|  |  |  |
| --- | --- | --- |
| Access Modifiers and Keywords | | |
| Sr No. | Names | IMP Points |
| 1 | Public | 1.Allows access to the class, method, or variable from any other class or project. |
| 2 | Private | 1.Can only be accessed within the same class.  2. restrict access to other classes can’t access outside not even by derived classes. |
| 3 | Protected | 1.Can be accessed within the same class and its derived (child) classes.  2.Used in inheritance to allow child classes to reuse parent class functionality. |
| 4 | Internal | 1. Allows access to the member within the same assembly (project) but not from other assemblies.  2. Cannot be accessed from outside the assembly, even if the class is public. |
| 5 | Static | 1. Used to define members (methods, fields, properties) that belong to the class itself rather than an instance of the class.  2. Static cannot access non-static members directly.  3. Static fields and methods require a class reference to access. |
| 6 | Non-Static | 1. Requires an instance of the class to access.  2. Used when different objects need to maintain their own separate values. |
| 7 | Const | 1. Used to declare a constant value that cannot be changed.  2. Must be initialized at declaration. |
| 8 | Readonly | 1.Used to define variables that can only be assigned during declaration or inside a constructor.  2.Cannot be modified outside the constructor or declaration.  3.Can be assigned runtime values, unlike const.  4.Useful when values need initialization but should not change after object creation. |
| 9 | Var | 1. Automatically determines the data type at compile-time.  2. Must be initialized at declaration, and the type cannot be changed later. |
| 10 | Dynamic | 1. Allows variables to store any type and determines their type at runtime. |
| 11 | Params | 1. Allows a method to accept a variable number of arguments as an array. |
| 12 | Ref | 1. Passes arguments by reference instead of by value.  2. ref requires the variable to be initialized before passing  3. ref can use for modifying existing values. |
| 13 | Out | 1. Passes arguments by reference  2. out must be assigned a value within the method.  3. Used when a method needs to return multiple values. |
| 14 | Get and Set | 1. Used to encapsulate fields by providing controlled access.  2. get retrieves the value, while set assigns a new value.  3. Must be inside a property declaration. |
| 15 | Sealed | 1. Prevents a class from being inherited or a method from being overridden.  2. Used to restrict further modifications and ensure security of class.  3. Cannot be used with abstract classes or methods. |
| 16 | Singleton | 1. Ensures a class has only one instance and provides a global point of access to it.  2. Requires a private constructor and a static instance variable.  3. Ensure thread safety when implementing the Singleton pattern.  4. Used for managing shared resources. |
| 17 | Lock | 1. Ensures thread safety by allowing only one thread to access a block of code at a time.  2. Requires an object (reference type) as a lock token. |
| 18 | Interface | 1. Defines a contract that implementing classes must follow.  2. Cannot contain implementation details (only method).  3. A class can implement multiple interfaces, but only inherit from one class. |
| 19 | Virtual | 1. Virtual allows method overriding. Allows a method in a base class to be overridden in a derived class.  2.Virtual methods must have a base implementation.  3.Virtual is optional to override. |
| 20 | Abstract | 1. Defines a method or class that must be implemented in a derived class.  2. abstract methods can only exist in abstract classes.  3. Abstract classes cannot be instantiated. |
| 21 | Method Overloading | 1.Overloading allows multiple methods with the same name but different parameters in the same class.  2. Overloading needs a different parameter signature. (name and parameters)  3. Overloading is determined at compile time.  4.Also called asStatic Binding. |
| 22 | Method Overriding | 1. Allows a derived class to provide a new implementation for a method defined in the base class.  2. Overriding requires the method to be virtual or abstract in the base class.  3. Overriding is determined at run time.  4. Also called asDynamic Binding. |
| 23 | Async | 1. Async is a modifier that indicates a method is asynchronous.  2. Must be used with a method (async method).  3. Async alone does not make a method asynchronous; it requires await inside. |
| 24 | Await | 1. Await is an operator used inside an async method to pause execution until the awaited task completes.  2. It ensures that the method resumes only after the awaited operation finishes.  3. Can only be used inside an async method.  4. If await is not used inside an async method, it will run synchronously. |
| 25 | Try | 1. Used to define a block of code that might throw an exception. It ensures that the code inside it is monitored for errors.  2. Must be followed by at least one catch block or a finally block. |
| 26 | Catch | 1. Used to handle exceptions thrown in the try block. It specifies what to do when an exception occurs.  2. Must come after a try block. You can have multiple catch blocks to handle different types of exceptions. |
| 27 | Throw | 1. Used to explicitly throw an exception, either a new one or rethrow an existing one.  2. throw must be used inside a catch block or within a method.  3. |
| 28 | Finally | 1.  Used to define a block of code that will always execute, regardless of whether an exception occurs or not.  2. Must come after a try block or a catch block. |
| 29 | As | 1. Used for safe type conversion; returns null if conversion fails instead of throwing an exception.  2. Can only be used with referencetypes or nullable value types. |
| 30 | Is | 1. Used to check if an object is of a specific type. Returns true if the object is of the specified type, otherwise false.(used for type checking)  2. Works with all types (value types, reference types, and nullable types). |
| 31 | Continue | 1. Used in loops (for, while, do-while, foreach) to skip the current iteration and move to the next one.  2. Can only be used inside loops.  3. Used to skip unnecessary iterations without exiting the loop. |
| 32 | Break | 1. Used to exit a loop (for, while, do-while, foreach) or a switch statement immediately.  2. Can only be used inside loops or switch statements.  3. Used to terminate a loop or switch case when a specific condition is met. |
| 33 | For | 1.Used to iterate a block of code a specific number of times. It is ideal when you know the number of iterations in advance.  2. Requires initialization, condition, and iteration expression in its syntax. |
| 34 | While | 1. Used to iterate a block of code as long as a condition is true. It is ideal when the number of iterations is unknown.  2. Needs a valid boolean condition to check before each iteration. |
| 35 | Do-while | 1.  it guarantees at least one execution of the loop body, even if the condition is false initially.  then continues based on a condition.  2. The condition check happens after executing the loop body. |
| 36 | Foreach | 1. Used to iterate over collections (e.g., arrays, lists) without needing an index.  2. Cannot modify the collection directly during iteration. |
|  |  |  |
| 37 | Enum | 1. Used to define a set of named constants. It is a value type that represents a group of related values.  2. The underlying type must be an integral type (byte, int, long, etc.).  3. Each member of enum must have a unique name; values can be assigned explicitly. |
| 38 | Checked | 1. Used to enable overflow checking for arithmetic operations. If an overflow occurs, it throws an OverflowException.  2. Works only with integral types (int, long, etc.). |
| 39 | Unchecked | 1. Used to disable overflow checking for arithmetic operations. allowing overflow without errors.  2. Use unchecked only when you want to ignore overflow and allow default behaviour. |
| 40 | Base | 1. Calls base class constructors, methods, or properties in an inherited class.  2. Can only be used inside a derived class. |
| 41 | New | 1. Use new to create objects.  2. Cannot create instances of abstract classes or interfaces. |
| 42 | Namespace | 1. Groups related classes, interfaces, and enums under a common name to avoid naming conflicts. |
| 43 | Using | 1. Allows access to types from a namespace. (importing namespace).  2. Manages resources automatically (e.g., IDisposable objects) when used in a using block. |
| 44 | Object | 1. Enables storing any data type in a variable. |
| 45 | If | 1. Executes a block of code only if the given condition is true.(used for decision making).  2. Condition must be boolean (true or false). |
| 46 | Else if | 1.Allows checking multiple conditions in sequence when the initial if condition is false.  2. Must always follow an if statement.  3. Can have multiple else if blocks, but they execute in order. If one else if condition is true, the rest are skipped. |
| 47 | Else | 1.Used with if to execute a block of code when the if condition is false.  2. Must follow an if or else if statement. |
| 48 | Goto | 1.it is jump statement which is used to jump to the specific section.  2. Must jump to a valid label within the same method. |
| 49 | value | 1. Represents the input value assigned to a property  2. Used to store or modify property values dynamically. |
| 50 | Let | 1. Used in LINQ queries to create a temporary variable that holds the result of an expression.  2. Can only be used in LINQ queries. |
| 51 | Into | 1. Used in LINQ queries to group results or continue a query after a select or group clause.  2. Must follow a select or group clause in LINQ. |
| 52 | Select | 1. Extracts specific data from a collection or query result.  2. Must be used at the end of a LINQ query. |
| 53 | From | 1. Defines the data source from which records will be retrieved.  2. Must be the first keyword in a LINQ query. |
| 54 | Where | 1. Filters records based on a condition.  2. Requires a boolean condition and Must follow the from keyword before select. |
| 55 | Groupby | 1.Groups data based on a specified key  2. Must be followed by into to create grouped results.  3. Used to organize data into groups. |
| 56 | Orderby | 1. Sorts the elements of a collection in ascending or descending order.  2. Must follow a from clause. |
| 57 | Single Responsibility Principle (SRP) | 1. A class should have only one reason to change, meaning it should have only one responsibility or job. |
| 58 | Open/Closed Principle (OCP) | 1. A class should be open for extension but closed for modification.  2. Avoid modifying existing code when adding new functionality. |
| 59 | Primary Key | 1. Uniquely identifies each record in a table.  2. Ensures no duplicate or NULL values in the column.  3. Must be unique, non-null, and only one per table. |
| 60 | Foreign Key | 1. Creates a relationship between two tables by referencing a primary key.  2. Must reference an existing primary key in another table.  3. Cannot contain values that do not exist in the referenced table.  4. Both tables must have compatible data types for the key column. |
| 61 | Combination Key | Combination Key is when two or more columns are combined to create a unique identifier |
| 62 | Composite Key | 1. A Composite Key is a Primary Key that consists of multiple columns. |
| 63 | Unique Key | 1. A Unique Key ensures that values in a column must be unique across all rows, but it can have NULL values. |
| 64 | Stored Procedure | 1. A Stored Procedure is a precompiled set of SQL statements stored in the database that can be executed as a single unit.(CRUD)  2. AS is used to define the procedure body. BEGIN ... END is used if using multiple statements.  EXEC or EXECUTE to call the procedure. |
| 65 | User-Defined Function (UDF) | 1. It is a reusable function that returns a single value or a table.  2. Must return a value (Scalar or Table).  Cannot modify database data (INSERT, UPDATE, DELETE not allowed).  Cannot use TRY...CATCH for error handling.  3. RETURNS keyword to define the return type.  AS to define the function body.  RETURN statement to return a value or table.  4. Can be used inside SELECT, WHERE, and JOIN queries.  Must be prefixed with dbo. when calling (SELECT dbo.FunctionName()). |
| 66 | DataReader | 1. Fetches data row by row (forward-only, read-only).  2. Requires an open database connection.  Cannot modify data or move backward. |
| 67 | DataTable | 1. Stores tabular data in memory,  Stores only one table (no multiple table support).  2. Can be used without a database connection.  3. Column definitions before adding rows (dt.Columns.Add(...)). |
| 68 | DataSet | 1. Holds multiple DataTables and their relationships in memory.  2. Cannot execute queries; it only stores fetched data.  3. Required DataAdapter to fill data into the DataSet. |
| 69 | DataAdapter | 1. Acts as a bridge between the database and DataTable/DataSet.  Used for fetching and updating data without keeping a connection open.  2. Works only with DataTable or DataSet (not DataReader). |
| 70 | SqlCommand | 1. Executes SQL queries (SELECT, INSERT, UPDATE, DELETE) and stored procedures.  2. Requires an active SqlConnection to execute commands. |
| 71 | SqlConnection | 1. Establishes a connection to a SQL Server database (mandatory for any database operation).  2. Must have a valid connection string (Data Source=...; Initial Catalog=...;).  3. Needs to be opened (conn.Open()) before executing commands and closed (conn.Close()) after use. |
| 72 | ExecuteReader() | 1. Retrieves multiple rows from the database (used for SELECT queries). Returns a SqlDataReader. |
| 73 | ExecuteNonQuery() | 1. Executes INSERT, UPDATE, DELETE, and returns the number of rows affected (does not return data). |
| 74 | ExecuteScalar() | 1. Returns a single value (e.g., COUNT(), MAX()). Used for aggregated queries. |
| 75 | CommandText | 1. Holds the SQL query or stored procedure name to execute. |
| 76 | CommandType | 1. Specifies whether the command is a SQL text query (CommandType.Text) or a Stored Procedure (CommandType.StoredProcedure).   |  | | --- | |  |  |  | | --- | |  | |
| 77 | Parameters | 1. Used to pass values safely into SQL queries (prevents SQL injection). |
| 78 | AddWithValue() | 1. Adds a parameter with a specified value directly. |
| 79 | Add() | |  | | --- | |  |  |  | | --- | | 1.Adds a parameter with more control over data type and value. | |
| 80 | Sprint Planning | 1. Decides what work will be done in the upcoming sprint. The team selects tasks from the backlog. |
| 81 | Daily Standup | 1. A short 15-minute meeting where team members discuss progress, roadblocks, and next steps. |
| 82 | Sprint Review | 1. Held at the end of a sprint to showcase completed work and gather feedback from stakeholders. |
| 83 | Sprint Retrospective | 1. A reflection meeting where the team discusses what went well, what didn’t, and how to improve in the next sprint. |
| 84 | CROSS JOIN | 1. Produces cartesian product of two tables. |
| 85 | Self Join | 1. Used to join a table with itself.  2. Must use aliases to differentiate. |
| 86 | HAVING | 1. Filter grouped data (used after GROUP BY).  2. Only used with GROUP BY. |
| 87 | GROUPING SETS | 1. Custom group combinations in a single query.  2. Used only with GROUP BY.  3. A GROUP BY clause with multiple sets. |
| 88 | CUBE | 1. Returns all combinations of grouped columns.  2. Works only with GROUP BY. |
| 89 | ROLLUP | 1. Returns hierarchical totals (top-down summary).  2. Grouping order matters.  3. GROUP BY with multiple columns. |
| 90 | EXISTS | 1. Checks if subquery returns rows (boolean check).  2. Subquery must return something. |
| 91 | ANY / ALL | 1. Compare a value to any or all values in a list/subquery.  2. Works with comparison operators (=, <, >). |
| 92 | CROSS APPLY | 1. Join with table-valued function, returns matching rows.  2. Works like INNER JOIN; skips NULLs. |
| 93 | OUTER APPLY | 1. Like LEFT JOIN for functions; includes unmatched rows.  2. None; shows NULLs for non-matching.  3. Use when you need all rows from left side. |
| 94 | UNION | 1. Combine distinct rows from two SELECTs.  2. Column count & types must match.  3. Same number of columns in both queries. |
| 95 | INTERSECT | 1. Get common rows from two SELECTs.  2. Column count & types must match.  3. Removes duplicates by default. |
| 96 | EXCEPT | 1. Returns rows in first query not in second.  2. Column count & types must match.  3. Also removes duplicates. |
| 97 | CTE (Common Table Expression) | 1. Define a temporary resultset for use within a SELECT/INSERT/UPDATE/DELETE.  2. Only valid within the same query.  3. Use WITH keyword before SELECT. |
| 98 | PIVOT | 1. Convert rows to columns .  2. Only works with aggregate functions.  3. Requires a fixed set of column values to pivot. |
| 99 | MERGE | 1. Insert, update, or delete records in a target table based on source table match.  2. Can't use multiple WHEN MATCHED THEN without conditions.  3. Requires Source & target tables, and a join condition. |
| 100 | Transaction | 1. Ensure atomicity – either all statements succeed, or none.  2. Must explicitly use BEGIN, COMMIT, and ROLLBACK.  3. Requires Set of DML statements (INSERT/UPDATE/DELETE). |
| 101 | CASE | 1. Apply conditional logic in queries (like IF...ELSE).  2. Must end with END, can’t return multiple types.  3.Requires A WHEN...THEN structure. |
| 102 | COALESCE | 1. Return first non-null value from a list.  2. All values should be of the same datatype.  3.Requires At least two arguments. |
| 103 | NULLIF | 1. Returns NULL if two values are equal, else returns the first one.  2. Accepts exactlytwo arguments.  3.Requires Both values for comparison. |
| 104 | ROLLBACK | 1. Undo all changes made in the current transaction.  2.Used to maintain data integrity when something goes wrong.  3. Can only rollback within an open transaction (i.e., after BEGIN and before COMMIT).  4. A transaction must be in progress (BEGIN TRANSACTION). |
| 105 | COMMIT | 1. Save all changes permanently to the database.  2. Once committed, changes can't be undone via ROLLBACK.  3. Requires A valid transaction and successful execution of statements. |
| 106 | Entity Framework | 1.Entity Framework (EF) is an open-source Object-Relational Mapper (ORM) for .NET. It helps developers interact with a relational database using .NET objects, instead of writing raw SQL queries. |
| 107 | DbContext | 1. Acts as a bridge between your C# code and the database.  2. Manages database connections, tracking changes, and saving data.  3. Must inherit from DbContext class.  4.Requires Connection string and defined DbSet properties. |
| 108 | DbSet | 1. Represents a table in the database.  2. Used to perform CRUD operations on the entities.  3. Should be declared as a property inside DbContext.  4. Generic type must be an entity class.  5. You can use LINQ like .Where(), .Add(), .Remove() directly on it. |
| 109 | Entity Classes | 1. Represent tables/records in code using C# classes.  2. Help EF understand the database schema via code-first approach.  3. Must have a primary key (default: Id or ClassNameId).  4. Properties must be public with getters and setters. |
| 110 | Database-First | 1. Use existing database to generate models and DbContext.  2. Database must be well-designed before generation.  3. Can’t make major schema changes from code.  4. Requires Existing DB, use EF Designer or Scaffold-DbContext. |
| 111 | Code-First | 1. Create models in C# code first, then generate the database from them.  2. You control schema through code, not directly in the DB.  3. Requires Entity classes, DbContext, and Add-Migration, Update-Database (Package Manager Console or CLI). |
| 112 | EF Properties : SaveChanges() | 1. Commits all changes (Add, Update, Delete) made in the DbContext to the database.  2. Must be called after all changes, otherwise nothing is saved to DB. |
| 113 | Entry() | 1. Provides access to change tracking info (state like Added, Modified, etc.) of a specific entity.  2. You must pass a valid entity object to context.Entry(entity). |
| 114 | Add() | 1. Adds a new entity to the DbContext so it will be inserted into DB on SaveChanges().  2. Must use SaveChanges() after to actually insert it.  3. A new instance of the entity class. |
| 115 | Remove() | 1. Marks an entity for deletion from the database.  2. Entity must be tracked or attached to the context.  3.Requires The entity instance to be removed. |
| 116 | Find() | 1. Quickly retrieves a record by primary key.  2. Only works with primary key search.  3. Requires The key value(s) of the entity. |
| 117 | Include() | 1. Eagerly loads related data (e.g., navigation properties like foreign key relationships).  2. Must be used before executing the query (e.g., with ToList(), FirstOrDefault()).  3.Requires A navigation property defined in the model. |
| 118 | LINQ | LINQ is a query language integrated into C# (and .NET languages) that allows you to write queries directly within your code to retrieve data from various sources. |
| 119 | LINQ Types : LINQ to Objects | 1. Used to query in-memory collections like arrays, lists, dictionaries, etc.  2. Only works on objects in memory, not on external data sources.  3. Requires using System.Linq and collections like List<T>, Array, etc.  4. It doesn't require a database; it's the simplest and most commonly used LINQ type. |
| 120 | LINQ to SQL | 1.Used to query and manipulate SQL Server database data directly using LINQ.  2. Only supports SQL Server; not cross-database compatible.  3. Requires a LINQ to SQL class (.dbml file) and a valid connection string.  4. |
| 121 | LINQ to Entities (Entity Framework) | 1. Used to query databases using Entity Framework (EF), working with strongly-typed entity classes.  2. Requires proper configuration of the EF model (Database First, Code First, or Model First).  3. DbContext, DbSet, entity classes, and a configured EF environment. |
| 122 | LINQ to XML | 1. Used to query, modify, and create XML documents using LINQ.  2. Works only with XML data, not other structured formats.  3. Requires System.Xml.Linq and classes like XElement, XDocument. |
| 123 | LINQ to DataSet | 1. Used to query data from ADO.NET DataSet and DataTable objects.  2. Operates only on disconnected data (DataSets/DataTables).  3. Requires a filled DataSet, usually from a SQL data adapter. |
| 124 | Query Syntax | 1. Similar to SQL.  2. Some complex operations like Aggregate, Zip, MaxBy, MinBy, etc., are not supported in Query Syntax — only available in Method Syntax.  3. Need a collection or data source (like List, array, DBSet, etc.).  4. Must import System.Linq. |
| 125 | Method Syntax | 1. Uses lambda expressions  2. Used to perform complex and advanced LINQ operations through method chaining.  3. Supports all LINQ operators — including those not supported in query syntax like Zip(), Aggregate(), MaxBy(), GroupJoin(), etc.  4. Must include using System.Linq; |
| 126 | Deferred Execution | 1. Executes the query only when the data is actually accessed (e.g., in a foreach loop or .ToList()).  2. Query runs only when iterated |
| 127 | Immediate Execution | 1. Executes the query immediately and stores the result into memory (e.g., via .ToList(), .Count() etc.).  2.Query runs instantly (e.g., when using ToList(), Count()) |
| 128 | Conversion Operators | 1. ToList: Converts a sequence to a List<T>.  2. ToArray: Converts a sequence to an array (T[]).  3. ToDictionary: Converts a sequence into a dictionary (Dictionary<TKey, TValue>).  4. Cast: Casts elements of a sequence to a specified type. |
| 129 | Partitioning Operators | 1. Take: Returns a specified number of contiguous elements from the start of a sequence.  2. Skip: Skips a specified number of elements and returns the remaining elements. |
| 130 | Quantifier Operators | 1. Any: Returns true if any element in a sequence satisfies a condition.  2. All: Returns true if all elements in a sequence satisfy a condition.  3. Contains: Checks if a sequence contains a specific element. |
| 131 | Projection Operators | 1. Select: Projects each element of a sequence into a new form.  2. SelectMany: Flattens a collection of collections into a single sequence. |
| 132 | Element Operators | 1. First: Returns the first element of a sequence, or throws an exception if the sequence is empty.  2. FirstOrDefault: Returns the first element of a sequence, or null/default value if the sequence is empty.  3. Last: Returns the last element of a sequence, or throws an exception if the sequence is empty.  4. LastOrDefault: Returns the last element of a sequence, or null/default value if the sequence is empty. |
|  |  | 5. Single: Returns the only element of a sequence, or throws an exception if there is more than one element.  6. SingleOrDefault: Returns the only element of a sequence, or null/default value if the sequence is empty; throws an exception if there’s more than one element.  7. ElementAt: Returns the element at a specific index in a sequence, or throws an exception if the index is out of bounds.  8. ElementAtOrDefault: Returns the element at a specific index in a sequence, or null/default value if the index is out of bounds. |
| 133 | LINQ ToLookup Method | 1. The ToLookup method in LINQ is used to create a lookup (a collection of keys and their associated values). It transforms a sequence of elements into a collection of keys, each of which can have multiple associated values. |
| 134 | LINQ Zip Method | 1. The Zip method in LINQ is used to merge two sequences by combining the elements of both sequences, element by element, into a new sequence. It pairs elements from two sequences and produces a sequence of pairs. |
| 135 | Dependency Injection | 1. DI helps in reducing the tight coupling between classes. It allows classes to depend on abstractions (interfaces or base classes) rather than concrete implementations.  2. It improves testability by making it easy to mock or substitute dependencies, which is essential for unit testing.  3. DI typically requires a DI container to manage the creation and life-cycle of objects.  4. The most common DI method is constructor injection, but there are also property injection and method injection, though they may lead to less maintainable code.  5. Dependencies must be initialized before use, and the container must be configured to resolve dependencies. |
| 136 | Unit Testing | 1. Unit testing ensures that individual units of code (usually functions or methods) work as expected. It verifies the correctness of the smallest testable parts of an application in isolation.  2. It helps in identifying bugs early, improving code quality, and reducing the chances of regressions. |
| 137 | Unit Test Frameworks | 1. Popular frameworks for unit testing in C# include xUnit, NUnit, and MSTest.  2. These provide structure for writing tests, assertions for verifying outcomes, and test runners to execute tests. |
| 138 | Mocking | 1. Mocking involves creating mock objects to simulate dependencies of the unit under test. This allows you to isolate the unit of work from its dependencies, enabling focused and efficient tests.  2. Mocking is done using libraries like Moq, NSubstitute, or Rhino Mocks in C#. |
| 139 | Test-Driven Development (TDD) | 1. TDD is an approach where you write the unit test before writing the actual code.  2. The cycle is: Write a failing test → Write code to pass the test → Refactor. This ensures that code is always written with tests in mind. |
| 140 | Assertions | 1. Assertions are statements in a unit test that check if the result of a unit under test matches the expected result. Common assertions include:   * Assert.Equal(expected, actual) * Assert.True(condition) * Assert.NotNull(object) |

**Student Contact**

Contact ID **PK**

Email

Mob

Student Id **FK**

**Student Geo Info**

Geo ID **PK**

Address

City

State

Zip Code

Student Id **FK**

**Student Guardian**

Guardian ID **PK**

Name

Relationship

Contact No

Student Id **FK**

1:M

1:1

1:M

**Student**

Student Id **PK**

First Name

Last Name

1:1

**Student Academics**

Academic ID **PK**

Course

CGPA

Student Id **FK**