

PROJECT TRAINING WORKSHOP

TECHNICAL .NET and SQL

UNDERSTANDING .NET

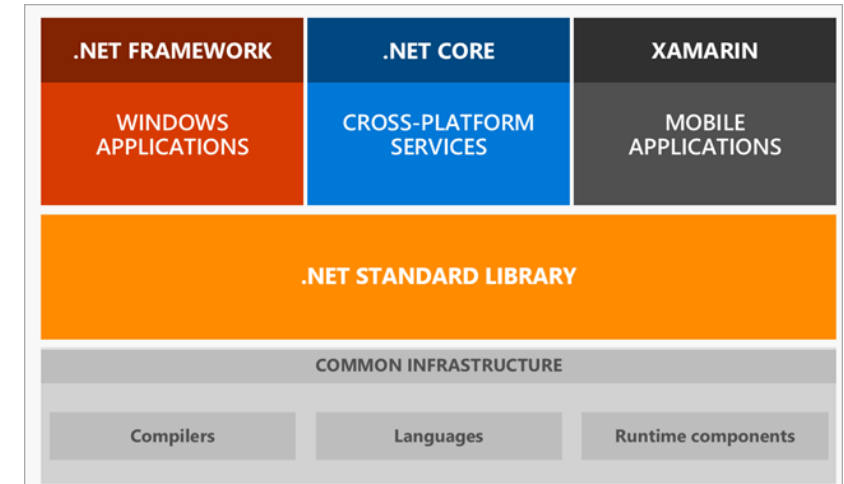
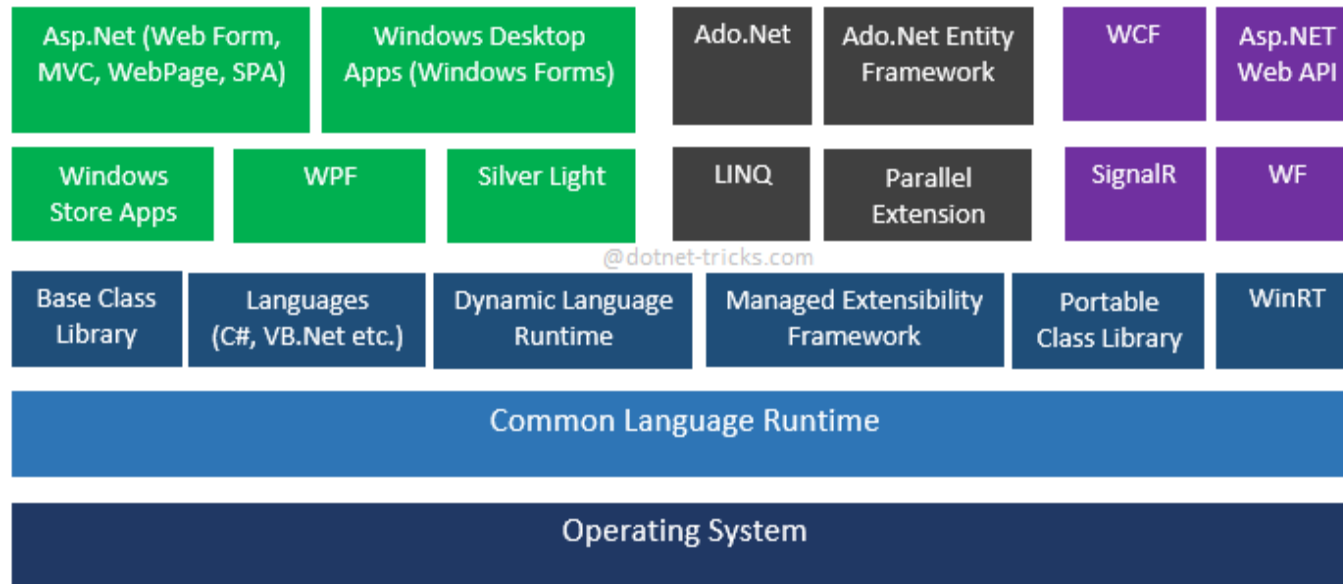
- What is this?



UNDERSTANDING .NET

- **What is .NET Framework?**

- **It is software development framework (tools, libraries, and runtime environments for building and running various types of applications)**



UNDERSTANDING .NET

- **What is role of .NET Framework?**

Cross-Platform Development	Language Versatility	Application Types	Common Language Runtime (CLR)
Base Class Libraries (BCL)	Integrated Development Environment (IDE)	ASP.NET and Web Development	Cloud and Microservices
Open Source	Performance		

- **What can be built with .NET?**

Web Application	Console Application	Windows Services	Windows Application
Internet Enabled Applications (Cloud)	Web Services	Extensions or Components	Libraries

COMPONENTS OF .NET

- **Common Language Runtime (CLR)**

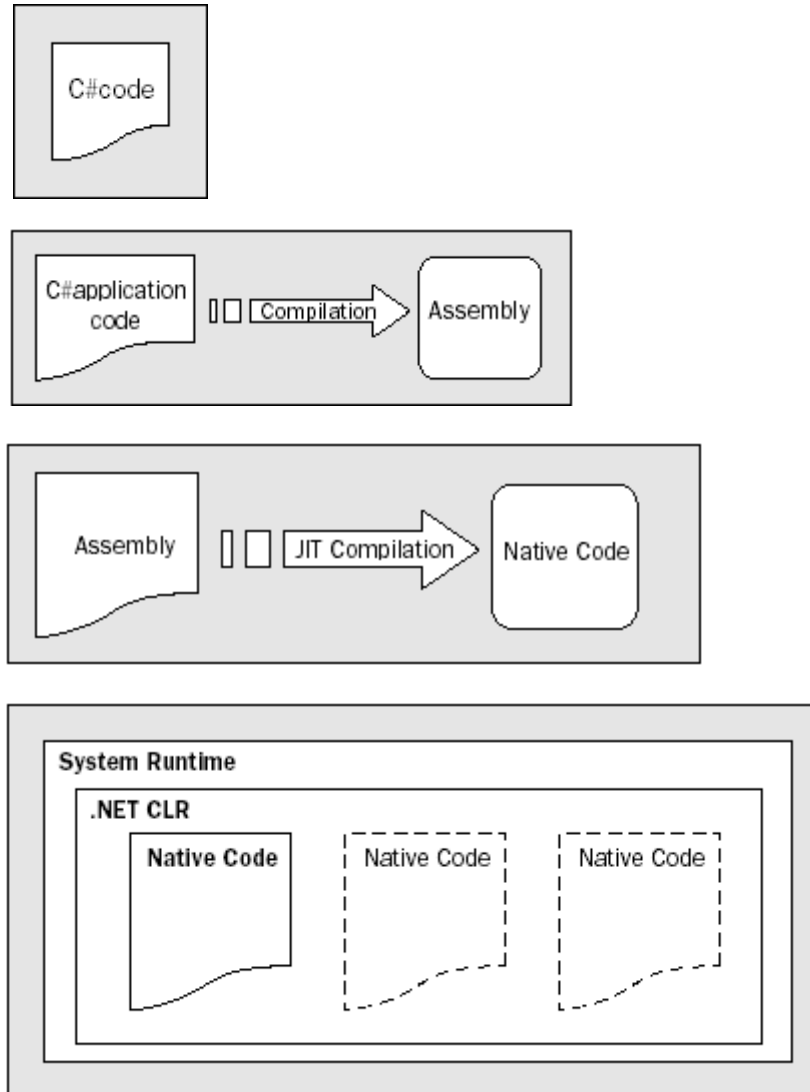
Garbage Collector	Code Manager	Common Type System	Common Language Specification
Class Loader			

- **Base Class Library (BCL)**

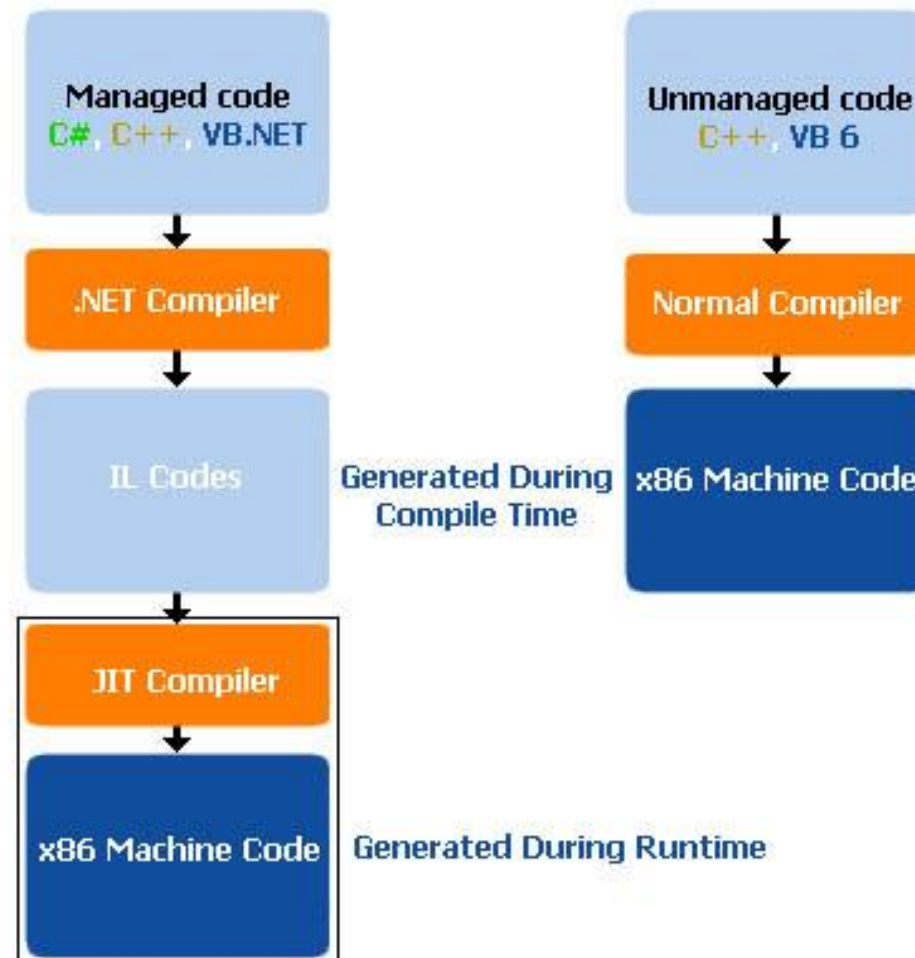
ADO.NET	Remoting	Reflection	Diagnostics
Threading	I/O		

- **Portable Class Library (PCL) , Dynamic Language Runtime (DLR) , Language Integrated Query (LINQ)**
- **Entity Framework (EF), Parallel Extention, Workflow Foundation (WF), Windows Communication Foundation (WCF)**

CODE EXECUTION PROCESS



MANAGED vs. UNMANAGED EXECUTION



UNDERSTANDING CLASS

SqlConnection Class

.NET Framework 4.6 and 4.5 | [Other Versions](#)

Namespace: [System.Data.SqlClient](#)
Assembly: [System.Data](#) (in [System.Data.dll](#))

Inheritance Hierarchy

[System.Object](#)
[System.MarshalByRefObject](#)
[System.ComponentModel.Component](#)
[System.Data.Common.DbConnection](#)
[System.Data.SqlClient.SqlConnection](#)

Syntax

C#




C++

F#





VB

```
public sealed class SqlConnection : DbConnection, ICloneable
```




Constructors

	Name	Description
	SqlConnection()	Initializes a new instance of the SqlConnection class.
	SqlConnection(String)	Initializes a new instance of the SqlConnection class when given a string that contains the connection string.
	SqlConnection(String, SqlCredential)	Initializes a new instance of the SqlConnection class given a connection string, that does not use Integrated Security = true and a SqlCredential object that contains the user ID and password.




Properties

	Name	Description
	AccessToken	Gets or sets the access token for the connection.
	ClientConnectionId	The connection ID of the most recent connection attempt, regardless of whether the attempt succeeded or failed.
	ColumnEncryptionTrustedMasterKeyPaths	Allows you to set a list of trusted key paths for a database server. If while processing an application query the driver receives a key path that is not on the list, the query will fail. This property provides additional protection against security attacks that involve a compromised SQL Server providing fake key paths, which may lead to leaking key store credentials.
	ConnectionString	Gets or sets the string used to open a SQL Server database. (Overrides DbConnection.ConnectionString .)



Methods

	Name	Description
	BeginTransaction()	Starts a database transaction.
	BeginTransaction(IsolationLevel)	Starts a database transaction with the specified isolation level.
	BeginTransaction(IsolationLevel, String)	Starts a database transaction with the specified isolation level and transaction name.

Events

	Name	Description
	Disposed	Occurs when the component is disposed by a call to the Dispose method. (Inherited from Component .)
	InfoMessage	Occurs when SQL Server returns a warning or informational message.
	StateChange	Occurs when the state of the event changes. (Inherited from DbConnection .)

Explicit Interface Implementations

	Name	Description
	IDbConnection.BeginTransaction()	Begins a database transaction. (Inherited from DbConnection .)
	IDbConnection.BeginTransaction(IsolationLevel)	Begins a database transaction with the specified IsolationLevel value. (Inherited from DbConnection .)

Remarks

A [SqlConnection](#) object represents a unique session to a SQL Server data source. With a client/server database system, it is equivalent to a network connection to the server. [SqlConnection](#) is used together with [SqlDataAdapter](#) and [SqlCommand](#) to increase performance when connecting to a Microsoft SQL Server database. For all third-party SQL Server products, and other OLE DB-supported data sources, use [OleDbConnection](#).

When you create an instance of [SqlConnection](#), all properties are set to their initial values. For a list of these values, see the [SqlConnection](#) constructor.

See [ConnectionString](#) for a list of the keywords in a connection string.

If the [SqlConnection](#) goes out of scope, it won't be closed. Therefore, you must explicitly close the connection by calling **Close** or **Dispose**. **Close** and **Dispose** are functionally equivalent. If the connection pooling value **Pooling** is set to **true** or **yes**, the underlying connection is returned.

UNDERSTANDING C#

- **What are the key concepts of C#?**

Object-Oriented Programming (OOP)	Strongly Typed	C# Syntax	.NET Framework
Common Language Infrastructure (CLI)	Managed Code	Garbage Collection	Assemblies
Namespace	Exception Handling	Generics	Delegates and Events
LINQ (Language Integrated Query)	Properties and Indexers	Attributes	Interfaces
Inheritance and Polymorphism	Value Types and Reference Types	Nullable Types	Async/Await
Task Parallel Library	Collections		

UNDERSTANDING C# - SYNTAX, GARBAGE COLLECTOR

This is namespace

```
namespace WebTrainingRoom
{
    public class Training
    {
        public Training()
        {
        }
        public string Subject { get; set; }
        public string Location { get; set; }
        public string Trainer { get; set; }
        public void Schedule()
        {
        }
        public void SendInvoice()
        {
        }
    }
}
```

Class name

Constructor

Property name

Access modifier

Property type

Method name

Method type

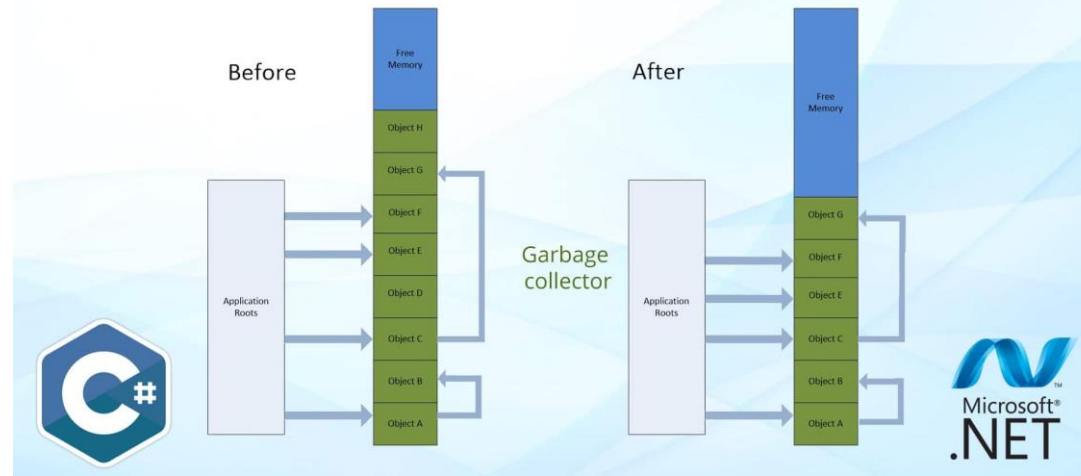
```
using System;
namespace Pragim
{
    class Sample
    {
        static void Main()
        {
            int[] Numbers = new int[3];

            Numbers[0] = 10;
            Numbers[1] = 20;
            Numbers[3] = "Name";
        }
    }
}
```

class System.String
Represents text as a series of Unicode characters.

Error:
Cannot implicitly convert type 'string' to 'int'

How does garbage collector work in .net c#



UNDERSTANDING C# - EXCEPTION HANDLING, GENERICS, EVENTS

```
public static void Main()
{
    try
    {
        LevelOne();
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine(ex);
    }
}

static void LevelOne()
{
    LevelTwo();
}

static void LevelTwo()
{
    LevelThree();
}

static void LevelThree()
{
    throw new ArgumentException("Exception in level three!");
}
```

Try-catch block found. Exception is handled.

No try-catch, going one level up.

No try-catch, going one level up.

An exception occurred. No try-catch, going up.

```
using System;

public class Program
{
    public static void Swap<T>(ref T a, ref T b)
    {
        T temp = a;
        a = b;
        b = temp;
    }

    public static void Main(string[] args)
    {
        int num1 = 5;
        int num2 = 10;

        Console.WriteLine($"Before Swap: num1 = {num1}, num2 = {num2}");

        Swap(ref num1, ref num2);

        Console.WriteLine($"After Swap: num1 = {num1}, num2 = {num2}");

        string str1 = "Hello";
        string str2 = "World";

        Console.WriteLine($"Before Swap: str1 = {str1}, str2 = {str2}");

        Swap(ref str1, ref str2);

        Console.WriteLine($"After Swap: str1 = {str1}, str2 = {str2}");
    }
}
```

```
class Publisher
{
    // Declare an event using EventHandler delegate
    public event EventHandler MyEvent;

    public void RaiseEvent()
    {
        MyEvent?.Invoke(this, EventArgs.Empty);
    }
}

class Subscriber
{
    public void OnEvent(object sender, EventArgs e)
    {
        Console.WriteLine("Event raised!");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Publisher publisher = new Publisher();
        Subscriber subscriber = new Subscriber();

        // Subscribe to the event
        publisher.MyEvent += subscriber.OnEvent;

        // Raise the event
        publisher.RaiseEvent();
    }
}
```

UNDERSTANDING C# - LINQ, INTERFACE, INDEXERS

```
class Program
{
    static void Main(string[] args)
    {
        // Sample data: a list of Person objects
        List<Person> people = new List<Person>
        {
            new Person { Name = "Alice", Age = 28 },
            new Person { Name = "Bob", Age = 35 },
            new Person { Name = "Charlie", Age = 22 },
            new Person { Name = "David", Age = 30 },
            new Person { Name = "Eve", Age = 25 }
        };

        // Using LINQ to query the list of people
        var youngAdults = from person in people
                           where person.Age >= 18 && person.Age <= 30
                           select person;

        // Display the result
        Console.WriteLine("Young adults (age 18-30):");
        foreach (var person in youngAdults)
        {
            Console.WriteLine($"{person.Name}, Age: {person.Age}");
        }
    }
}

class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

```
class Program
{
    class Team
    {
        private string[] players = new string[11];

        // Indexer to access players by jersey number
        public string this[int jerseyNumber]
        {
            get => players[jerseyNumber - 1];
            set => players[jerseyNumber - 1] = value;
        }

        // Property for the team name
        public string TeamName { get; set; }
    }

    static void Main()
    {
        Team soccerTeam = new Team();

        // Setting the team name using the property
        soccerTeam.TeamName = "Reds";

        // Using the indexer to assign player names
        soccerTeam[7] = "David Beckham";
        soccerTeam[10] = "Lionel Messi";

        // Accessing the team name and player names
        Console.WriteLine($"Team Name: {soccerTeam.TeamName}");
        Console.WriteLine($"Player 7: {soccerTeam[7]}");
        Console.WriteLine($"Player 10: {soccerTeam[10]}");
    }
}
```

```
interface IShape
{
    double CalculateArea();
}

class Circle : IShape
{
    public double Radius { get; set; }

    public Circle(double radius)
    {
        Radius = radius;
    }

    public double CalculateArea()
    {
        return Math.PI * Radius * Radius;
    }
}

class Rectangle : IShape
{
    public double Width { get; set; }
    public double Height { get; set; }

    public Rectangle(double width, double height)
    {
        Width = width;
        Height = height;
    }

    public double CalculateArea()
    {
        return Width * Height;
    }
}

class Program
{
    static void Main()
    {
        IShape circle = new Circle(5);
        IShape rectangle = new Rectangle(4, 8);

        Console.WriteLine($"Circle Area: {circle.CalculateArea()}");
        Console.WriteLine($"Rectangle Area: {rectangle.CalculateArea()}");
    }
}
```



UNDERSTANDING C# - Parallel Programming

```
class Program
{
    0 references
    static async Task Main()
    {
        Console.WriteLine("Start of the program.");

        // Simulate an asynchronous operation
        int result = await PerformAsyncOperation();

        Console.WriteLine($"Result of the async operation: {result}");

        Console.WriteLine("End of the program.");
    }

    1 reference
    static async Task<int> PerformAsyncOperation()
    {
        Console.WriteLine("Start of PerformAsyncOperation.");

        // Simulate a delay of 2 seconds asynchronously
        await Task.Delay(12000);

        Console.WriteLine("End of PerformAsyncOperation.");

        // Return a result
        return 42;
    }
}
```

```
class Program
{
    static void Main()
    {
        // Value type example
        int value1 = 5;
        int value2 = value1; // Copy the value of 'value1' to 'value2'
        value2 = 10;         // Modify 'value2'

        Console.WriteLine("Value type example:");
        Console.WriteLine($"value1: {value1}"); // Output: value1: 5
        Console.WriteLine($"value2: {value2}"); // Output: value2: 10

        // Reference type example
        MyClass obj1 = new MyClass(1);
        MyClass obj2 = obj1; // 'obj2' references the same object as obj1
        obj2.Value = 10;     // Modify 'obj2'

        Console.WriteLine("\nReference type example:");
        Console.WriteLine($"obj1.Value: {obj1.Value}"); // Output: 1
        Console.WriteLine($"obj2.Value: {obj2.Value}"); // Output: 10
    }
}

class MyClass
{
    public int Value { get; set; }

    public MyClass(int value)
    {
        Value = value;
    }
}
```

```
class Program
{
    static async Task Main()
    {
        Console.WriteLine("Start of the program.");

        // Start multiple tasks in parallel
        Task task1 = Task.Run(() => DoWork(1));
        Task task2 = Task.Run(() => DoWork(2));

        await Task.WhenAll(task1, task2); // Wait for both tasks to complete

        Console.WriteLine("End of the program.");
    }

    static void DoWork(int id)
    {
        Console.WriteLine($"Task {id} is working...");
        Task.Delay(2000).Wait(); // Simulate work
        Console.WriteLine($"Task {id} completed.");
    }
}
```

UNDERSTANDING C# - COLLECTIONS

```
class Program
{
    static void Main()
    {
        List<string> names = new List<string>();

        // Adding elements to the list
        names.Add("Alice");
        names.Add("Bob");
        names.Add("Charlie");

        // Iterating through the list
        foreach (string name in names)
        {
            Console.WriteLine(name);
        }

        // Accessing elements by index
        Console.WriteLine("First name: " + names[0]);

        // Removing an element
        names.Remove("Bob");
        Console.WriteLine("After removing 'Bob':");
        foreach (string name in names)
        {
            Console.WriteLine(name);
        }
    }
}
```

```
class Program
{
    static void Main()
    {
        Dictionary<string, int> ages = new Dictionary<string, int>();

        // Adding key-value pairs to the dictionary
        ages["Alice"] = 30;
        ages["Bob"] = 25;
        ages["Charlie"] = 35;

        // Accessing values by key
        Console.WriteLine("Age of Alice: " + ages["Alice"]);

        // Iterating through the dictionary
        foreach (var pair in ages)
        {
            Console.WriteLine($"{pair.Key}: {pair.Value} years old");
        }

        // Checking if a key exists
        if (ages.ContainsKey("David"))
        {
            Console.WriteLine("David's age: " + ages["David"]);
        }
        else
        {
            Console.WriteLine("David's age not found.");
        }
    }
}
```

```
class Program
{
    static void Main()
    {
        HashSet<string> uniqueNames = new HashSet<string>();

        // Adding elements to the hash set
        uniqueNames.Add("Alice");
        uniqueNames.Add("Bob");
        uniqueNames.Add("Alice"); // Duplicate, won't be added

        // Iterating through the hash set
        foreach (string name in uniqueNames)
        {
            Console.WriteLine(name);
        }

        // Checking if an element exists
        bool containsBob = uniqueNames.Contains("Bob");
        Console.WriteLine($"Contains 'Bob': {containsBob}");
    }
}
```

TECHNICAL .NET

- **ASSESSMENT**
 - **Make a calculator program using Console/Windows/Web application as group**

PROJECT TRAINING WORKSHOP

DATABASE

UNDERSTANDING DATABASE

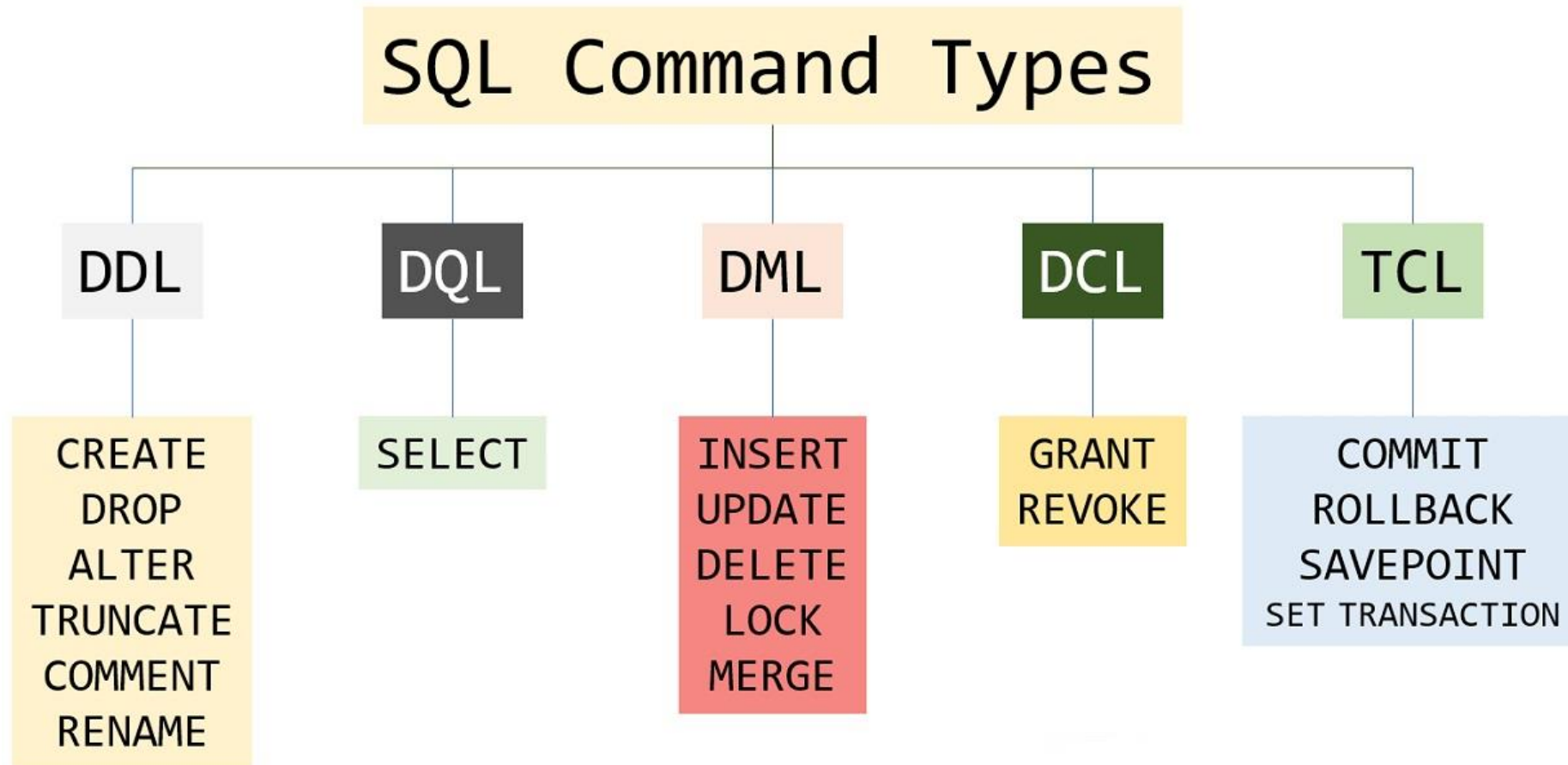
- **What is Database?**

- A database is a structured collection of data that is organized in a way that allows for efficient storage, retrieval, updating, and management of that data.

- **What are key concepts of Database?**

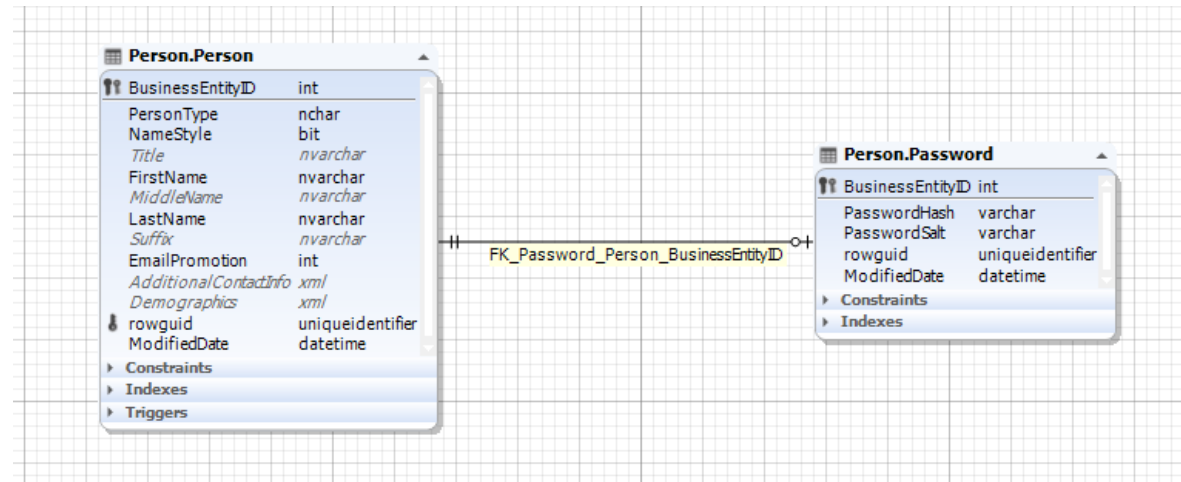
Data/ Data Types	Table	Row	Column
Schema	Queries (CRUD & Joins)	Indexes	Relationships (PK, FK, CK)
Constraints	ACID Properties Atomicity, Consistency, Isolation Durability	Normalization	Data Security
Backup and Recovery	Database Management System (DBMS)		

UNDERSTANDING QUERIES



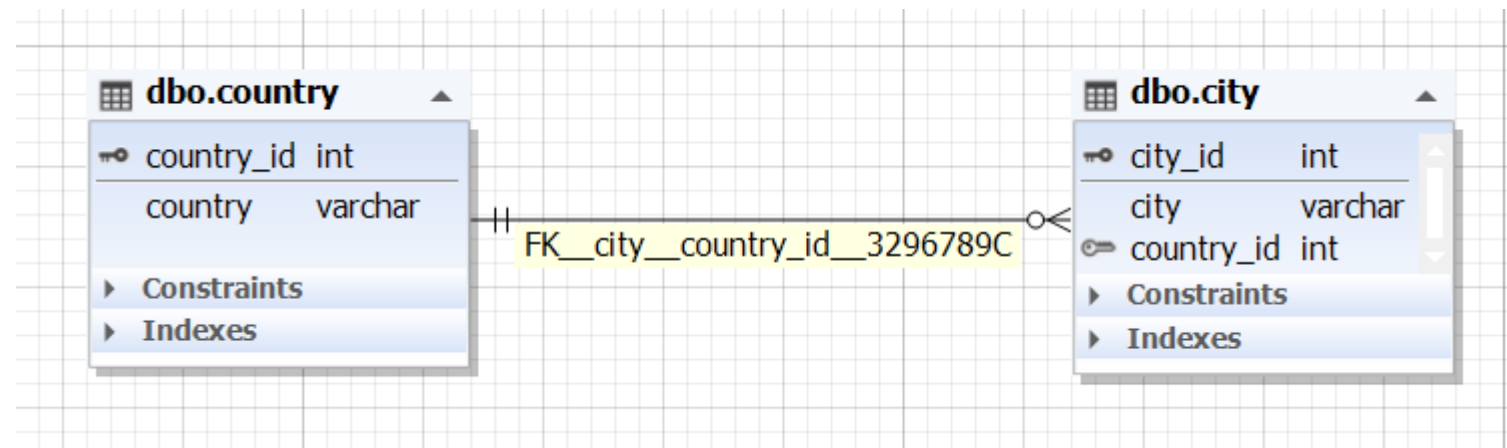
UNDERSTANDING RELATIONSHIPS (1-1)

A one-to-one relationship in a database occurs when each row in table 1 has only one related row in table 2



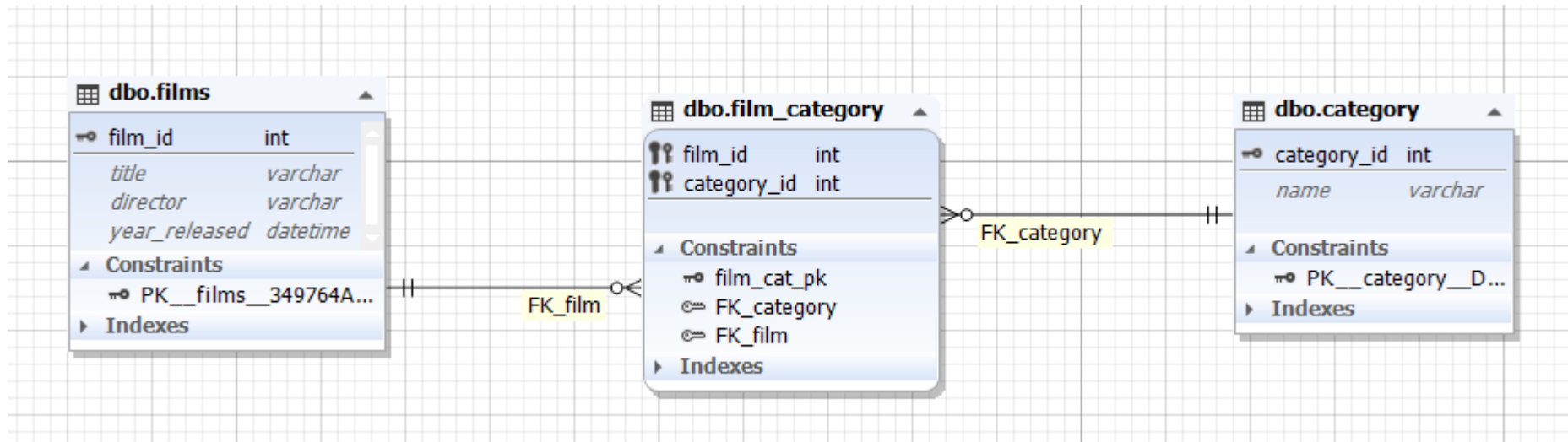
UNDERSTANDING RELATIONSHIPS (1-M)

A one-to-many relationship occurs when one record in table 1 is related to one or more records in table 2



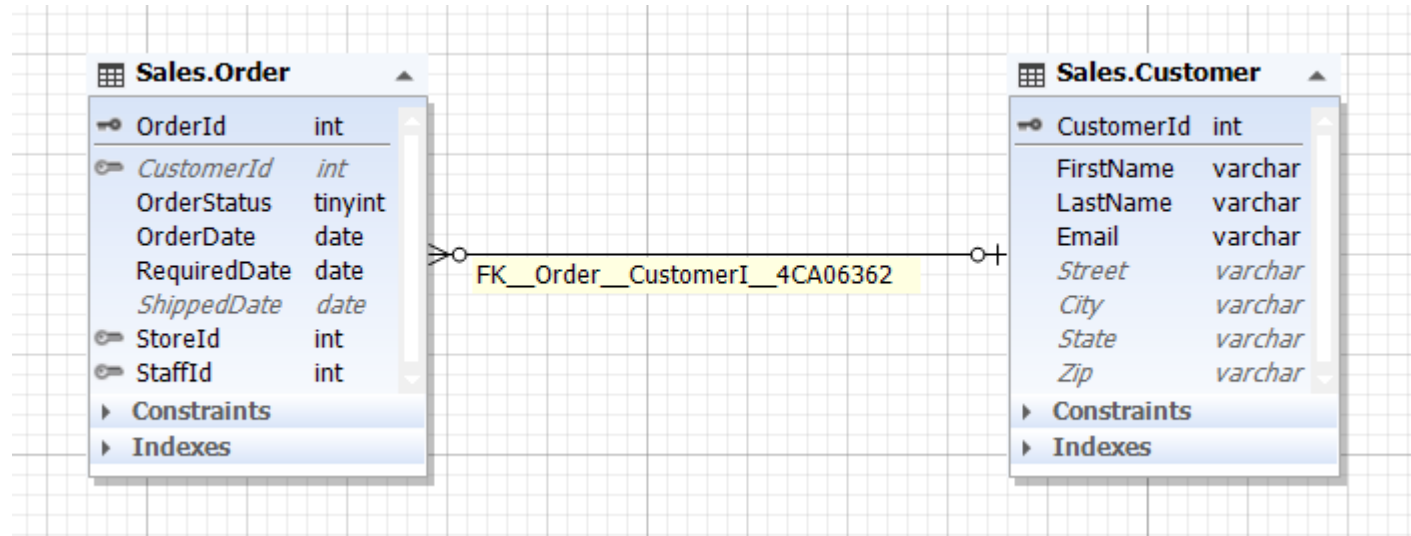
UNDERSTANDING RELATIONSHIPS (M-M)

A many-to-many relationship occurs when multiple records in one table are related to multiple records in another table.



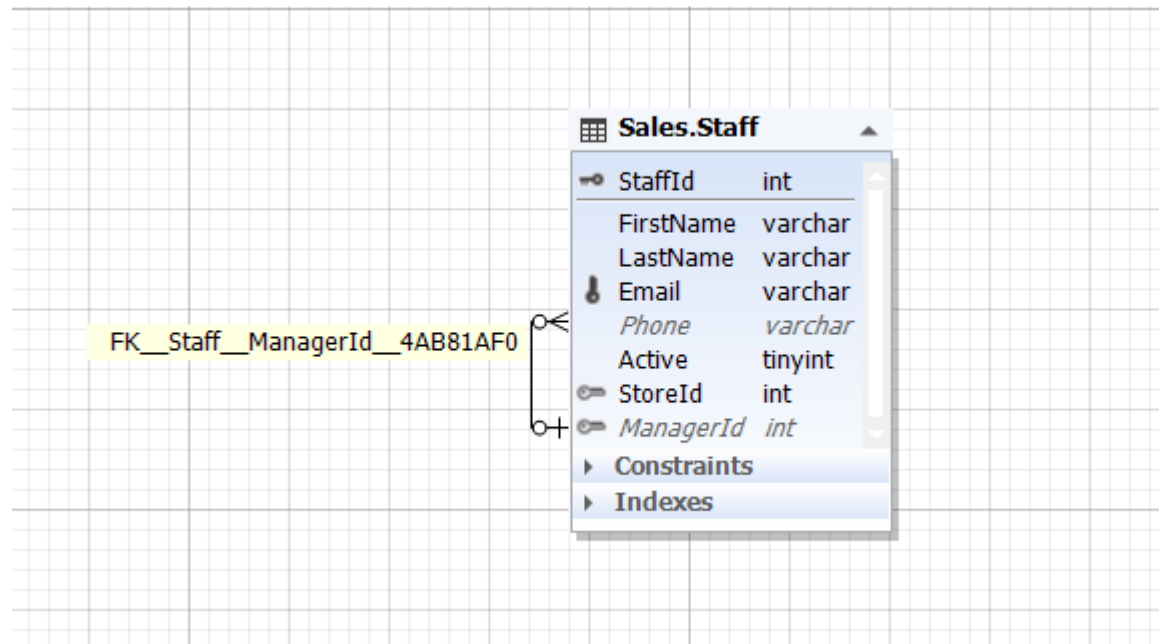
UNDERSTANDING RELATIONSHIPS (M-1)

There is not much difference between one-to-many and many-to-one relationships



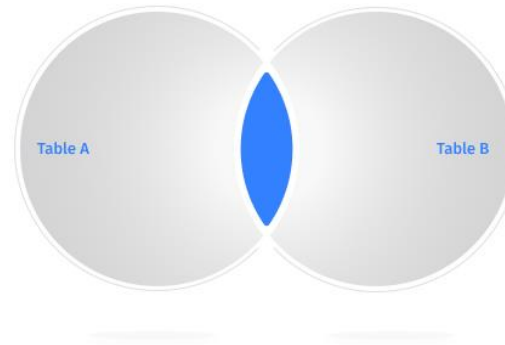
UNDERSTANDING RELATIONSHIPS (SELF)

A self-referencing relationship (also known as a recursive relationship) in a database occurs when a column in a table relates to another column in the same table



UNDERSTANDING JOIN (INNER)

INNER JOIN

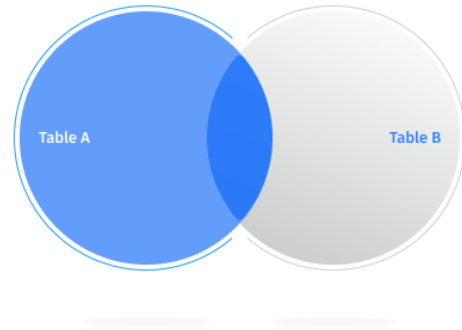


INNER JOINS are used to fetch only common matching records for which the join condition is met

```
SELECT columns  
FROM tableA  
INNER JOIN  
tableB  
ON tableA.column = tableB.column;
```


UNDERSTANDING JOIN (LEFT OUTER)

LEFT OUTER JOIN



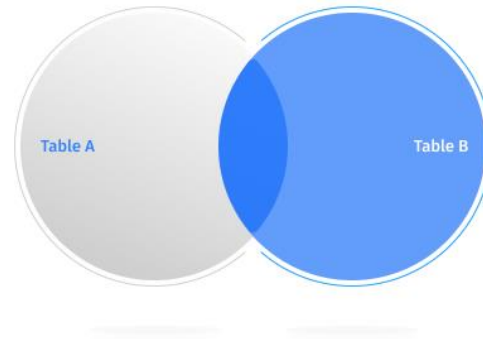
LEFT JOINS allow retrieving all records from Table A, along with those records from Table B for which the join condition is met

```
SELECT  
columns  
FROM  
tableA  
LEFT [OUTER] JOIN  
tableB  
ON tableA.column = tableB.column;
```

UNDERSTANDING JOIN (RIGHT OUTER)

RIGHT JOINS allow retrieving all records from Table B, along with those records from Table A for which the join condition is met

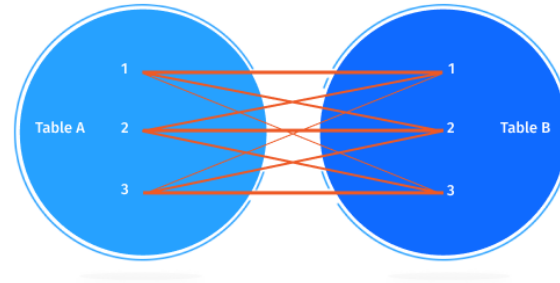
RIGHT OUTER JOIN



```
SELECT  
columns  
FROM  
tableA  
RIGHT [OUTER] JOIN  
tableB  
ON tableA.column = tableB.column;
```

UNDERSTANDING JOIN (CROSS)

CROSS JOIN

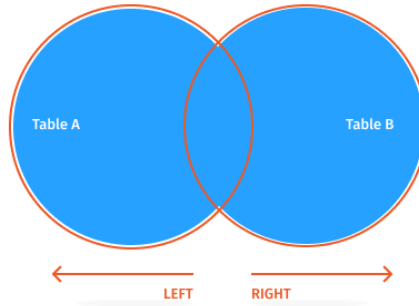


CROSS JOIN, also known as a cartesian join, retrieves all combinations of rows from each table

```
SELECT  
columns  
FROM  
tableA  
CROSS JOIN  
tableB;
```

UNDERSTANDING JOIN (FULL)

FULL JOIN



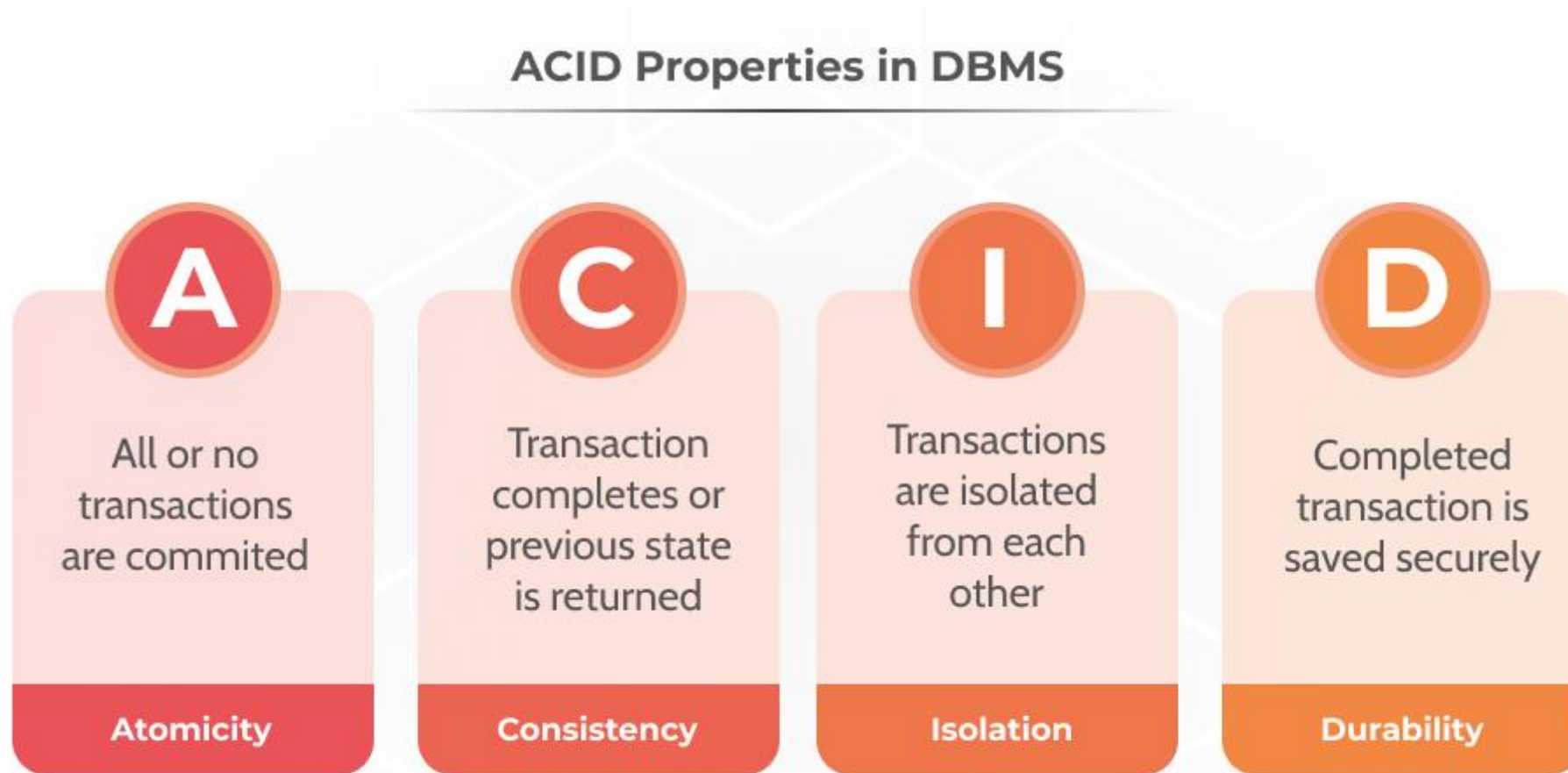
FULL JOIN returns all records when there is a match in left (tableA) or right (tableB) table records

```
SELECT * FROM tableA LEFT JOIN tableB ON  
tableA.id = tableB.id
```

UNION

```
SELECT * FROM tableA RIGHT JOIN tableB ON  
tableA.id = tableB.id
```

UNDERSTANDING ACID



UNDERSTANDING NORMALIZATION

1. Unnormalized table:

Student#	Advisor	Adv-Room	Class1	Class2	Class3
1022	Jones	412	101-07	143-01	159-02
4123	Smith	216	101-07	143-01	179-04

2. First normal form: No repeating groups

Student#	Advisor	Adv-Room	Class#
1022	Jones	412	101-07
1022	Jones	412	143-01
1022	Jones	412	159-02
4123	Smith	216	101-07
4123	Smith	216	143-01
4123	Smith	216	179-04

UNDERSTANDING NORMALIZATION

3. Second normal form: Eliminate redundant data

Students:

Student#	Advisor	Adv-Room
1022	Jones	412
4123	Smith	216

Registration:

Student#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	101-07
4123	143-01
4123	179-04

UNDERSTANDING NORMALIZATION

4. Third normal form: Eliminate data not dependent on key

Students:

Student#	Advisor
1022	Jones
4123	Smith

Faculty:

Name	Room	Dept
Jones	412	42
Smith	216	42

TECHNICAL .NET

- **ASSESSMENT**
 - **Create your own Database for any real time business as a group activity**