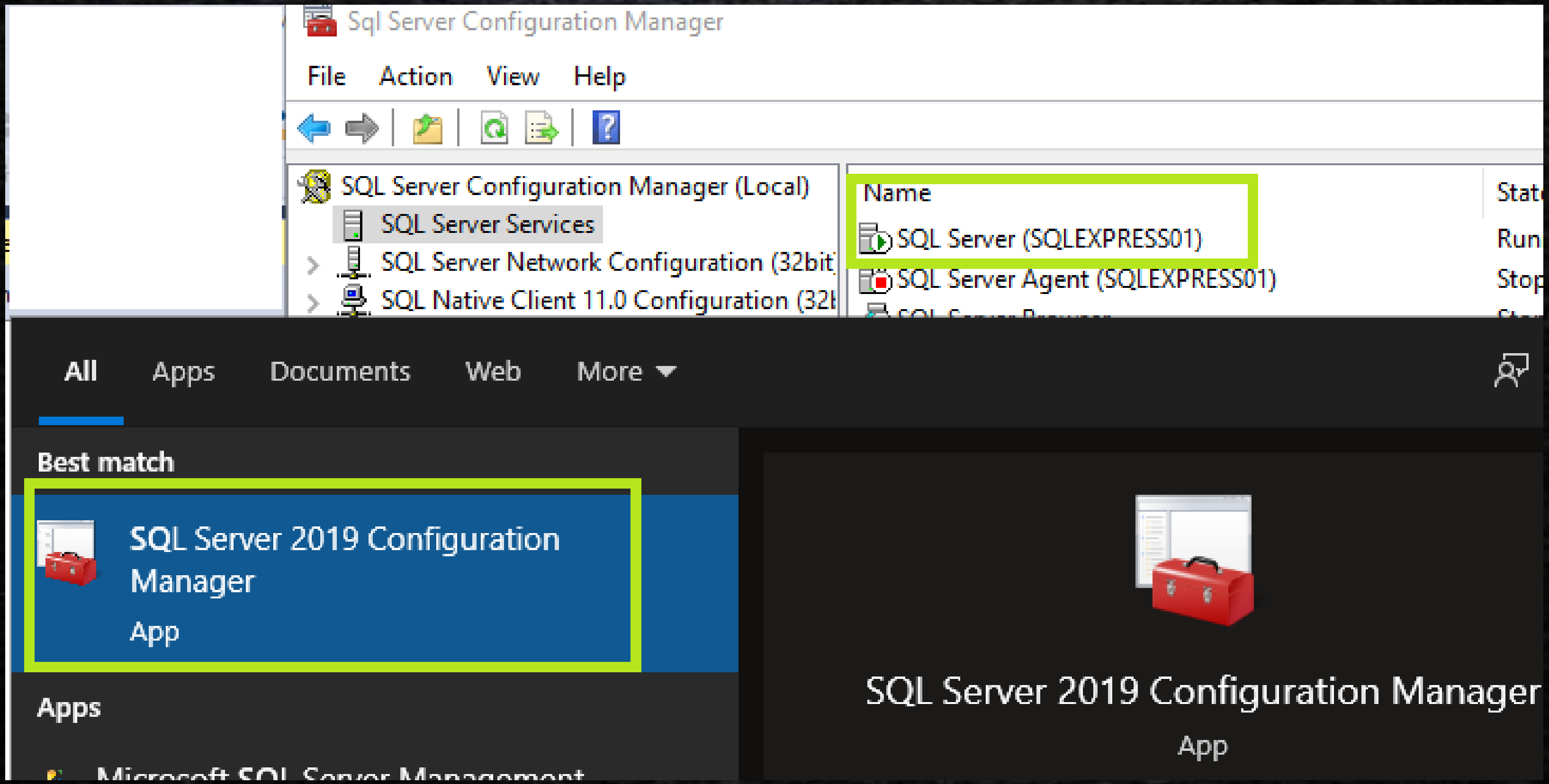
Close and Quit setup, then Restart your computer

Open Microsoft SQL Server Management Studio

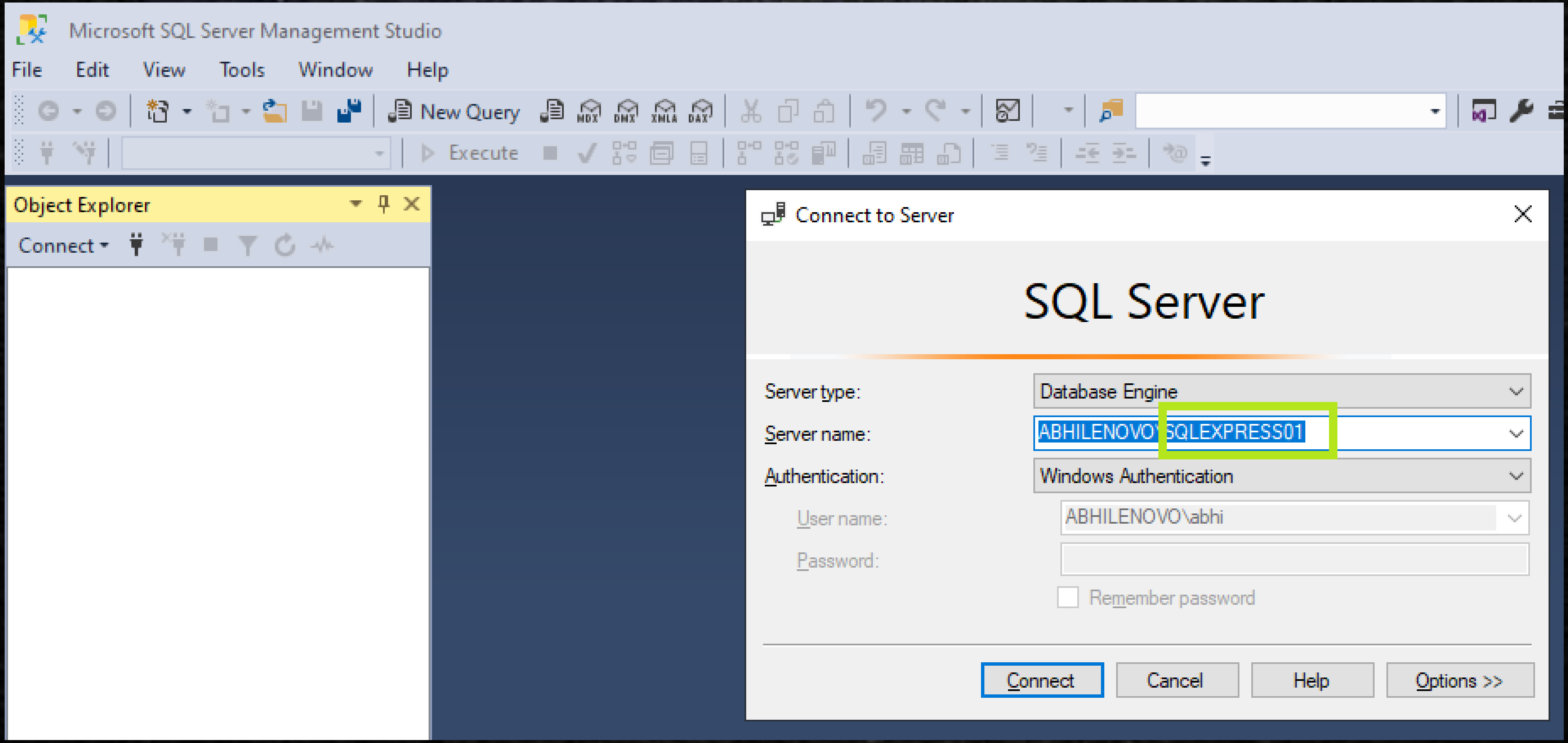




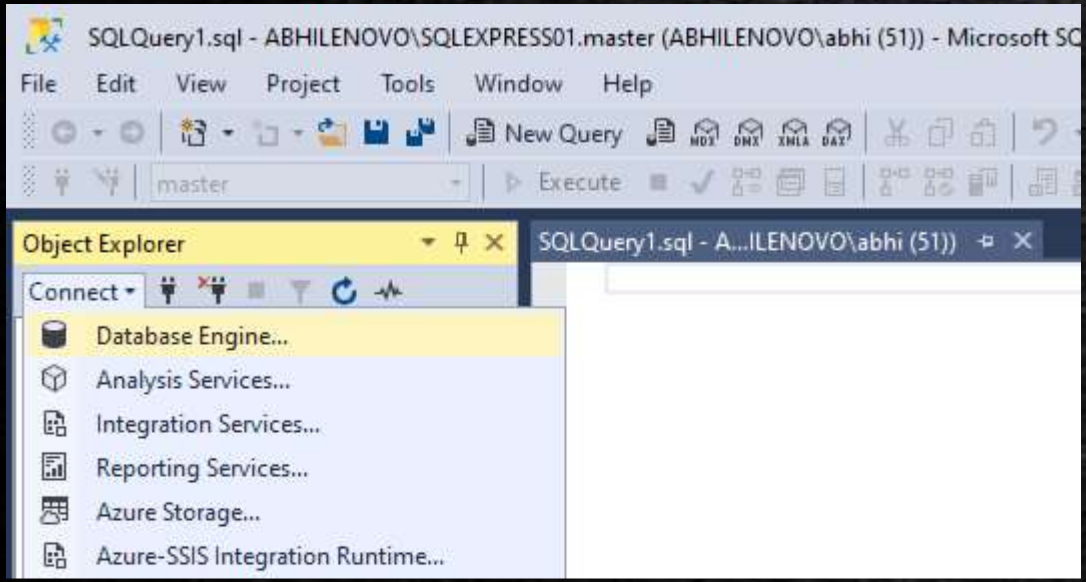
Open Microsoft SQL Server Config and find server name

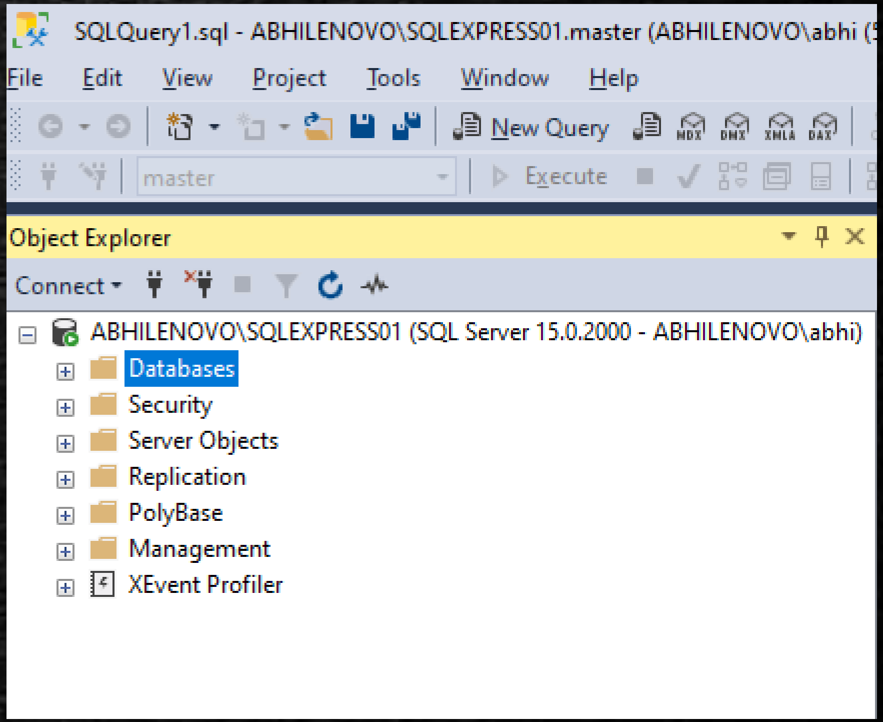


In SQL Server Management Studio connect to that server

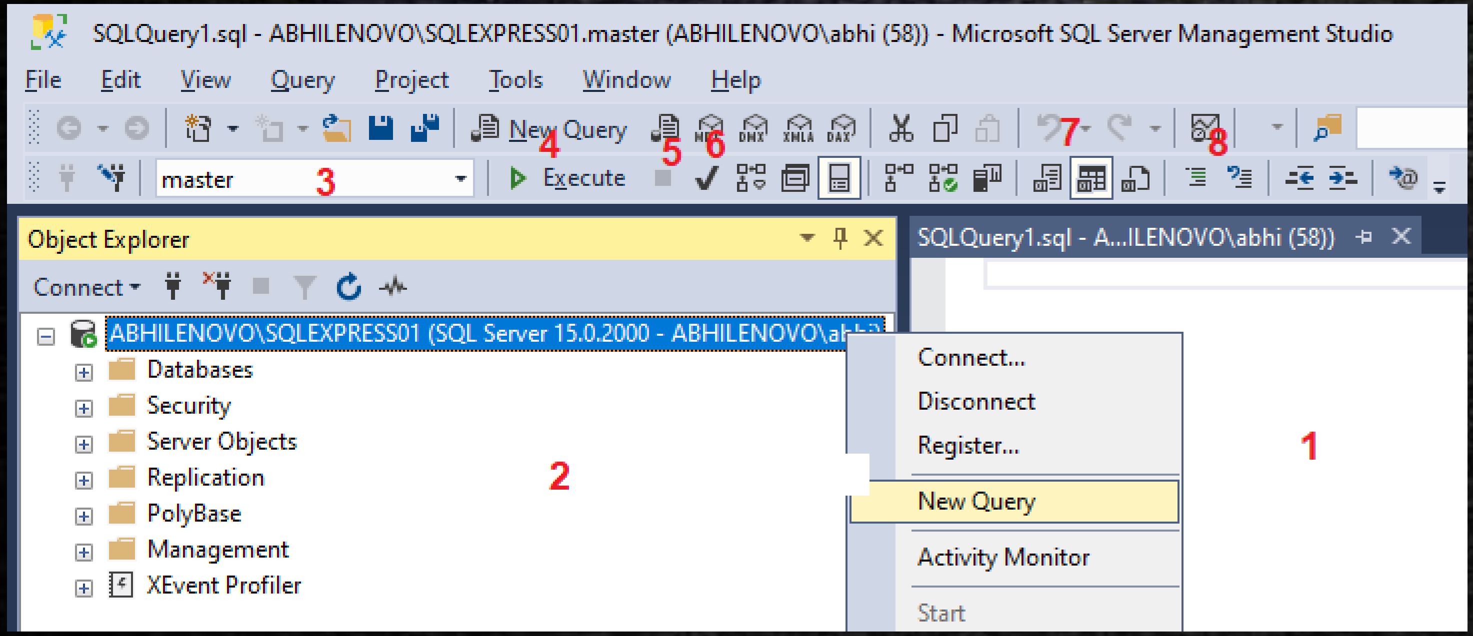


In SQL Server Management Studio connect to that server





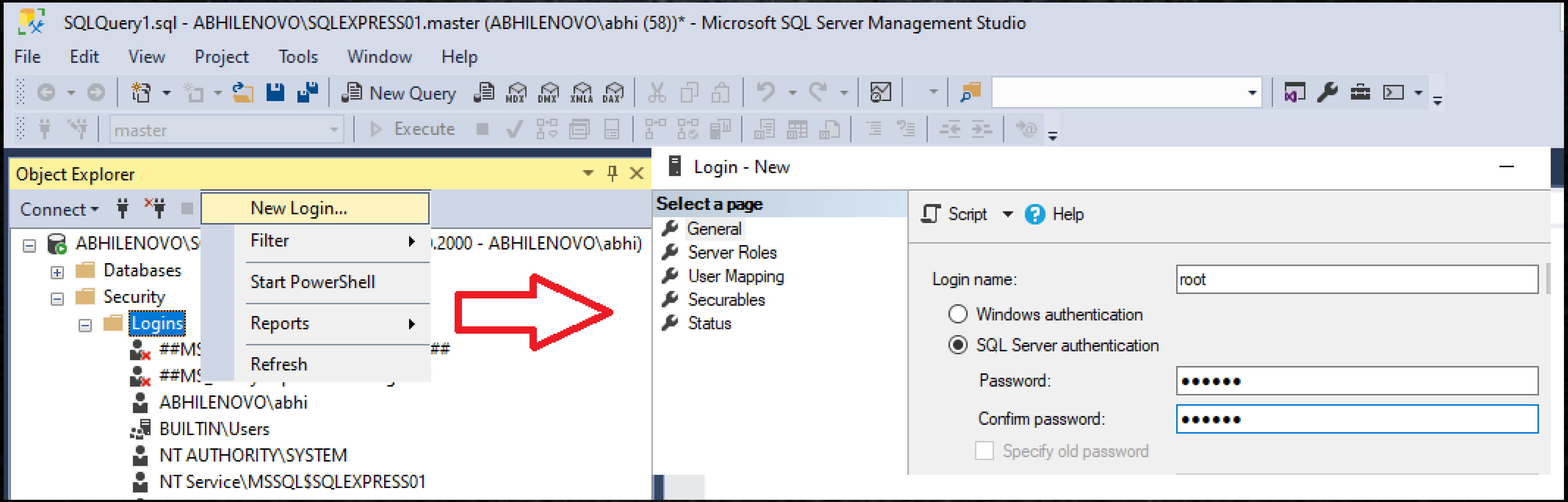
In SQL Server Management Studio Interface



7. Change Query Result Destination
8. Comment or uncomment (comment using - - before line)

1. Query Editor: This section is used to write the queries.
2. Object Explorer: Shows the database objects contained on the server in a tree format.
3. Databases Selection Dropdown : Select database to run the query
4. Execution button: Execute the query and get results
5. Cancel Query : Stop currently running query
6. Parse : Validate query syntax without checking the db objects

Option to Set a custom Login for SQL Server



- Just in case If we want to set a custom login for SQL Server.
- For our exercises we will be using the default 'Windows Authentication'

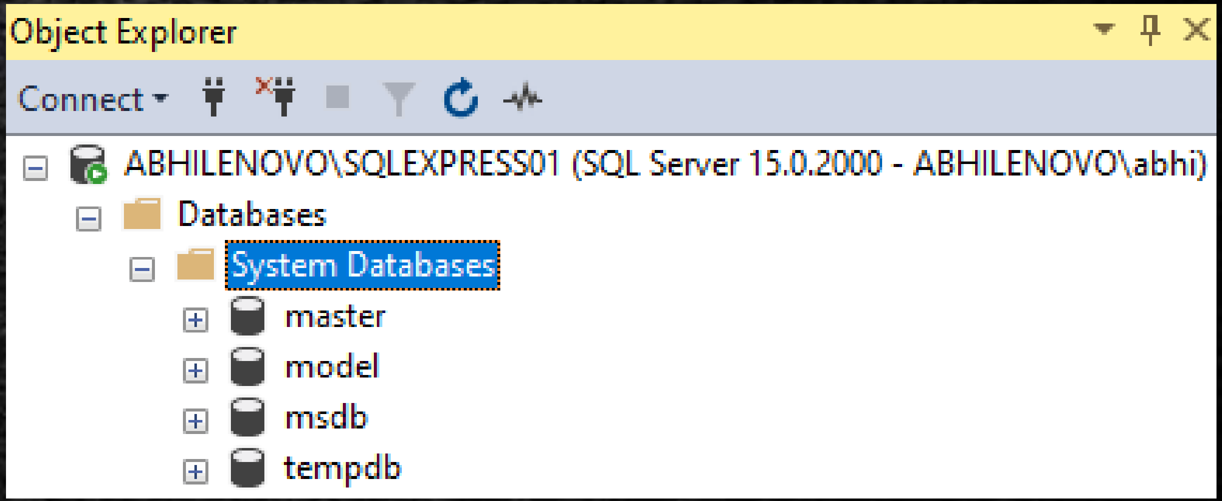
SQL Basic DB Operations

CREATE, USE, DROP and BACKUP DB

Types of DB in SQL Server

SQL Server has two types of database:

- System databases
- User Databases



- System databases are created automatically while installing the MS SQL Server.

- It is essential to run the server efficiently.

SQL Basic DB Operations

CREATE, USE, DROP and BACKUP DB

Create a new user database

A new database in SQL Server can be created in two ways:

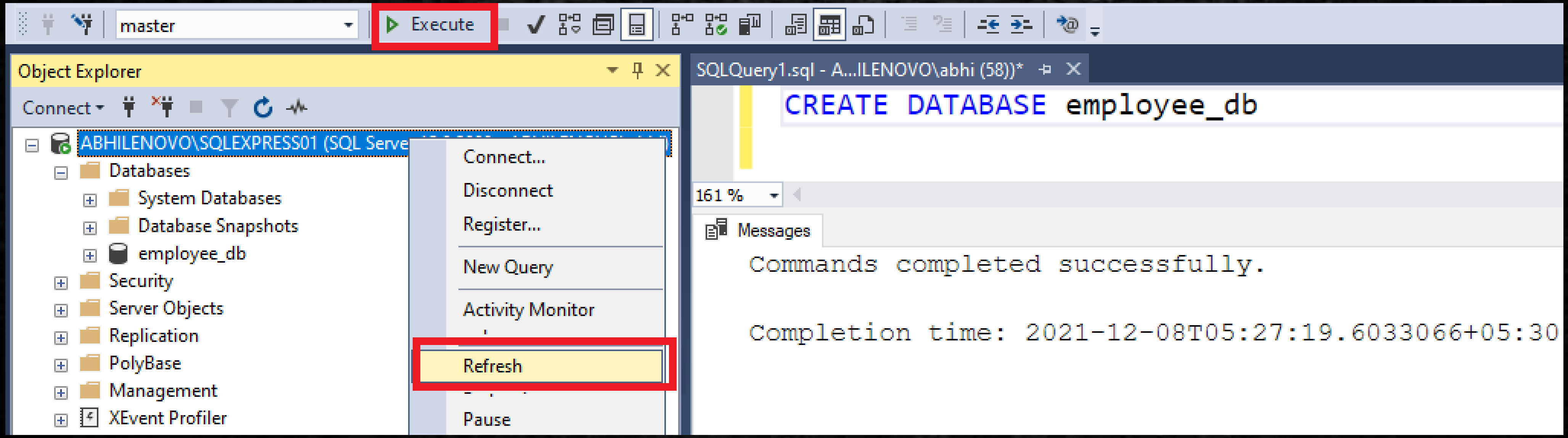
- Transact-SQL Command
- SQL Server Management Studio

*T-SQL, stands for Transact-SQL and is referred to as TSQL, is an extension of the SQL language by Microsoft and Sybase, used primarily within Microsoft SQL Server. This means that it provides all the functionality of SQL but with some added extras.

- Using the TSQL* Command: 'CREATE DATABASE db_name '

Sybase is now a part of SAP

But in earlier days Sybase and Microsoft has worked together.

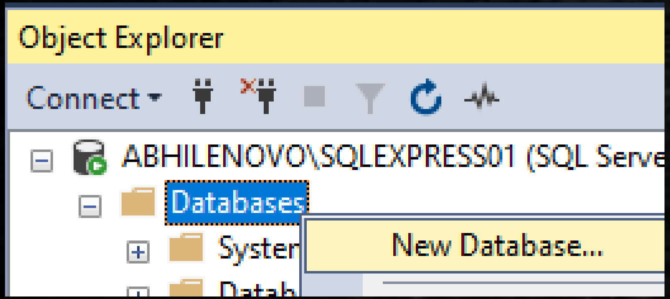


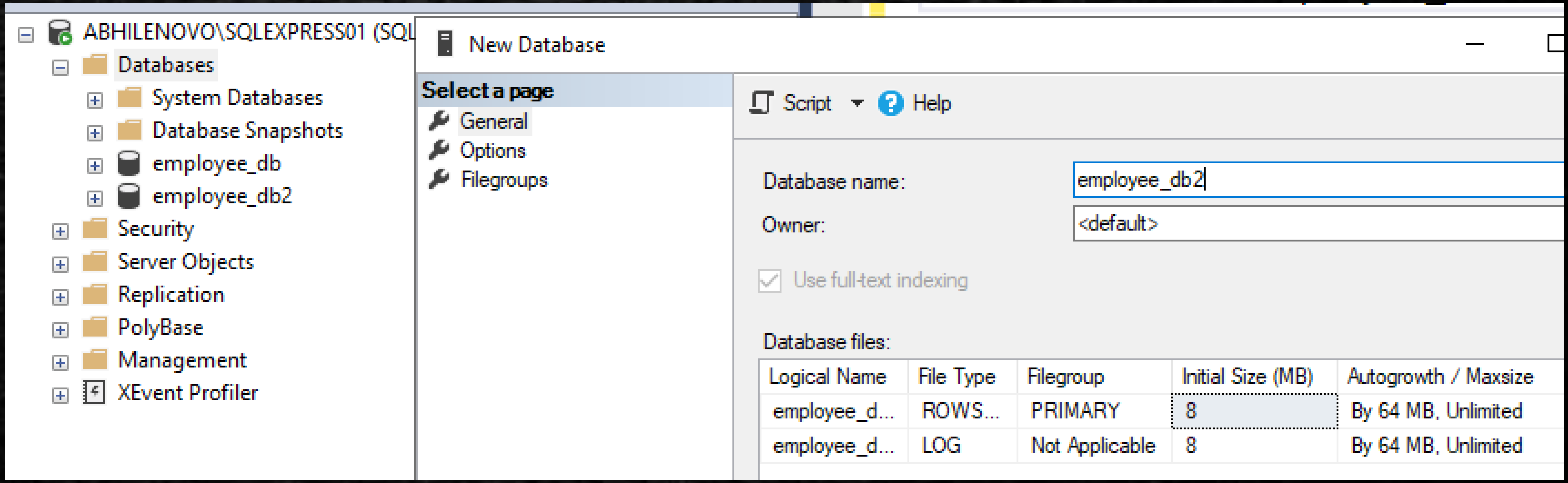
Create a new user database

A new database in SQL Server can be created in two ways:

- Transact-SQL Command
- SQL Server Management Studio

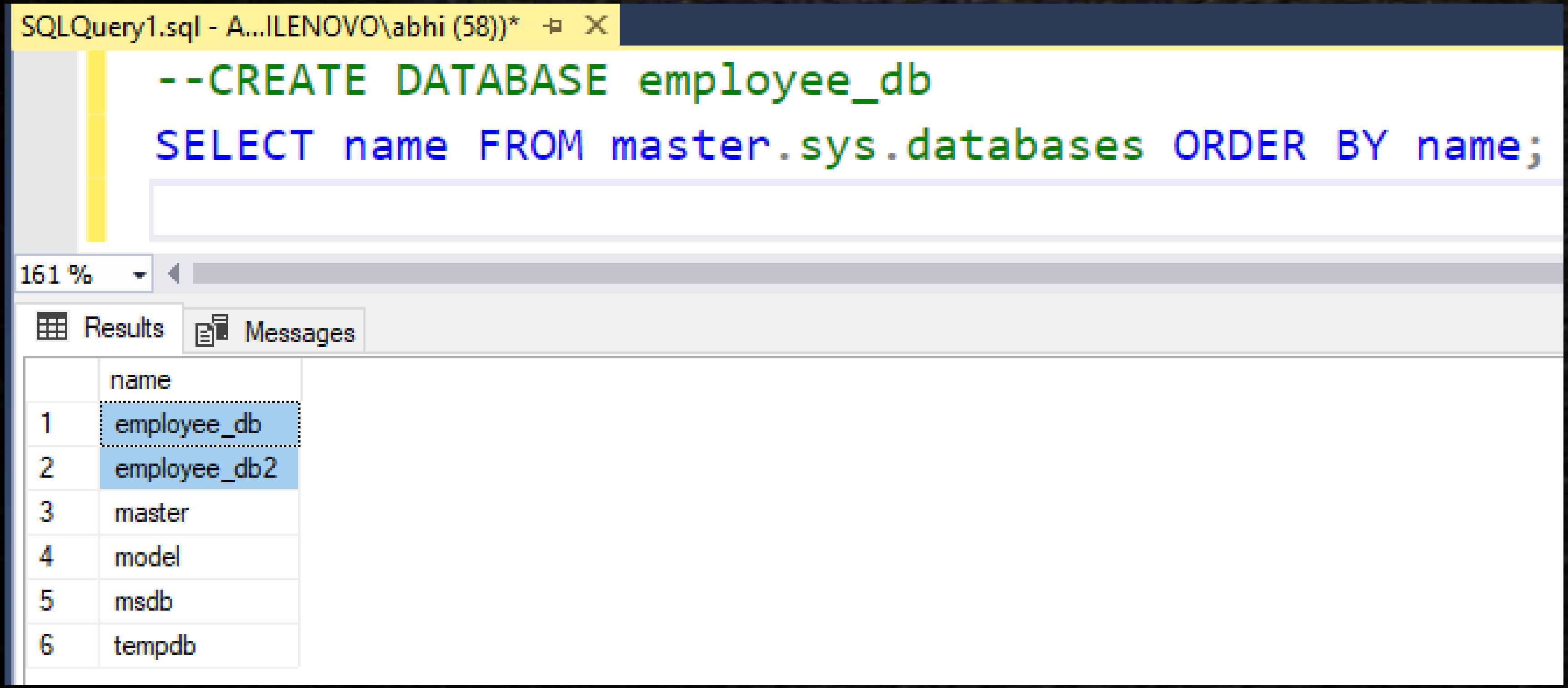
- Using SQL Server Management Studio

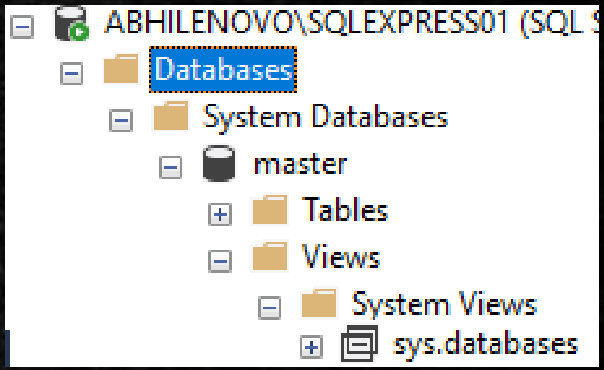




List All Databases (MSSQL)

Executing this stored procedure will display the 'view' of all databases

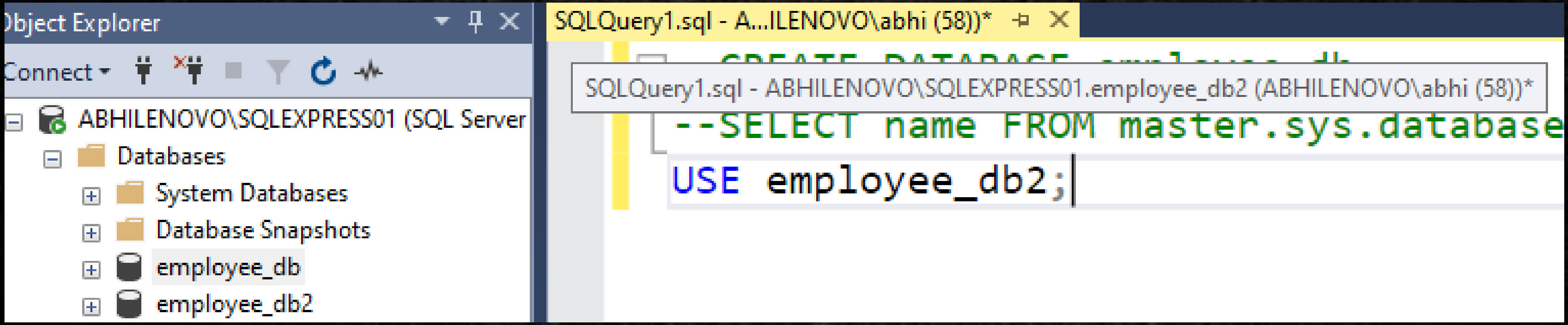




Select a database (SQL)

You may either click on the db on the left side and execute query OR

Use the 'USE db_name;' SQL query



Backup a database (SQL)

You may either right click on the db and select 'Tasks >> Backup' OR

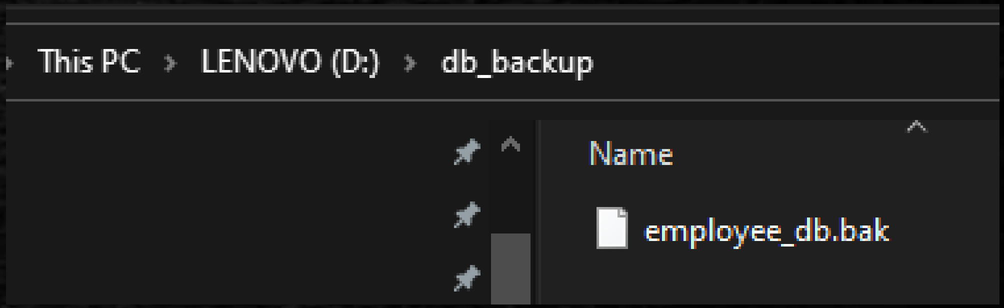
Use the 'BACKUP DATABASE db_name TO DISK = 'path';' SQL query

```sql
BACKUP DATABASE employee_db TO DISK = 'D:\db_backup\employee_db.bak'
```

61 %

Messages

```
Processed 376 pages for database 'employee_db', file 'employee_db' on file 1.
Processed 2 pages for database 'employee_db', file 'employee_db_log' on file 1.
BACKUP DATABASE successfully processed 378 pages in 0.289 seconds (10.204 MB/sec).
```



Backup a database (SQL)

You may either right click on the db and select 'Tasks >> Backup' OR

Use the 'BACKUP DATABASE db_name TO DISK = 'path' WITH DIFFERENTIAL;' SQL query to backup only the changes

```sql
BACKUP DATABASE employee_db TO DISK = 'D:\db_backup\employee_db.bak' WITH DIFFERENTIAL
```

Messages

```
Processed 56 pages for database 'employee_db', file 'employee_db' on file 2.
Processed 2 pages for database 'employee_db', file 'employee_db_log' on file 2.
BACKUP DATABASE WITH DIFFERENTIAL successfully processed 58 pages in 0.062 seconds (7.245 MB/sec).
```

Restore database from backup

You may either right click on the db and select 'Tasks >> Restore >>Database'
OR

Use the RESTORE DATABASE employee_db

FROM DISK = 'C:\dbbakup\employee_db.bak'

WITH REPLACE;

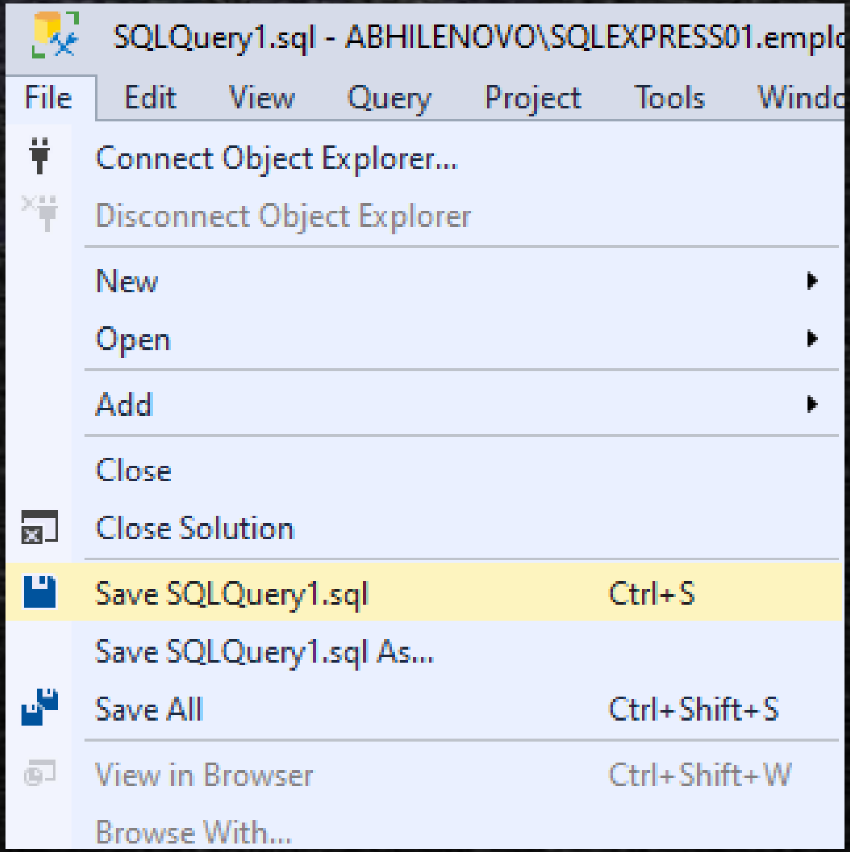
--WITH REPLACE if you want to overwrite the existing db

```
--use any other database to restore the backup
use master

RESTORE DATABASE employee_db
FROM DISK = 'C:\dbbakup\employee_db.bak'
WITH REPLACE
```

Backup the SQL file

We can re-use the queries by saving them as a .sql file

File >> Save

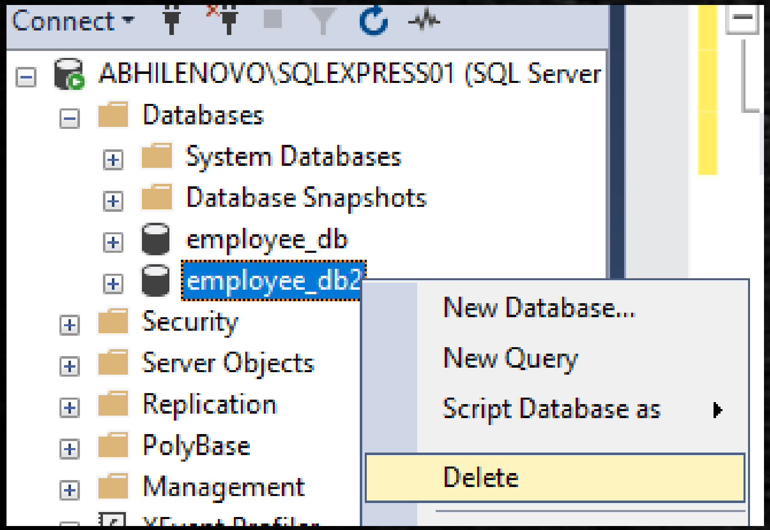


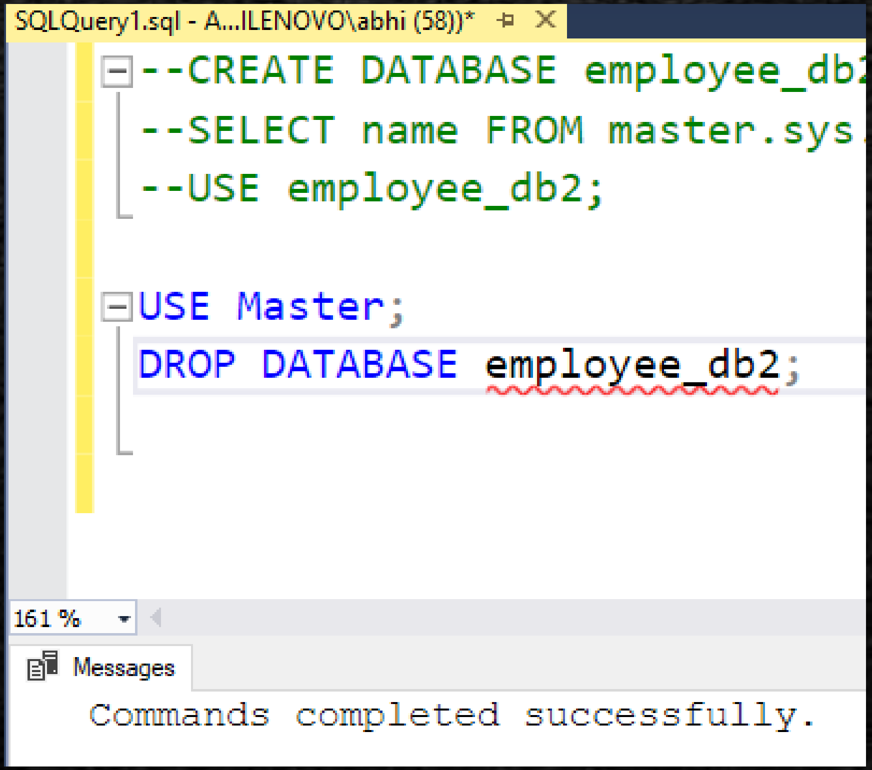
Delete a database (SQL)

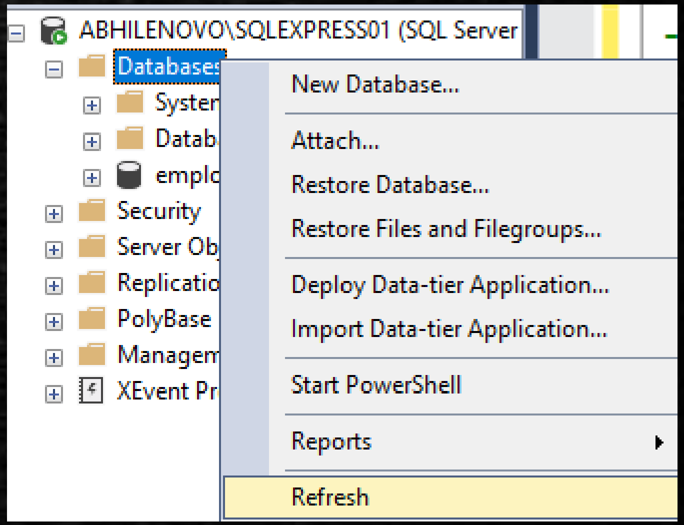
You may either right click on the db and select 'Delete' OR

Use the 'DROP DATABASE db_name;' SQL query

(Make sure to 'unuse' the db before drop)







Database Schema in MSSQL

The ‘container’ for DB Objects

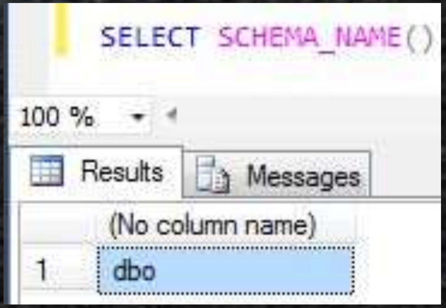
Database Schema in SQL Server

- Schema is a logical collection of database objects such as tables, views, stored procedures, indexes, triggers, functions.
- It can be thought of as a ‘container’, created by a database user.
- The database user who creates a schema is the schema owner.

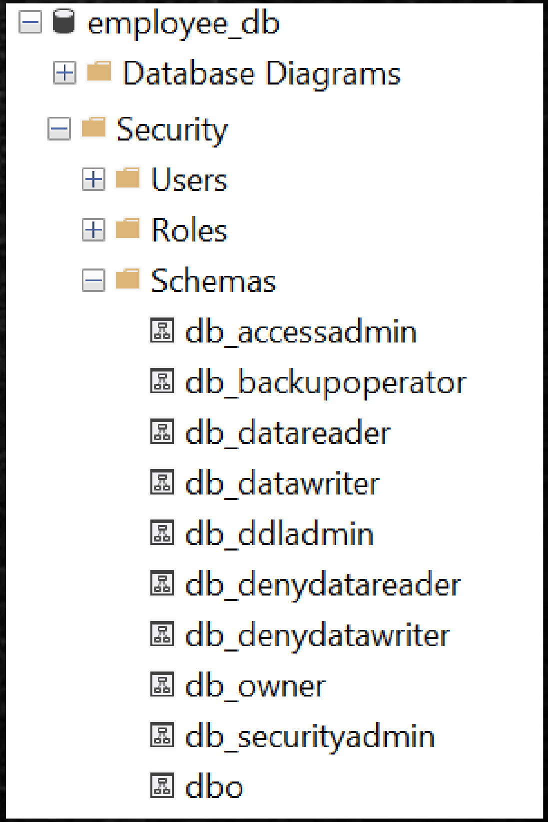
- More details about schema
- The schema ownership is transferrable.
- Database objects can be moved among the schemas.
- A single schema can be shared among multiple users.
- A user can be dropped without dropping the database objects associated with the user.
- Dbo (or database owner) is the default schema for a newly created database.

Viewing Current Schema of DBO

- Using the "SCHEMA_NAME" function we can determine the default schema for the database.



- OR we can also use the SSMS interface also.



Create a New Schema

- This command will create a new schema called myschema1 and the default user dbo will be the schema owner.

```
CREATE SCHEMA myschema1
```

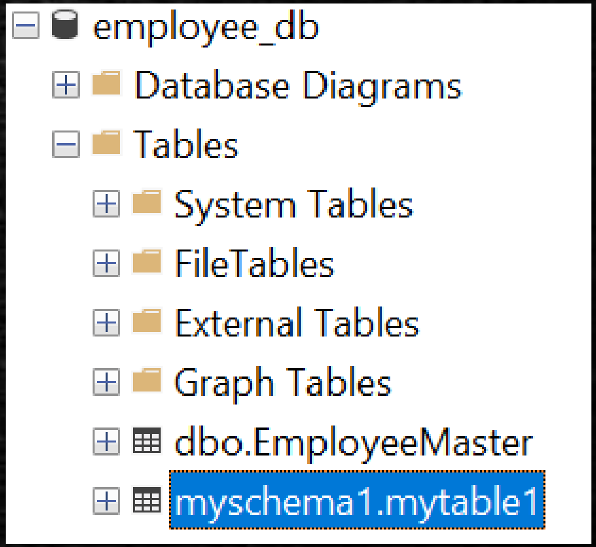
- Or we can explicitly specify the schema owner.

```
CREATE SCHEMA myschema2 AUTHORIZATION dbo
```

Create Objects Under the Schema

- After we create a schema, you can create objects under this schema and grant permissions to other users.

```
CREATE TABLE myschema1.mytable1
(
     ID int,
     FirstName nvarchar(50) NOT NULL,
     LastName nvarchar(50) NOT NULL
);
```



Alter an Already Created Schema

- We can use the ALTER SCHEMA statement to transfer database objects from one schema to another schema in the same database.

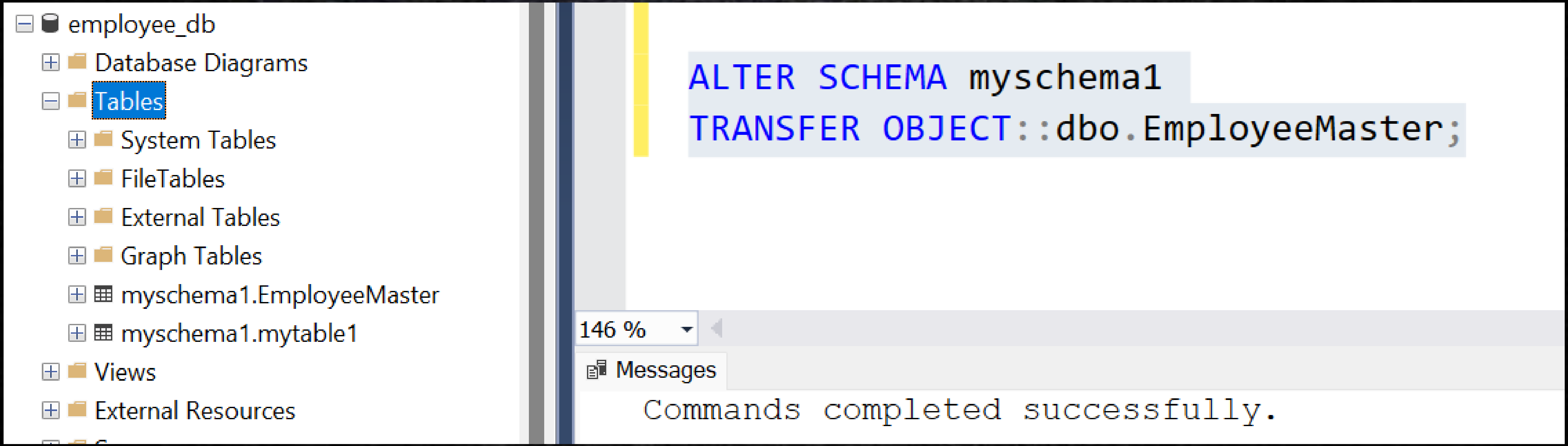
Syntax

```
ALTER SCHEMA <schema_name>
TRANSFER [entity_type::]securable_name;
```

schema_name is the name of a schema in the current database, into which the securable (table, view, stored procedure, etc) will be moved.

entity_type can be Object, Type or XML Schema Collection.

Alter an Already Created Schema



Change Ownership of an Already Created Schema

- We can use ALTER AUTHORIZATION statement to change the owner of the schema.

```
ALTER AUTHORIZATION ON SCHEMA :: myschema1 TO dbo
```

Delete a Schema

- DROP SCHEMA deletes a schema from the database.
- The schema that is being dropped must not contain any database objects.

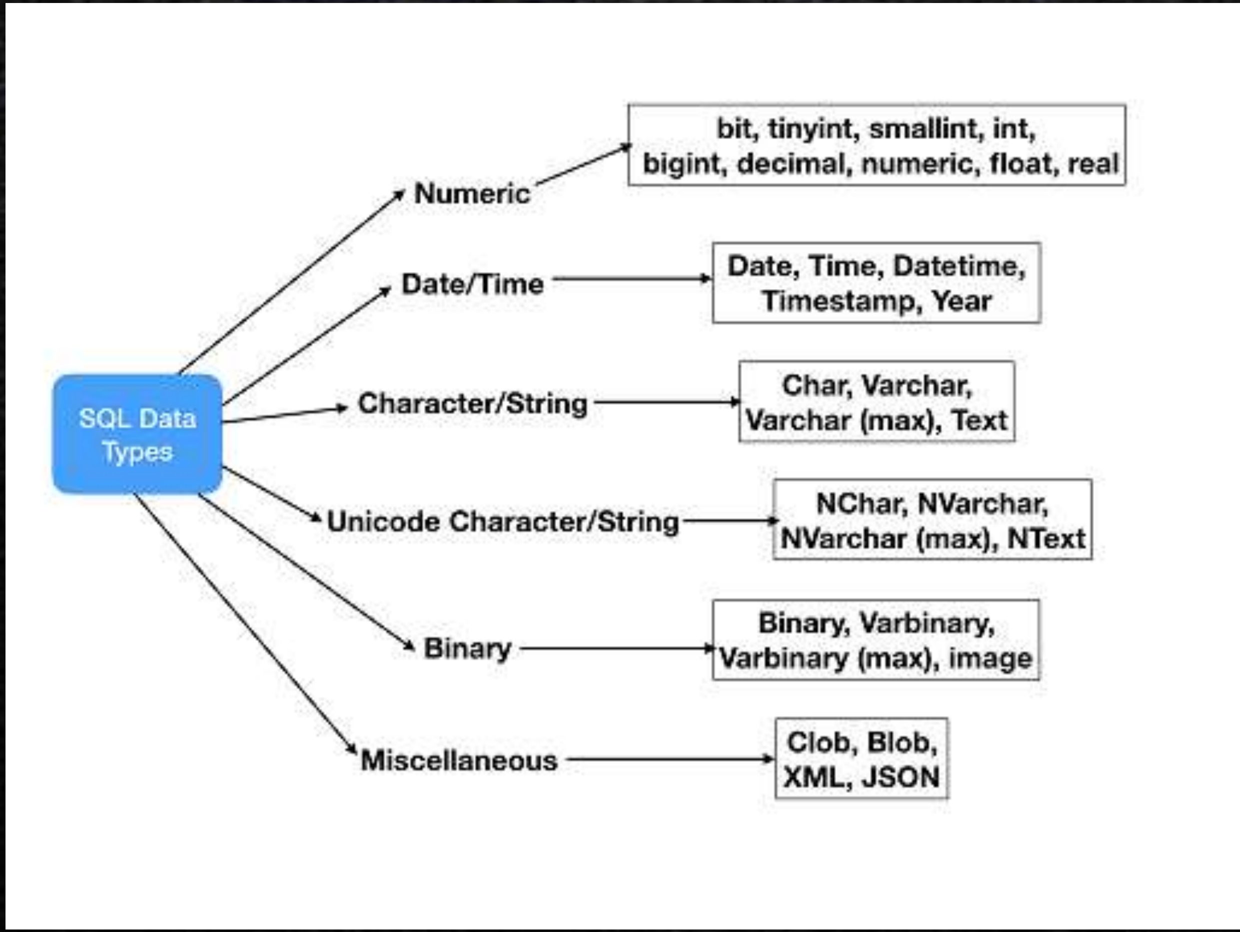
```
ALTER SCHEMA dbo
TRANSFER OBJECT::myschema1.EmployeeMaster;

DROP SCHEMA IF EXISTS myschema1
```

SQL Basic Table Operations

Basic DDL Operations: CREATE, DROP, ALTER

SQL Data Types

Using Data Type, we specify what type of data is expected inside of each column. The common data types are :



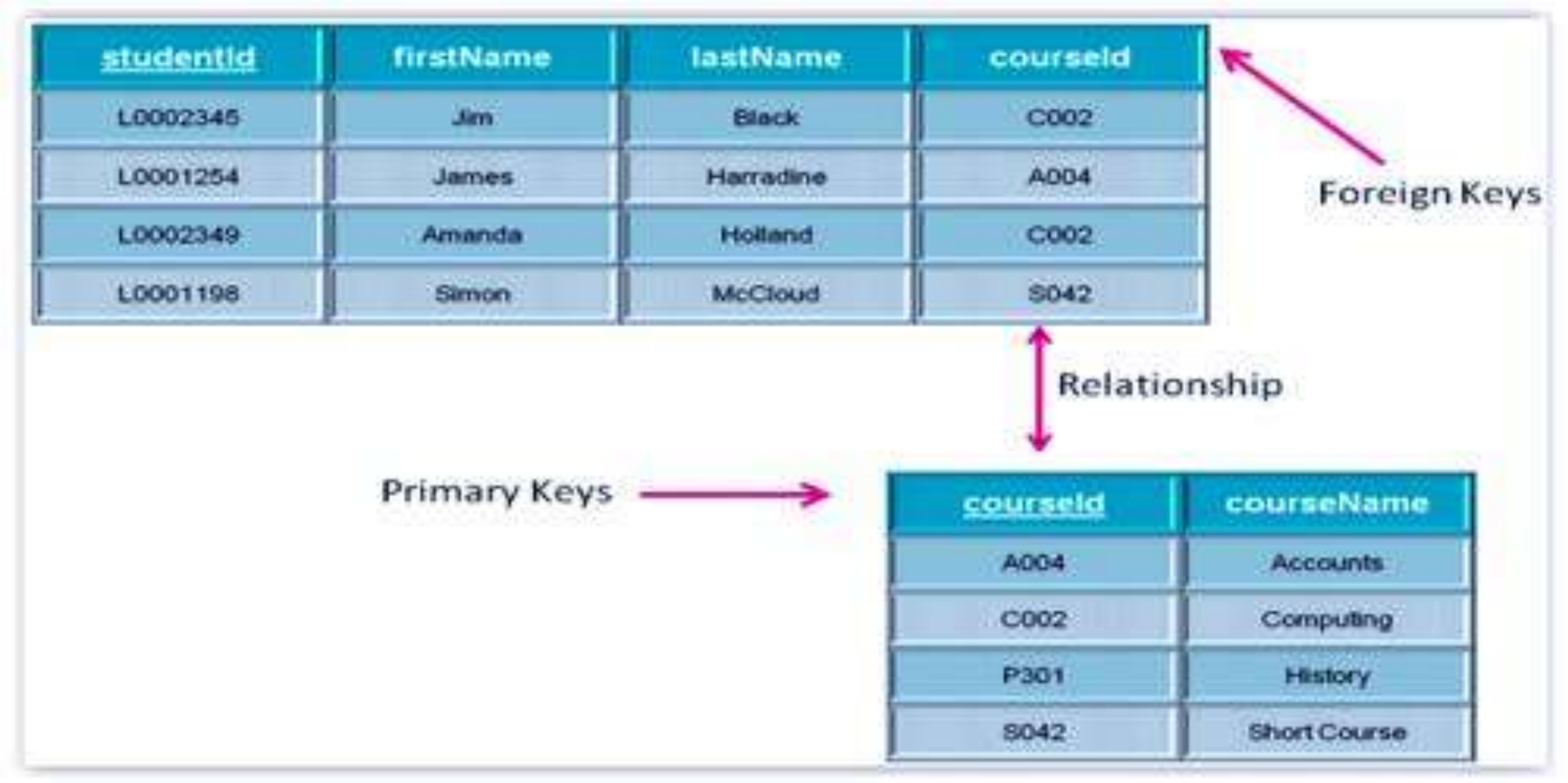
SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Column level constraints apply to a column, and table level constraints apply to the whole table.

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE.
- IDENTITY data type generates autoincrementing integer
- FOREIGN KEY - links between tables
- CHECK - Ensures values satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - To create and get data from database very quickly

Primary Key vs Foreign Key

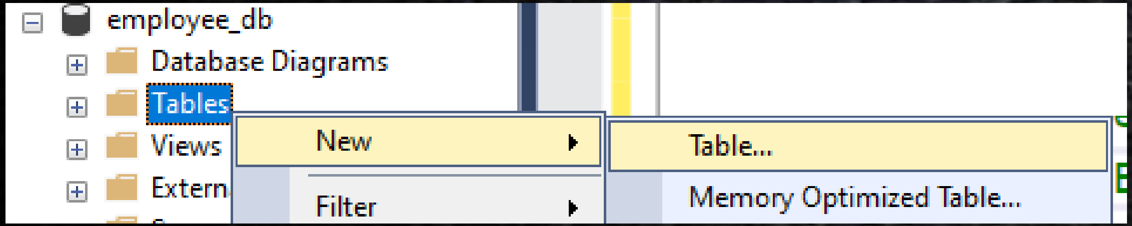


studentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042

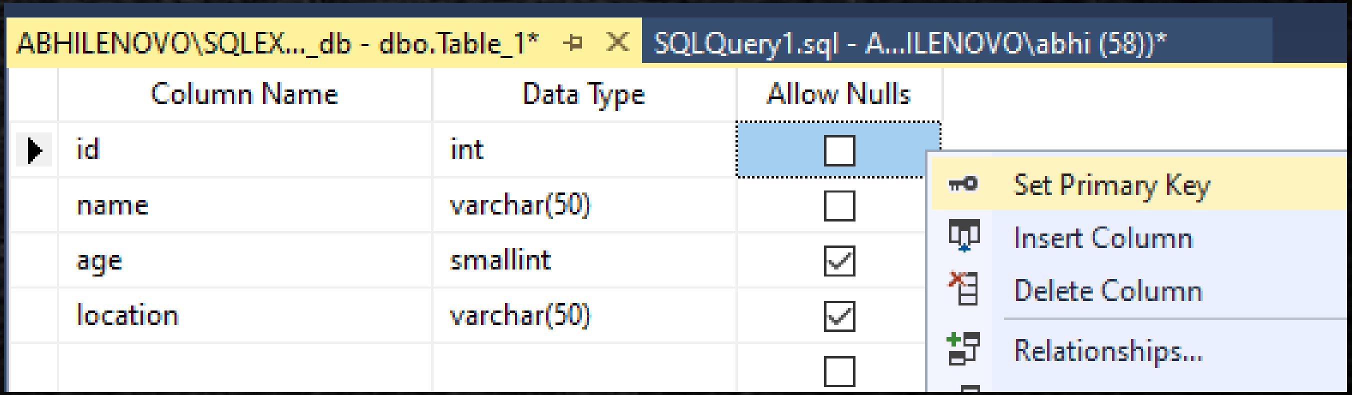
courseId	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short Course

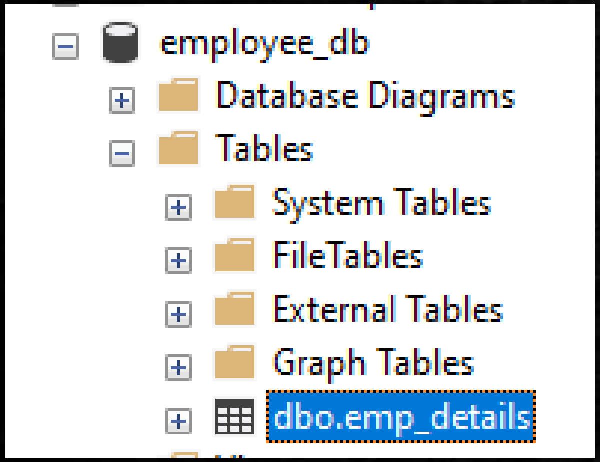
Create Table in the database

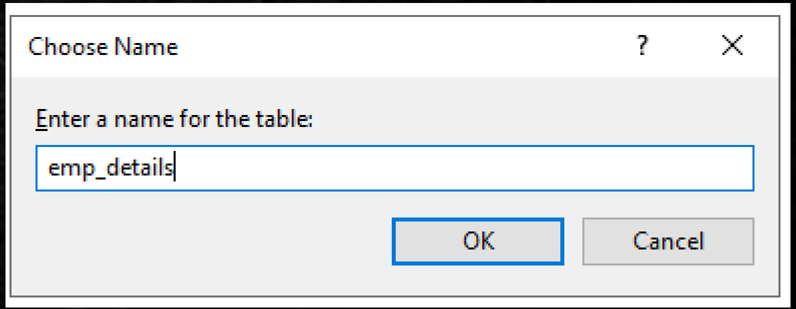
Using Management Studio:



After adding columns press ctrl+S to save the table

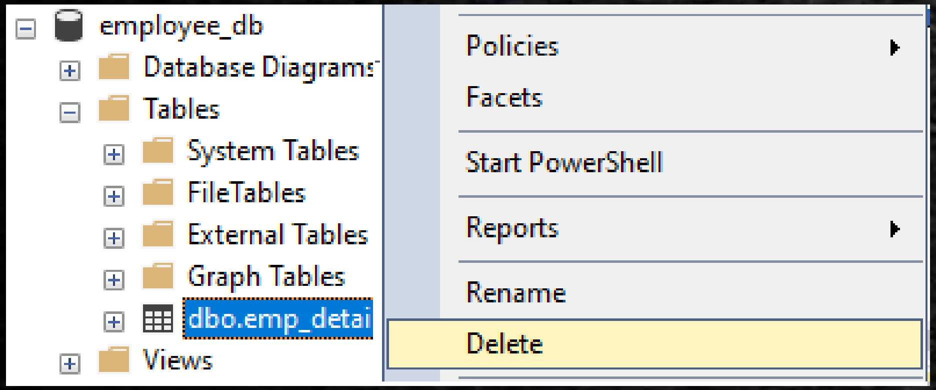


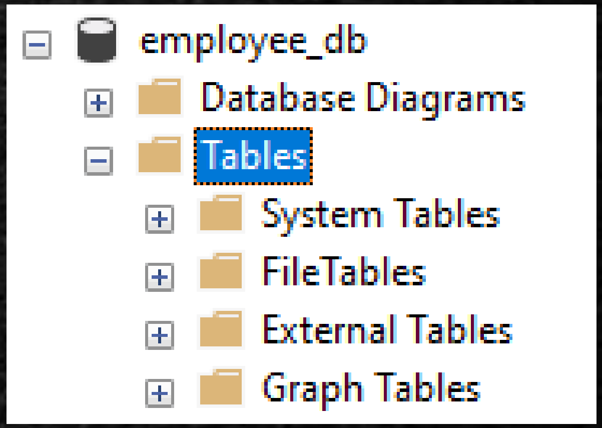




Delete Table from the database

Using Management Studio:





CREATE a table in database

Use the CREATE TABLE [database_name.] table_name (

```
    column_definition1,
    column_definition2,
    ........,
    table_constraints
);
```

SQL query to create new table

CREATE an employee table in database

Use the CREATE TABLE table_name (

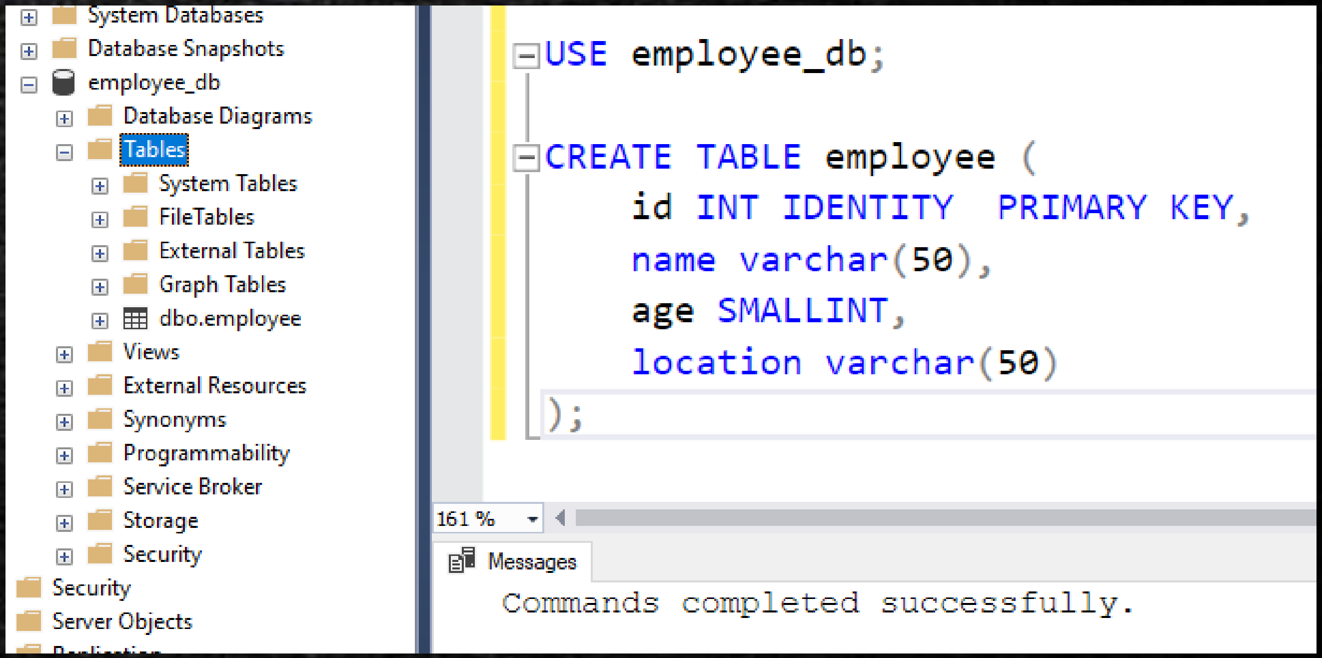
column_definition1,

column_definition2,

........,

table_constraints

);



SQL query to create new table

ALTER a table in database

ALTER TABLE is used to add, delete, or edit columns in an existing table.

```
ALTER TABLE table_name ADD column_name datatype;

ALTER TABLE table_name DROP COLUMN column_name;

ALTER TABLE table_name ALTER COLUMN column_name datatype;
```

ALTER employee table example

```sql
--CREATE TABLE employee (
--      id INT IDENTITY  PRIMARY KEY,
--      name varchar(50),
--      age SMALLINT,
--      location varchar(50)
--);

ALTER TABLE employee
ADD DOB date;
```

Messages

Commands completed successfully.

DESCRIBE (View) Table Schema

SQL Server has built-in system stored procedure sp_help which we can Execute to view table schema

```sql
--ALTER TABLE employee
--ADD DOB date;

EXEC sp_help employee;
```

161 %

Results | Messages

	Name	Owner	Type	Created_datetime
1	employee	dbo	user table	2021-12-08 07:09:16.723

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLe
1	id	int	no	4	10	0	no	(n/a)	(n/a)
2	name	varchar	no	50			yes	no	yes
3	age	smallint	no	2	5	0	yes	(n/a)	(n/a)
4	location	varchar	no	50			yes	no	yes
5	DOB	date	no	3	10	0	yes	(n/a)	(n/a)

	Identity	Seed	Increment	Not For Replication
1	id	1	1	0

	RowGuidCol
1	No rowguidcol column defined.

	Data_located_on_filegroup
1	PRIMARY

	index_name	index_description	index_keys
1	PK__employee__3213E83F9F9C90AC	clustered, unique, primary key located on PRIMARY	id

INSERT Data into table

We can

- Add data in a single row
- Add data in multiple rows

```
INSERT INTO [table_name]
(col_name1, col_name2, ... )
VALUES
(value1, value2, ... );
```

```
INSERT INTO [table_name]
(col_name1, col_name2, ... )
VALUES
(value1, value2, ... )
(value1, value2, ... )
(value1, value2, ... )
(value1, value2, ... );
```

INSERT Data into table

```sql
INSERT INTO employee (name, age, location,dob)
VALUES ('Tom', 2, 'USA','2018-10-20'),
('Jerry', 1, 'USA','2018-10-20'),
('Mickey', 3, 'USA','2018-10-20');
```

lessages

```
(3 rows affected)

Completion time: 2021-12-08T08:07:32.7467824+05:30
```

Column names are optional.
Just need to have
INSERT INTO employee
VALUES …

Fetch data in the table using SELECT statement

To fetch all the columns

SELECT * FROM table_name;

GO is the Block Delimiter.
Just like ; is the Query Delimiter

```
INSERT INTO employee (name, age, location,dob)
VALUES ('Tom', 2, 'USA','2018-10-20'),
('Jerry', 1, 'USA','2018-10-20'),
('Mickey', 3, 'USA','2018-10-20');

GO

select * from employee;
```

161 %

Results | Messages

	id	name	age	location	DOB
1	1	donald	2	USA	NULL
2	2	Jerry	1	USA	NULL
3	3	Mickey	3	USA	NULL
4	4	Tom	2	USA	2018-10-20
5	5	Jerry	1	USA	2018-10-20
6	6	Mickey	3	USA	2018-10-20
7	7	Tom	2	USA	2018-10-20
8	8	Jerry	1	USA	2018-10-20
9	9	Mickey	3	USA	2018-10-20

Query executed successfully. ABHILEN

Update rows of data using UPDATE statement

```
UPDATE table_name
SET column1 = new_value1,
      column2 = new_value2, ...
[WHERE Clause] ;
```

Update a row of data using UPDATE statement

```sql
UPDATE employee
SET name = 'donald'
WHERE name='Tom';
```

161 %

Results | Messages

```
(1 row affected)
```

```sql
select * from employee;
```

161 %

Results | Messages

	id	name	age	location	DOB
1	1	donald	2	USA	NULL
2	2	Jerry	1	USA	NULL
3	3	Mickey	3	USA	NULL

DELETE a row of data using DELETE statement

DELETE FROM table_name
[WHERE Clause] ;

```
DELETE FROM employee WHERE name='Tom'
GO
select * from employee;
```

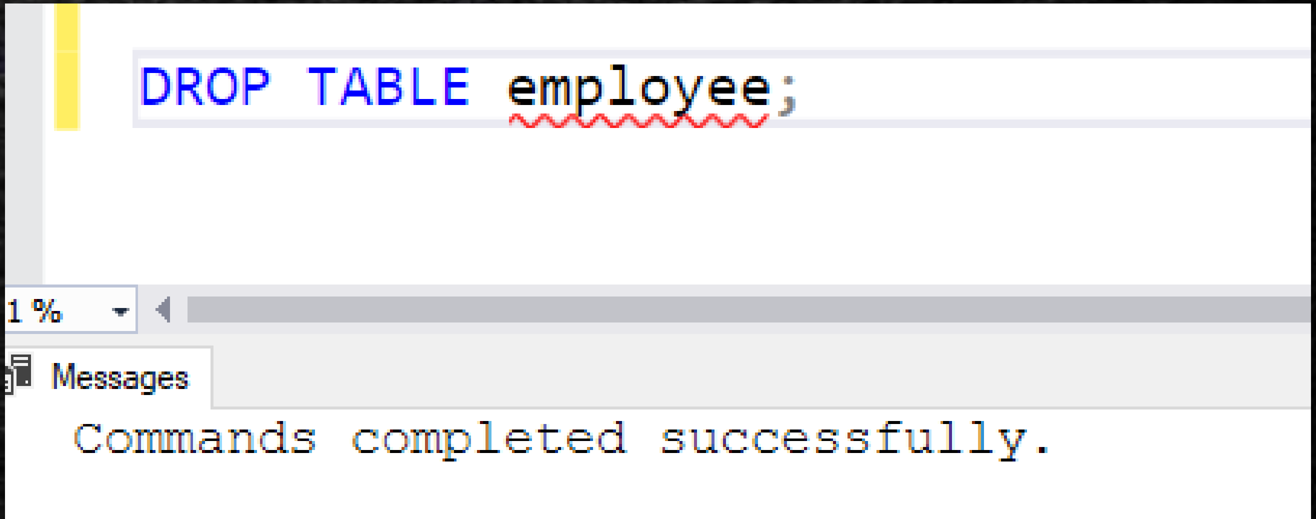
161 %

Results | Messages

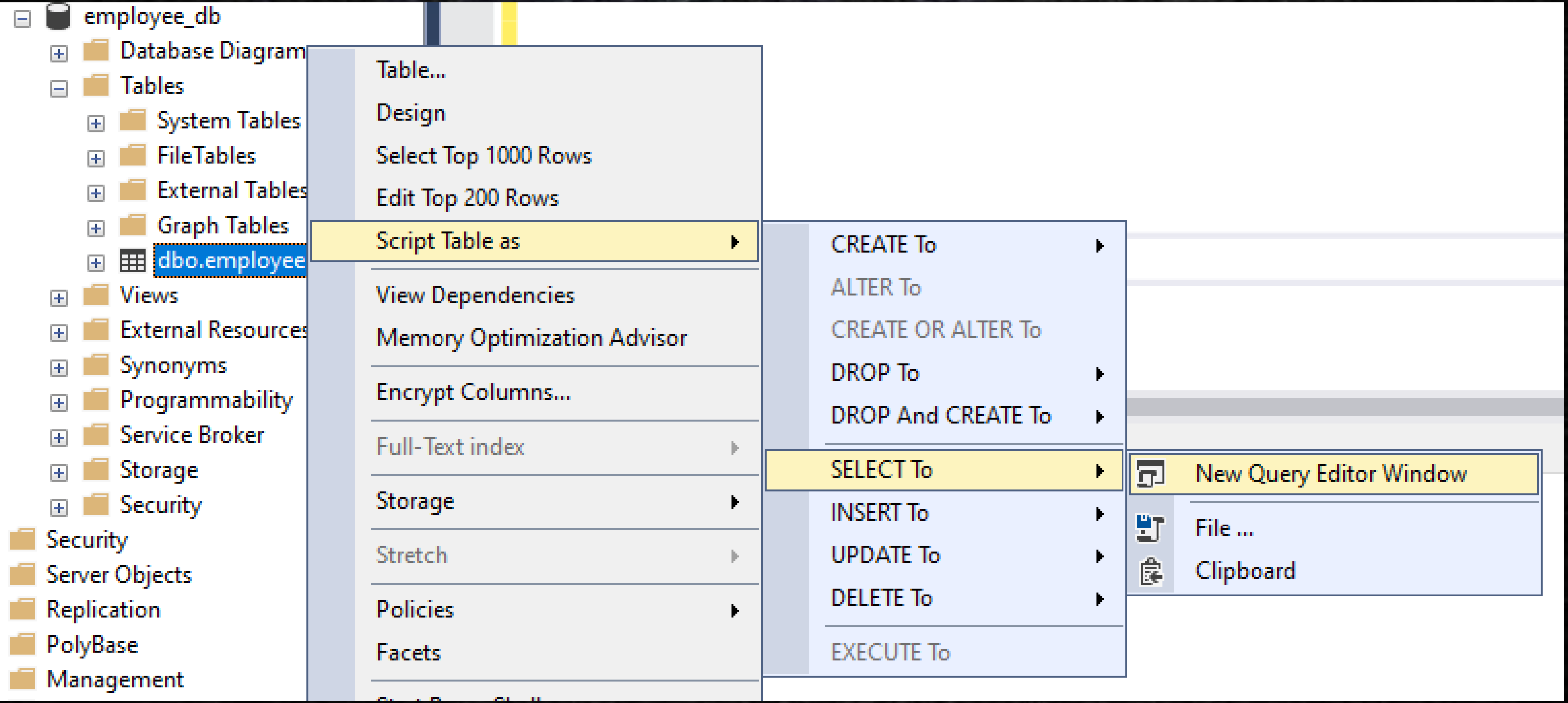
	id	name	age	location	DOB
1	1	donald	2	USA	NULL
2	2	Jerry	1	USA	NULL
3	3	Mickey	3	USA	NULL

DROP a table

```
USE db_name;
DROP TABLE table_name ;
```



CREATE, DROP , SELECT, UPDATE, DELETE using SQL Server



Import a Sample Database

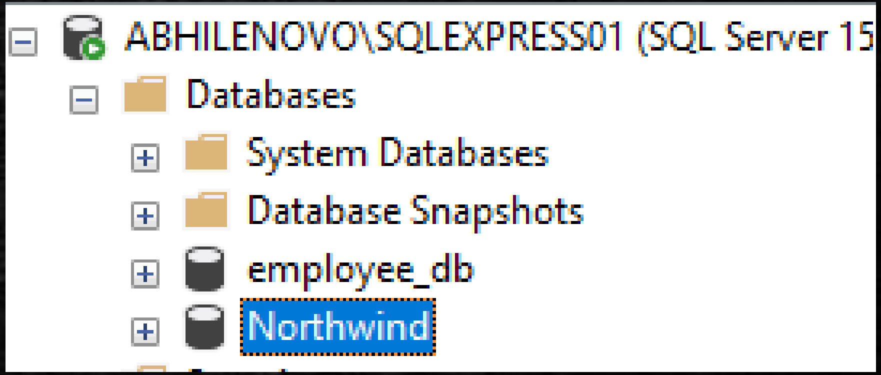
Import the Northwind database from Microsoft

Northwind and pubs sample databases for Microsoft SQL Server

The Northwind and Pubs databases are available for free by Microsoft and can be downloaded and used in any SQL Server

https://github.com/microsoft/sql-server-samples/tree/master/samples/databases/northwind-pubs

View the Raw SQL of northwind, copy it to query window and run it.



SQL Basic Aggregate Functions

Basic Aggregate Operations: MIN, MAX, SUM, AVG, COUNT

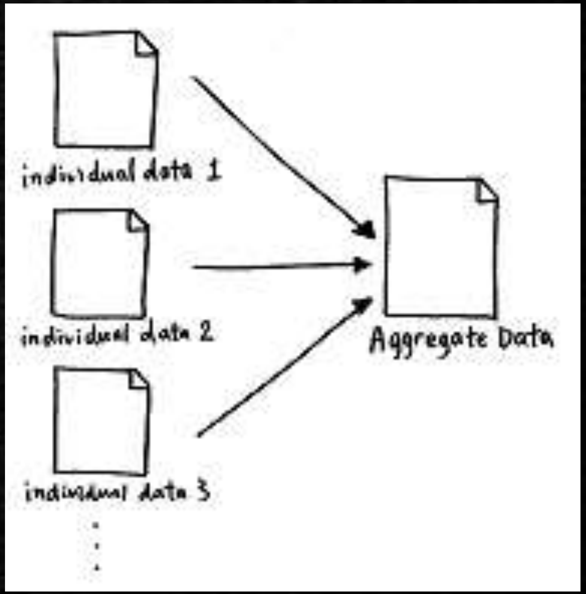
Microsoft® SQL Server®

What are Aggregate Functions in SQL?

An aggregate function allows you to perform a calculation on a set of values to return a single scalar value.

The most commonly used SQL aggregate functions:

MIN – gets the minimum value in a set of values.
MAX – gets the maximum value in a set of values.
SUM – calculates the sum of values.
AVG – calculates the average of a set of values.
COUNT – counts rows in a specified table or view.



SQL MIN Aggregate Function

MIN is used to get the minimum or smallest value of a specified column or expression.

MIN ignores NULL values from the table.

The Syntax is

```
SELECT MIN column(s)
FROM table_name(s)
[WHERE conditions];
```

SQL MIN Aggregate Function Examples

```
SELECT
    MIN(unitprice)
FROM
 Northwind.dbo.products;
```

Results | Messages

	(No column name)
1	2.50

```
SELECT
    MIN(unitprice) AS 'min unit price'
FROM
    products;
```

Results | Messages

	min unit price
1	2.50

SQL MIN Aggregate Function Examples

```
-- Using a subquery that uses the MIN() function

SELECT
    productid, productname, unitprice
FROM
    products
WHERE
    unitprice = (SELECT MIN(unitprice) FROM products);

--Will be equal to

SELECT
    productid, productname, unitprice
FROM
    products
WHERE
    unitprice = 2.50;
```

Results Messages

	productid	productname	unitprice
1	33	Geitost	2.50

SQL MAX Aggregate Function

MAX is used to get the maximum or largest value of a specified column or expression.

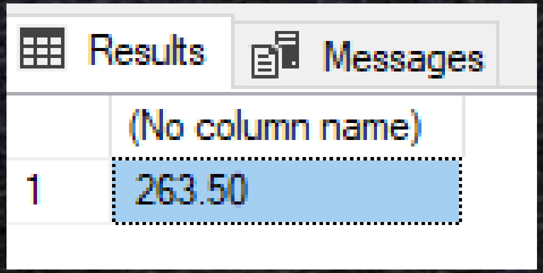
MIN ignores NULL values from the table.

The Syntax is

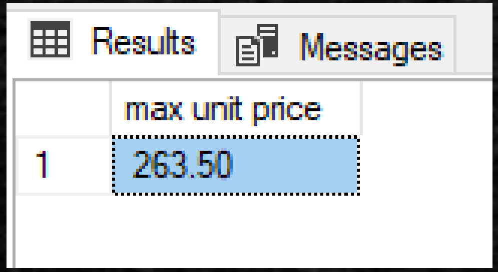
```
SELECT MAX column(s)
FROM table_name(s)
[WHERE conditions];
```

SQL MAX Aggregate Function Examples

```sql
SELECT
     MAX(unitprice)
FROM
     products;
```




```sql
SELECT
     MAX(unitprice) AS 'max unit price'
FROM
     products;
```



SQL MAX Aggregate Function Examples

```
-- Using a subquery that uses the MAX() function

SELECT
    productid, productname, unitprice
FROM
    products
WHERE
    unitprice = (SELECT MAX(unitprice) FROM products);

--Will be equal to

SELECT
    productid, productname, unitprice
FROM
    products
WHERE
    unitprice = 263.50;
```

Results | Messages

	productid	productname	unitprice
1	38	Côte de Blaye	263.50

SQL MAX Aggregate Function Examples

```
-- Using a subquery that uses the MAX() function

SELECT
    productid, productname, unitprice
FROM
    products
WHERE
    unitprice = (SELECT MAX(unitprice) FROM products);

--Will be equal to

SELECT
    productid, productname, unitprice
FROM
    products
WHERE
    unitprice = 263.50;
```

Results | Messages

	productid	productname	unitprice
1	38	Côte de Blaye	263.50

SQL AVG Aggregate Function

AVG is used to get the average value of a specified column or expression.

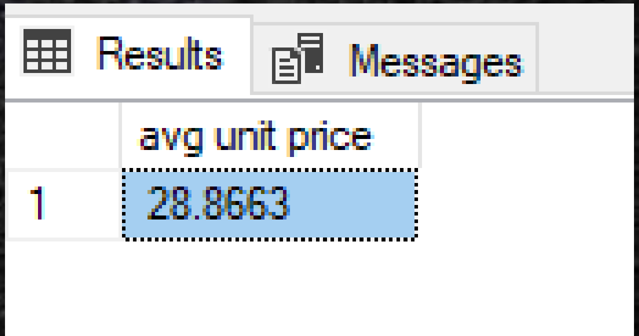
AVG ignores NULL values from the table.

The Syntax is

```
SELECT AVG column(s)
FROM table_name(s)
[WHERE conditions];
```

SQL AVG Aggregate Function Examples

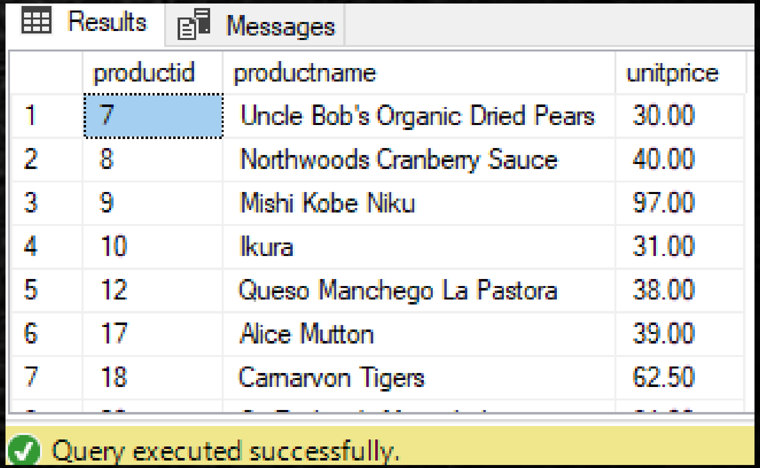
```
SELECT
    AVG(unitprice)
FROM
    products;
```




```
SELECT
    AVG(unitprice) AS 'avg unit price'
FROM
    products;
```

SQL AVG Aggregate Function Examples

```
-- Using a subquery that uses the AVG() function

SELECT
    productid, productname, unitprice
FROM
    products
WHERE
    unitprice > (SELECT AVG(unitprice) FROM products);

--Will be equal to

SELECT
    productid, productname, unitprice
FROM
    products
WHERE
    unitprice > 28.8663;
```


Results Messages

	productid	productname	unitprice
1	7	Uncle Bob's Organic Dried Pears	30.00
2	8	Northwoods Cranberry Sauce	40.00
3	9	Mishi Kobe Niku	97.00
4	10	Ikura	31.00
5	12	Queso Manchego La Pastora	38.00
6	17	Alice Mutton	39.00
7	18	Carnarvon Tigers	62.50

Query executed successfully.

SQL SUM Aggregate Function

SUM is used to get the total value of a specified column or expression.

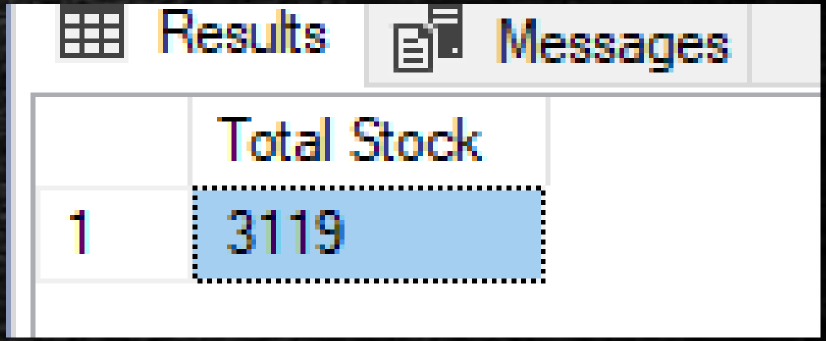
SUM ignores NULL values from the table.

The Syntax is

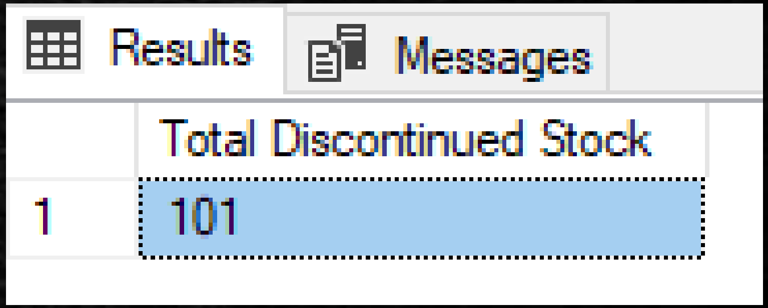
```
SELECT SUM column
FROM table_name
[WHERE conditions];
```

SQL SUM Aggregate Function Examples

```
SELECT
      SUM(UnitsInStock) AS 'Total Stock'
FROM
      products
```




```
SELECT
      SUM(UnitsInStock) AS 'Total Discontinued Stock'
FROM
      products
WHERE
      Discontinued = 1
```



SQL COUNT Aggregate Function

COUNT is used for calculating the total number of rows present in the table.

The Syntax is

```
SELECT COUNT column
FROM table_name
[WHERE conditions];
```

SQL COUNT Aggregate Function Examples

```
SELECT
      COUNT(ProductID) AS 'Products Count'
FROM
      products
```

Results | Messages

	Products Count
1	77

```
SELECT
      COUNT(ProductID) AS 'No of Discontinued Products'
FROM
      products
WHERE
      Discontinued = 1
```

Results | Messages

	No of Discontinued Products
1	8

SQL Server - Basic Clauses

Basic Clauses : DISTINCT, GROUP BY, WHERE, ORDER BY, HAVING, SELECT, GROUPING SETS

What are clauses in SQL?

A clause is just a logical part of an SQL statement

The most commonly used SQL Clauses are:

- DISTINCT
- GROUP BY
- WHERE
- ORDER BY
- HAVING
- SELECT
- GROUPING SETS

DISTINCT Clause

- The result set of a SELECT statement may contain duplicate rows.
- To eliminate the duplicates, use the DISTINCT operator
- We can use the DISTINCT operator in the SELECT statement only.

The syntax is:

```
SELECT DISTINCT column(s)
FROM table_name;
```

DISTINCT Clause Examples

```
SELECT City FROM Northwind.dbo.Customers

SELECT DISTINCT City FROM Customers

SELECT DISTINCT City, Region FROM Customers

SELECT DISTINCT City, Region FROM Customers
WHERE Country='UK'
```

Results | Messages

	City
85	Torino
86	Toulouse
87	Tsawassen
88	Vancouver
89	Versailles
90	Walla Walla
91	Warszawa

Query executed successfully.

Results | Messages

	City
63	Torino
64	Toulouse
65	Tsawassen
66	Vancouver
67	Versailles
68	Walla Walla
69	Warszawa

Query executed successfully.

Results | Messages

	City	Region
63	Torino	NULL
64	Toulouse	NULL
65	Tsawassen	BC
66	Vancouver	BC
67	Versailles	NULL
68	Walla Walla	WA
69	Warszawa	NULL

Query executed successfully.

Results | Messages

	City	Region
1	Cowes	Isle of Wight
2	London	NULL

GROUP BY Clause

- GROUP BY statement groups rows that have the same values into temporary summary rows
- It is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG())

The syntax is:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

GROUP BY Clause Examples

```
SELECT COUNT(CustomerID) AS 'No of Customers', Country
FROM Customers
GROUP BY Country;
```

Results Messages

	No of Cust	Country
1	3	Argentina
2	2	Austria
3	2	Belgium
4	9	Brazil
5	3	Canada
6	2	Denmark
7	2	Finland

```
SELECT COUNT(CustomerID) AS 'No of Customers', Country
FROM Customers
GROUP BY Country;
ORDER BY COUNT(CustomerID)
```

Results Messages

	No of Customers	Country
1	1	Norway
2	1	Poland
3	1	Ireland
4	2	Portugal
5	2	Sweden
6	2	Switzerland

WHERE Clause

- The WHERE clause in SQL Server is used to filter records from the table.

- Often used with SELECT, the WHERE clause can also work with the UPDATE and DELETE query.

The syntax is:

```
SELECT column_name(s)
FROM table_name
WHERE condition;
```

WHERE Clause Operators

- The WHERE clause also supports these operators to filter the records:

Operator Name	Operator Symbol
Equal	=
Less Than	<
Greater Than	>
Less Than or Equal	<=
Greater Than or Equal	>=
Not Equal	<>
Search for a specific pattern	LIKE
Find records within given range	BETWEEN
Used to specify multiple values	IN

WHERE Clause Examples

```
--Using = operator
-- For string compare use ''
SELECT CompanyName, city
  FROM Suppliers
  WHERE Country = 'USA'
  ORDER BY CompanyName;

--Using BETWEEN operator
SELECT * FROM Employees
WHERE EmployeeID BETWEEN 1 AND 5
```

Results | Messages

	CompanyName	city
1	Bigfoot Breweries	Bend
2	Grandma Kelly's Homestead	Ann Arbor
3	New England Seafood Cannery	Boston
4	New Orleans Cajun Delights	New Orleans

Results | Messages

	EmployeeID	LastName	FirstName	Title	Title
1	1	Davolio	Nancy	Sales Representative	Ms.
2	2	Fuller	Andrew	Vice President, Sales	Dr.
3	3	Leverling	Janet	Sales Representative	Ms.
4	4	Peacock	Margaret	Sales Representative	Mrs
5	5	Buchanan	Steven	Sales Manager	Mr.

WHERE Clause Examples

```
--Using IN operator
SELECT * FROM Employees
WHERE EmployeeID IN (1,2,3)
```

Results | Messages

	EmployeeID	LastName	FirstName	Title
1	1	Davolio	Nancy	Sales Representative
2	2	Fuller	Andrew	Vice President, Sales
3	3	Leverling	Janet	Sales Representative

```
--Using LIKE operator
SELECT * FROM Employees
WHERE FirstName Like 'Robert'
```

Results | Messages

	EmployeeID	LastName	FirstName	Title
1	7	King	Robert	Sales Representat

ORDER BY Clause

- Used to arrange the table's data in ascending or descending order based on the given column or list of columns.
-
- Often used with SELECT

The syntax is:

```
SELECT column_name(s)
FROM table_name
WHERE conditions
ORDER BY column_name [ASC | DESC];
```

ORDER BY Clause Examples

```
SELECT FirstName, BirthDate FROM Employees
ORDER BY BirthDate DESC
```

Results | Messages

	FirstName	BirthDate
1	Anne	1966-01-27 00:00:00.000
2	Janet	1963-08-30 00:00:00.000
3	Michael	1963-07-02 00:00:00.000
4	Robert	1960-05-29 00:00:00.000
5	Laura	1958-01-09 00:00:00.000
6	Steven	1955-03-04 00:00:00.000
7	Andrew	1952-02-19 00:00:00.000

```
--First sort by BD, then by First name
SELECT FirstName, BirthDate FROM Employees
ORDER BY BirthDate DESC,
FirstName ASC;
```

Results | Messages

	FirstName	BirthDate
1	Anne	1966-01-27 00:00:00.000
2	Janet	1963-08-30 00:00:00.000
3	Michael	1963-07-02 00:00:00.000
4	Robert	1960-05-29 00:00:00.000

HAVING Clause

- The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions. Eg: we cannot use `WHERE AVG(UnitPrice)>20`

The syntax for HAVING Clause is:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

HAVING Clause Examples

```
SELECT ProductName,UnitPrice FROM Products
GROUP BY ProductName, UnitPrice
HAVING AVG(UnitPrice)>20
```

Results | Messages

	ProductName	UnitPrice
1	Gustaf's Knäckebröd	21.00
2	Queso Cabrales	21.00
3	Louisiana Fiery Hot Pepper Sauce	21.05
4	Chef Anton's Gumbo Mix	21.35
5	Flotemysost	21.50
6	Chef Anton's Cajun Seasoning	22.00
7	Tofu	23.25

SELECT Clause

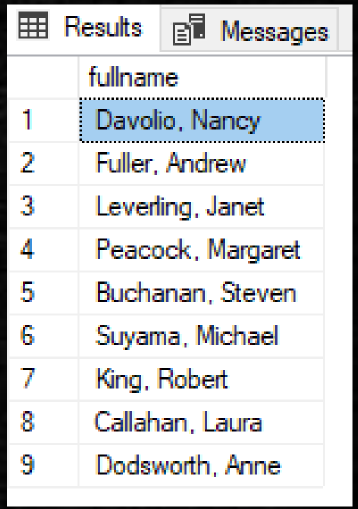
When working with a database, querying data from a table is one of the most common tasks that we have to deal with on a regular basis.

To query data from one or more tables, you use the SELECT statement. The basic syntax is:

```
SELECT column_name(s)
FROM table_name
WHERE condition
```

SELECT Clause Examples

```sql
SELECT * FROM Products

SELECT ProductName,UnitPrice FROM Products

--a simple expression:
SELECT 1 + 1

--combine string using CONCAT()
SELECT CONCAT(LastName,', ',FirstName) AS fullname
FROM employees
```



Results Messages

	fullname
1	Davolio, Nancy
2	Fuller, Andrew
3	Leverling, Janet
4	Peacock, Margaret
5	Buchanan, Steven
6	Suyama, Michael
7	King, Robert
8	Callahan, Laura
9	Dodsworth, Anne

GROUPING SETS Clause

- GROUPING SET is introduced in SQL Server 2008.
- GROUPING SET is able to generate a result set that can be generated by a UNION ALL of multiple simple GROUP BY clauses.
- It's very handy as the query handles the filter and report without having to code

```
SELECT column_name(s)
FROM table_name
Group BY
   GROUPING SETS
   ( (set1), (set2), ..)
```

GROUPING SETS Clause Example

```
use employee_db
go
CREATE TABLE EmployeeMaster
(
        Id INT IDENTITY PRIMARY KEY,
        EmployeeCode varchar(10),
        EmployeeName varchar(25),
        DepartmentCode varchar(10),
        LocationCode varchar(10),
        salary int
)
```

GROUPING SETS Clause Example

```
TRUNCATE TABLE EmployeeMaster;
GO;

INSERT into EmployeeMaster(EmployeeCode,
EmployeeName, DepartmentCode, LocationCode ,salary)
VALUES
('E0001', 'Hulk', 'IT','TVM', 4000),
('E0002', 'Spiderman', 'IT','TVM',  4000),
('E0003', 'Ironman', 'QA','KLM', 3000),
('E0004', 'Superman', 'QA','KLM', 3000),
('E0005', 'Batman', 'HR','TVM', 5000),
('E0005', 'Raju', 'HR','KTM', 5000),
('E0005', 'Radha', 'HR','KTM', 5000)
```

GROUPING SETS Clause Example

```
select * from employeemaster
```

	Id	EmployeeCode	EmployeeName	DepartmentCode	LocationCode	salary
1	1	E0001	Hulk	IT	TVM	4000
2	2	E0002	Spiderman	IT	TVM	4000
3	3	E0003	Ironman	QA	KLM	3000
4	4	E0004	Superman	QA	KLM	3000
5	5	E0005	Batman	HR	TVM	5000
6	6	E0005	Raju	HR	KTM	5000
7	7	E0005	Radha	HR	KTM	5000

We need to get some summarized data, like total cost by Employee, total cost by Department, total cost by location and total cost for all employees with all locations in a single result set.

GROUPING SETS Clause Example

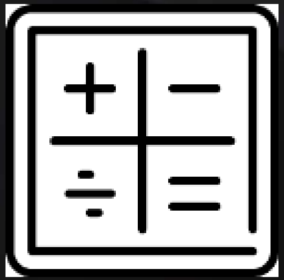
```
SELECT EmployeeCode, EmployeeName, DepartmentCode,
LocationCode, SUM(salary) TotalCost

from EmployeeMaster
Group BY
   GROUPING SETS
   (
      (EmployeeCode, EmployeeName,
       DepartmentCode, LocationCode),

       (DepartmentCode),

       (LocationCode),

       ()
   )
```

Results | Messages

	EmployeeCode	EmployeeName	DepartmentCode	LocationCode	TotalCost
1	E0003	Ironman	QA	KLM	3000
2	E0004	Superman	QA	KLM	3000
3	NULL	NULL	NULL	KLM	6000
4	E0005	Radha	HR	KTM	5000
5	E0005	Raju	HR	KTM	5000
6	NULL	NULL	NULL	KTM	10000
7	E0001	Hulk	IT	TVM	4000
8	E0002	Spiderman	IT	TVM	4000
9	E0005	Batman	HR	TVM	5000
10	NULL	NULL	NULL	TVM	13000
11	NULL	NULL	NULL	NULL	29000
12	NULL	NULL	HR	NULL	15000
13	NULL	NULL	IT	NULL	8000
14	NULL	NULL	QA	NULL	6000

SQL Server - Basic Operators

Basic Operators : Comparison Operators, UNION, INTERSECT, IN, NOT, BETWEEN, IS NULL, NOT NULL, LIKE, EXIST



Operators in SQL

An operator is a word or a character used in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic

The most commonly used SQL Operators are:

- Comparison Operators
- UNION
- INTERSECT
- IN
- NOT
- BETWEEN
- IS NULL and NOT NULL
- LIKE
- EXIST

Comparison Operators in SQL

Used to test for equality and inequality.

Used in the WHERE clause to determine which records to select.

Index	Comparison Operator	Description
1)	=	equal
2)	<>	not equal
3)	!=	not equal .
4)	>	greater than
5)	>=	greater than or equal
6)	<	less than
7)	<=	less than or equal
8) 9)	!> !<	not greater than not less than

10)	IN ( )	matches a value in a list.
11)	NOT	negate a condition.
12)	BETWEEN	specify within a range (inclusive) value.
13)	IS NULL	specifies null value.
14)	IS NOT NULL	specifies non-null value.
15)	LIKE	pattern matching with % and _
16)	EXISTS	if subquery returns at least one row.

Comparison Operators in SQL Examples

```
SELECT * from EmployeeMaster WHERE salary = 3000

SELECT * from EmployeeMaster WHERE salary < 3000
SELECT * from EmployeeMaster WHERE salary <= 3000

SELECT * from EmployeeMaster WHERE salary > 3000
SELECT * from EmployeeMaster WHERE salary >= 3000

SELECT * from EmployeeMaster WHERE salary !> 3000
SELECT * from EmployeeMaster WHERE salary !< 3000
```

Other Operators in SQL

10)	IN ( )	matches a value in a list.
11)	NOT	negate a condition.
12)	BETWEEN	specify within a range (inclusive) value.
13)	IS NULL	specifies null value.
14)	IS NOT NULL	specifies non-null value.
15)	LIKE	pattern matching with % and _
16)	EXISTS	if subquery returns at least one row.

IN, NOT Operators

```
SELECT * from EmployeeMaster WHERE salary IN
(3000,5000)

//which is equal to
SELECT * from EmployeeMaster WHERE  salary = 3000 OR
salary = 5000

//IN For String
SELECT * from EmployeeMaster WHERE employeename IN('Raju',
'Radha')

SELECT * from EmployeeMaster WHERE employeename NOT
IN('Raju', 'Radha')
```

BETWEEN, NULL, NOT Operators

```
SELECT * from EmployeeMaster WHERE salary BETWEEN 3000
AND 5000

SELECT * from EmployeeMaster WHERE salary IS NOT NULL

SELECT * from EmployeeMaster WHERE salary IS NULL
```

LIKE Operators in SQL using Wildcard Search

Wildcard	Explanation
%	to match any string of any length (including zero length).
[ ]	to match on any character in the [ ] brackets (for example, [abc] would match on a, b, or c characters)
[^]	It is used to match on any character not in the [^] brackets (for example, [^abc] would match on any character that is not a, b, or c characters)

```
SELECT * from EmployeeMaster WHERE employeename LIKE ‘super'

SELECT * from EmployeeMaster WHERE employeename LIKE ‘sup%'

SELECT * from EmployeeMaster WHERE employeename LIKE ‘%man'

SELECT * from EmployeeMaster WHERE employeename NOT LIKE '%ra%'
```

LIKE Operators in SQL using Wildcard Search

[ ]	to match on any character in the [ ] brackets (for example, [abc] would match on a, b, or c characters)
[^]	It is used to match on any character not in the [^] brackets (for example, [^abc] would match on any character that is not a, b, or c characters)

```
//will return 8 letter names starting with Su, containing p or j in between and ending in erman
SELECT * from EmployeeMaster WHERE employeename LIKE
'Su[pj]erman%'

//will return 4 letter names starting with ra, containing n or j in between and ending in u
SELECT * from EmployeeMaster WHERE employeename LIKE 'ra[nj]u%'

//will return 4 letter names starting with ra, NOT containing n or j in between and ending in u
SELECT * from EmployeeMaster WHERE employeename LIKE 'ra[^nj]u%'

SELECT * from EmployeeMaster WHERE employeename NOT LIKE 'raj%'
```

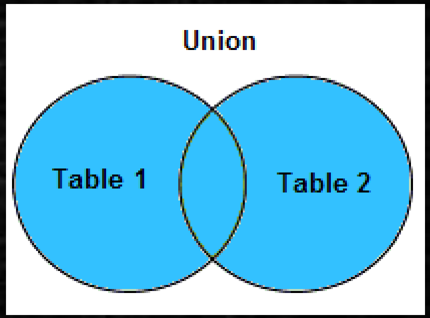
EXISTS Operator in SQL

```
select * from EmployeeMaster WHERE EXISTS
(select * from EmployeeMaster where EmployeeName  LIKE
'superman')
```

UNION Operator

UNION operator is used to combine the result-set of two or more SELECT statements.

```
SELECT expression1, expression2, ...
FROM table1
[WHERE conditions]
UNION
SELECT expression1, expression2, ...
FROM table2
[WHERE conditions];
```



UNION Operator Example

```
use employee_db
go
CREATE TABLE EmployeeMaster2
(
        Id INT IDENTITY PRIMARY KEY,
        EmployeeCode varchar(10),
        EmployeeName varchar(25),
        DepartmentCode varchar(10),
        LocationCode varchar(10),
        salary int
)
```

UNION Operator Example

```
TRUNCATE TABLE EmployeeMaster2;
GO;

INSERT into EmployeeMaster2(EmployeeCode, EmployeeName,
DepartmentCode, LocationCode ,salary)
VALUES
('E0001', 'Arun', 'IT','TVM', 5000),
('E0002', 'Varun', 'IT','TVM',  4000),
('E0003', 'Kiran', 'QA','KLM', 3050),
('E0004', 'Superman', 'QA','KLM', 3000),
('E0005', 'Midhun', 'HR','TVM', 1000),
('E0005', 'Singh', 'HR','KTM', 6000),
('E0005', 'Jyothi', 'HR','KTM', 4000)
```

UNION Operator Example

```
select * from EmployeeMaster
UNION
select * from EmployeeMaster2
```

	Id	EmployeeCode	EmployeeName	DepartmentCode	LocationCode	Salary
1	1	E0001	Hulk	IT	TVM	4000
2	2	E0002	Spiderman	IT	TVM	4000
3	3	E0003	Ironman	QA	KLM	3000
4	4	E0004	Superman	QA	KLM	3000
5	5	E0005	Batman	HR	TVM	5000
6	6	E0006	Raju	HR	KTM	5000
7	7	E0007	Radha	HR	KTM	5000
8	1	E0001	Arun	IT	TVM	5000
9	2	E0002	Varun	IT	TVM	4000
10	3	E0003	Superman	QA	KLM	3000
11	4	E0004	Kiran	QA	KLM	3050
12	5	E0005	Midhun	HR	TVM	1000
13	6	E0005	Singh	HR	KTM	6000
14	7	E0005	Jyothi	HR	KTM	4000

By default UNION will not
Fetch Duplicates. Using ALL will get duplicates

```
select * from EmployeeMaster
UNION ALL
select * from EmployeeMaster2
```

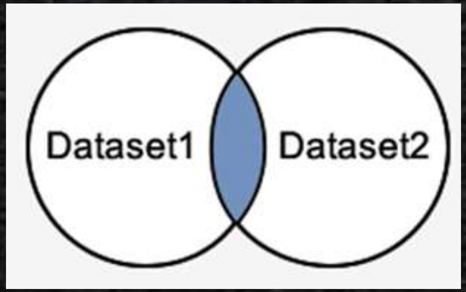
UNION with multiple expressions

```
select employeename, salary FROM EmployeeMaster
WHERE salary > 3000
UNION
select employeename, salary from EmployeeMaster2
```

	employeename	salary
1	Arun	5000
2	Batman	5000
3	Hulk	4000
4	Jyothi	4000
5	Kiran	3050
6	Midhun	1000
7	Radha	5000
8	Raju	5000
9	Singh	6000
10	Spiderman	4000
11	Superman	3000
12	Varun	4000

INTERSECT Operator

INTERSECT operator is used to fetch only the records that are in common between two SELECT statements or data sets.

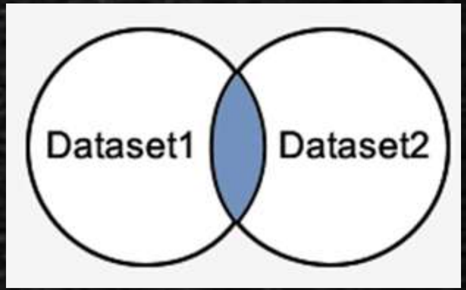


Conditions for INTERSECT operation

- The number of expressions in both SELECT statements must be same.
- Respective columns in each of the SELECT statements must have similar data types.

INTERSECT Operator

```
SELECT expression
FROM table1
[WHERE conditions]
INTERSECT
SELECT expression
FROM table2
[WHERE conditions]
```



INTERSECT Operator Example

```
select * from EmployeeMaster
INTERSECT
select * from EmployeeMaster2
```

Results | Messages

	Id	EmployeeCode	EmployeeName	DepartmentCode	LocationCode	Salary
1	4	E0004	Superman	QA	KLM	3000

INTERSECT with multiple expressions

```sql
select employeename, salary FROM EmployeeMaster
WHERE salary > 2000
INTERSECT
select employeename, salary from EmployeeMaster2
```

Results Messages

	employeename	salary
1	Superman	3000

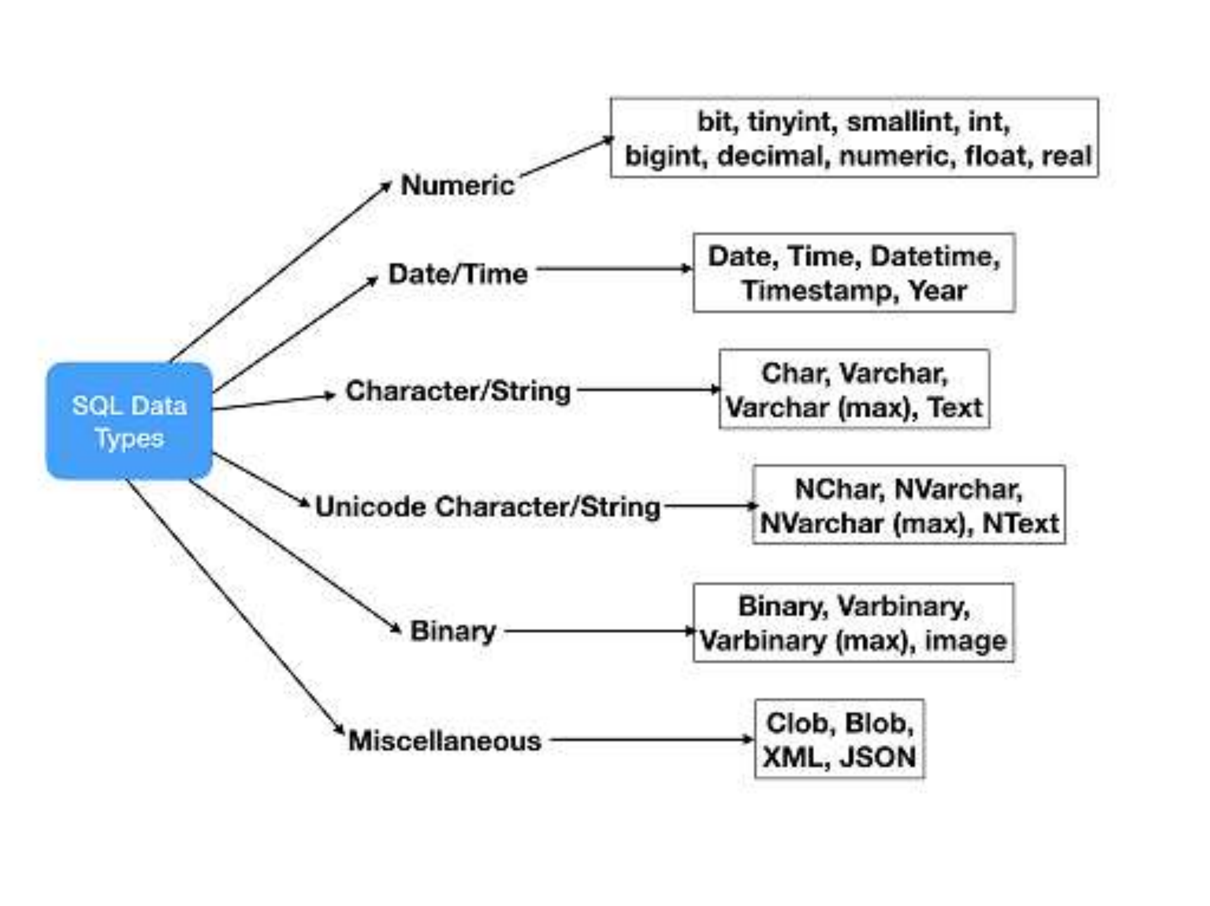
SQL Server – Data Types

Going In-depth with Popular Data Types

SQL Data Types

SQL Server supports a variety of SQL standard data types.
They can be categorized as the following :

- Exact numeric
- Approximate numeric
- Date and time
- Character strings
- Unicode character strings
- Binary strings
- Other data types



Exact numeric data type

will store exact numbers such as integer, decimal, and money.

Data Type	Descriptions
Bit (1 byte)	It is an integer type that allows us to store 0, 1, and NULL values.
Tinyint (1 byte)	It allows us to store whole numbers from 0 to 255.
Smallint (2 bytes)	It allows us to store whole numbers between -32768 to 32767.
Int (4 bytes)	It allows to store whole numbers between -2,147,483,648 and 2,147,483,647

Exact numeric data type

Bigint (8 bytes)	whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807
Decimal(p, s) (5 to 17 bytes)	fixed precision numbers. p indicates the maximum total number of digits that can be stored both to the left and the right of the decimal point. By default, it is 18 but can be in a range of 1 to 38. s indicates the maximum number of digits stored to the right of the decimal point. By default, its value is 0 but can be from 0 to p.
Numeric(p, s) (5 to 17 bytes)	It is similar to the decimal data type
Smallmoney (4 bytes)	It allows storing monetary or currency values.
Money (8 bytes)	It allows to store monetary or currency values.

Approximate numeric data type

will store floating-point and real values. It is mainly used in scientific calculations.

Data Type	Lower range	Upper Range	Storage	Precision
float(n)	1.79E+308	1.79E+308	depends on n. 4 or 8 bytes	7 digit
real	3.40E+38	3.40E+38	4 byte	15 digit

Date and Time data types

will store the temporal values such as date and time, including time offset in a column.

Data Type	Descriptions	Lower Range	Upper Range	Storage
date	To store dates in SQL Server. By default, its format is YYYY-MM-DD value is 1900-01-01.	0001-01-01	9999-12-31	3 bytes
datetime2	specifies date and time with fractional seconds With accuracy of 100 nanoseconds. It provides precision from 0 to 7 digits. By default, its precision is 7, and the format is YYYY-MM-DD hh:mm: ss[.fractional seconds]. (previously it was 'datetime' which is being deprecated)	0001-01-01 00:00:00	9999-12-31 23:59:59.999 9999	6 to 8 bytes

Date and Time data types

datetimeoffset	same as datetime2 with the addition of a time zone offset. timezone offset value -14:00 through +14:00.	0001-01-01 00:00:00	9999-12-31 23:59:59.9999999	10 bytes
smalldatetime	specifies a date along with the time of day and an accuracy of 1 minute. time is calculated on a 24-hour clock, with seconds starting at zero (:00) and no fractional seconds.	1900-01-01 00:00:00	2079-06-06 23:59:59	4 bytes
time	specifies time data only with an accuracy of 100 nanoseconds. It is based on a 24-hour clock without time zone. By default, its format is hh:mm:ss[.nnnnnnn].	00:00:00.0000000	23:59:59.9999999	3 to 5 bytes

Character or string data type

Can be used for character data type only, which can be fixed or variable in length.

Data Type	Descriptions	Lower Range	Upper Range	Storage
char(n)	used to store fixed-length non-Unicode character data.	0 characters	8000 characters	n bytes
varchar(n)	used to store variable-length non-Unicode character data.	0 characters	8000 characters	n bytes + 2 bytes
varchar(max)	stores variable-length data. It is recommended to avoid this data type unless required because of its huge memory storage.	0 characters	2^31 characters	n bytes + 2 bytes
text	variable-length character string. It is also recommended to avoid this data type because it would be deprecated in future releases.	0 characters	2,147,483,647 chars	n bytes + 4 bytes

Character or string data type (Unicode)

define the full range of Unicode character sets encoded in the UTF-16 character set.

Data Type	Descriptions	Lower Range	Upper Range	Storage
nchar	used to store fixed-length Unicode character data.	0 characters	4000 characters	2 times n bytes
nvarchar	used to store variable-length Unicode character data.	0 characters	4000 characters 2 times	n bytes + 2 bytes

Binary data types

allows storing image, audio, and video files of fixed and variable length into a table.

Data Type	Descriptions	Lower Range	Upper Range	Storage
binary	It is used to store fixed-length binary strings.	0 bytes	8000 bytes	n bytes
varbinary	It is used to store variable-length binary string.	0 bytes	8000 bytes	The actual length of data entered + 2 bytes
image	similar to the varbinary data type that can store up to 2 GB. It is recommended to avoid this data type because it would be deprecated in future releases.	0 bytes	2,147,483,647 bytes	

Other data types

Data Type	**Description**
cursor	variables or stored procedure OUTPUT parameter that contains a reference to a cursor
rowversion	automatically generated, unique binary numbers within a database.
hierarchyid	represent a tree position in a tree hierarchy
uniqueidentifier	16-byte GUID
sql_variant	store values of other data types
XML	store XML data in a column, or a variable of XML type
Spatial Geometry type	represent data in a flat coordinate system.
Spatial Geography type	store ellipsoidal (round-earth) data, such as GPS latitude and longitude coordinates.
table	store a result set temporarily for processing at a later time

Popular Data Types : Example : Create Table

```
CREATE DATABASE data_types_eg;
GO
USE data_types_eg;
GO


CREATE TABLE data_types_eg(
    bit_col BIT,
    char_col CHAR(3),
     date_col DATE,
     date_time_col DATETIME2(3),
     date_time_offset_col DATETIMEOFFSET(2),
     dec_col DECIMAL (4, 2),
     num_col NUMERIC (4, 2),
```

Popular Data Types : Example : Create Table

```
    bigint_col bigint,
    int_col INT,
    smallint_col SMALLINT,
    tinyint_col tinyint,
    nchar_col NCHAR(10),
    nvarchar_col NVARCHAR(10),
    time_col TIME(0),
    varchar_col VARCHAR(10)
);
```

Popular Data Types : Example : Insert Data

```
INSERT INTO data_types_eg (

bit_col,
char_col,
date_col,
date_time_col,
date_time_offset_col,
dec_col,
num_col,
```

Popular Data Types : Example : Insert Data

```
bigint_col,
int_col,
smallint_col,
tinyint_col,
nchar_col,
nvarchar_col,
time_col,
varchar_col
)
```

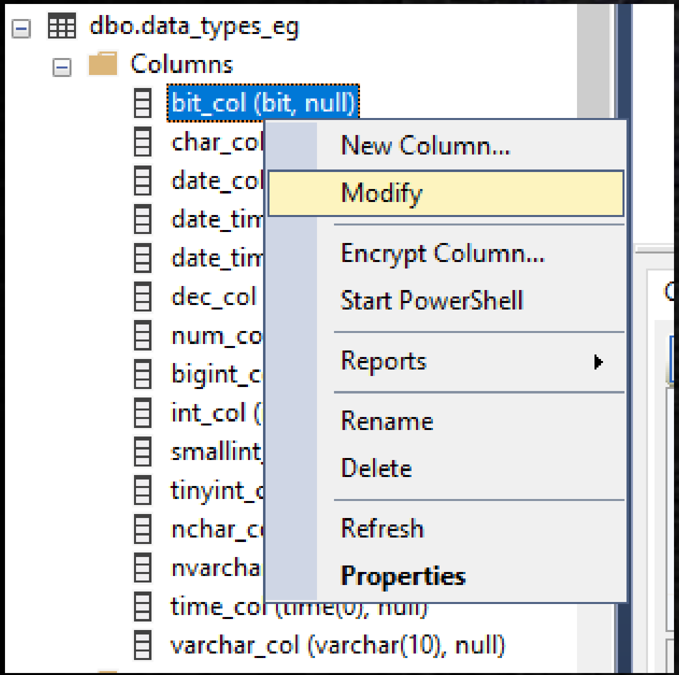
Popular Data Types : Example : Insert Data

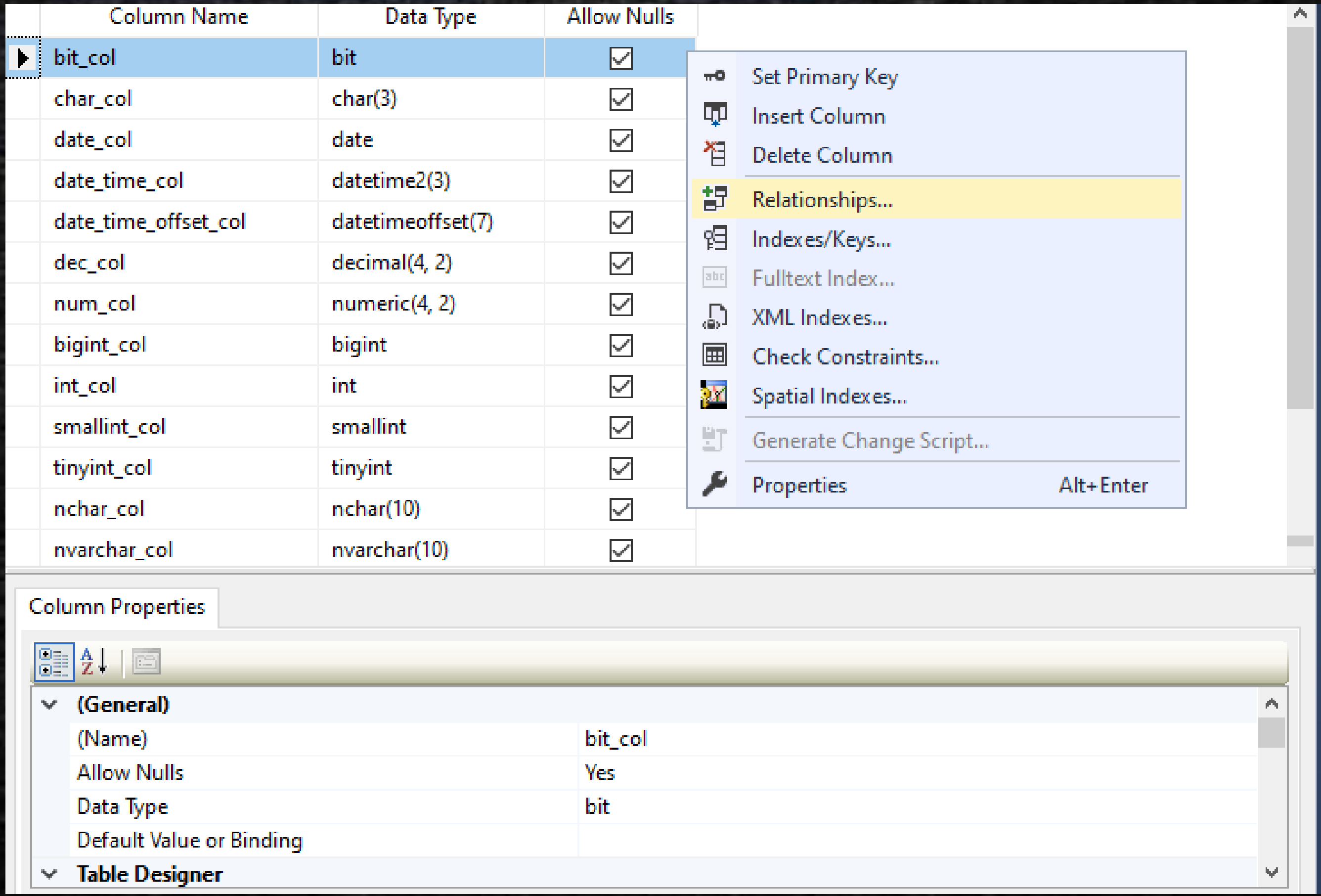
```
VALUES
    (
        1,
        'ABC',
        '2019-01-01',
        '2018-06-23 07:30:20',
        '2020-12-20 17:20:12.56 +05:30',
        10.05,
        20.05,
```

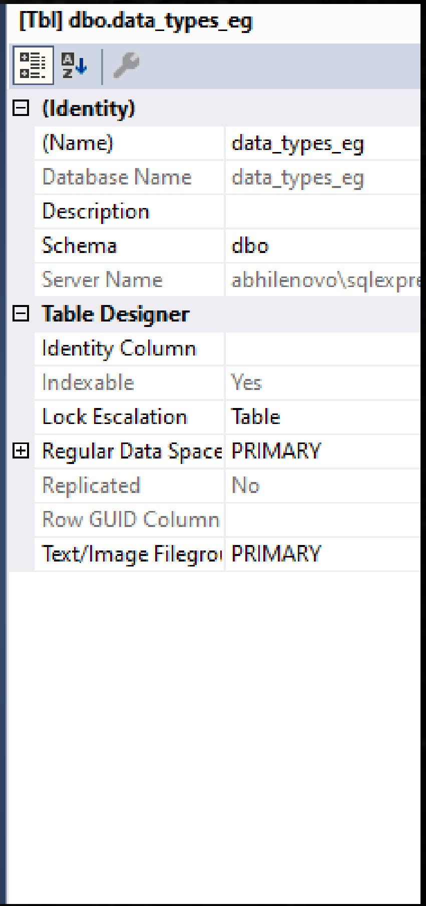
Popular Data Types : Example : Insert Data

```
        9223372036854775807,
        2147483647,
        32767,
        255,
        N'いえ',
        N'こんにちは',
        '09:10:00',
        'John Doe'
    );
```

Modify Columns, Types using SSMS UI







SQL Server – Constraints

Going In-depth with SQL Constraints

Microsoft® SQL Server®

SQL Constraints

Constraints are rules and restrictions applied on a column or a table such that unwanted data can't be inserted into tables.

We can create constraints on single or multiple columns of any table to maintain the data integrity and accuracy in the table.

Popular Constraints are :

- PRIMARY KEY
- FOREIGN KEY
- CHECK Constraint
- UNIQUE Constraint
- DEFAULT Constraint
- NOT NULL Constraint

PRIMARY KEY Constraint

- A PRIMARY KEY constraint declares a column or multiple columns whose values should be unique

- If tried to insert or update a duplicate primary key, SQL engines will issue an error message.

- PRIMARY KEY constraint helps enforce the integrity of data automatically.

PRIMARY KEY Constraint Single Column example

```
CREATE TABLE usage_logs (
   logid INT NOT NULL IDENTITY PRIMARY KEY,
   message char(255) NOT NULL
)
```

	Column Name	Data Type	Allow Nulls
	logid	int	☐
	message	char(255)	☐
			☐

The LogID column is defined as the `PRIMARY KEY` with :

`NOT NULL` : the value in the column cannot be NULL.

`IDENTITY` : the database engine generates a sequence for the column whenever a new row is inserted into the table.

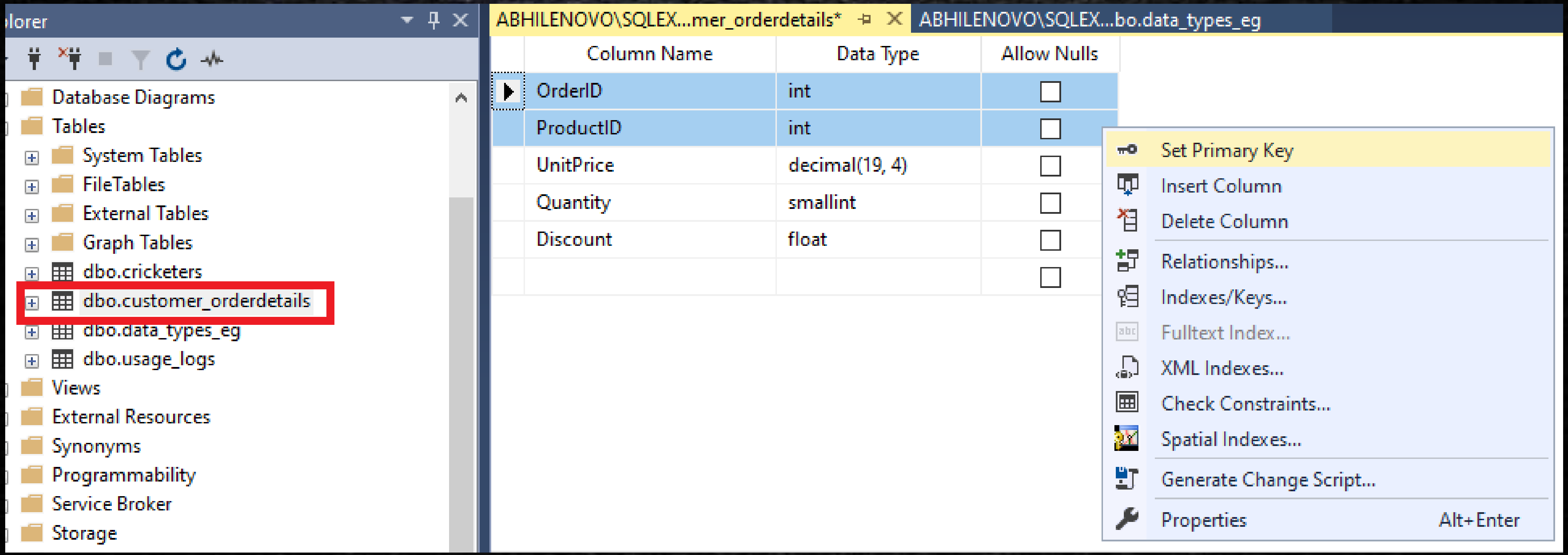
The AUTO_INCREMENT optional attribute can be defined as IDENTITY in SQL server, AUTO_INCREMENT in MySQL, SERIAL in PostgreSQL.

PRIMARY KEY Constraint Multi Column example

```
CREATE TABLE customer_orderdetails (
   OrderID int NOT NULL,
   ProductID int NOT NULL,
   UnitPrice decimal(19,4) NOT NULL,
   Quantity smallint NOT NULL,
   Discount float NOT NULL,
   PRIMARY KEY (OrderID,ProductID),
 )
```

Column Name	Data Type	Allow Nulls
OrderID	int	☐
ProductID	int	☐
UnitPrice	decimal(19, 4)	☐
Quantity	smallint	☐
Discount	float	☐
		☐

PRIMARY KEY Constraint Using Table Design View



Create Primary Key - Using ALTER TABLE

Syntax:

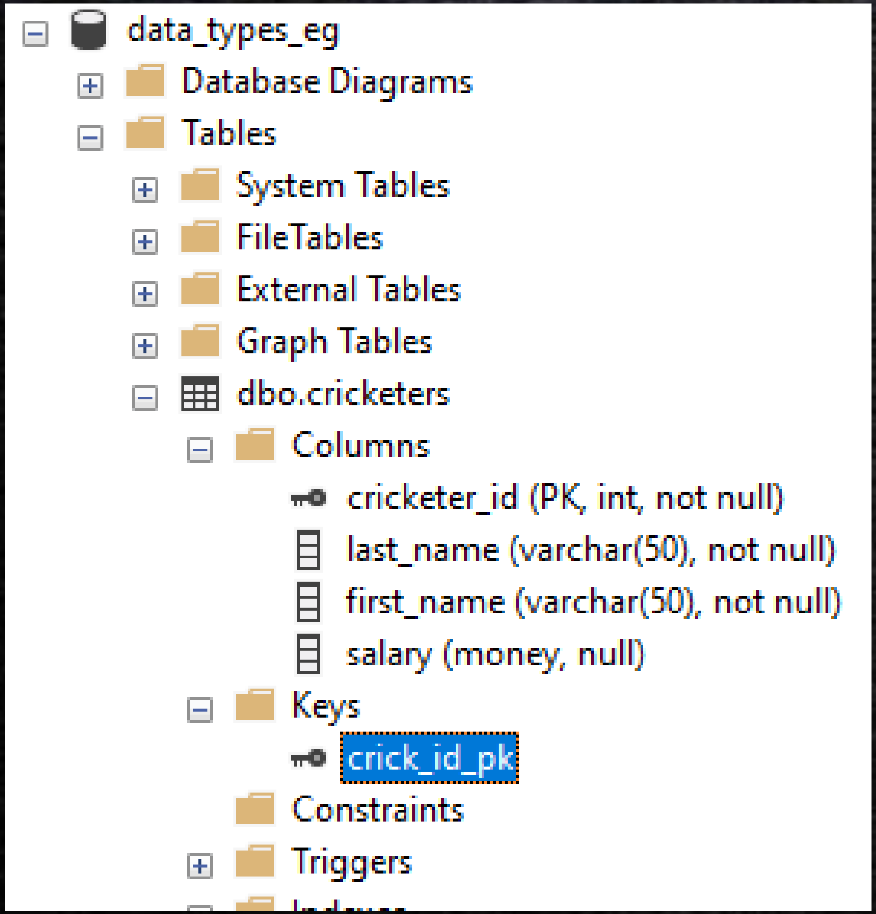
```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
PRIMARY KEY (column1, column2, ... column_n);
```

We can use the ALTER TABLE statement to create a primary key on column(s) that are already defined as NOT NULL

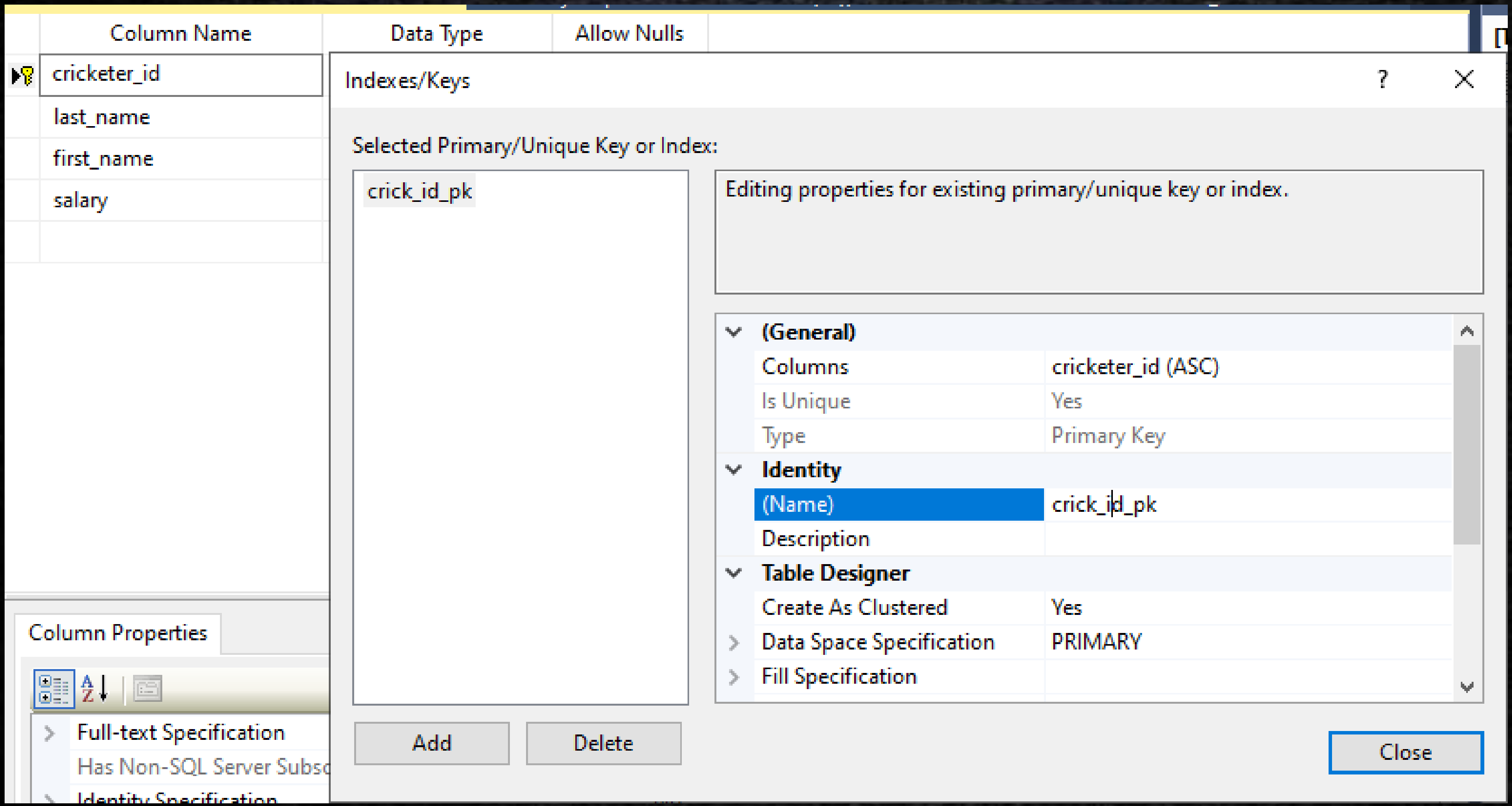
Create Primary Key - Using ALTER TABLE

```
CREATE TABLE cricketers
( cricketer_id INT NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  first_name VARCHAR(50) NOT NULL,
  salary MONEY
);

ALTER TABLE cricketers ADD CONSTRAINT
crick_id_pk PRIMARY KEY (cricketer_id);
```

To View Constraint Name : Method 1



To View Constraint Name : Method 2



To View Constraint Name : Method 3

```
EXEC sp_help cricketers
```

194 %

Results | Messages

3	first_name	varchar	no	50			no	no	no	Latin1_General_CI_AI
4	salary	money	no	8	19	4	yes	(n/a)	(n/a)	NULL

	Identity	Seed	Increment	Not For Replication
1	No identity column defined.	NULL	NULL	NULL

	RowGuidCol
1	No rowguidcol column defined.

	Data_located_on_filegroup
1	PRIMARY

	index_name	index_description	index_keys
1	crick_id_pk	clustered, unique, primary key located on PRIMARY	cricketer_id

	constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_keys
1	PRIMARY KEY (clustered)	crick_id_pk	(n/a)	(n/a)	(n/a)	(n/a)	cricketer_id

Enable Primary Key

```
ALTER INDEX constraint_name ON table_name
REBUILD;
```

```
ALTER INDEX fk_inv_product ON myinventory
REBUILD;
```

Disable Primary Key

```
ALTER INDEX constraint_name ON table_name
DISABLE;

ALTER INDEX fk_inv_product ON myinventory
DISABLE;
```

Drop Primary Key - Using ALTER TABLE

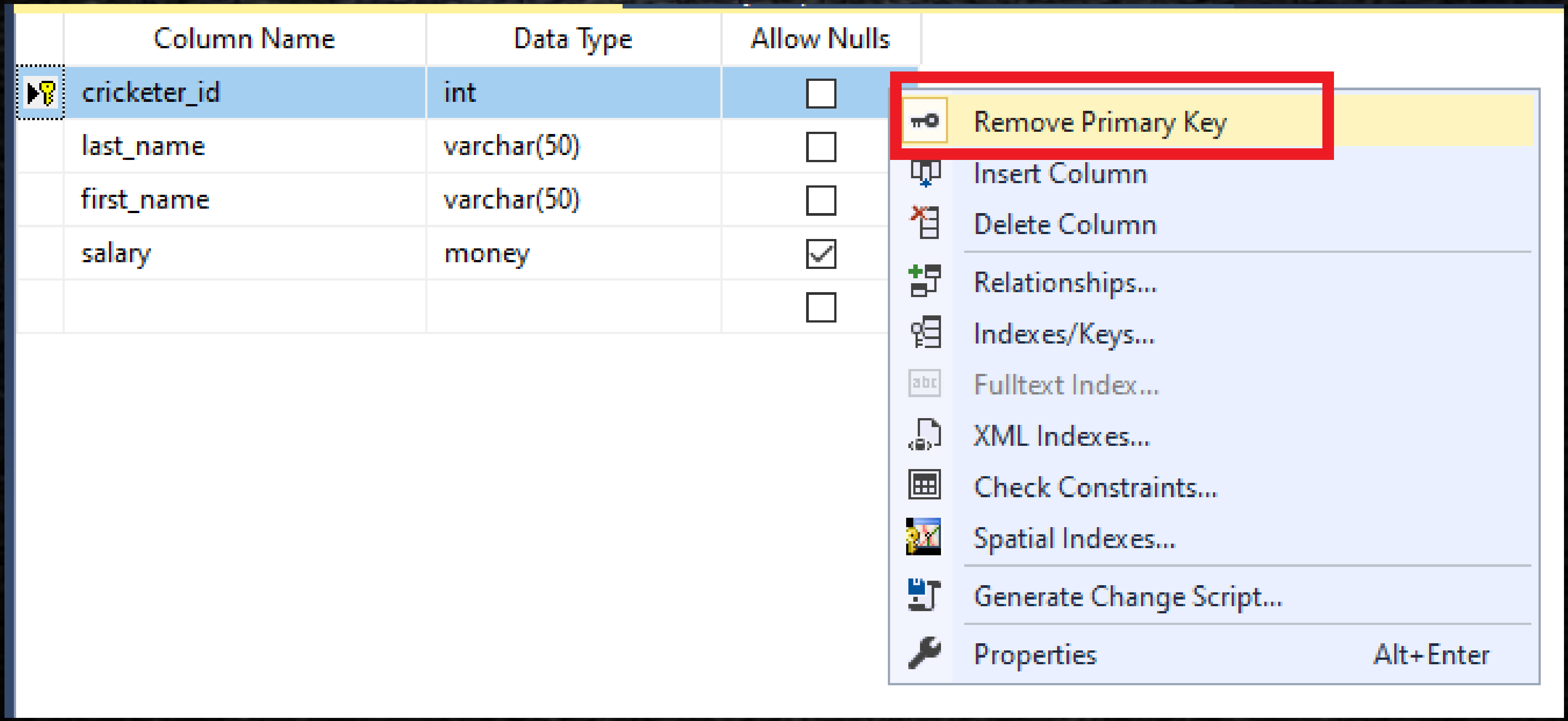
Syntax:

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

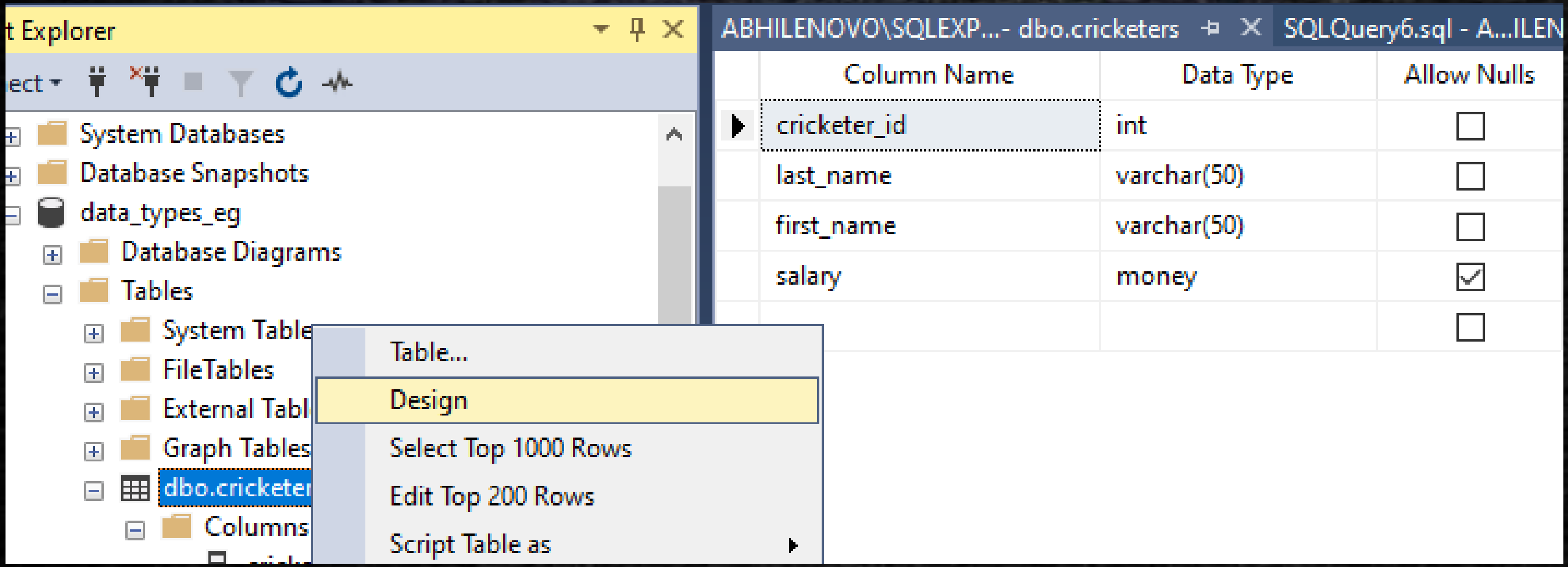
Example:

```
ALTER TABLE cricketers
DROP CONSTRAINT crick_id_pk
```

DROP PRIMARY KEY Using Table Design View



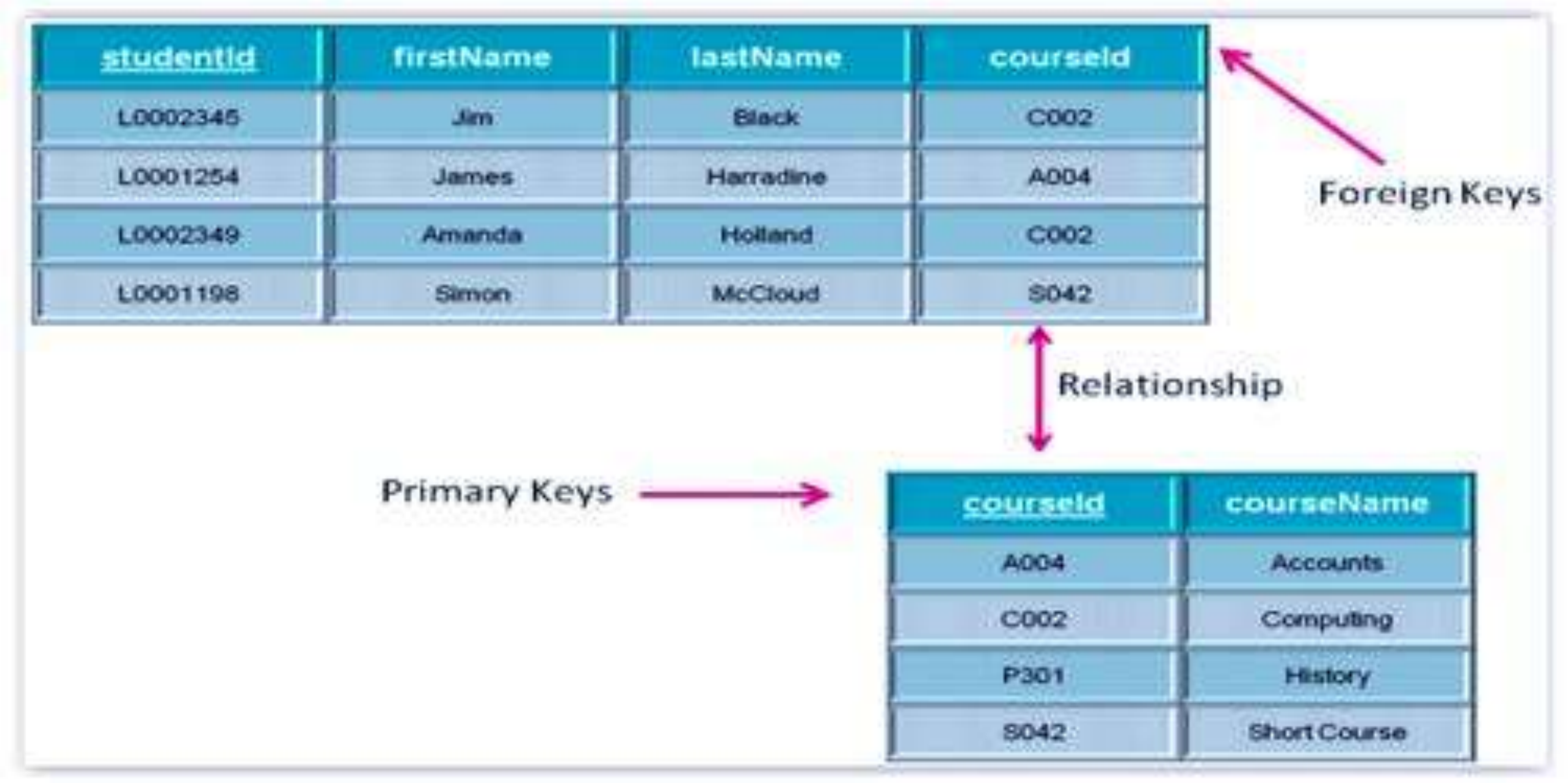
Drop Primary Key - Check



OR:

```
EXEC sp_help cricketers
```

Primary Key vs Foreign Key

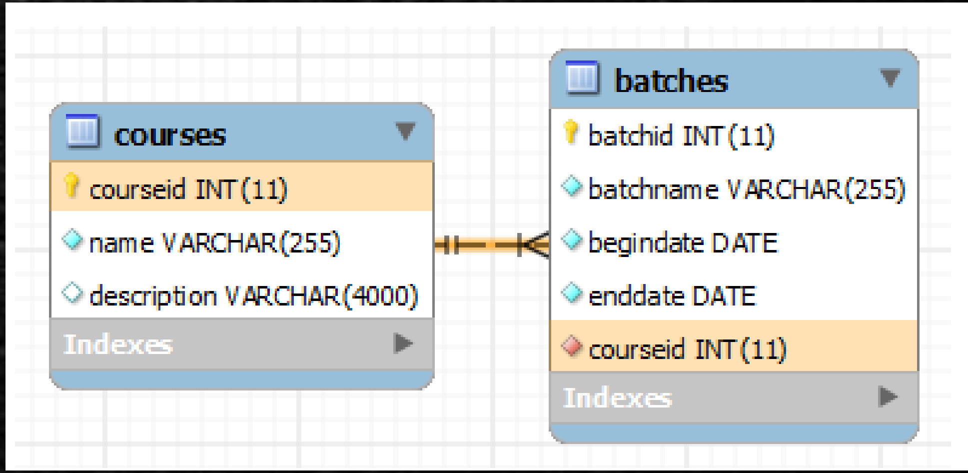


studentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042

courseId	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short Course

Foreign Key - Introduction

- A foreign key is used to enforce a relationship between two tables.
- The table that contains the foreign key is called foreign key table. The referenced table is called parent table foreign key is called child table.
- It specifies that a value in one table must also appear in another table.

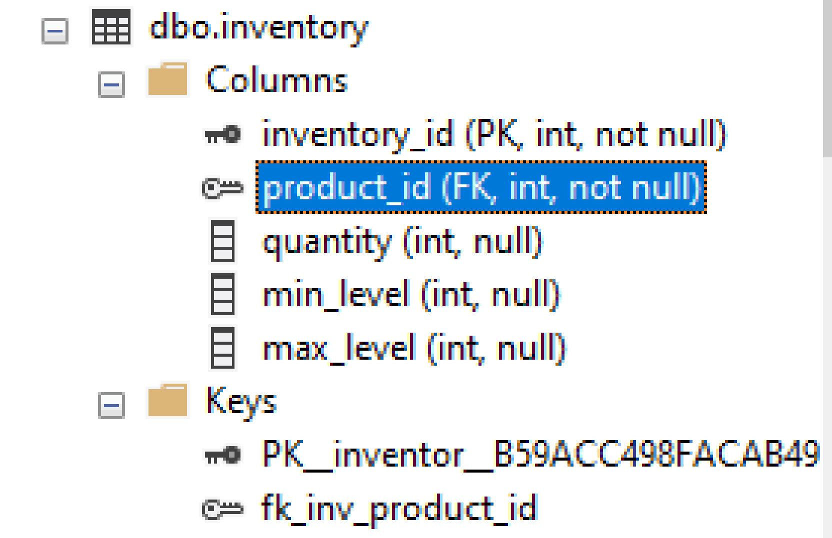


Create Foreign Key - Using CREATE TABLE

```
CREATE TABLE child_table
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...
  CONSTRAINT fk_name
    FOREIGN KEY (child_col1, child_col2, ...)
    REFERENCES parent_table (parent_col1, parent_col2, ...)
);
```

Create Foreign Key - Using CREATE TABLE

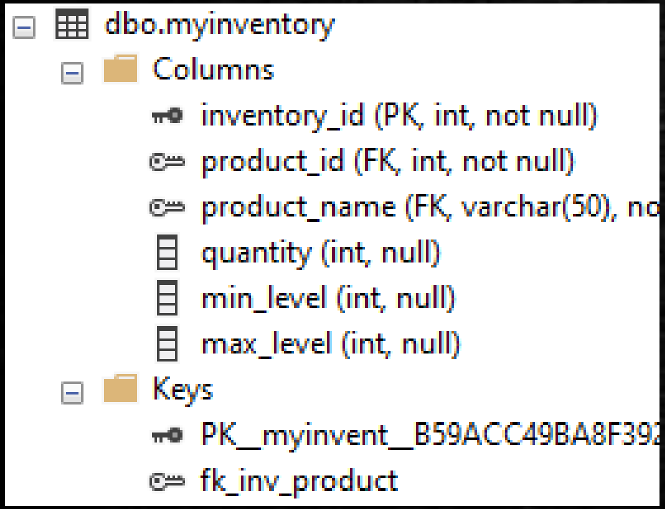
```
CREATE TABLE myproducts
( product_id INT NOT NULL IDENTITY PRIMARY KEY,
  product_name VARCHAR(50) NOT NULL,
  category VARCHAR(25)
);
```




```
CREATE TABLE myinventory
( inventory_id INT PRIMARY KEY,
  product_id INT NOT NULL,
  quantity INT,
  min_level INT,
  max_level INT,
  CONSTRAINT fk_inv_product_id
    FOREIGN KEY (product_id)
    REFERENCES products (product_id)
);
```

Multiple Foreign Key - Using CREATE TABLE

```
CREATE TABLE myproducts
( product_id INT NOT NULL IDENTITY,
  product_name VARCHAR(50) NOT NULL,
  category VARCHAR(25)
  CONSTRAINT myproducts_pk PRIMARY KEY (product_id, product_name) );
```

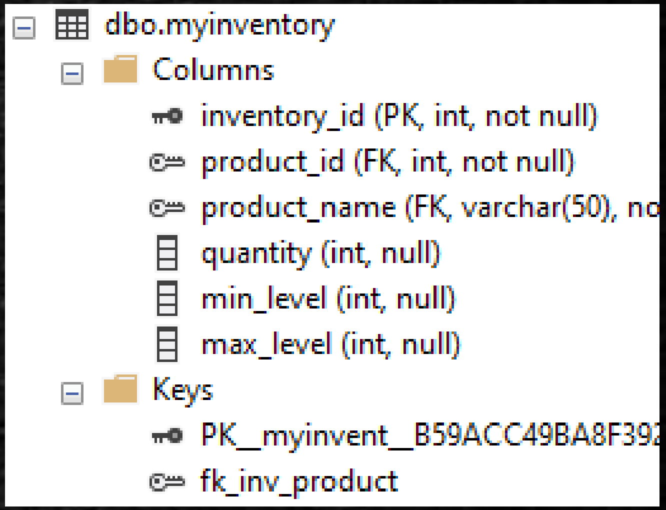



```
CREATE TABLE myinventory
( inventory_id INT PRIMARY KEY,
  product_id INT NOT NULL,
  product_name VARCHAR(50) NOT NULL,
  quantity INT,
  min_level INT,
  max_level INT,
  CONSTRAINT fk_inv_product
    FOREIGN KEY (product_id, product_name)
    REFERENCES products(product_id, product_name) );
```

Disable Foreign Key

```
ALTER TABLE table_name
NOCHECK CONSTRAINT fk_name;
```

```
ALTER TABLE myinventory
NOCHECK CONSTRAINT fk_inv_product;
```



Enable Foreign Key

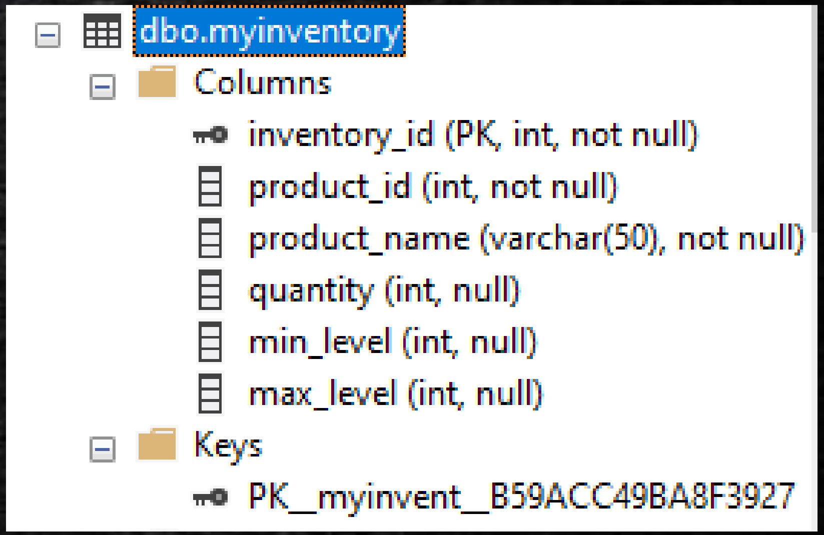
```
ALTER TABLE table_name
CHECK CONSTRAINT fk_name;
```

```
ALTER TABLE myinventory
CHECK CONSTRAINT fk_inv_product;
```

DROP Foreign Key

```
ALTER TABLE table_name
DROP CONSTRAINT fk_name;
```

```
ALTER TABLE myinventory
DROP CONSTRAINT fk_inv_product;
```



NOT NULL Constraint

- By default SQL store NULL values.

- Using a NOT NULL constraint We can restrict NULL value from being inserted into a prescribed column

- This is applicable for INSERT or UPDATE operations

NOT NULL Constraint on Table Creation

```
CREATE TABLE usage_logs (
  logid INT NOT NULL ,
  message char(255)
)
```

Column Name	Data Type	Allow Nulls
logid	int	☐
message	char(255)	☑
		☐

The LogID is defined with `NOT NULL`

So the value in the column cannot be NULL. If Null inserted, will get error

'Column does not allow nulls'

NOT NULL Constraint : Alter Table

```
ALTER TABLE usage_logs
ALTER COLUMN message
varchar(200) NOT NULL;
```

Column Name	Data Type	Allow Nulls
logid	int	☐
message	char(255)	☐
		☐

The LogID and message columns are now defined with `NOT NULL`

So the value in the column cannot be NULL. If Null inserted, will get error

'Column does not allow nulls'

DROP NOT NULL Constraint

```
ALTER TABLE usage_logs
ALTER COLUMN message
varchar(200) NULL;
```

Column Name	Data Type	Allow Nulls
logid	int	☐
message	char(255)	☑
		☐

The message column can now have Null values

UNIQUE Constraint

- UNIQUE constraint ensures that no duplicate values can be inserted into a column
- Unlike PRIMARY KEY, it allows one null value.
- This is applicable for INSERT or UPDATE operations

UNIQUE Constraint on Table Creation

```
CREATE TABLE usage_logs (
   logid INT NOT NULL UNIQUE,
   message char(255)
)
```

Column Name	Data Type	Allow Nulls
logid	int	☐
message	char(255)	☑
		☐

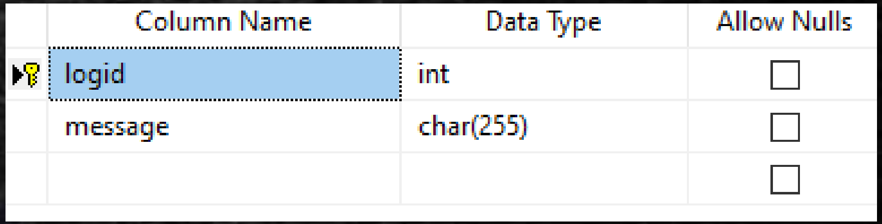
The LogID is defined with `NOT NULL and UNIQUE`

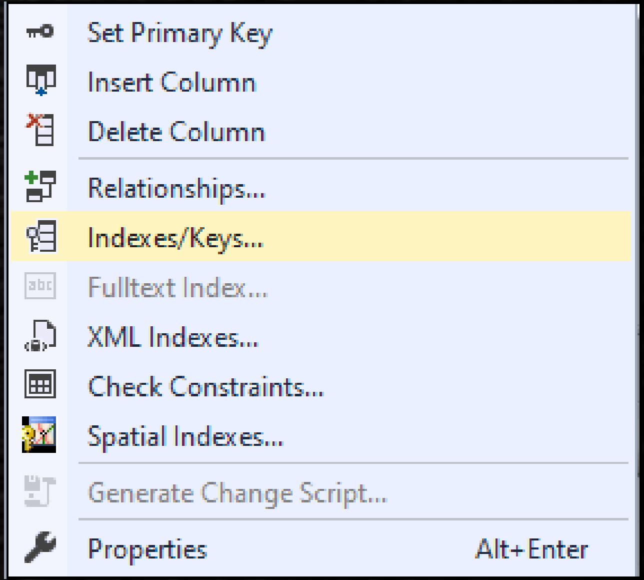
If Null or repeating value inserted, will get error

'Column does not allow nulls, Cannot insert Duplicate Key'

UNIQUE Constraint : Alter Table

```
ALTER TABLE usage_logs
ADD CONSTRAINT uniq_msg
UNIQUE (message);
```





The LogID is defined with `NOT NULL and UNIQUE`

If Null or repeating value inserted, will get error

'Column does not allow nulls, Cannot insert Duplicate Key'

DROP UNIQUE Constraint

```
ALTER TABLE usage_logs
DROP CONSTRAINT uniq_msg;
```

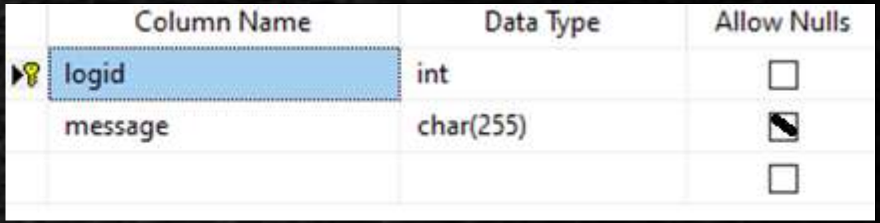
The message column can now have duplicate values

CHECK Constraint

- Used to limit the range of values in a column.
- Assures no corrupted information is entered in a column.
- We can specify more than one check constraint for a specific column.
- This is applicable for INSERT or UPDATE operations

CHECK Constraint on Table Creation

```
CREATE TABLE usage_logs (
   logid INT NOT NULL UNIQUE CHECK (logid > 10) ,
   message char(255)
)
```



The LogID is defined with `NOT NULL, UNIQUE and CHECK`

If less than 10 value inserted, will get error

'Conflicted with CHECK constraint'

CHECK Constraint : Alter Table

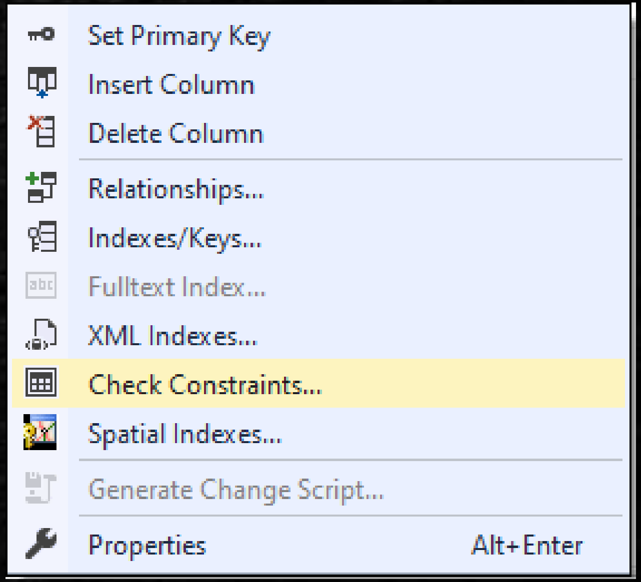
```
ALTER TABLE usage_logs
ADD CONSTRAINT chk_id
CHECK (logid > 10);
```

Column Name	Data Type	Allow Nulls
logid	int	☐
message	char(255)	☐
		☐

The LogID is defined with `NOT NULL, UNIQUE and CHECK`

If less than 10 value inserted, will get error

'Conflicted with CHECK constraint''



DROP CHECK Constraint

```
ALTER TABLE usage_logs
DROP CONSTRAINT chk_id;
```

The message column can now have less than 10 values

DEFAULT Constraint

- Used to insert the default value in the column when the user does not specify any value.

- It can be a constant value, system-defined value, or NULL.

- This is applicable for INSERT or UPDATE operations

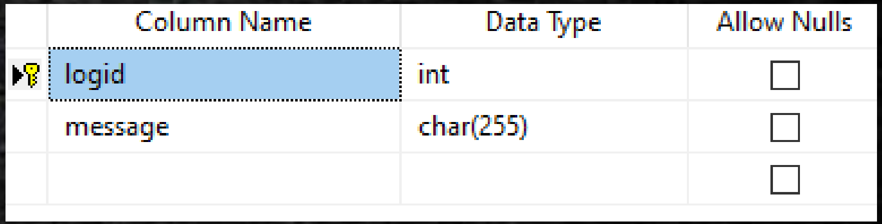
DEFAULT Constraint on Table Creation

```
CREATE TABLE usage_logs (
  logid INT NOT NULL UNIQUE,
  message char(255),
  msgdate DATETIME NOT NULL DEFAULT GETDATE()
)
```

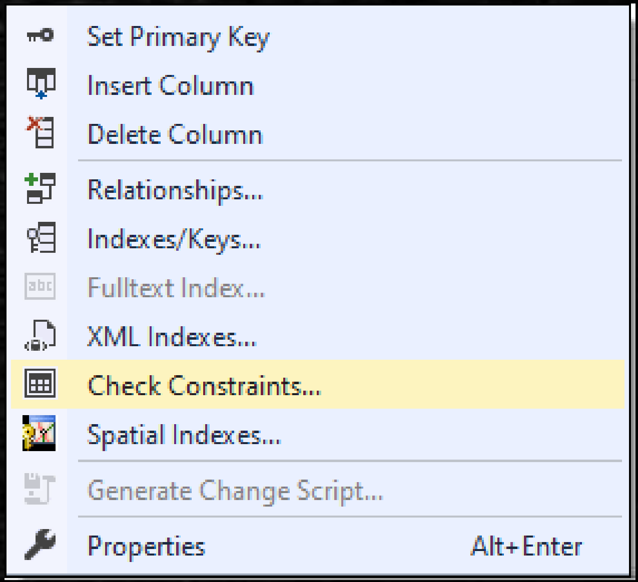
The `msgdate` if not specified, will have the date for today inserted.

DEFAULT Constraint : Alter Table

```
ALTER TABLE usage_logs
ADD CONSTRAINT def_date
DEFAULT (GETDATE()) FOR msgdate;
```



The `msgdate` if not specified, will have the date for today inserted.



DROP DEFAULT Constraint

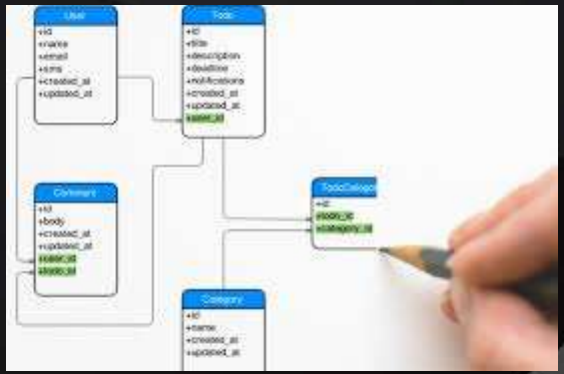
```
ALTER TABLE usage_logs
DROP CONSTRAINT def_date;
```

The message column can now have blank when not specified

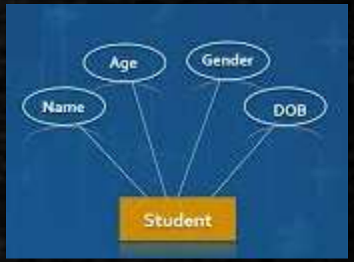
ER diagram

The Entity Relationship Diagram

Microsoft® SQL Server®



The Entity-Relationship Data Model



- What is a Data Model?

- The model which is used to move from informal description of what user wants to precise description of what can be implemented in a Database Management System.

Steps in developing a database for a SW project

- Requirements analysis
- Conceptual Database design
- (ER Model is used at this stage.)

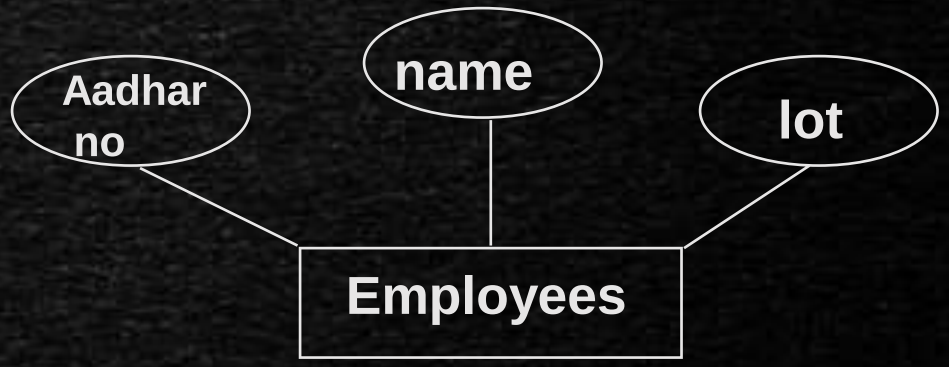
- Logical Database design
- Schema refinement
- Physical Database design
- Applications and security

Constructing model using the E R Diagram.

- The major task is Identifying: entities, attributes, and their relationships to construct model using the Entity Relationship Diagram.

- In short we can say :

- Entity → table
- Attribute → column
- Relationship → line



What is an Entity?

- "...anything (people, places, objects, events, etc.) about which we store information
- (e.g. supplier, employee, utility bag, car seat, etc.)."

- Tangible: customer, product
- Intangible: order, accounting receivable

Entity Instance

A single occurrence of an entity.

Here we have 6 instances

Attribute: First name

Entity: student

instance

Student ID	Last Name	First Name
2144	Arnold	Betty
3122	Taylor	John
3843	Simmons	Lisa
9844	Macy	Bill
2837	Leath	Heather
2293	Wrench	Tim

What is an Attribute?

- Attributes are data objects that either identify or describe entities (property of an entity).

- In other words, it is a descriptor whose values are associated with individual entities of a specific entity type

What is a Relationship?

- Relationships are associations between entities.
- Typically, a relationship is indicated by a verb connecting two or more entities.
- Employees are assigned to projects
- Relationships should be classified in terms of cardinality.
- One-to-one, one-to-many, etc.

Cardinality of a Relationship?

- The cardinality is the number of occurrences in one entity which are associated to the number of occurrences in another.

- There are three basic cardinalities (degrees of relationship).

- one-to-one (1:1), one-to-many (1:M), and many-to-many (M:N)

- Eg: one department assigned to multiple employees

Identifier Attributes

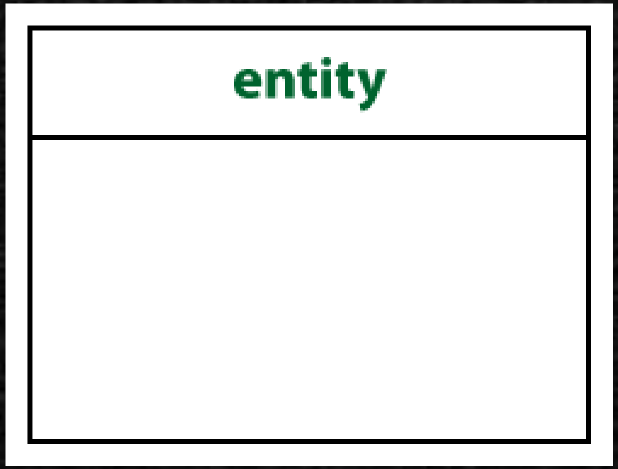
- Attributes uniquely identify entity instances.
- In relational database it will be most probably the primary key.
- Identifiers are represented by underlying the name of the attribute(s)
- Employee (Employee_ID), student (Student_ID)

Crow's Foot Notation for ER Diagram

- Known as Information Engineering (IE) notation
- It is the most popular notation method

1. Entity:

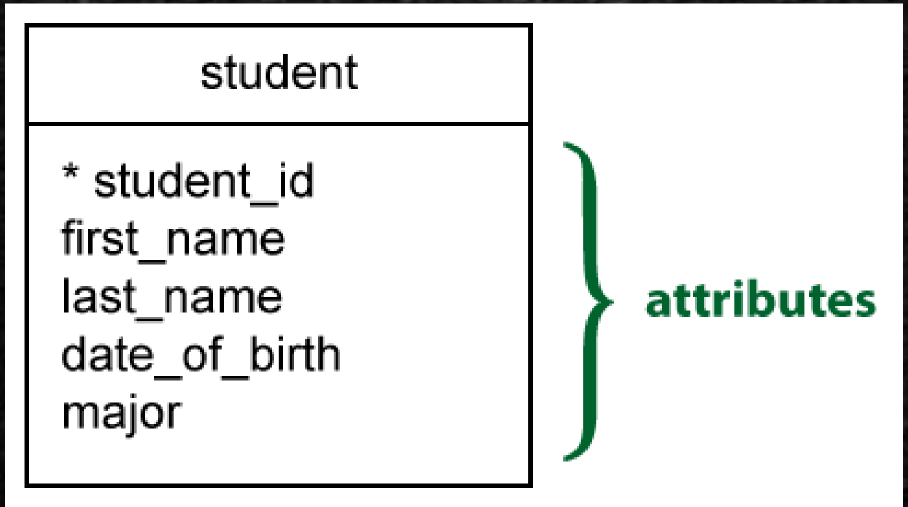
- Represented by a rectangle, with its name on the top.



Crow's Foot Notation for ER Diagram

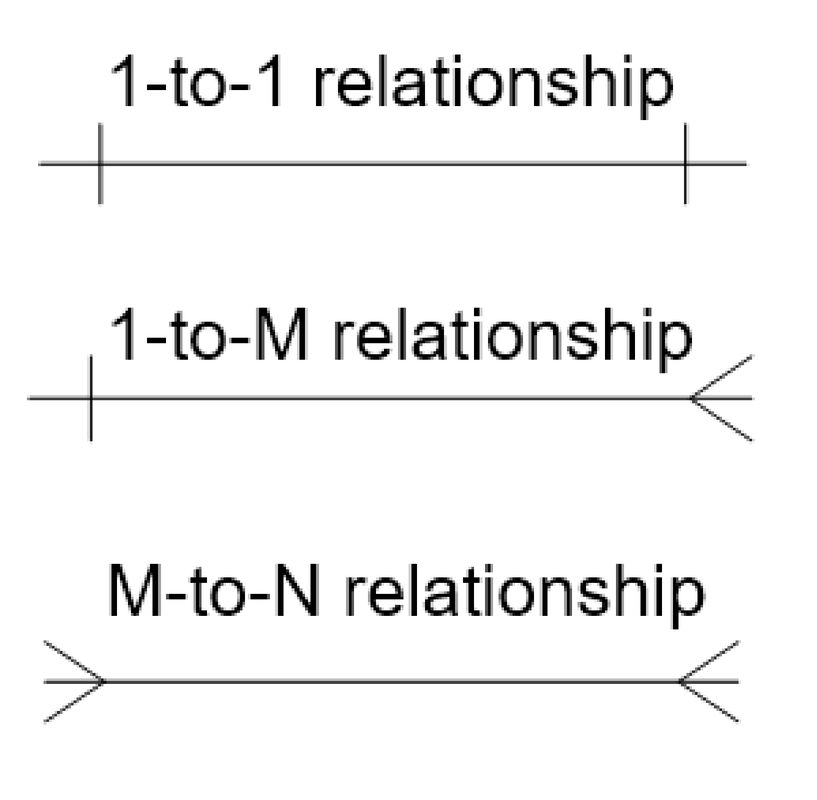
2. Attributes:

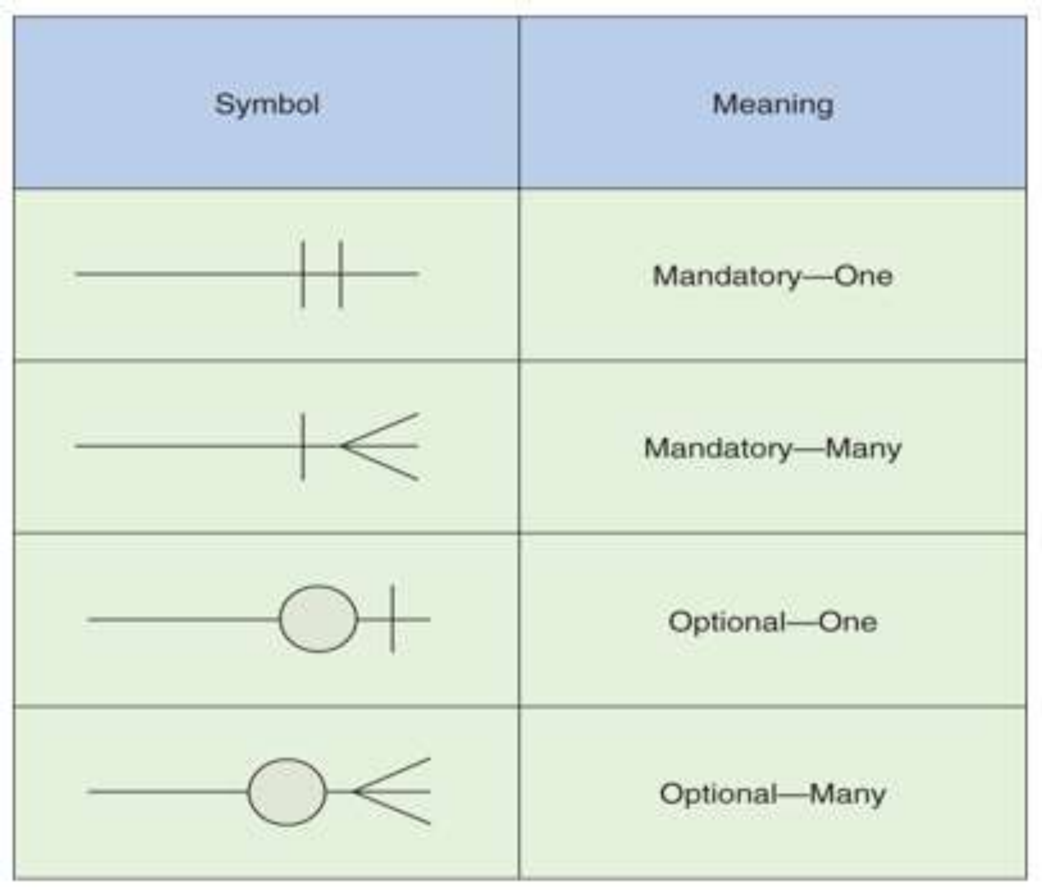
- Represented by underlining the name of the attribute(s)



Crow's Foot Notation for ER Diagram

3. Cardinality Types:



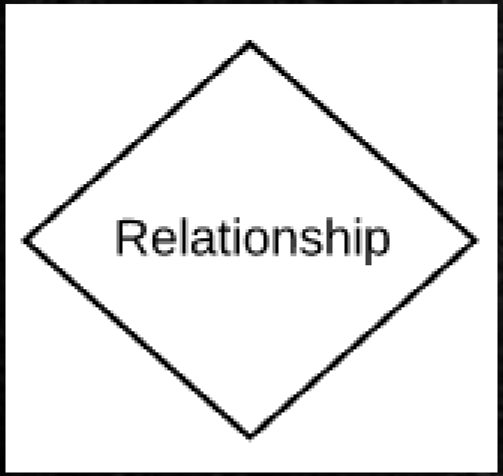


Symbol	Meaning
	Mandatory—One
	Mandatory—Many
	Optional—One
	Optional—Many

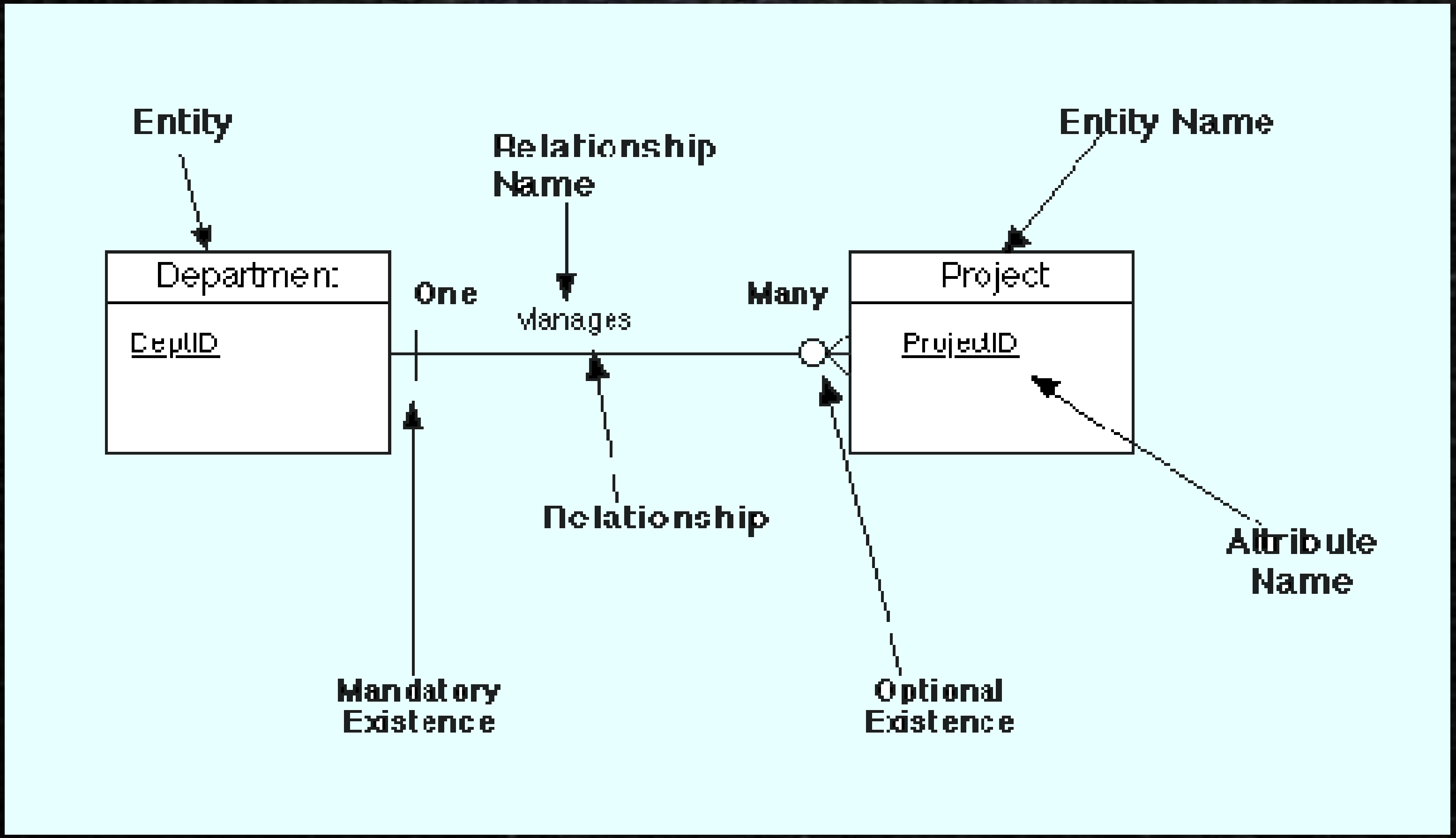
Crow's Foot Notation for ER Diagram

4. Relationship:

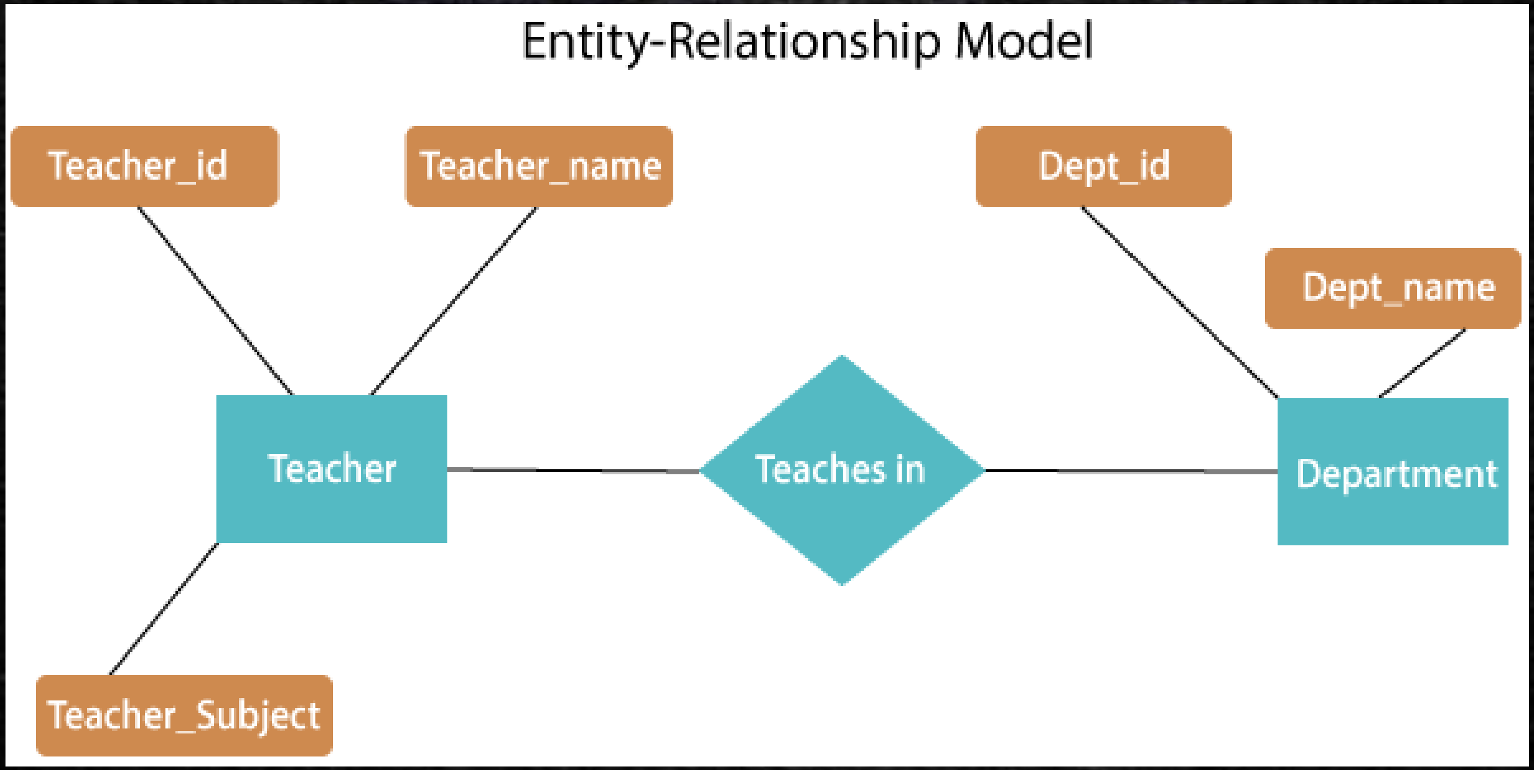
- Relationships are associations between or among entities.



An Example ER Model Diagram

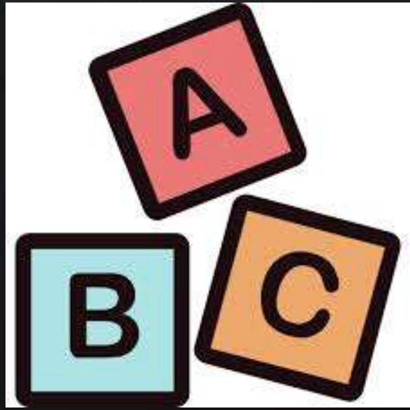


An Example ER Model Diagram



SQL Server String Functions

The Built In Functions to Manipulate Strings



SQL Server String Functions

- There are many built-in string functions in SQL Server that we can use to manipulate the character data and for processing the string data type. .

- These functions deal with string and character data in a variety of data types such as varchar, nvarchar, and char.

- It accepts a string value as an input and returns a string value regardless of the data type (string or numeric).

ASCII, CHARINDEX, CONCAT

```
SELECT ASCII('A')
-- returns '65', the ASCII value of A

SELECT CHARINDEX('World', 'Hello World')
-- returns '7', the position of search string

SELECT CONCAT('Hello', 'World');
-- returns 'Hello World'
```

SOUNDEX(), DIFFERENCE, LEFT() RIGHT()

```
SELECT SOUNDEX('Test'), SOUNDEX('Testing');
-- returns 'T235', Returns 4 char code to denote how the words
sound similar

SELECT DIFFERENCE('Test','Testing');
-- returns '3', Comparing two SOUNDEX values, return integer

SELECT LEFT('Hello World', 5), RIGHT('Hello World', 5);
-- specific number of characters from the left-side or right-side
-- returns Hello and World
```

LOWER() UPPER(), LTRIM() RTRIM(), REPLICATE()

```
SELECT LOWER('Hello'), UPPER('Hello');
-- returns hello and HELLO

--remove additional spaces from an string's left or right
SELECT RTRIM('Hello   '), LTRIM('   World');

SELECT REPLICATE('Hello', 3) AS Result;
-- repeat the string a specified number of times.
```

Those were the popular ones. There are still more string manipulation techniques

Please try doing more research with keyword 'SQL String Functions'...

SQL DATE and TIME Functions

The Built In Functions to Manipulate Date Data Type

SQL Server Date Time Functions

- The date and time function is used to handle date and time data effectively.

- The default format of date and time in SQL are :

- DATE: YYYY-MM-DD
- DATETIME: YYYY-MM-DD HH: MI: SS
- TIMESTAMP: YYYY-MM-DD HH: MI: SS
- YEAR: YYYY or YY

CURRENT_TIMESTAMP, GETDATE, GETUTCDATE

```
SELECT CURRENT_TIMESTAMP AS DATE;
--returns current date time


--returns current date time
SELECT GETDATE() AS Date;


--time based on the UTC timestamp
SELECT UTCDATE() AS Date;


--time with more precision
SELECT SYSDATETIME() AS Date;
```

DATENAME, DATEPART

```
--to extract the part of the date day, month, or year.
SELECT DATENAME(day, '2021/12/10') AS Result1,
DATENAME(month, '2021/12/10') AS Result2,
DATENAME(year, '2021/12/10') AS Result3;

-- to extract the part of the date as an integer value
SELECT DATEPART(day, '2021/12/10') AS Result1,
DATEPART(month, '2021/12/10') AS Result2,
DATEPART(year, '2021/12/10') AS Result3;
```

YEAR, MONTH, DAY, DATEPART

```
--to extract the part of the date day, month, or year.
SELECT YEAR('2021/12/10') AS Result1,
MONTH('2021/12/10') AS Result2,
DAY('2021/12/10') AS Result3;

-- to extract the part of the date as an integer value
SELECT DATEPART(day, '2021/12/10') AS Result1,
DATEPART(month, '2021/12/10') AS Result2,
DATEPART(year, '2021/12/10') AS Result3;
```

DATEDIFF

```
--to extract the difference of days, months, or years.
SELECT DATEDIFF(dd,'2020/2/3', '2021/3/5') AS TotalDays,
 DATEDIFF(MM,'2020/2/3', '2021/3/5') AS TotalMonths,
 DATEDIFF(WK,'2020/2/3', '2021/3/5') AS TotalWeeks;
```

Those were the popular ones.
There are still more date/time manipulation techniques
Please try doing more research with keyword 'SQL Date Functions'...

SQL MATHEMATICAL Functions

The Built In Functions to Perform Mathematical Operations

SQL Server Mathematical Functions

- There are several mathematical functions to perform basic mathematical calculations.

- Eg: to find the square root, logarithmic, round, floor, elementary exponential value, and trigonometric functions.

SQRT, ABS, CEILING, FLOOR

```
SELECT SQRT(25) AS Result1
--returns square root

--returns absolute value
SELECT ABS(-20) AS Result1;

--returns next highest value
SELECT CEILING(22.19) AS Result1,

--returns next lowest value
SELECT FLOOR(22.19) AS Result1,
```

RAND, POWER, LOG, SIGN

```
SELECT POWER(3,2) AS Result1,
--returns exponential


--returns natural logarithm
SELECT LOG(20) AS Result1;


--returns sign
SELECT SIGN(-22) AS Result1,
```

Those were the popular ones.

There are still more mathematical functions

Please try doing more research with keyword 'SQL mathematical Functions'

SQL CONVERSION Functions

The Built In Functions to Perform Data Conversion

SQL Server CONVERT Function

- Used to convert the data type of a value into the other type.

- The syntax of a CONVERT function in SQL Server is:

CONVERT (data_type, expr, length)

CONVERT() EXAMPLES

```
SELECT CONVERT(int, 30.55);
--convert expression to int


--converts string expression to datetime
SELECT CONVERT(datetime, '2020-08-25');


--convert to varchar of length 100
SELECT CONVERT(varchar, '2020-08-25', 101);
```

SQL Server CAST Function

- The CAST() function also converts a value (of any type) into a specified datatype.

- The syntax of a CAST function is:

  CAST(expression AS datatype(length))

- CONVERT is SQL Server specific, CAST is ANSI.

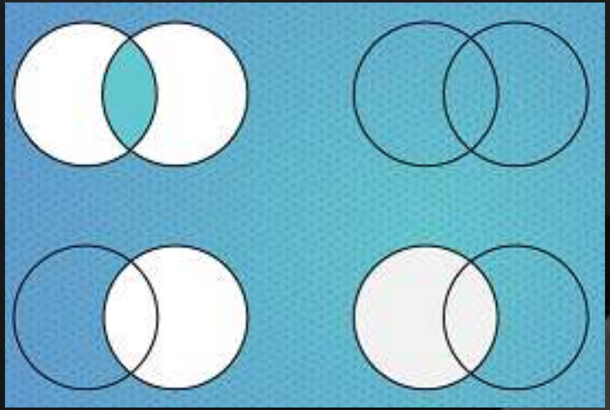
CAST() EXAMPLES

```
SELECT CAST(20.65 AS varchar);
--convert expression to varchar


--converts a value to a datetime datatype:
SELECT CAST('2020-08-25' AS datetime);
```

SQL JOINS

The Built In Functions to Perform Data Conversion

Microsoft® SQL Server®



SQL Server JOIN Clause

- In RDBMS, we store our data in multiple tables that are linked together by a common key value

- We usually need to fetch data from two or more tables into the desired output based on some conditions.

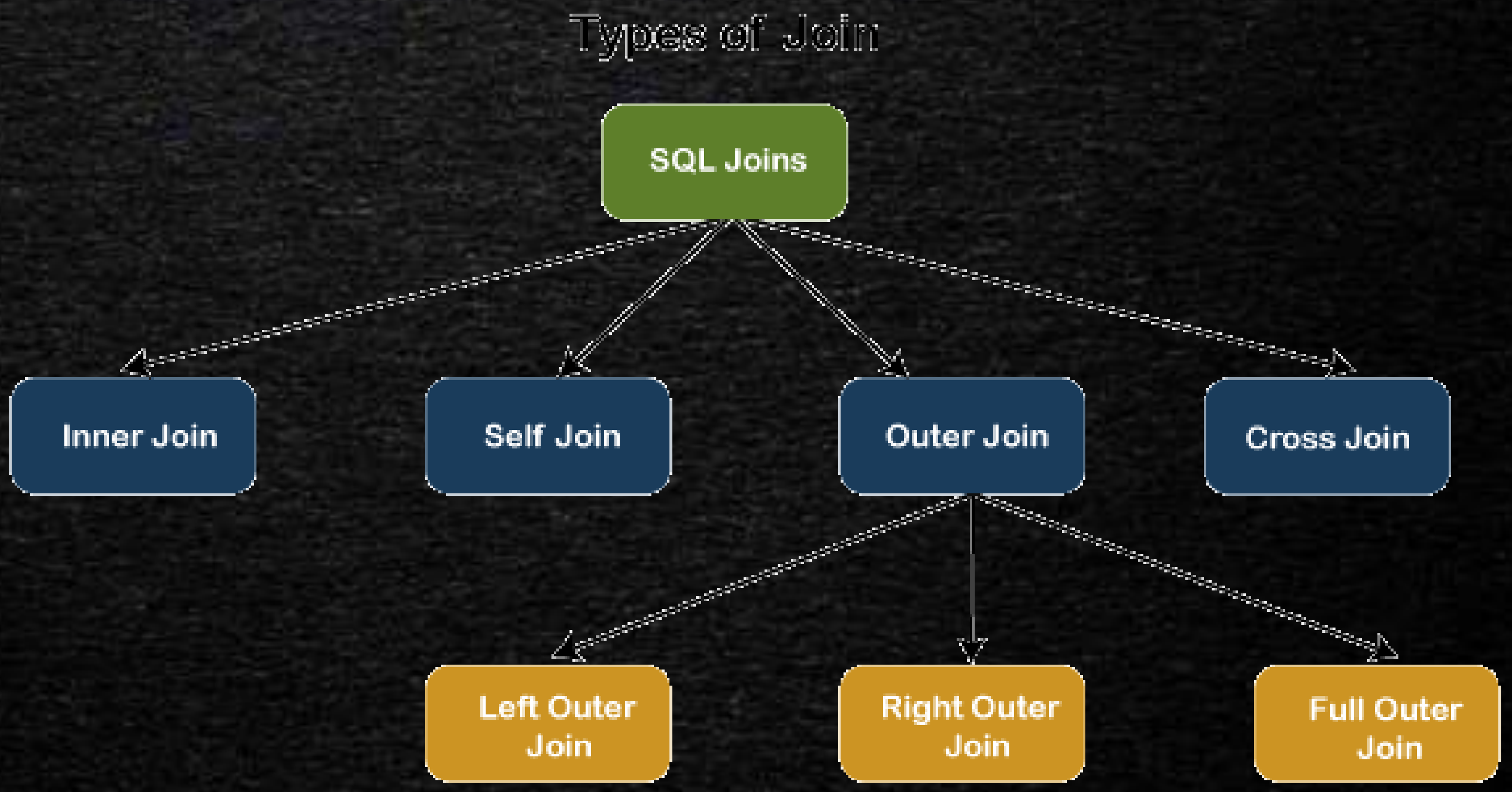
- The JOIN clause in SQL allows us do this.

- We can join the tables using a SELECT statement and a join condition.

Types of JOINS in SQL Server

- SQL Server mainly supports four types of JOINS.

- INNER JOIN
- OUTER JOIN
- CROSS JOIN
- SELF JOIN



Create Tables Trainee, Fee and Semester

```
CREATE DATABASE training

USE training


CREATE TABLE trainee (
   id int PRIMARY KEY IDENTITY,
   admission_no varchar(45) NOT NULL,
   first_name varchar(45) NOT NULL,
   last_name varchar(45) NOT NULL,
   age int,
   city varchar(25) NOT NULL
);
```

```
CREATE TABLE fee (
   admission_no varchar(45) NOT NULL,
   sem_no int NOT NULL,
   course varchar(45) NOT NULL,
   amount int,
);
```

```
CREATE TABLE semester (
   sem_no int NOT NULL,
   sem_name varchar(10),
);
```

Fill the Tables With Dummy Data

```
INSERT INTO trainee (admission_no, first_name, last_name,
age,city)
VALUES (3354,'Spider', 'Man', 13, 'Texas'),
 (2135, 'James', 'Bond', 15, 'Alaska'),
 (4321, 'Jack', 'Sparrow', 14, 'California'),
 (4213,'John', 'McClane', 17, 'New York'),
 (5112, 'Optimus', 'Prime', 16, 'Florida'),
 (6113, 'Captain', 'Kirk', 15, 'Arizona'),
 (7555,'Harry', 'Potter', 14, 'New York'),
 (8345, 'Rose', 'Dawson', 13, 'California');
```

Fill the Tables With Dummy Data

```
INSERT INTO semester (sem_no, sem_name)
VALUES
(1,'First Sem'),
(2, 'Second Sem'),
(3, 'Third Sem'),
(4, 'Fourth Sem');
```

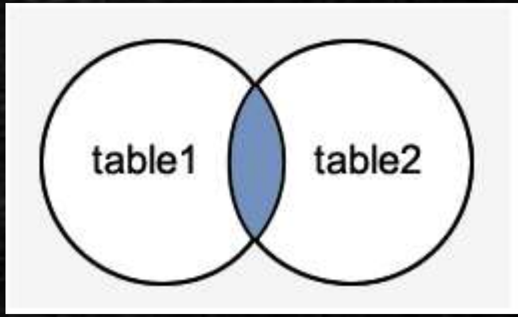
Fill the Tables With Dummy Data

```
INSERT INTO fee (admission_no, sem_no, course, amount)
VALUES (3354, 1, 'Java', 20000),
(7555, 1, 'Android', 22000),
(4321, 2, 'Python', 18000),
(8345, 2, 'SQL', 15000),
(9345, 2, 'Blockchain', 16000),
(9321, 3, 'Ethical Hacking', 17000),
(5112, 1, 'Machine Learning', 30000);
```

Run select query for tables and verify the data

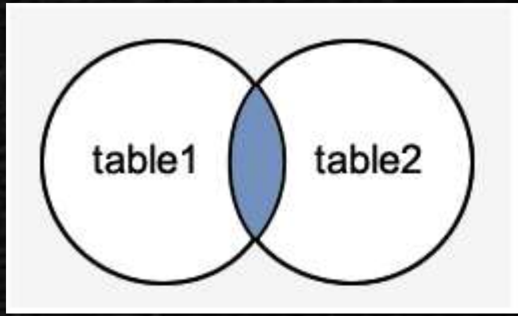
INNER JOIN

- INNER JOIN It is the simple and most popular form of join
- It is the default join (Same result even if INNER keyword is not specified during JOIN)
- It Returns records that have matching values in both tables



INNER JOIN syntax

SELECT columns
FROM table1
INNER JOIN table2 ON condition1
INNER JOIN table3 ON condition2



INNER JOIN Example

```
SELECT trainee.admission_no, trainee.first_name,
trainee.last_name, fee.course, fee.amount

FROM trainee

INNER JOIN fee ON trainee.admission_no = fee.admission_no;
```

Results | Messages

	admission_no	first_name	last_name	course	amount_paid
1	3354	Spider	Man	Java	20000
2	4321	Jack	Sparrow	Python	18000
3	5112	Optimus	Prime	Machine Learning	30000
4	7555	Harry	Potter	Android	22000
5	8345	Rose	Dawson	SQL	15000

INNER JOIN with 3 Tables Example

```
SELECT trainee.admission_no, trainee.first_name,
trainee.last_name, fee.course, fee.amount, semester.sem_name
FROM trainee
INNER JOIN fee ON trainee.admission_no = fee.admission_no
INNER JOIN semester ON semester.sem_no = fee.sem_no
```

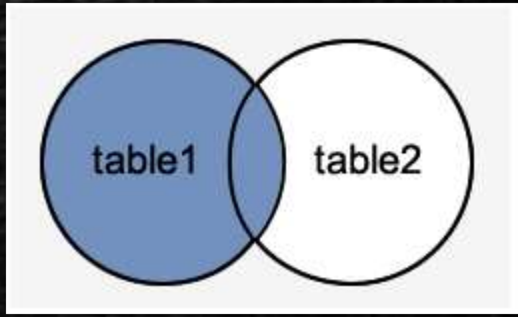
Results Messages

	admission_no	first_name	last_name	course	amount_paid	sem_name
1	3354	Spider	Man	Java	20000	First Sem
2	4321	Jack	Sparrow	Python	18000	Second Sem
3	5112	Optimus	Prime	Machine Learning	30000	First Sem
4	7555	Harry	Potter	Android	22000	First Sem

LEFT (OUTER) JOIN

- LEFT JOIN returns all records from the left table (trainees), even if there are no matches in the right table (fee).

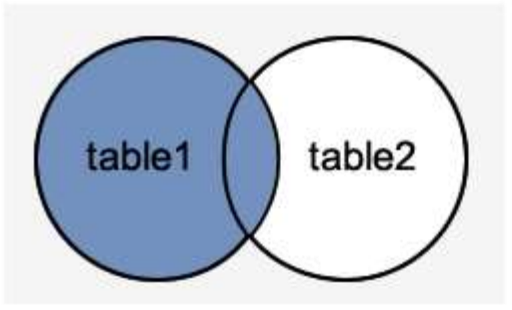
- It will return NULL when no matching record is found in the right side table



- Since OUTER is an optional keyword, it is also known as LEFT JOIN.

LEFT (OUTER) JOIN

```
SELECT column_lists
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```



LEFT (OUTER) JOIN Example

```
SELECT trainee.admission_no, trainee.first_name,
trainee.last_name, fee.course, fee.amount

FROM trainee

LEFT OUTER JOIN fee ON trainee.admission_no = fee.admission_no;
```

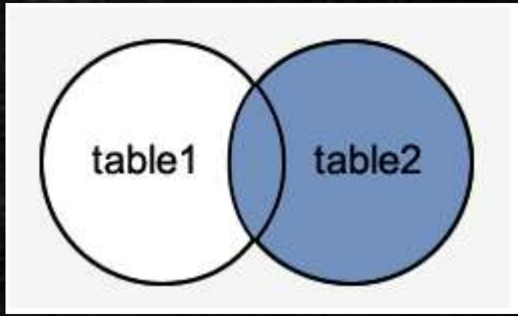
Results | Messages

	admission_no	first_name	last_name	course	amount_paid
1	3354	Spider	Man	Java	20000
2	2135	James	Bond	NULL	NULL
3	4321	Jack	Sparrow	Python	18000
4	4213	John	McClane	NULL	NULL
5	5112	Optimus	Prime	Machine Learning	30000
6	6113	Captain	Kirk	NULL	NULL
7	7555	Harry	Potter	Android	22000
8	8345	Rose	Dawson	SQL	15000

RIGHT (OUTER) JOIN

- RIGHT JOIN returns all records from the right table (fee), even if there are no matches in the left table (trainees).

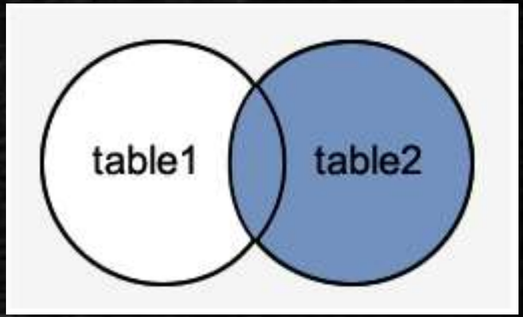
- It will return NULL when no matching record is found in the left side table



- Since OUTER is an optional keyword, it is also known as RIGHT JOIN.

RIGHT (OUTER) JOIN

```
SELECT column_lists
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```



RIGHT (OUTER) JOIN Example

```sql
SELECT trainee.admission_no, trainee.first_name,
trainee.last_name, fee.course, fee.amount

FROM trainee

RIGHT OUTER JOIN fee ON trainee.admission_no = fee.admission_no;
```

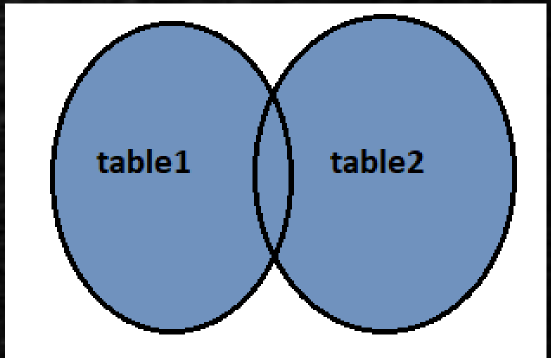
Results | Messages

	admission_no	first_name	last_name	course	amount_
1	3354	Spider	Man	Java	20000
2	7555	Harry	Potter	Android	22000
3	4321	Jack	Sparrow	Python	18000
4	8345	Rose	Dawson	SQL	15000
5	NULL	NULL	NULL	Blockchain	16000
6	NULL	NULL	NULL	Ethical Hacking	17000
7	5112	Optimus	Prime	Machine Learning	30000

FULL (OUTER) JOIN

- FULL JOIN returns a result that includes all rows from both tables.

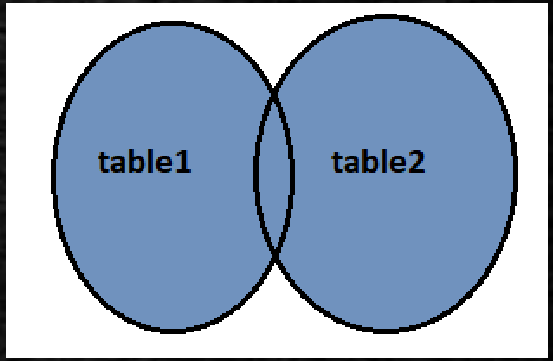
- It will return NULL when no matching record is found in the left side or right side table



- FULL OUTER JOIN and FULL JOIN are the same.

FULL (OUTER) JOIN

```
SELECT column_lists
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```



FULL (OUTER) JOIN Example

```
SELECT trainee.admission_no, trainee.first_name,
trainee.last_name, fee.course, fee.amount
FROM trainee
FULL OUTER JOIN fee ON trainee.admission_no = fee.admission_no;
```

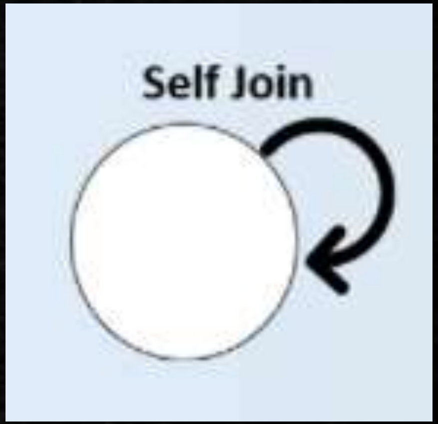
Results | Messages

	admission_no	first_name	last_name	course	amount
1	3354	Spider	Man	Java	20000
2	2135	James	Bond	NULL	NULL
3	4321	Jack	Sparrow	Python	18000
4	4213	John	McClane	NULL	NULL
5	5112	Optimus	Prime	Machine Learning	30000
6	6113	Captain	Kirk	NULL	NULL
7	7555	Harry	Potter	Android	22000
8	8345	Rose	Dawson	SQL	15000
9	NULL	NULL	NULL	Blockchain	16000
10	NULL	NULL	NULL	Ethical Hacking	17000

SELF JOIN

- A table is joined to itself using the SELF JOIN.

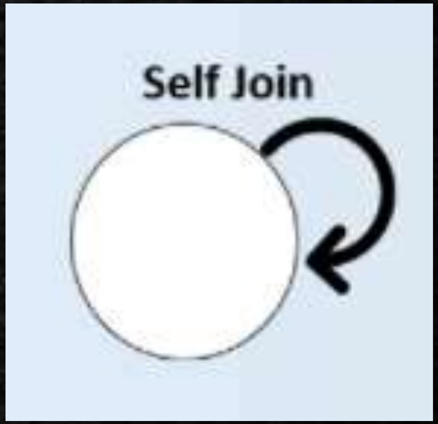
- Just like in other joins, instead of two tables, the copy of the same table referenced by different alias names is used.



- SELF JOIN can be thought of as a JOIN of two copies of the same tables and is used to extract hierarchical data and comparing rows inside a single table.

SELF JOIN

```
SELECT T1.col_name, T2.col_name...
FROM table1 T1, table1 T2
WHERE join_condition;
```



SELF JOIN Example – List People in order of city

```
SELECT t1.first_name, t1.last_name, t2.city
FROM trainee t1 , trainee t2
WHERE t1.admission_no = t2.admission_no
AND t1.city = t2.city
ORDER BY t2.city;
```

- Here we used the `admission_no` and `city` column as a join condition to get the data from both tables.

	first_name	last_name	city
1	James	Bond	Alaska
2	Captain	Kirk	Arizona
3	Rose	Dawson	California
4	Jack	Sparrow	California
5	Optimus	Prime	Florida
6	Harry	Potter	London
7	Spider	Man	Texas
8	John	McClane	Texas

SELF JOIN Example – List People in order of city

```
SELECT t1.first_name, t1.last_name, t2.city
FROM trainee t1 , trainee t2
WHERE t1.admission_no != t2.admission_no
AND t1.city = t2.city
ORDER BY t2.city;
```

Results | Messages

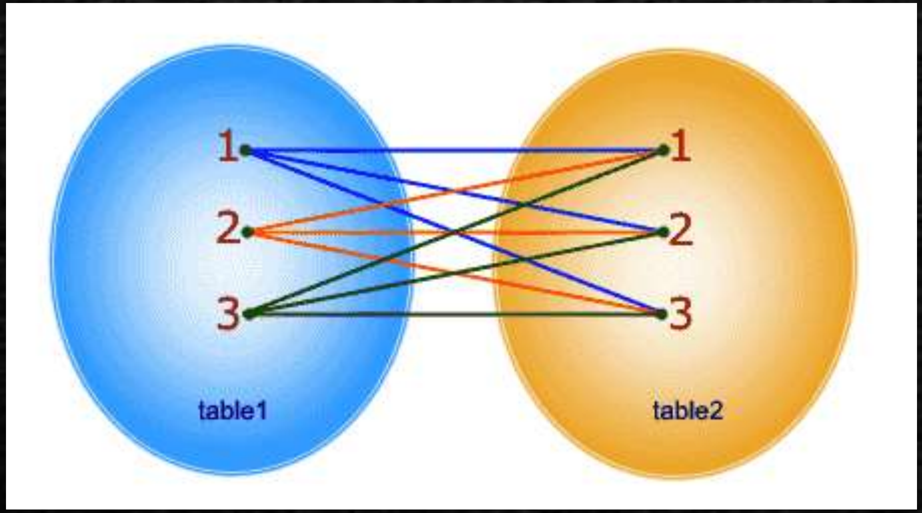
	first_name	last_name	city
1	Jack	Sparrow	California
2	Rose	Dawson	California
3	Spider	Man	Texas
4	John	McClane	Texas

- Here we are mentioning that the city name should be equal but admission no should be different.
- This will give us the repeating list of cities
- We can also use '<>' instead of '!=' in SQL

CROSS JOIN

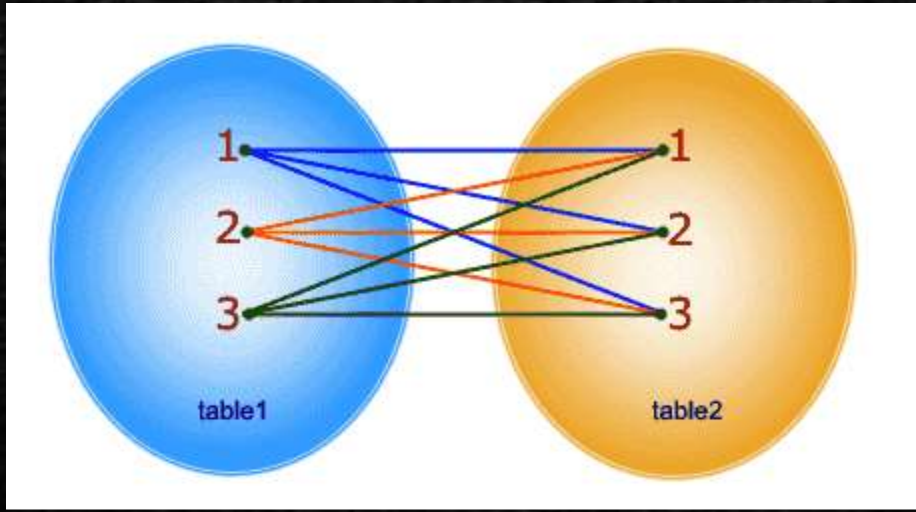
- The SQL CROSS JOIN produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table.

- This kind of result is called as Cartesian Product.



CROSS JOIN

```
SELECT column_lists
FROM table1
CROSS JOIN table2;
```



CROSS JOIN Example

```
SELECT * from trainee
CROSS join fee
```

Results | Messages

	id	admission_no	first_name	last_name	age	city	admission_no	sem_no	course	amount
1	1	3354	Spider	Man	33	Texas	3354	1	Java	20000
2	2	3355	James	Bond	27	Alaska	3354	1	Java	20000
3	3	3356	Jack	Sparrow	33	California	3354	1	Java	20000
4	4	3357	John	McClane	40	Texas	3354	1	Java	20000
5	5	3358	Optimus	Prime	20	Florida	3354	1	Java	20000
6	6	3359	Captain	Kirk	37	Arizona	3354	1	Java	20000
7	7	3360	Harry	Potter	12	London	3354	1	Java	20000
8	8	3361	Rose	Dawson	22	California	3354	1	Java	20000
9	1	3354	Spider	Man	33	Texas	3355	2	Android	22000
10	2	3355	James	Bond	27	Alaska	3355	2	Android	22000
11	3	3356	Jack	Sparrow	33	California	3355	2	Android	22000
12	4	3357	John	McClane	40	Texas	3355	2	Android	22000
13	5	3358	Optimus	Prime	20	Florida	3355	2	Android	22000
14	6	3359	Captain	Kirk	37	Arizona	3355	2	Android	22000
15	7	3360	Harry	Potter	12	London	3355	2	Android	22000
16	8	3361	Rose	Dawson	22	California	3355	2	Android	22000
17	1	3354	Spider	Man	33	Texas	3356	2	Python	18000
18	2	3355	James	Bond	27	Alaska	3356	2	Python	18000
19	3	3356	Jack	Sparrow	33	California	3356	2	Python	18000
20	4	3357	John	McClane	40	Texas	3356	2	Python	18000
21	5	3358	Optimus	Prime	20	Florida	3356	2	Python	18000
22	6	3359	Captain	Kirk	37	Arizona	3356	2	Python	18000

CROSS JOIN Example

```
SELECT trainee.admission_no, trainee.first_name,
trainee.last_name, fee.course, Fee.amount_paid
FROM trainee
CROSS JOIN fee
WHERE trainee.admission_no = fee.admission_no;
```

- If WHERE clause is used with CROSS JOIN, it functions like an INNER JOIN.

Results | Messages

	admission_no	first_name	last_name	course	amount
1	3354	Spider	Man	Java	20000
2	3355	James	Bond	Android	22000
3	3356	Jack	Sparrow	Python	18000
4	3357	John	McClane	SQL	15000
5	3358	Optimus	Prime	Blockchain	16000
6	3359	Captain	Kirk	Ethical Hacking	17000
7	3360	Harry	Potter	Machine Learning	16000

SQL SERVER STORED PROCEDURES

The Block of Reusable Statements

Stored Procedures

- A stored procedure is a group of SQL statements compiled into a single execution plan.

- It is compiled once and can be used again and again.

- To supply data to the procedure we must have to use parameters in procedure.

- To return any value from procedure we use return statement.

Features of Stored Procedures

- Server Traffic Reduction : instead of sending several SQL statements, we need to send only the procedure name

- Reusable: Prevents unnecessary rewrites of the same code.

- Improved Performance: Procedure is usually processed quicker because its pre-compiled and the query processor does not have to create a new plan.

- Specific to a Vendor: Stored procedures written in one platform cannot run on another.

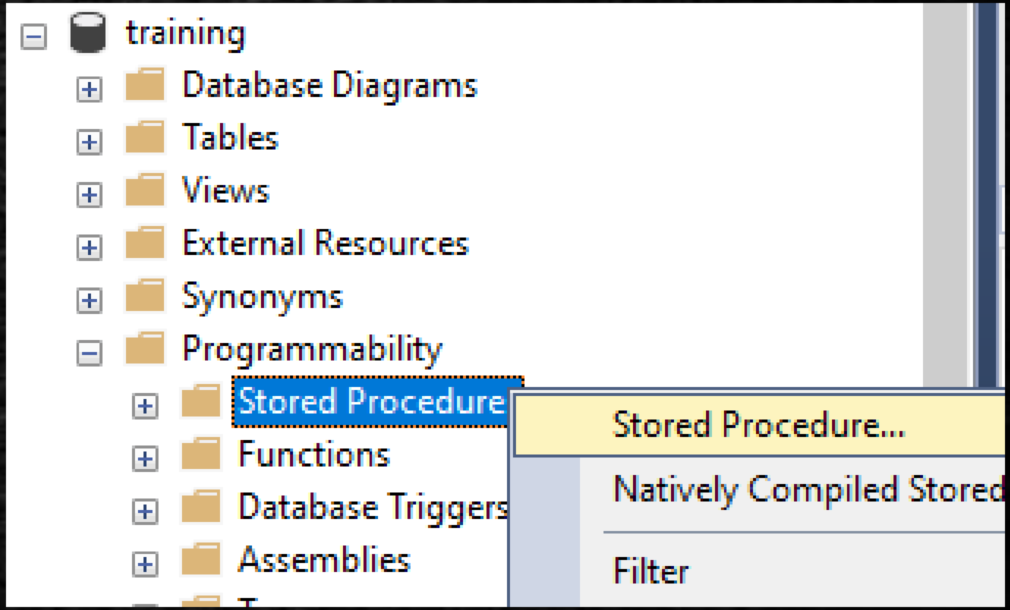
Creating a Stored Procedure

```
CREATE PROCEDURE procedure_name
AS
BEGIN
	query statements….
END;
```

Example : Creating a Simple Stored Procedure

```
CREATE PROCEDURE traineeAgewiseList
AS
BEGIN
    SELECT first_name, age, city
    FROM trainee
    ORDER BY age;
END;
```

- Creating SP using MSMS



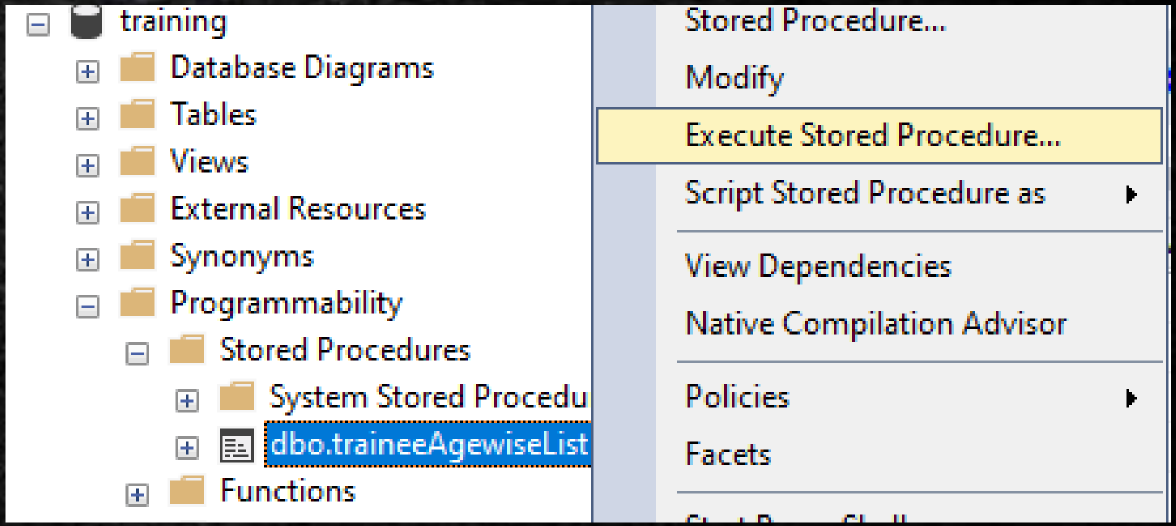
Example : Executing the Stored Procedure

```
EXEC traineeAgewiseList
```

Results | Messages

	first_name	age	city
1	Harry	12	London
2	Optimus	20	Florida
3	Rose	22	California
4	James	27	Alaska
5	Jack	33	California
6	Spider	33	Texas
7	Captain	37	Arizona
8	John	40	Texas

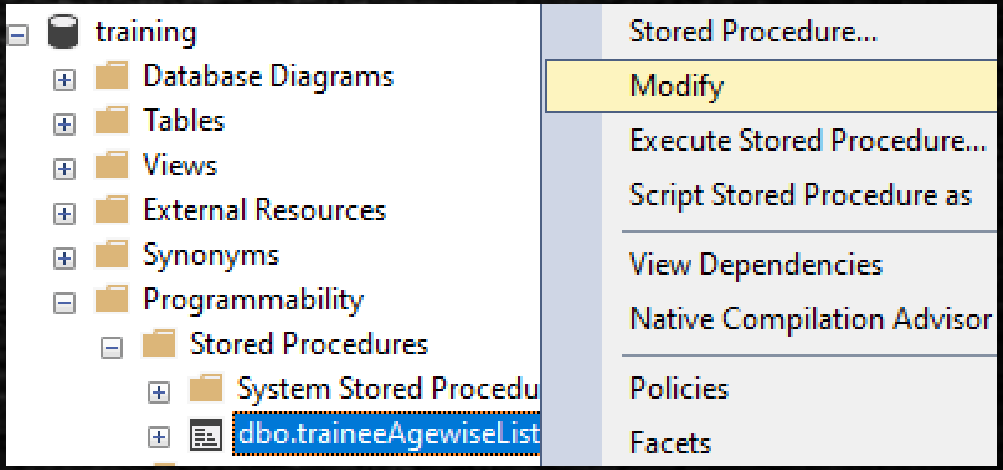
- Executing SP using MSMS



Example : Modifying a Stored Procedure

```
ALTER PROCEDURE traineeAgewiseList
AS
BEGIN
      SELECT first_name, last_name, age, city
      FROM trainee
      ORDER BY age;
END;
```

- Altering SP using MSMS



Example : Listing the Stored Procedures in current db

```
SELECT * FROM sys.procedures;
```

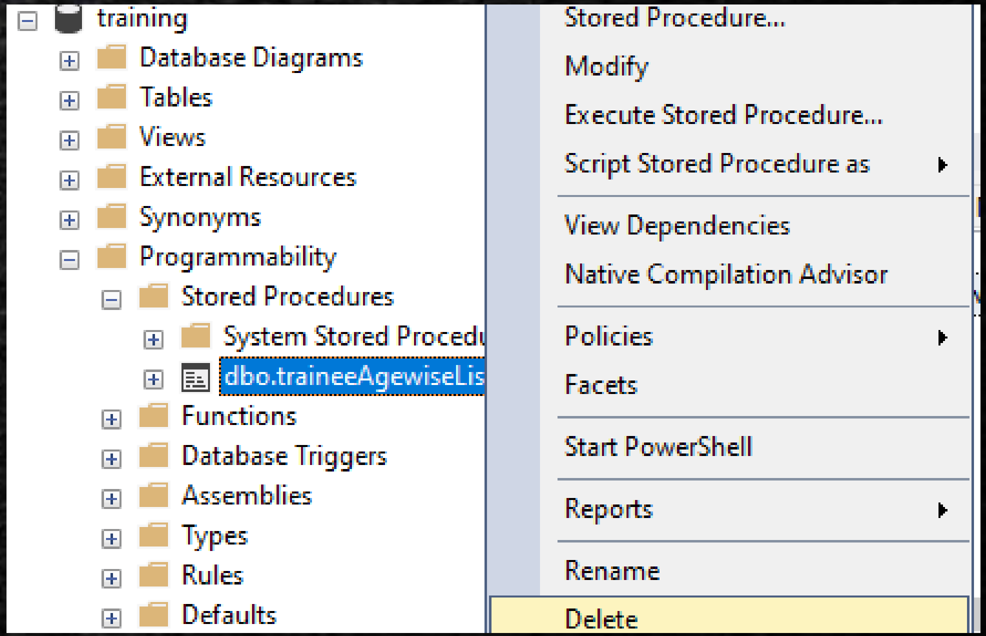
Results | Messages

	name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date	is_ms_
1	traineeAgewiseList	1205579333	NULL	1	0	P	SQL_STORED_PROCEDURE	2021-12-16 05:54:26.693	2021-12-16 06:03:37.160	0

Example : Deleting the Stored Procedure

```
DROP PROCEDURE procedure_name;
```

- Deleting SP using MSMS



Input Parameters in Stored Procedure

- We can create input parameters stored procedures.

- It enable us to pass one or more parameters to get the filtered result.

Example : Parameters in the Stored Procedure

```
CREATE PROCEDURE getTraineesFromCity (@city VARCHAR(50))
AS
BEGIN
   SET NOCOUNT ON;
   SELECT first_name, last_name, age, city
   FROM trainee
   WHERE city = @city
END
```

Example : Parameters in the Stored Procedure

```
exec getTraineesFromCity 'texas'
```

Results Messages

	first_name	last_name	age	city
1	Spider	Man	33	Texas
2	John	McClane	40	Texas

Return Parameters from Stored Procedure

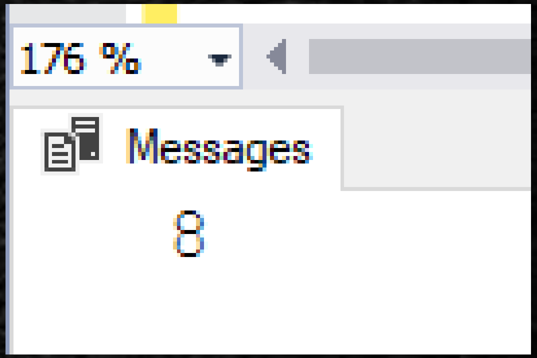
- We can provide output parameters in a stored procedure.
- Using the syntax
  - parameter_name data_type OUTPUT

Example : Return parameter from Stored Procedures

```
CREATE PROCEDURE getTraineeCount (@traineeCount INT OUTPUT)
AS
BEGIN
    SELECT @traineeCount = COUNT(id)FROM trainee;
END;
```

Example : Receive and process value

```
-- Declare an int to hold output
DECLARE @TraineeCount INT

-- Execute SP with keyword OUTPUT
EXEC getTraineeCount @TraineeCount OUTPUT

-- Print the result
PRINT @TraineeCount
```




```
-- Run these all at once, because after execution the memory is cleared
```

SQL SERVER SUBQUERY

The nested query

Subquery in SQL

- A subquery is a query that is nested inside a SELECT, INSERT, UPDATE, or DELETE statement, or inside another subquery.

- It is embedded within the WHERE clause.

Example : Subquery with SELECT

```
SELECT *
    FROM trainee
    WHERE id IN (SELECT id
          FROM trainee
          WHERE age > 20) ;
```

Results | Messages

	id	admission_no	first_name	last_name	age	city
1	1	3354	Spider	Man	33	Texas
2	2	3355	James	Bond	27	Alaska
3	3	3356	Jack	Sparrow	33	California
4	4	3357	John	McClane	40	Texas
5	6	3359	Captain	Kirk	37	Arizona
6	8	3361	Rose	Dawson	22	California

Example : Subquery with DELETE

```
DELETE
    FROM trainee
    WHERE admission_no IN (SELECT admission_no
          FROM trainee
          WHERE admission_no = 3356) ;
```

Results | Messages

	id	admission_no	first_name	last_name	age	city
1	1	3354	Spider	Man	33	Texas
2	2	3355	James	Bond	27	Alaska
3	3	3356	Jack	Sparrow	33	California
4	4	3357	John	McClane	40	Texas
5	6	3359	Captain	Kirk	37	Arizona
6	8	3361	Rose	Dawson	22	California

SQL SERVER VIEWS

A virtual table just to view result

Views in SQL

- A view is a virtual table based on the result-set of an SQL statement.
- Just like normal tables, a view also contains rows and columns
- We can have SQL statements and functions added to a view

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Create View and Call the View

```
CREATE VIEW [texas trainees] AS
SELECT first_name, last_name, city
FROM trainee
WHERE city = 'Texas';
```

The view can be queried later using

```
SELECT * FROM [texas trainees];
```

Results | Messages

	first_name	last_name	city
1	John	McClane	Texas

SQL SERVER TRIGGERS

Fires on Event

SQL Trigger

- SQL trigger is a database object which fires when an event occurs in a database.

- We can execute a SQL query that will do a 'process' when a change occurs on a database table such as insertion, updating, deletion etc

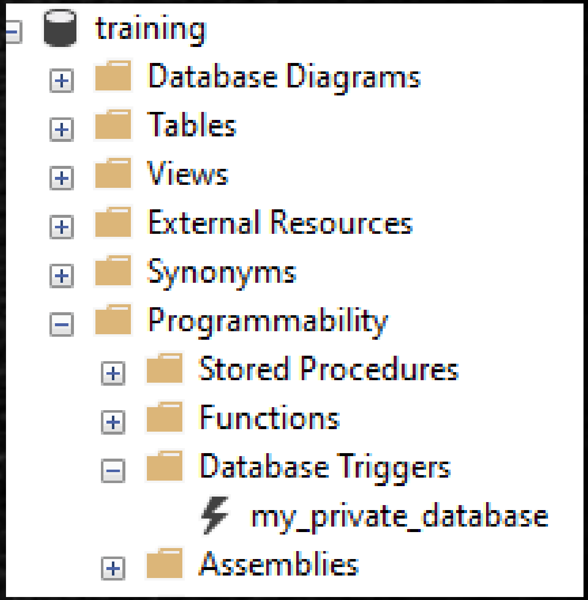
Types of SQL Triggers

There are two types of triggers:

- DDL Trigger
  - Fired in response to DDL (Data Definition Language) events like Create, Alter and Drop
- DML Trigger
  - Fired in response to DML (Data Manipulation Language) events like Insert, Update, and Delete.
  - There are two types of DML triggers
    - AFTER Triggers and INSTEAD OF triggers

DDL Trigger Example

- **DDL Trigger** - Is applied to the whole database




```
use training;
create trigger my_private_database
on database
for
create_table,alter_table,drop_table
as
print'These operations are not allowed in this db'
rollback;
```

DDL Trigger Example

- When attempted to create, alter or drop any table in a database then the following message appears

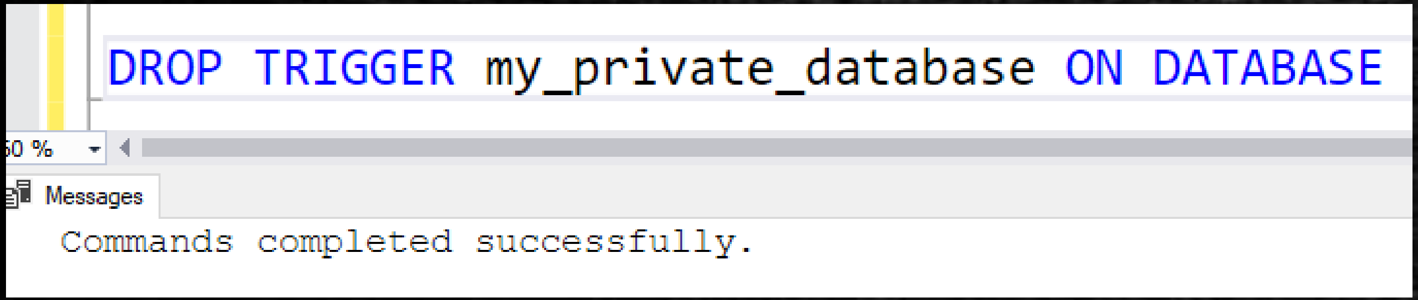
```
CREATE TABLE test (
    id int,
    name varchar(20)
);
```

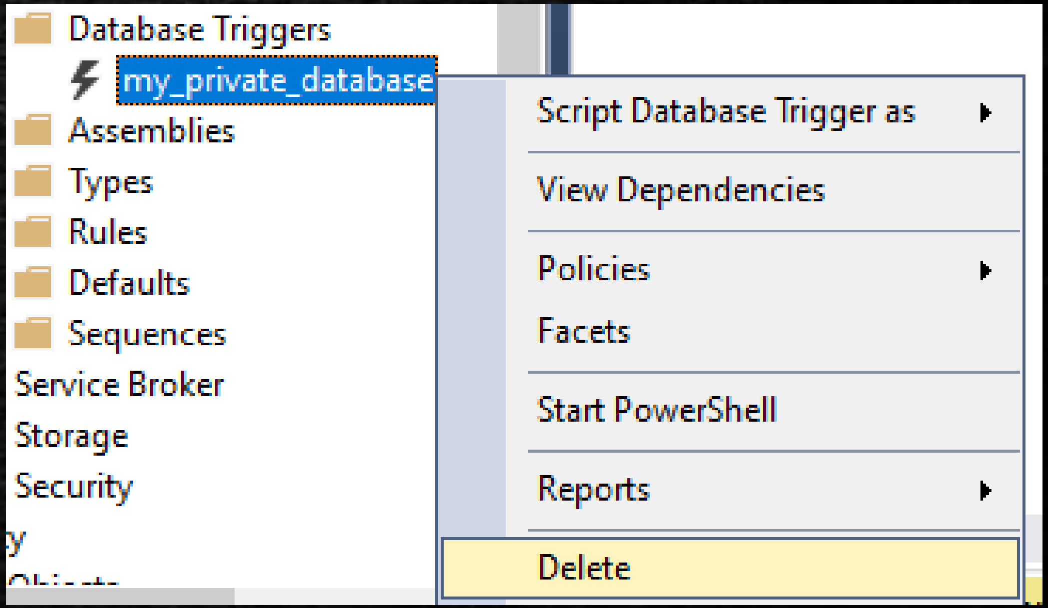
Messages

```
These operations are not allowed in this db
Msg 3609, Level 16, State 2, Line 10
```

DDL Trigger Deletion

```
Use training

DROP TRIGGER my_private_database
ON DATABASE
```





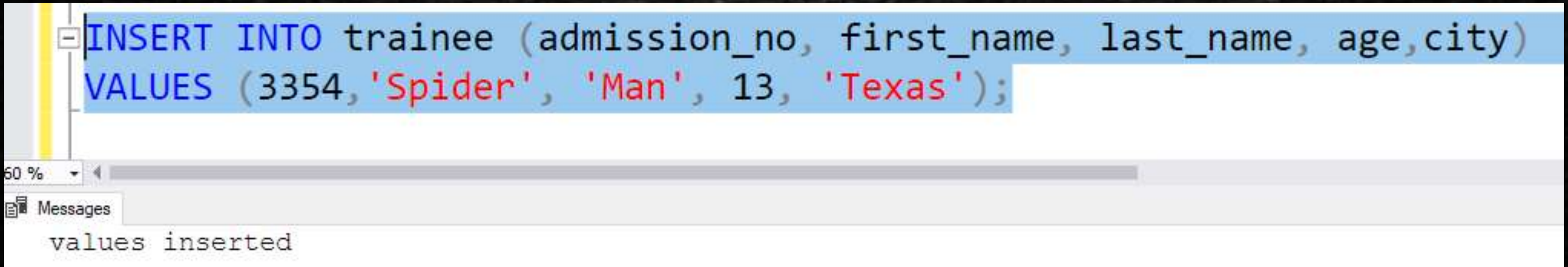
DML : AFTER Trigger

- AFTER triggers are executed after the action of an INSERT, UPDATE, or DELETE statement.

```
CREATE TRIGGER my_private_table2
ON trainee
AFTER INSERT
AS
BEGIN
    PRINT 'values inserted'
END
```

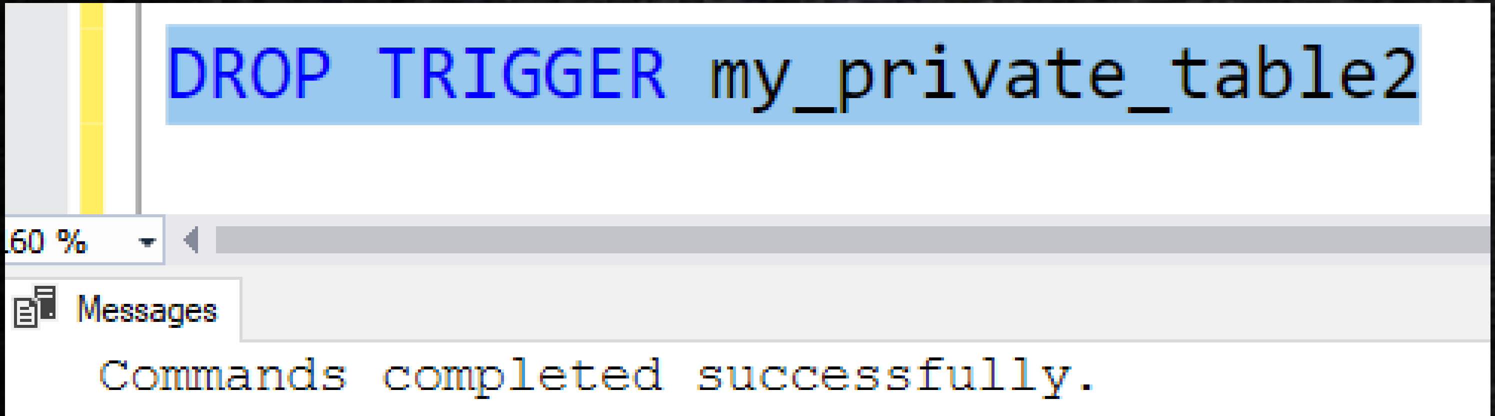
DML : AFTER Trigger

- After an insert, update or delete of any row in the table, the following message appears



DML: AFTER Trigger Deletion

```
DROP TRIGGER my_private_table2
```



DML : AFTER Trigger : Automatic value insertion example

- Eg: After the AFTER triggers are executed after the action of INSERT, we are trying to automatically save a copy of values to the backup table.

```
CREATE TRIGGER my_private_table2
ON trainee
AFTER INSERT
AS
```

DML : AFTER Trigger : Automatic value insertion example

```
BEGIN
    SET NOCOUNT ON; --do not show the number of affected rows

    DECLARE @id INT
    DECLARE @admission_no INT
    DECLARE @age INT
    DECLARE @first_name VARCHAR(45)
    DECLARE @last_name VARCHAR(45)
    DECLARE @city VARCHAR(45)
```

DML : AFTER Trigger : Automatic value insertion example

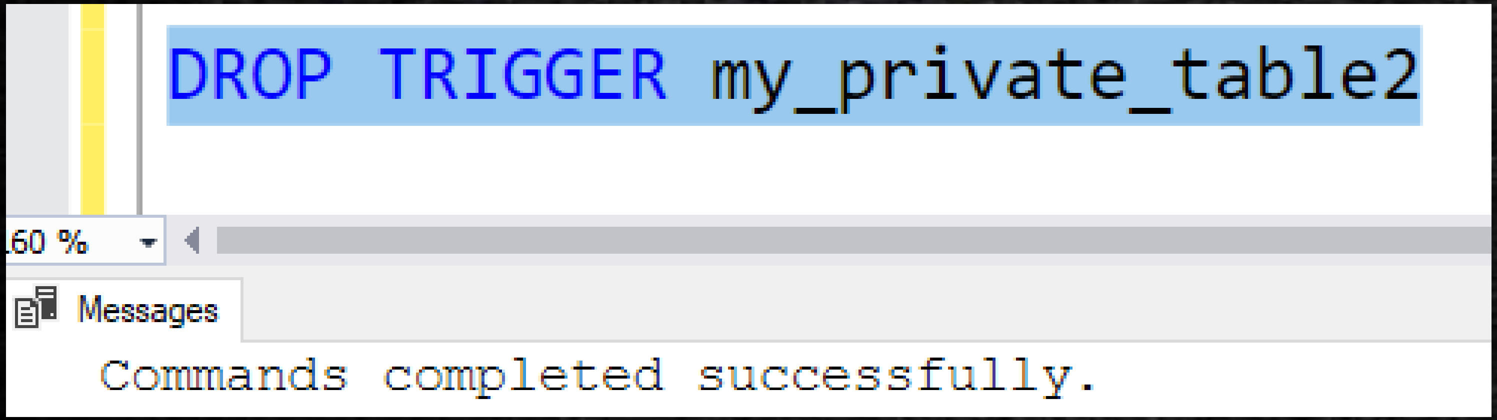
```
SELECT @id = I.id,
@admission_no = I.admission_no,
@first_name = I.first_name,
@last_name = I.last_name,
@age =I.age,
@city =I.city
FROM INSERTED I
```

DML : AFTER Trigger : Automatic value insertion example

```
    INSERT INTO trainee_backup
    VALUES(@id,@admission_no, @first_name,
    @last_name, @age, @city)

    PRINT 'values inserted in trainee and
    backup tables'
END

SELECT * FROM trainee
SELECT * FROM trainee_backup
```

DML: AFTER Trigger Deletion

```
DROP TRIGGER my_private_table2
```



DML : INSTEAD OF Trigger

- The database engine will execute only the trigger instead of executing the statement.

```
CREATE TRIGGER my_private_table2
ON trainee
INSTEAD OF INSERT
AS
BEGIN
    PRINT 'values not inserted'
END
```

DML : INSTEAD OF Trigger

- When attempted to insert, the following message appears without executing the insert

```
INSERT INTO trainee (admission_no, first_name, last_name, age,city)
VALUES (3354,'Spider', 'Man', 13, 'Texas');
```

Messages

```
values not inserted
```

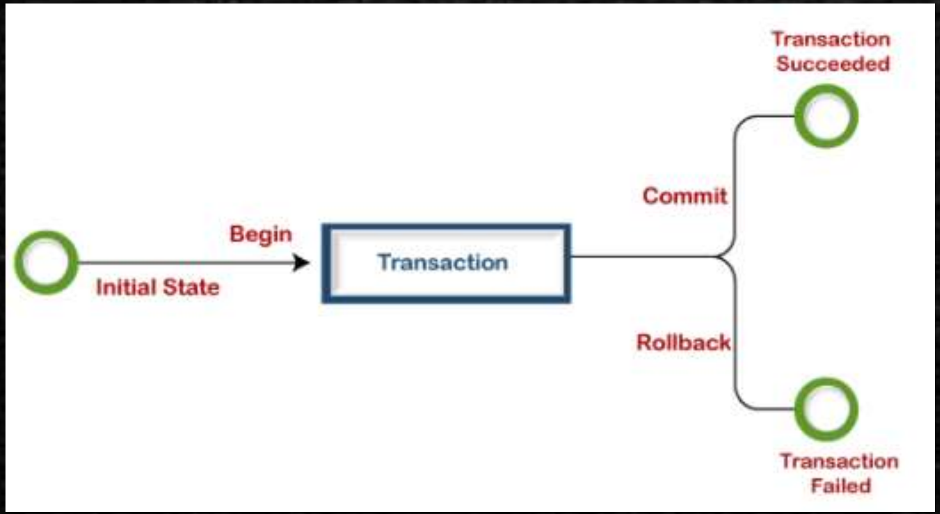
SQL SERVER TRANSACTIONS

Sequential queries

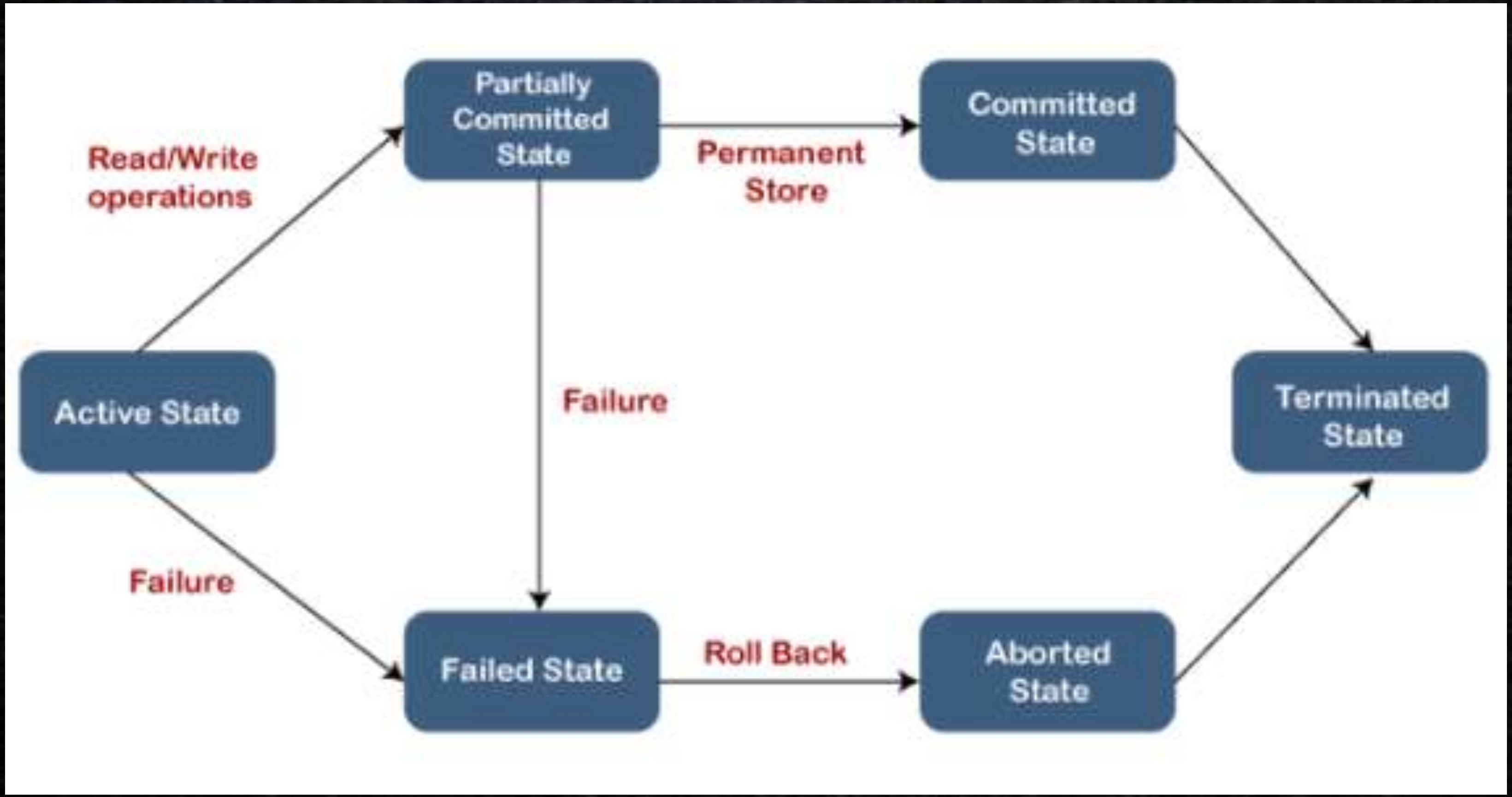
SQL SERVER TRANSACTIONS

- A transaction in SQL Server is a sequential DML statements or queries to perform single or multiple tasks.
- For Each transaction one of these two things should happen:

1. All modifications were successful and transaction is committed.
2. OR, all modifications are undone and transaction is in rollback.



SQL SERVER TRANSACTION STATES



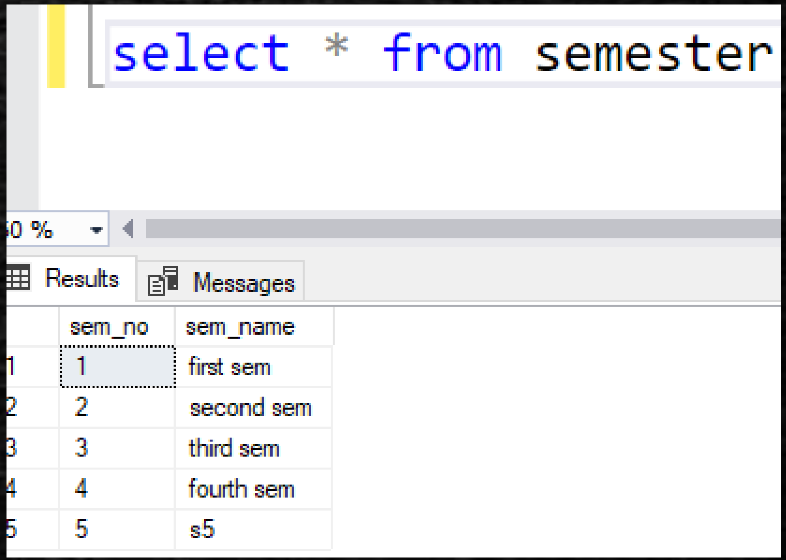
Transactions in A Bank ATM Scenario

- When a customer withdraws money using ATM, the steps happening in the background are :

- Check the availability of the requested amount in account.
- Deduct the amount from the account if there is sufficient balance and then updates the account balance.
- Log about the transaction is either successful or failed. If successful, modify in the database.
- Otherwise, if failed, the transaction will be rolled back into its previous state.

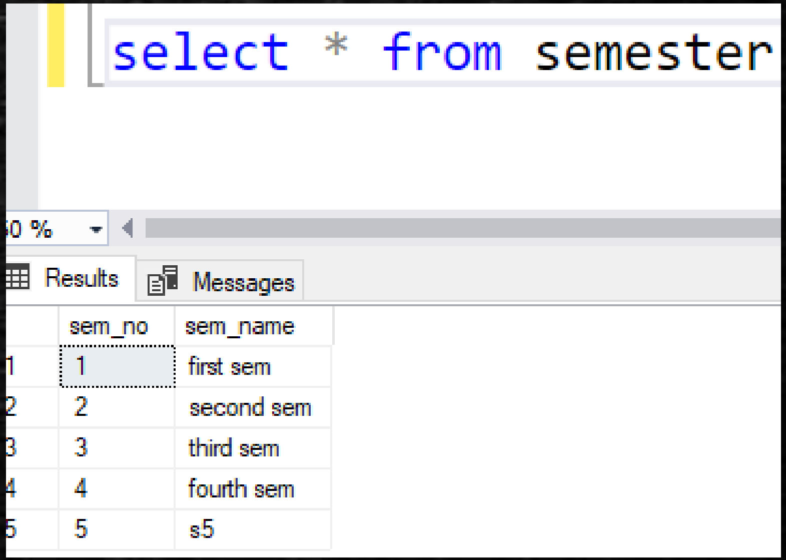
COMMIT of Transaction Example

```
-- Start a new transaction
BEGIN TRANSACTION

-- SQL Statements
 INSERT INTO semester (sem_no, sem_name) VALUES (5, 'sem 5')
 UPDATE semester SET sem_name = 's5' WHERE sem_no = 5

 -- Commit changes
COMMIT TRANSACTION
```



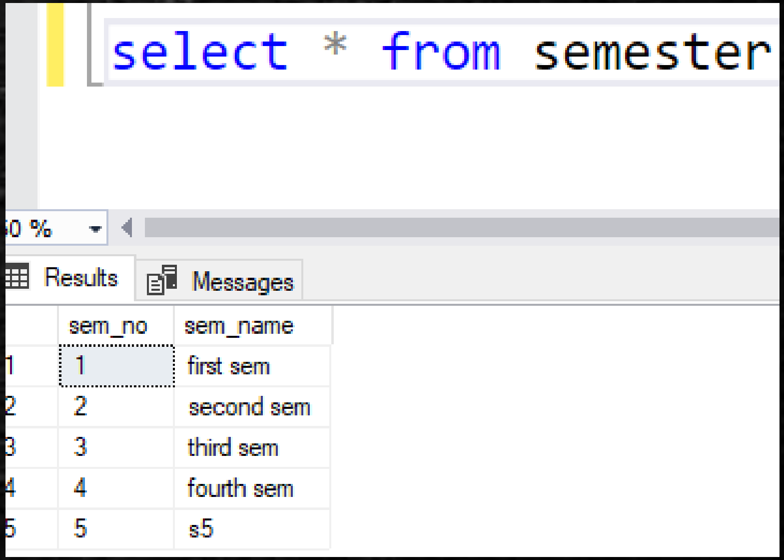
Manual ROLLBACK of Transaction Example

```
-- Start a new transaction
BEGIN TRANSACTION

-- SQL Statements
 INSERT INTO semester (sem_no, sem_name) VALUES (6, 'sem 6')
 UPDATE semester SET sem_name = 's6' WHERE sem_no = 6

--Undo Changes
ROLLBACK TRANSACTION
```



ROLLBACK on Transaction Error Example

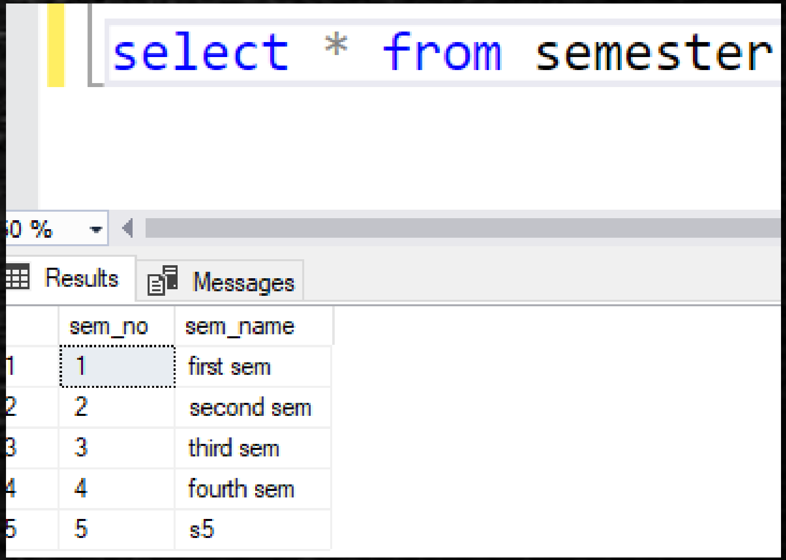
```
BEGIN TRANSACTION

 INSERT INTO semester (sem_no, sem_name) VALUES (6, 'sem 6')
 UPDATE semester SET sem_no = 'six' WHERE sem_no = 6

-- Check for error using system variable @@ERROR
IF(@@ERROR > 0)
BEGIN
    ROLLBACK TRANSACTION
END
ELSE
BEGIN
   COMMIT TRANSACTION
END
```



	sem_no	sem_name
1	1	first sem
2	2	second sem
3	3	third sem
4	4	fourth sem
5	5	s5

Automatic ROLLBACK of Transaction Example

```
-- Start a new transaction
BEGIN TRANSACTION

-- SQL Statements
 INSERT INTO semester (sem_no, sem_name) VALUES (6, 'sem 6')
 UPDATE semester SET sem_no = 'six' WHERE sem_no = 6

 -- Commit changes
COMMIT TRANSACTION
```



```
select * from semester
```

Results | Messages

	sem_no	sem_name
1	1	first sem
2	2	second sem
3	3	third sem
4	4	fourth sem
5	5	s5

NORMALIZATION

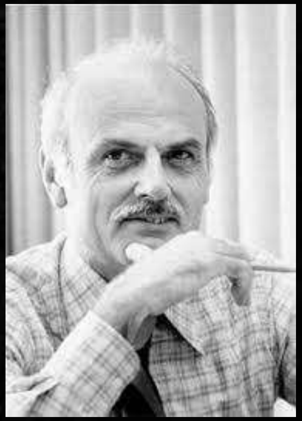
Keep everything in range and order

Normalization in DBMS

- Normalization is a db design technique that reduces data redundancy and eliminates Anomalies during operations.

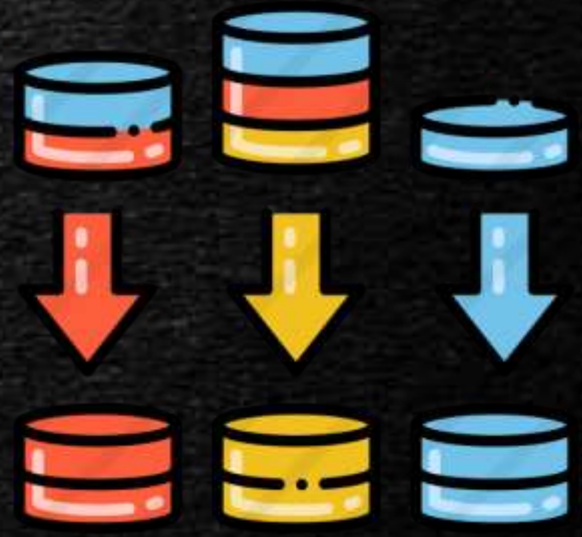
- Normalization rules divides larger tables into smaller tables and links them using relationships.

- The relational model inventor Edgar Codd proposed the theory of normalization of data with the introduction of the First Normal Form, Second and Third Normal Form.



List of Database Normal Forms

- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)
- BCNF (Boyce-Codd Normal Form)
- 4NF (Fourth Normal Form)
- 5NF (Fifth Normal Form)
- 6NF (Sixth Normal Form)



- In most practical applications, normalization achieves its best in 3rd Normal Form.

1NF (First Normal Form) Rules

- Every table cell should contain a single value.
- Every record needs to be unique.
- Here is an 1NF Table Example

Full Names	Physical Address	Movies rented	Salutation
Janet Jones	First Street Plot No 4	Pirates of the Caribbean	Ms.
Janet Jones	First Street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

2NF (Second Normal Form) Rules

- Rule 1- It should already be in 1NF
- Rule 2- Single Column Primary Key that is independent.

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

3NF (Third Normal Form) Rules

- Rule 1- It should already be in 2NF
- Rule 2- The tables has no transitive functional dependencies

Membership ID	Full names	Physical Address	Salutation ID
1	Janet Jones	First Street Plot No 4	2
2	Robert Phil	3rd Street 34	1
3	Robert Phil	5th Avenue	1

Membership ID	Movies rented
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

Salutation ID	Salutation
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

BACKUP AND RESTORE

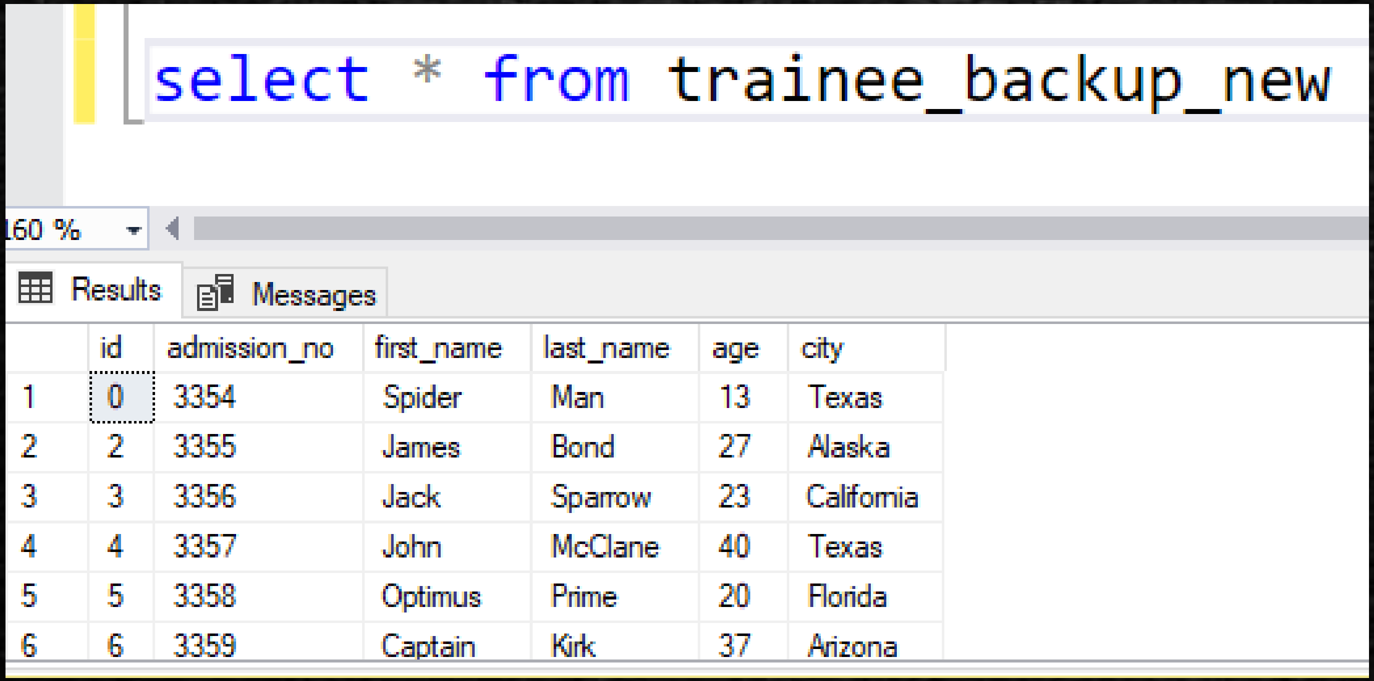
Backup and Restore Database

Create copy of table using SELECT INTO

```
--SYNTAX TO COPY ALL CONTENT OF TABLE
SELECT *
INTO table2 [IN other_dbname]
FROM table1
WHERE condition;


--SYNTAX TO COPY ONLY SPECIFIC COLUMNS OF TABLE
SELECT *
INTO table2 [IN other_dbname]
FROM table1
WHERE condition;
```

INSERT INTO example

```
SELECT *
INTO trainee_backup_new
FROM trainee_backup
```



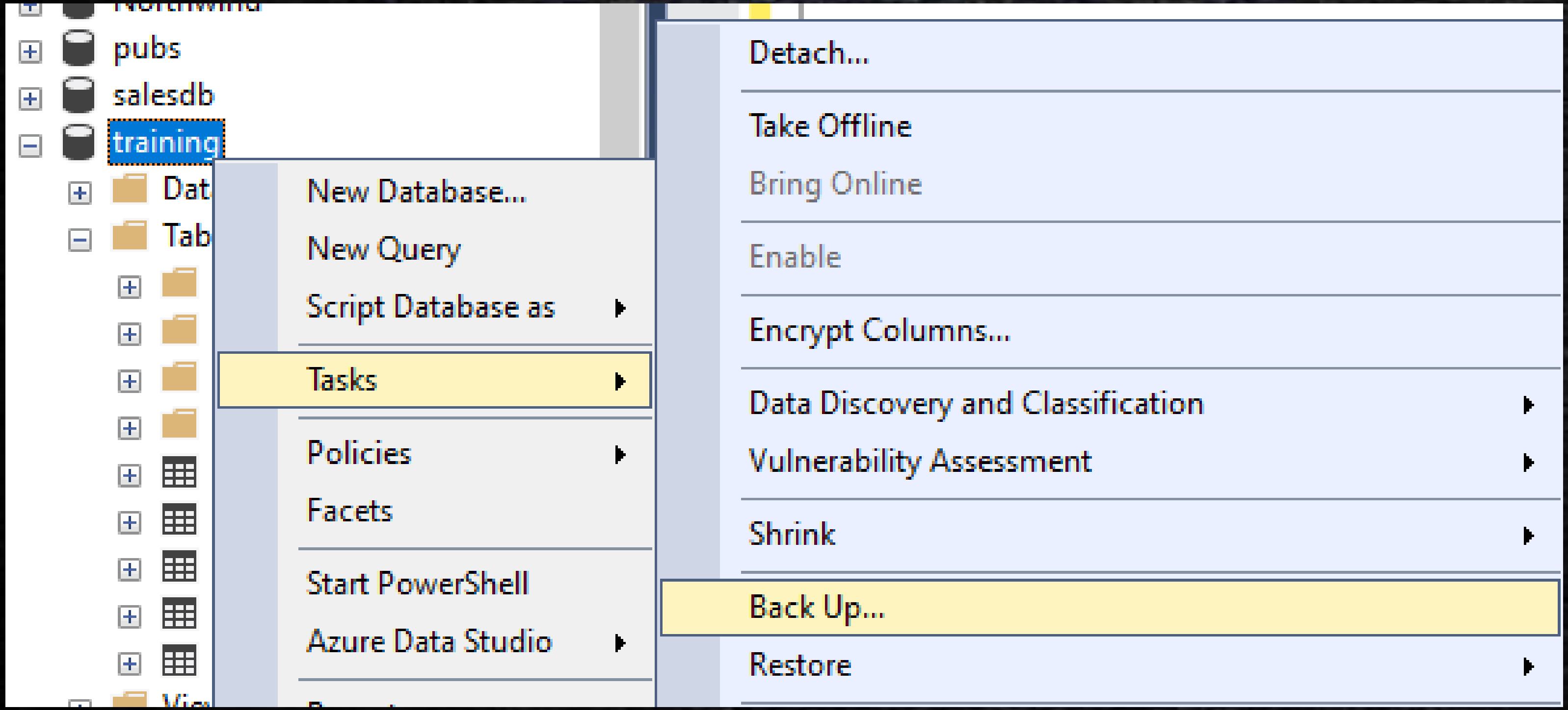
```
select * from trainee_backup_new
```

160 %

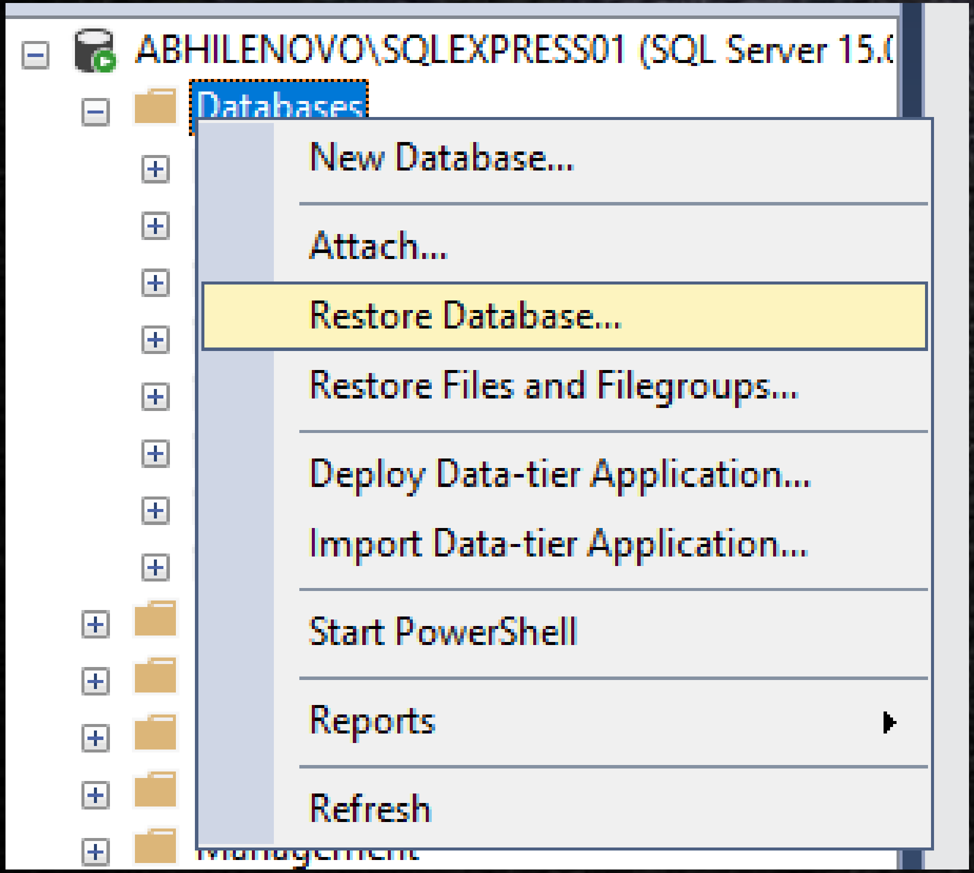
Results | Messages

	id	admission_no	first_name	last_name	age	city
1	0	3354	Spider	Man	13	Texas
2	2	3355	James	Bond	27	Alaska
3	3	3356	Jack	Sparrow	23	California
4	4	3357	John	McClane	40	Texas
5	5	3358	Optimus	Prime	20	Florida
6	6	3359	Captain	Kirk	37	Arizona

CREATE DATABASE BACKUP



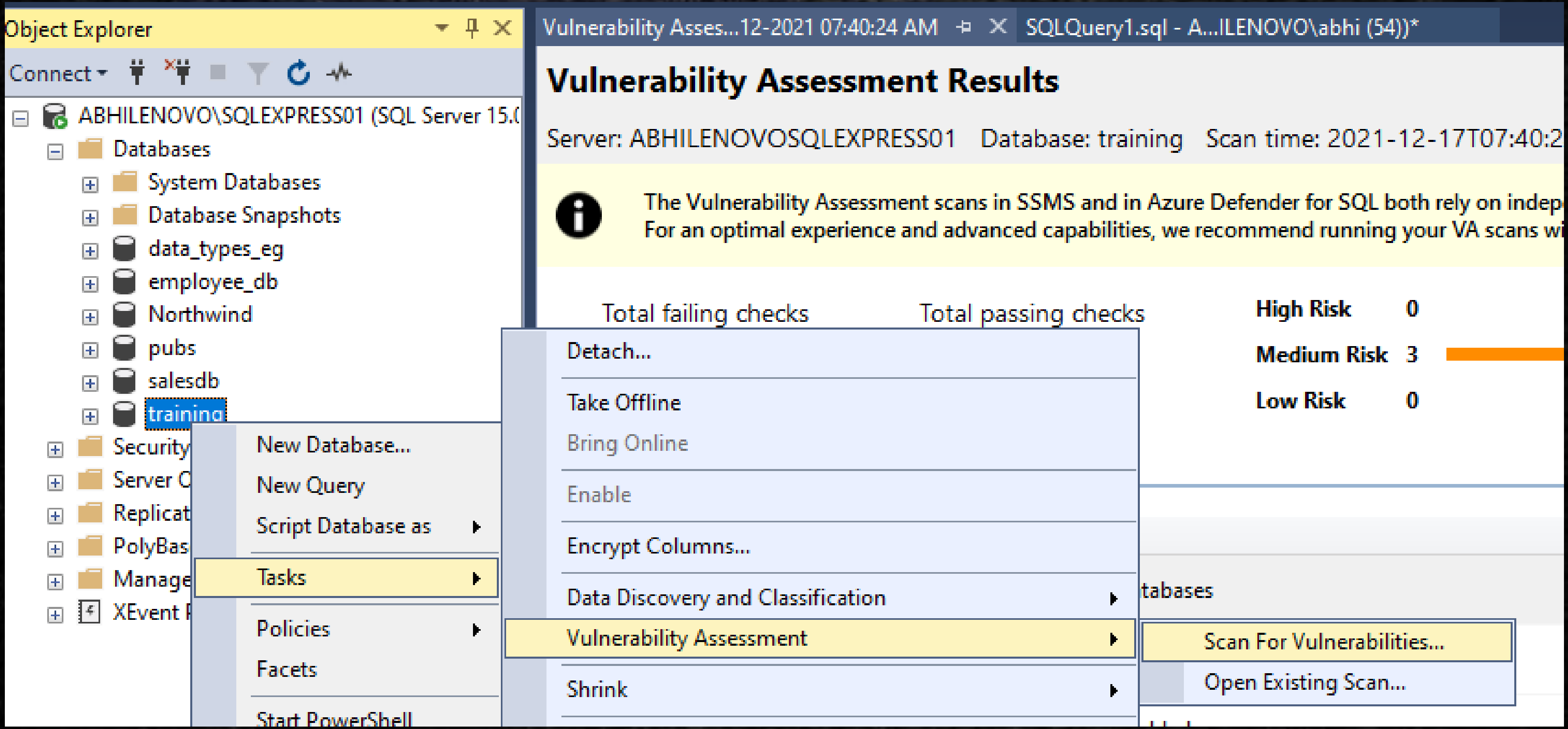
RESTORE DATABASE BACKUP



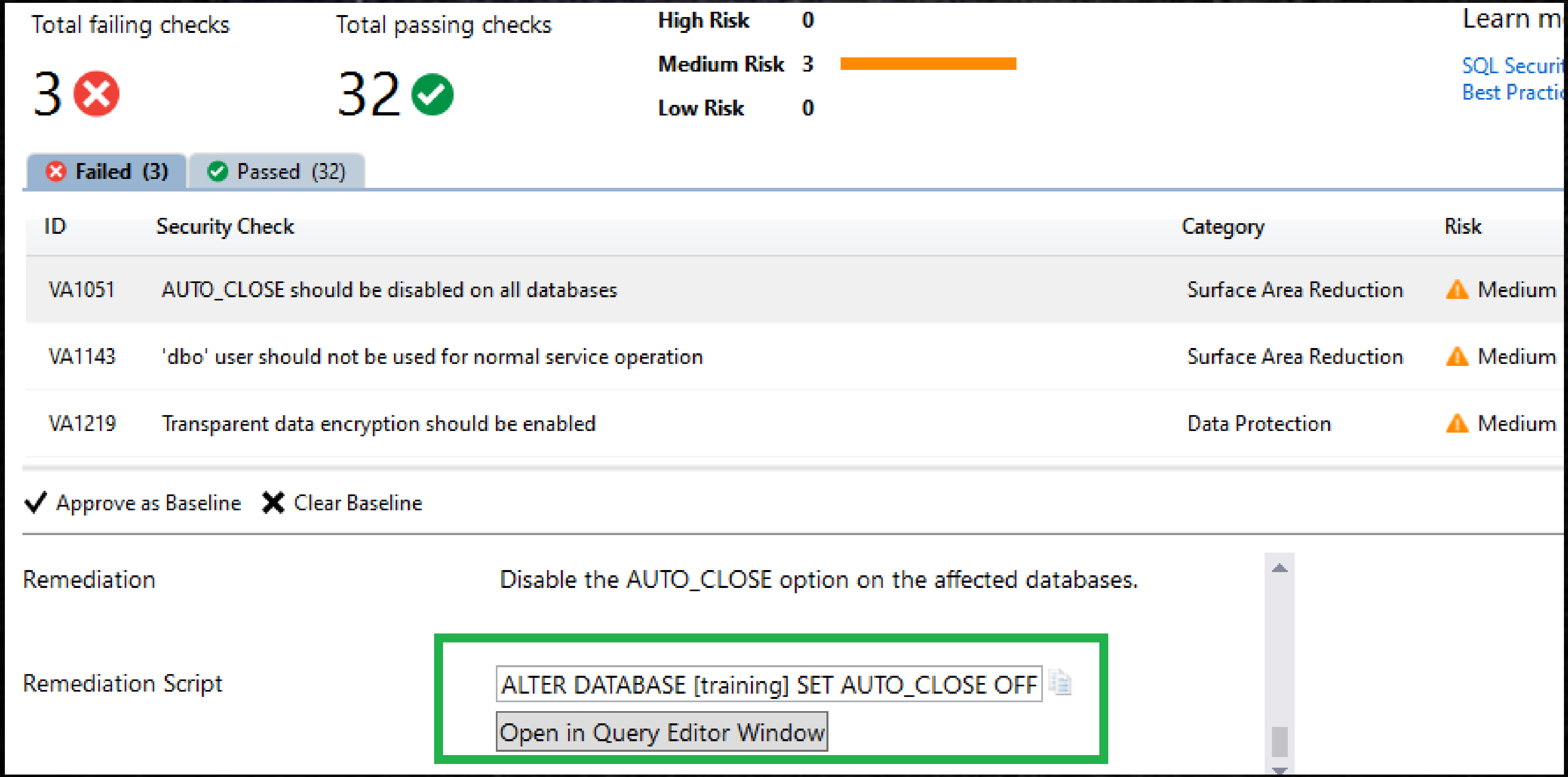
SECURING THE DATABASE

Assess the security risks and preventive measures

GETTING A VULNERABLITY ASSESMENT REPORT



IMPLEMENTING RECOMMENDED FIXES



SQL INJECTION

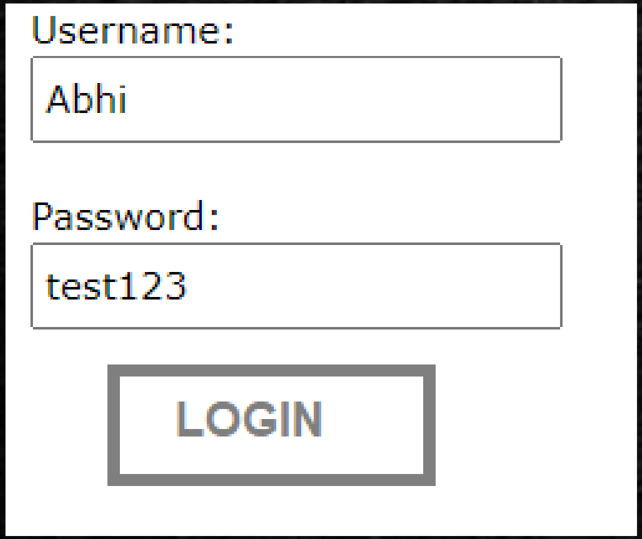
- SQL injection is a query injection hacking technique
- It is one of the most common web hacking techniques.
- SQL injection is the sending of malicious code in SQL statements, via web page input options.

SQL INJECTION Demo 1

A common example of SQL injection would be the Based on

""="" is Always True

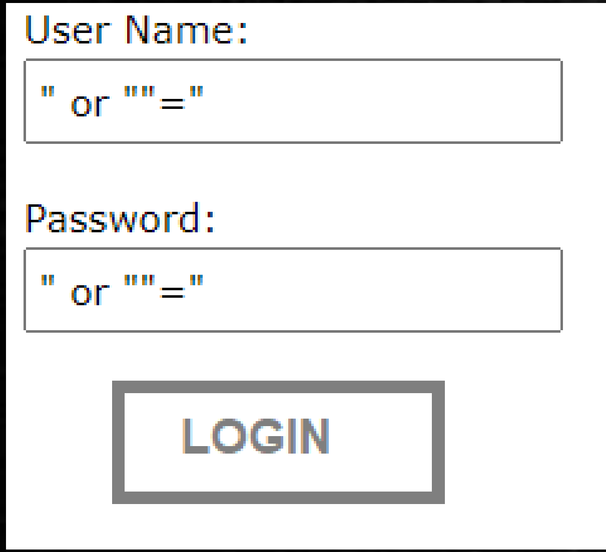
For example, A Web page may contain two text boxes for username and password




```
SELECT * FROM Users WHERE
username = 'abhi' and password = 'test123'
```

SQL INJECTION Demo 1

A hacker could pass " or ""=" string into the username and password fields, which will generate an always true query in the backend and will get the list of all users

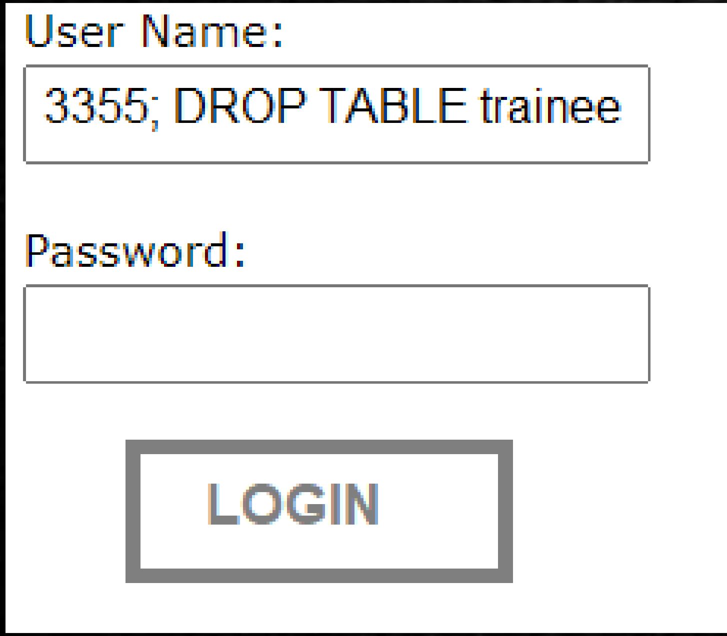



```
SELECT * FROM Users WHERE
username = "" or ""="“
and password = "" or ""=""
```

SQL INJECTION Demo 2

Another common example of SQL injection would be by creating a batched statement by attaching a small query using text boxes like..

DROP TABLE trainee




```
SELECT * FROM trainee WHERE
admission_no = '3355'; DROP TABLE trainee
```