**Database challenge**

**Question 1:**

Diagram

Description automatically generated

**Question 2:**

update [EmployeeFile] set[EmployeeFile].email=REPLACE([Employee File].email,Substring([EmployeeFile].email,charindex

('@',[EmployeeFile].email)+1,charindex('.',[Employee File].email,charindex('@',[EmployeeFile].email,0))-(charindex(‘@’,[Employee File.email+1)) ‘company’)

WHERE [EmployeeFile].email LIKE '%\_@\_%.\_%'

**Back-end code Challenge**

**Question 1:**

List<string> likes = new list<string> {“Alex”,”Jacob”};

if (likes.count ==0)

{

console.writeline(“no likes”);

}

else if (likes.count ==1)

{

console.writeline($”{likes[0]} likes this”);

}

else if (likes.count == 2)

{

console.writeline($”{likes[0]} and {likes[1]} likes this”);

}

else if (likes.count == 3)

{

console.writeline($”{likes[0]}, {likes[1]} and {likes[2]} likes this”);

}

else if (likes.count >= 4)

{

console.writeline($”{likes[0]}, {likes[1]} and {likes.count-2} others likes this”);

}

**Question 2:**

**Bonus: what can be done to reduce tight coupling in a class?** Inheritance/interfaces will be easier to access one class features of another class through interfaces. Interfaces is a design of the class which has no logic but rather what a class must do instead of how to do it like what an Inheritance method would have, loosely coupled development will allow future developers to easily maintain those classes.

public Interface IRobotService

{

Robot BuildRobot(enum RobotType);

}

public Interface IPartsService

{

List<Parts> GetRobotPartsFor(Enum RobotType);

List<Parts> GetCarPartsFor(Enum CarType);

}

public Interface ICarService

{

car BuildCar(Enum CarType);

}

public class RobotService : IRobotService

{

public Robot BuildRobot(Enum RobotType)

{

if(RobotType == RoboticDog)

var parts = GetRobotPartsFor(RoboticDog);

return \_robotService.BuildRobotDog(parts);

else if(RobotType == RoboticCat)

var parts = GetRobotPartsFor(RoboticCat);

return \_robotService.BuildRobotCat(parts);

else if(RobotType == RoboticDrone)

var parts = GetRobotPartsFor(RoboticDrone);

return \_robotService.BuildRobotDrone(parts);

else if (RobotType == RoboticCar)

var parts = GetRobotPartsFor(RoboticCar);

return \_robotService.BuildRobotCar(parts);

else

return null;

}

}

public class PartsService : IPartsService

{

public List<Parts> GetRobotPartsFor(Enum RobotType)

{

return \_partsService.GetParts(RobotType);

}

public List<Parts> GetCarPartsFor(Enum CarType)

{

return \_partsService.GetParts(CarType);

}

}

public class CarService : ICarService

{

public Car BuildCar(Enum CarType)

{

if(CarType == Toyota)

var parts = GetCarPartsFor(Toyota);

return \_carService.BuildCar(parts);

else if(RobotType == Ford)

var parts = GetCarPartsFor(Ford);

return \_carService.BuildCar(parts);

else if(RobotType == Opel)

var parts = GetCarPartsFor(Opel);

return \_carService.BuildCar(parts);

else if (RobotType == Honda)

var parts = GetCarPartsFor(Honda);

return \_carService.BuildCar(parts);

else

return null;

}

}

public class Factory

{

private Interface IRobotService \_iRobotService;

private Interface IPartsService \_iPartsService;

private Interface ICarService \_iCarService;

public class Factory(IRobotService robotService,

IPartsService partsService,

ICarService carService)

{

\_iRobotService = robotService;

\_iPartsService = partsService;

\_iCarService = carService;

}

public Robot BuildRobot(Enum robotType)

{

return \_\_iRobotService.BuildRobot(robotType);

}

public car BuildCar(Enum carType)

{

return \_iCarService.BuildCar(carType);

}

public List<Parts> GetRobotPartsFor(Enum robotType)

{

return \_iPartsService.GetParts(robotType);

}

public List<Parts> GetCarPartsFor(Enum carType)

{

return \_iPartsService.GetParts(carType);

}

**Front-end code Challenge**

**Question 1:**

1. #firstDiv = red and #secondDiv = orange
2. document.getElementById('firstDiv').style.backgroundColor = 'pink' ;
3. document.getElementById('secondDiv').classList.add("yellow-card");

**Question 2:**

1. Both instances are the same
2. Use the equal value and equal type operator (===)

**Question 3:**

1. - ViewPort Meta tag – adjust website to screen size <https://www.browserstack.com/guide/how-to-make-website-mobile-friendly>

* using bootstrap – bootstrap will allow developers to use a number of components in css classes and allow developers to maintain easily although a lot of bootstrap frameworks does automatically.

<https://docs.microsoft.com/en-us/archive/msdn-magazine/2015/june/web-development-building-responsive-web-sites-with-bootstrap>

<body>

  <div class="container">

    <hr />

    <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2 gray">1</div>

    <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2 orange">2</div>

    <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2 gold">3</div>

<div class="col-xs-6 col-sm-4 col-md-3 col-lg-2 lightGreen">4</div>

</div> <!-- /container -->

</body>

* responsive images – allow for images to be displayed accurately on different screen resolutions.

<style>

img {

max-width: 100%;

}

</style>

<picture>

<source type="image/webp" srcset="https://my-image.com/my-image-100.webp 1x, https://my-image.com/my-image-200.webp 2x">

<source type="image/png" srcset="https://my-image.com/my-image-100.png 1x, https://my-image.com/my-image-200.png 2x">

<img alt="my image" src="https://my-image.com/my-image-200.png" loading="lazy" width="100" height="100">

</picture>

1. Reduces single HTTP requests to the server

<https://stackoverflow.com/questions/12804955/what-is-the-benefit-of-asp-net-bundling-and-minification-in-runtime#:~:text=Bundling%20reduces%20the%20number%20of,comments%20and%20other%20unnecessary%20characters>

1. Run a sass compiler to convert sass to css

<https://blog.bitsrc.io/the-complete-beginners-guide-to-sass-ee8d5278f4>

1. Java script transpiling – converts jc code using the latest features to suit older browsers

Babel – java script compiler

<https://www.browserstack.com/guide/resolve-javascript-cross-browser-compatibility-issues>