**JavaScript**:

* In HTML, JavaScript code is inserted between **<script>** and </script> tags.
* Writing into an HTML element, using **innerHTML**. *document.getElementById("demo").innerHTML = 5 + 6*;
* Writing into the HTML output using **document.write**().
* Writing into an alert box, using **window.alert()**.
* Writing into the browser console, using **console.log()**
* Variables declared with the **let** keyword can have **Block Scope**. Declared with {var x=3;}
* Variables defined with **const** behave like **let** variables, except they cannot be reassigned

**React.js** [**https://reactjs.org/**](https://reactjs.org/)

* **React** is a JavaScript library for building user interfaces.
* **Declarative**: React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes. **Declarative** views make your code more predictable and easier to debug.
* **Component-Based**: Build encapsulated components that manage their own state, then compose them to make complex UIs. Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.
* React can also **render on the server using Node** and power mobile apps using React Native.
* Input data that is passed into the component can be accessed by **render()** via **this.props**.
* Whenever you get confused by something in JavaScript, [MDN](https://developer.mozilla.org/en-US/docs/Web/JavaScript) and [javascript.info](https://javascript.info/) are great websites to check. There are also [community support forums](https://reactjs.org/community/support.html) where you can ask for help.
* Javascript basic: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript>
* Getting started with react: <https://reactjs.org/docs/getting-started.html>
* React tutorial: <https://www.taniarascia.com/getting-started-with-react/>
* Practical approach to learn: <https://reactjs.org/tutorial/tutorial.html>
* Step-by-step approach: <https://reactjs.org/docs/hello-world.html>
* In classic Object Oriented Programming, objects are collections of data and methods that operate on that data. JavaScript is a prototype-based language that contains no class statement, as you'd find in C++ or Java. Instead, JavaScript uses functions as classes.

**SQL Server Services and Tools**

Microsoft provides both data management and business intelligence (BI) tools and services together with SQL Server.

* For data management, SQL Server includes SQL Server Integration Services (SSIS), SQL Server Data Quality Services, and SQL Server Master Data Services. To develop databases, SQL Server provides SQL Server Data tools; and to manage, deploy, and monitor databases SQL Server has SQL Server Management Studio (SSMS).
* For data analysis, SQL Server offers SQL Server Analysis Services (SSAS). SQL Server Reporting Services (SSRS) provides reports and visualization of data. The Machine Learning Services technology appeared first in SQL Server 2016 which was renamed from the R Services.

|  |  |
| --- | --- |
| **HTML** | **HTML5** |

|  |  |  |
| --- | --- | --- |
| definition | A hypertext markup language (HTML) is the primary language for developing web pages. | HTML5 is a new version of HTML with new functionalities with markup language with Internet technologies. |
| media support | **HTML** does not have support for video and audio. | **HTML5** supports both video and audio. |
| Storage | The HTML browser uses cache memory / cookies as temporary storage. | HTML5 has the storage options like:**application cache, SQL database,** and **web storage**. |
| Browser compatibility | HTML is compatible with almost all browsers as it has been present for a long time, and the browser made modifications to support all the features. | In HTML5, we have many new tags, elements, and some tags that have been **removed/modified**, so only some browsers are fully compatible with **HTML5**. |
| Threading | In HTML, the browser interface and JavaScript running in the same thread. | The HTML5 has the JavaScript Web Worker API, which allows the browser interface to run in multiple threads. |
| Storage | Uses cookies to store data. | Uses local storage instead of cookies |
| Vector and Graphics | Vector graphics are possible with the help of technologies like **VML, Silverlight, Flash,etc**. | Vector graphics is an integral part of **HTML5, SVG** and **canvas**. |
| Shapes | It is not possible to create shapes like **circles, rectangles, triangles**. | We can draw shapes like **circles, rectangles, triangles**. |
| Doc type | Doctype declaration in html is too long <! DOCTYPE HTML PUBLIC "- // W3C // DTD HTML 4.01 // EN" "http://www.w3.org/TR/html4/strict.dtd"> | The DOCTYPE declaration in html5 is very simple "<! DOCTYPE html> |
| Character Encoding | Character encoding in HTML is too long. <! DOCTYPE HTML PUBLIC "- // W3C // DTD HTML 4.0 Transitional // EN"> | Character encoding declaration is simple <meta charset = "UTF-8"> |
|  | Elements like nav, header were not present. Cannot handle inaccurate syntax | New element for web structure like nav, header, footer etc. capable of handling inaccurate syntax. |

**RESTful Web Services with REST API**

* **REST** is any interface between systems using HTTP to obtain data and generate operations on those data in all possible formats, such as XML and JSON
* **Restful Web Services** is a lightweight, maintainable, and scalable service that is built on the REST architecture. Restful Web Service, expose API from your application in a secure, uniform, stateless manner to the calling client. The calling client can perform predefined operations using the Restful service.
* The underlying protocol for REST is HTTP. REST stands for REpresentational State Transfer.
* <https://www.guru99.com/restful-web-services.html#:~:text=Restful%20Web%20Services%20is%20a,operations%20using%20the%20Restful%20service>.
* REST is a way to access resources which lie in a particular environment. For example, you could have a server that could be hosting important documents or pictures or videos. If a client, say a web browser needs any of these resources, it has to send a request to the server to access these resources. Now REST services defines a way on how these resources can be accessed.
* The key elements of a RESTful implementation are as follows:
* Resources
* Request Verbs (like GET, POST, PUT, DELETE)
* Request Headers: Type of resource and authorization
* Request Body (specially for POST)
* Response body
* Response Status code

**Why Restful**

* Heterogeneous languages and environments – This is one of the fundamental reasons which is the same as we have seen for SOAP as well.
* It enables web applications that are built on various programming languages to communicate with each other.
* With the help of Restful services, these web applications can reside on different environments, some could be on Windows, and others could be on Linux.

**ASP.NET Core MVC Framework**

* <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0>

**Factory Method** is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

**Problem**: Imagine that you’re creating a logistics management application. The first version of your app can only handle transportation by trucks, so the bulk of your code lives inside the Truck class. if later you decide to add another type of transportation to the app, you will probably need to make all of these changes again. As a result, we will end up with pretty nasty code.

**Solutions**: The Factory Method pattern suggests that you replace direct object construction calls (using the new operator) with calls to a special factory method. the objects are still created via the new operator, but it’s being called from within the factory method. Objects returned by a factory method are often referred to as products.

we just need to move the constructor call from one part of the program to another. And we can override the factory method in a subclass and change the class of products being created by the method.

There’s a slight **limitation** that subclasses may return different types of products only if these products have a common base class or interface.

Also, the factory method in the base class should have its return type declared as this interface.

**Use**: We can use the Factory Method when you want to save system resources by reusing existing objects instead of rebuilding them each time.

**Pros:** You can avoid tight coupling between the creator and the concrete products. Single Responsibility Principle. **You can move the product creation code into one place in the program, making the code easier to support**. You can introduce new types of products into the program without breaking existing client code.

**Cons**: The code may become more complicated since you need to introduce a lot of new subclasses to implement the pattern**.------------------------------------------------------------------------------**

**Singleton** is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.

**Problem**: It solves two problems:

1. Ensure that a class has just a single instance. 2. Provide a global access point to that instance.

**Solutions**: Make the default constructor private, to prevent other objects from using the new operator with the Singleton class.

Create a static creation method that acts as a constructor. Doing so, this method calls the private constructor to create an object and saves it in a static field. All following calls to this method return the cached object.

**Use**: Use the Singleton pattern when a class in your program should have just a single instance available to all clients; for example, a single database object shared by different parts of the program. **Use the Singleton pattern when you need stricter control over global variables.**

**Pros:** You can be sure that a class has only a single instance and gain a global access point to that instance. The singleton object is initialized only when it’s requested for the first time.

**Cons**: **Violates the Single Responsibility Principle**. The pattern solves two problems at the time. The Singleton pattern can mask **bad design**, for instance, when the components of the program know too much about each other. The pattern requires special treatment in a multithreaded environment.

**OOP Architectural Design Patterns:**

* **Design patterns** are typical solutions to commonly occurring problems in software design. They are like pre-made blueprints that you can customize to solve a recurring design problem in your code.
* **Creational patterns** provide object creation mechanisms that increase flexibility and reuse of existing code. – Singleton, Factory Method, Builder, Prototype, Abstract Factory
* **Structural patterns** explain how to assemble objects and classes into larger structures, while keeping the structures flexible and efficient. – Adapter, Bridge, Composite, Façade, Decorator, Flyweight
* **Behavioral patterns** take care of effective communication and the assignment of responsibilities between objects. - Observer, strategy, State, Command, Iterator, Mediator, Memento, Visitor, Template method
* **Singleton**: Singleton is used when a class can only have a single instance and shared among all requests. For example, connection pool, cache, global variables. Singleton class needs to be sealed to prevent inheriance, having private (or hidden) constructor to avoid instance to be created. A lock needs to be placed to make it thread safe when creating the object.

public sealed class Cache

{

private static volatile Cache instance;

private static object lockObject = new Object();

private Cache() { }

public static Cache CacheObject

{

get

{

if (instance == null)

{

lock (lockObject)

{

if (instance == null)

{ InitializeInstance(); }

}

}

return instance;

}

}

private static void InitializeInstance()

{

instance = new Cache();

}

}

* **Factory Method**: Factory Method provides an interface to create objects in a superclass, but allows a subclass factory to create different type of object. **For example**, we have Manager class inherances Employee class. ManagerFactory inherances EmployeeFactory.

public class Employee

{ //...}

public class Manager : Employee{ //... }

public class EmployeeFactory

{

public virtual Employee CreateEmloyee()

{ return new Employee(); }

}

public class ManagerFactory : EmployeeFactory

{

public override Employee CreateEmloyee()

{ return new Manager(); }

}

* **Strategy**: Strategy patterns are groups of interchangeable functions that can be initialized for different purpose. This can be implemented by using abstract methods or interfaces.
* For example we have Fruit class. We can create apple and orange subclasses based on Fruit class and give them different discount algorithms.
* **Builder**: Builder pattern uses an interface class to separate the construction processes of a complex object from its representation so that the same construction processes can create different representations.
* Builder pattern includes 4 parts: a builder interface, a concrete builder, a director and a product.
* **Adapter**: When a client have a interface not compatible with an adaptee class, an adapter class is created to convert the adaptee interface to the client interface so the client can use the adaptee. *Use cases can be a hardware driver, or integrating a third party libary*.
* **State**: State pattern create an object to changed beheviour depends on its state.

For example, there're a few processes to handle before it completes.

* **Observer**: A subject can have multiple observer classes. When the subject class attributes changed, it can notify all observer classes. A good example is a UI object can have multiple event handlers. For example, Scheduler class has a list of attendees. When scheduled time changed, it notify all attendees.

public class Attendee

{

public void Update(DateTime dt) { //.... }

}

public class Scheduler

{

List attendees = new List();

public void Add(Attendee obj)

{

attendees.Add(obj);

}

public void ChangeSchedule(DateTime dt)

{

//.....

Notify(dt);

}

public void Notify(DateTime dt)

{

foreach (Attendee obj in attendees)

{

obj.Update(dt);

}

}

}

* **Decorator**: Decorator Pattern is also called wrapper pattern. Without creating subclass, it create a wrapper class with the same interface as another class.
* **Flyweight** is used to saving memory by creating shared objects
* **Facade**: Provide a single interface to a set of interfaces, to hide the complicity of the system and make it easy to use.

**Template**: Template define a skeleton of an algorithm in an operation, lets subclasses redefine certain steps without changing the algorithms structure. For example, there's same steps to get checking and saving account interest rate, but different algorithms.

* **SQL** stands for **Structured Query Language**. SQL is a standard language for accessing and manipulating databases.
* RDBMS stands for **Relational Database Management System**. RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access
* The data in RDBMS is stored in database objects called tables. A **table** is a collection of related data entries and it consists of columns and rows.
* SQL keywords are **NOT case sensitive**: select is the same as SELECT
* **SQL** is a query language, whereas **MySQL** is a relational database that uses **SQL** to query a database. You can use **SQL** to access, update, and manipulate the data stored in a database. **SQL** is used for writing queries for databases, **MySQL** facilitates data storing, modifying, and management in a tabular format.

**SELECT** - extracts data from a database

**UPDATE** - updates data in a database

**DELETE** - deletes data from a database

**INSERT INTO** - inserts new data into a database

**CREATE DATABASE** - creates a new database

**ALTER DATABASE** - modifies a database

**CREATE TABLE** - creates a new table

**ALTER TABLE** - modifies a table

**DROP TABLE** - deletes a table

**TRUNCATE TABLE** statement is used to delete the data inside a table, but not the table itself.

**CREATE INDEX** - creates an index (search key)

**DROP INDEX** - deletes an index

**SELECT DISTINCT** statement is used to return only distinct (different) values.

**SELECT INTO** statement copies data from one table into a new table.

**ORDER BY** keyword is used to sort the result-set in ascending or descending order.

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

**GROUP BY** statement groups rows that have the same values into summary rows. GROUP BY statement

is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-

set by one or more columns.

**Stored procedure** is prepared SQL code that you can save, so the code can be reused over & over again.

[**NOT NULL**](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value

[**UNIQUE**](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different

[**PRIMARY KEY**](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.

[**FOREIGN KEY**](https://www.w3schools.com/sql/sql_foreignkey.asp) - Uniquely identifies a row/record in another table.

[**CHECK**](https://www.w3schools.com/sql/sql_check.asp) - Ensures that all values in a column satisfies a specific condition.

[**DEFAULT**](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column when no value is specified.

[**INDEX**](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

* Joins: **INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN**
* **INNER JOIN** Customers ON Orders.CustomerID=Customers.CustomerID;

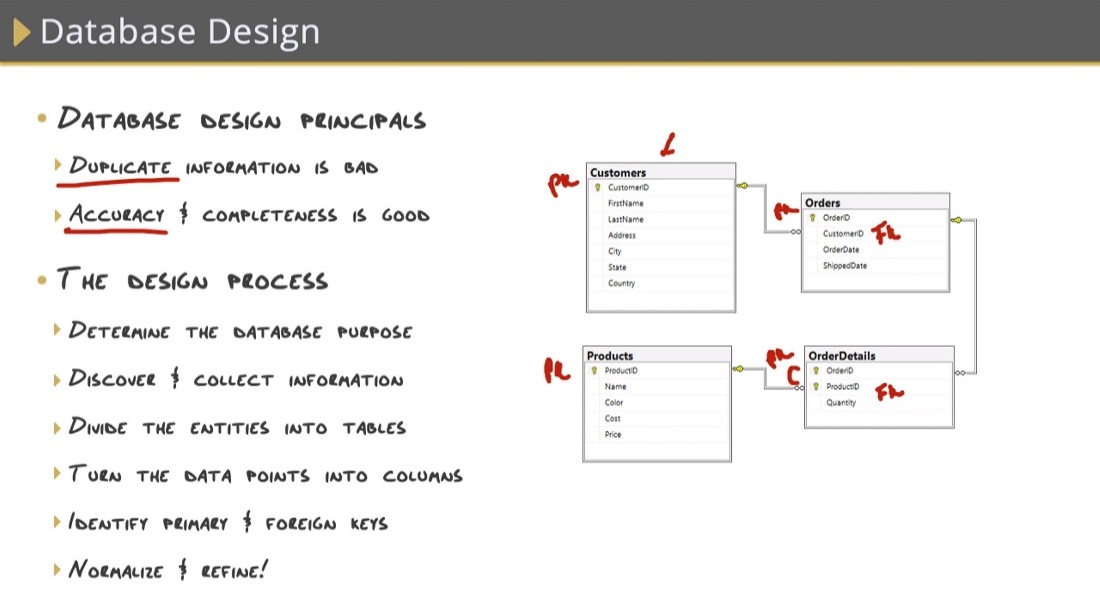
SELECT column\_name(s) FROM table1

INNER JOIN table2 ON table1.column\_name = table2.column\_name;

CREATE TABLE Orders (  
    OrderID int NOT NULL PRIMARY KEY,  
    OrderNumber int NOT NULL,    PersonID int FOREIGN KEY REFERENCES Persons(PersonID) );

**SQL Database Design**

* Avoid duplicate and redundant information data.
* Ensure accuracy and completeness/ data dependecy of the data



* What is **Normalization?** It is a technique of organizing the data in database. It is systematic approach of decomposing the tables to reduce redundancy.
* **1st Normal Form**: Remove repeating groups from table, create separate table for each set of related data, identify each set of data with primary key
* **2nd Normal Form:** Table should not contain partial dependency.
* **3rd** **Normal Form**: There should be any transitive dependency for no-prime attributes.
* **Boyce-Codd Normal Form BCNF**: Every functional dependency A🡪B, then A has to be the super key of that table. **Automacity**- should not have multiple values for a row.