

Database Management Systems Course

Name: Harun Yahya Ünal

Project description: This Project is an AI based tutoring solution for the Database Management Course. The AI is used for the following features:

- Answering students' questions about the course
- Measuring the student's level of knowledge about the course

Other non-AI based features implemented include the following.

- In the quiz section, it gives score to the student based on the number of correct answers. This score is used to calculate the overall score.

Features that may be implemented in the future are as follows

- A model may be implemented to generate feedback according to the overall score.
- A study part may be implemented to provide course notes for the section that has low overall score.
- A practice part may be implemented to practice that student can write a database code then ai evaluates the code, gives feedback.

The Project used the following AI models

Model Name	Reference (downloaded from)	Used for
distilbert-base-cased-distilled-squad	Hugging Face Transformers	It is used for answering the questions of students about a selected section.
all-MiniLM-L6-v2	Hugging Face Sentence Transformers	It is used for generating a knowledge score for student for a selected section.

Describe your data model and how it is implemented here

In this project, I used MSSQL for database. There are 3 tables in my database:

- **Sections Table:** It contains section names and detailed information about each section.
- **Quizzes Table:** It contains the 5 quizzes for each section.
- **Overall Reports Table:** It contains the scores from quizzes and student knowledge part.

Diagram of the database with relations:



To implement the database, I used Entity Framework Core, an Object-Relational Mapping (ORM) framework. This tool helps us work with database tables by using C# classes instead of writing SQL code.

Each database table is represented by a C# class. Entity Framework converts these classes into tables in the database. To make changes to the database, we use a feature called **migrations**.

We also create a **DbContext** class, which is like a connection between the application and the database. For example, if we write `_context.Sections.ToList()`, it gives us a list of all the sections in the database, including their details. This makes it easy to work with data in our code.

Describe your modules: what it does, how it does in here including the source code for each module

I have two main parts in my project: the Website and the LLM parts.

1. Website Part:

I used ASP.NET MVC technology to build the website. The main modules are:

Model: Represents the database tables as C# classes. Each model corresponds to a table in the database.

<pre>using System.ComponentModel.DataAnnotations; using System.Linq; using System.Web; namespace DBMSCourse.Models { public class Section { [Key] public int SectionId { get; set; } public string SectionName { get; set; } public string DetailedInfo { get; set; } } }</pre>	<pre>using System.ComponentModel.DataAnnotations; using System.Linq; using System.Web; namespace DBMSCourse.Models { public class Quiz { [Key] public int QuestionId { get; set; } public int SectionId { get; set; } public Section Section { get; set; } public string QuestionText { get; set; } public string AnswerA { get; set; } public string AnswerB { get; set; } public string AnswerC { get; set; } public string AnswerD { get; set; } public string CorrectAnswer { get; set; } } }</pre>
<pre>using System.ComponentModel.DataAnnotations; namespace DBMSCourse.Models { public class OverallReport { [Key] public int ReportId { get; set; } public int SectionId { get; set; } public Section Section { get; set; } public decimal? QuizScore { get; set; } public decimal? KnowledgeCheckScore { get; set; } public decimal? SectionOverallScore { get; set; } } }</pre>	

View: Contains HTML, CSS, and JavaScript files for the user interface. Each page has a view. It gets the request from user with HTTP protocols with JavaScript code.

HomePage:

```
@{
    Layout = null;
    var sections = new DBMSCourse.Repositories.SectionRepository(new DBMSCourseContext()).GetAllSections();
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home Page</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f9;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }

        .home-container {
            background: #ffffff;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
            border-radius: 10px;
            padding: 20px;
            max-width: 400px;
            width: 90%;
            text-align: center;
        }

        select {
            width: 100%;
            padding: 10px;
            margin-bottom: 20px;
            border: 1px solid #ddd;
            border-radius: 5px;
            font-size: 1em;
        }

        .nav-button {
            width: 100%;
            padding: 10px;
            margin: 10px 0;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            font-size: 1em;
            color: white;
        }
```

```

        transition: background-color 0.3s ease;
    }

    .quiz-btn {
        background-color: #007bff;
    }

    .quiz-btn:hover {
        background-color: #0056b3;
    }

    .knowledge-btn {
        background-color: #28a745;
    }

    .knowledge-btn:hover {
        background-color: #1e7e34;
    }

    .qa-btn {
        background-color: #ffc107;
    }

    .qa-btn:hover {
        background-color: #e0a800;
    }

    .score-btn {
        background-color: #dc3545;
    }

    .score-btn:hover {
        background-color: #bd2130;
    }
</style>
</head>
<body>
    <div class="home-container">
        <h2>Welcome to the DBMS Course</h2>
        <select id="sectionSelect">
            <option value="">Select a Section</option>
            @foreach (var section in sections)
            {
                var secId = section.SectionId;
                var secName = section.SectionName;
                <option value="@secId">@secName</option>
            }
        </select>
        <button class="nav-button quiz-btn" onclick="navigateTo('QuizPage')">Quiz</button>
        <button class="nav-button knowledge-btn" onclick="navigateTo('CheckStudentInfoPage')">Student
Knowledge</button>
        <button class="nav-button qa-btn" onclick="navigateTo('QAPage')">Question & Answer</button>
        <button class="nav-button score-btn" onclick="navigateTo('OverallReportPage')">Overall Score</button>
    </div>

```

```

<script>
  function navigateTo(page) {
    const section = document.getElementById('sectionSelect').value;
    if (!section && page != "OverallReportPage") {
      alert('Please select a section.');
```

QuizPage:

```

@{
  int? sectionId = ViewBag.SectionId;
  var context = new DBMSCourseContext();
  var quizRepository = new DBMSCourse.Repositories.QuizRepository(context);
  var reportRepository = new DBMSCourse.Repositories.OverallReportRepository(context);
  var quizzes = quizRepository.GetQuizzesBySection(sectionId);
}

<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Quiz Page</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f4f4f9;
      margin: 0;
      padding: 0;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
    }

    .quiz-container {
      background: #ffffff;
      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
      border-radius: 10px;
      padding: 20px;
      max-width: 500px;
      width: 90%;
    }

    .question {
      font-size: 1.2em;
      font-weight: bold;
      margin-bottom: 20px;

```

```
}

.options {
  list-style-type: none;
  padding: 0;
}

.options li {
  margin: 10px 0;
}

.custom-radio {
  display: flex;
  align-items: center;
  background-color: #f9f9f9;
  border: 2px solid #ddd;
  border-radius: 5px;
  padding: 10px;
  cursor: pointer;
  transition: border-color 0.3s ease, background-color 0.3s ease;
}

.custom-radio:hover {
  border-color: #007bff;
  background-color: #eef4ff;
}

.custom-radio input {
  display: none;
}

.custom-radio span {
  margin-left: 10px;
  font-size: 1em;
  color: #333;
}

.custom-radio input:checked + span {
  font-weight: bold;
  color: #007bff;
}

.custom-radio input:checked + span:before {
  content: "✓ ";
  color: #28a745;
  font-weight: bold;
}

.button-container {
  display: flex;
  justify-content: space-between;
  margin-top: 20px;
}

.submit-btn, .skip-btn {
```

```
padding: 10px 20px;
border: none;
border-radius: 5px;
cursor: pointer;
font-size: 1em;
transition: background-color 0.3s ease;
}

.submit-btn {
background-color: #28a745;
color: white;
}

.submit-btn:hover {
background-color: #218838;
}

.skip-btn {
background-color: #ffc107;
color: black;
}

.skip-btn:hover {
background-color: #e0a800;
}

.question-container {
display: none;
}

.question-container.active {
display: block;
}
</style>
<script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
</head>
<body>
<div class="quiz-container">
  @for (int i = 0; i < quizzes.Count; i++)
  {
    <div class="question-container" id="question-@i">
      <div class="question">
        @quizzes[i].QuestionText
      </div>
      <ul class="options">
        <li>
          <label class="custom-radio">
            <input type="radio" name="quiz-option-@quizzes[i].QuestionId" value="@quizzes[i].AnswerA">
            <span>@quizzes[i].AnswerA</span>
          </label>
        </li>
        <li>
          <label class="custom-radio">
            <input type="radio" name="quiz-option-@quizzes[i].QuestionId" value="@quizzes[i].AnswerB">
            <span>@quizzes[i].AnswerB</span>
          </label>
        </li>
      </ul>
    </div>
  }
</div>
```



```

        </label>
    </li>
    <li>
        <label class="custom-radio">
            <input type="radio" name="quiz-option-@quizzes[i].QuestionId" value="@quizzes[i].AnswerC">
            <span>@quizzes[i].AnswerC</span>
        </label>
    </li>
    <li>
        <label class="custom-radio">
            <input type="radio" name="quiz-option-@quizzes[i].QuestionId" value="@quizzes[i].AnswerD">
            <span>@quizzes[i].AnswerD</span>
        </label>
    </li>
</ul>
<div class="button-container">
    <button class="skip-btn" onclick="skipQuestion(@i)">Skip Question</button>
    <button class="submit-btn" onclick="submitAnswer(@quizzes[i].QuestionId, @i)">Submit Answer</button>
</div>
</div>
}
</div>

<script>
    let currentQuestionIndex = 0;
    let numberOfCorrectAnswer = 0;

    function showQuestion(index) {
        const allQuestions = document.querySelectorAll('.question-container');
        allQuestions.forEach((q, i) => {
            q.classList.remove('active');
            if (i === index) {
                q.classList.add('active');
            }
        });
    }

    function updateCorrectAnswerCount(sectionId, numberOfCorrectAnswer) {
        $.ajax({
            url: '/QuizPage/UpdateCorrectAnswerCount',
            type: 'POST',
            data: {
                sectionId: sectionId,
                numberOfCorrectAnswer: numberOfCorrectAnswer
            },
            success: function (response) {
                if (response.success) {
                    console.log("Correct answer count updated successfully.");
                } else {
                    console.error("Failed to update correct answer count:", response.message);
                }
            },
            error: function (error) {
                console.error("Error while updating correct answer count:", error);
            }
        });
    }

```

```

    });
}

function submitAnswer(quizId, index) {
    const selectedOption = document.querySelector(`input[name="quiz-option-${quizId}"]:checked`);
    if (!selectedOption) {
        alert("Please select an option.");
        return;
    }

    fetch('/QuizPage/CheckAnswer', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            quizId: quizId,
            selectedOption: selectedOption.value
        })
    })
    .then(response => response.json())
    .then(data => {
        if (data.success) {
            alert(data.isCorrect ? "Correct!" : "Incorrect!");
            if (data.isCorrect) {
                numberOfCorrectAnswer++
                updateCorrectAnswerCount(@sectionId, numberOfCorrectAnswer);
            }
        } else {
            alert("Error: " + data.message);
        }
        goToNextQuestion();
    })
    .catch(error => console.error('Error:', error));
}

function skipQuestion(index) {
    alert(` You skipped question ${index + 1}.` );
    goToNextQuestion();
}

function goToNextQuestion() {
    currentQuestionIndex++;
    if (currentQuestionIndex < @quizzes.Count) {
        showQuestion(currentQuestionIndex);
    } else {
        alert("Quiz Completed!");
    }
}

showQuestion(currentQuestionIndex);
</script>
</body>
</html>

```

QAPage:

```
@{
    int? sectionId = ViewBag.SectionId;
    var section = new DBMSCourse.Repositories.SectionRepository(new
DBMSCourseContext()).GetSectionById(sectionId);
    var sectionName = section.SectionName;
    var sectionInfo = section.DetailedInfo;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Question Answering</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f9;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }

        .qa-container {
            background: #ffffff;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
            border-radius: 10px;
            padding: 20px;
            max-width: 600px;
            width: 90%;
        }

        .qa-title {
            font-size: 1.5em;
            font-weight: bold;
            color: #333;
            margin-bottom: 20px;
            text-align: center;
        }

        .qa-input-container {
            display: flex;
            flex-direction: column;
            gap: 15px;
            margin-bottom: 20px;
        }

        .qa-input {
            width: 100%;
            padding: 10px;
            border: 2px solid #ddd;
```

```

border-radius: 5px;
font-size: 1em;
transition: border-color 0.3s ease;
}

.qa-input:focus {
  border-color: #007bff;
  outline: none;
}

.qa-submit-btn {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  background-color: #007bff;
  color: white;
  font-size: 1em;
  cursor: pointer;
  transition: background-color 0.3s ease;
  text-align: center;
}

.qa-submit-btn:hover {
  background-color: #0056b3;
}

.qa-response {
  background-color: #f9f9f9;
  border: 1px solid #ddd;
  border-radius: 5px;
  padding: 15px;
  font-size: 1em;
  color: #333;
  min-height: 50px;
}

.qa-response-title {
  font-weight: bold;
  margin-bottom: 10px;
}

.qa-response p {
  margin: 0;
}
</style>
</head>
<body>
<div class="qa-container">
  <div class="qa-title">@sectionName</div>
  <div class="qa-title">Ask Your Question</div>
  <div class="qa-input-container">
    <input type="text" id="question" class="qa-input" placeholder="Type your question here...">
    <button class="qa-submit-btn" onclick="submitQuestion()">Submit</button>
  </div>
  <div class="qa-response" id="response">

```

```

    <div class="qa-response-title">Answer:</div>
    <p id="response">The response will appear here.</p>
  </div>
</div>

<script>
async function submitQuestion() {
  const question = document.getElementById('question').value;
  const sectionId = @ViewBag.SectionId;

  if (!question) {
    alert('Please enter a question.');
```

 return;
 }

 try {
 const response = await fetch('@Url.Action("SubmitQuestion", "QAPage")', {
 method: 'POST',
 headers: {
 'Content-Type': 'application/json'
 },
 body: JSON.stringify({ sectionId, question })
 });

 const result = await response.json();

 if (result.success) {
 document.getElementById('response').textContent = result.answer;
 } else {
 document.getElementById('response').textContent = 'Error: ' + result.message;
 }
 } catch (error) {
 document.getElementById('response').textContent = 'Error: Unable to connect to the server.';
 }
}
</script>

</body>
</html>

CheckStudentInfoPage:

```

@{
  int? sectionId = ViewBag.SectionId;
  var section = new DBMSCourse.Repositories.SectionRepository(new
  DBMSCourseContext()).GetSectionById(sectionId);
  var sectionName = section.SectionName;
  var sectionInfo = section.DetailedInfo;
}

<!DOCTYPE html>
<html lang="en">

```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student Knowledge Checking</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      background-color: #f4f4f9;
    }

    .container {
      width: 80%;
      margin: 50px auto;
      padding: 20px;
      background: #fff;
      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
      border-radius: 8px;
      display: flex;
      flex-wrap: wrap;
      gap: 20px;
    }

    .info-section, .input-section {
      flex: 1;
      min-width: 300px;
    }

    .info-section {
      border-right: 2px solid #ddd;
      padding-right: 20px;
    }

    .info-title {
      font-size: 20px;
      font-weight: bold;
      margin-bottom: 10px;
      color: #333;
    }

    .info-content {
      font-size: 16px;
      line-height: 1.6;
      color: #555;
    }

    .input-title {
      font-size: 20px;
      font-weight: bold;
      margin-bottom: 10px;
      color: #333;
    }

    .input-area {
```

```

width: 100%;
height: 200px;
padding: 10px;
font-size: 16px;
border: 1px solid #ddd;
border-radius: 4px;
resize: none;
margin-bottom: 10px;
}

.submit-btn {
padding: 10px 20px;
font-size: 16px;
background-color: #007BFF;
color: #fff;
border: none;
border-radius: 4px;
cursor: pointer;
}

.submit-btn:hover {
background-color: #0056b3;
}

.result-section {
width: 100%;
margin-top: 20px;
}

.result-title {
font-size: 18px;
font-weight: bold;
margin-bottom: 10px;
color: #333;
}

.result-content {
font-size: 16px;
color: #007BFF;
}
</style>
<script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
</head>
<body>
<div class="container">
<!-- Detailed Section Information -->
<div class="info-section">
<div class="info-title">@sectionName</div>
<div class="info-content" id="section-info">@sectionInfo</div>
</div>

<!-- Student Input Area -->
<div class="input-section">
<div class="input-title">Enter Your Knowledge</div>

```

```
<textarea class="input-area" id="student-input" placeholder="Type what you know about this
section..."></textarea>
<button class="submit-btn" onclick="checkKnowledge()">Submit</button>
</div>
</div>

<!-- Result Section -->
<div class="result-section">
  <div class="result-title">Evaluation Result:</div>
  <div class="result-content" id="evaluation-result">
    Your result will appear here after evaluation.
  </div>
</div>

<script>
async function checkKnowledge() {
  const studentInput = document.getElementById("student-input").value.trim();
  const resultElement = document.getElementById("evaluation-result");
  const sectionInfo = document.getElementById("section-info").textContent;
  const sectionId = @sectionId;

  if (!studentInput) {
    resultElement.textContent = "Please enter your knowledge before submitting.";
    resultElement.style.color = "red";
    return;
  }

  try {
    const response = await fetch('/CheckStudentInfoPage/GetOverallScore', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        studentInput: studentInput,
        sectionId: sectionId
      })
    });

    const data = await response.json();

    if (data.success) {
      resultElement.textContent = `Score: ${data.score}/100`;
      resultElement.style.color = "#007BFF";

      // Skoru veritabanına kaydet
      await fetch('/CheckStudentInfoPage/UpdateStudentScore', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          sectionId: parseInt(sectionId), // Doğru tipte gönderildiğine emin ol
          score: parseFloat(data.score)
        })
      });
    } else {
      resultElement.textContent = `Error: ${data.message}`;
    }
  }
}
```



```

        resultElement.style.color = "red";
    }
} catch (error) {
    resultElement.textContent = "Error connecting to the server.";
    resultElement.style.color = "red";
}
}

</script>
</body>
</html>

```

OverallReportPage:

```

@{
    var reports = new DBMSCourse.Repositories.OverallReportRepository(new
    DBMSCourseContext()).GetOverallReports();
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Overall Report</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f4f4f9;
            color: #333;
        }

        .container {
            width: 90%;
            max-width: 800px;
            margin: 40px auto;
            background: #ffffff;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
            border-radius: 10px;
            padding: 20px;
        }

        h1 {
            text-align: center;
            color: #007bff;
            margin-bottom: 20px;
        }

        .table-container {
            overflow-x: auto;
        }
    </style>

```

```
table {
  width: 100%;
  border-collapse: collapse;
  margin: 20px 0;
  font-size: 1em;
}

table thead {
  background-color: #007bff;
  color: white;
}

table th, table td {
  text-align: left;
  padding: 12px;
  border-bottom: 1px solid #ddd;
}

table tr:nth-child(even) {
  background-color: #f9f9f9;
}

table tr:hover {
  background-color: #f1f1f1;
}

.score {
  font-weight: bold;
}

.btn-back {
  display: inline-block;
  padding: 10px 20px;
  margin-top: 20px;
  background-color: #28a745;
  color: white;
  text-decoration: none;
  border-radius: 5px;
  text-align: center;
  transition: background-color 0.3s ease;
}

.btn-back:hover {
  background-color: #218838;
}

.footer {
  text-align: center;
  margin-top: 20px;
  font-size: 0.9em;
  color: #888;
}
</style>
</head>
```

```

<body>
  <div class="container">
    <h1>Overall Report</h1>
    <div class="table-container">
      <table>
        <thead>
          <tr>
            <th>Section</th>
            <th>Quiz Score</th>
            <th>Knowledge Check Score</th>
            <th>Overall Score</th>
          </tr>
        </thead>
        <tbody>
          @foreach (var report in reports)
          {
            <tr>
              <td>@report.Section.SectionName</td>
              <td class="score">@report.QuizScore</td>
              <td class="score">@report.KnowledgeCheckScore</td>
              <td class="score">@report.SectionOverallScore</td>
            </tr>
          }
        </tbody>
      </table>
    </div>
    <a href="/" class="btn-back">Back to Home</a>
  </div>
  <div class="footer">
    &copy; @DateTime.Now.Year DBMS Course Report
  </div>
</body>
</html>

```

Controller: It processes the HTTP requests (GET, POST, PUT, DELETE) coming from user. When a user opens a page, a GET request is sent to the view to render the webpage. When a form is submitted, a POST request sends the data to the database via models. Also manages communication with the LLM models through a RESTful API. Each view is associated with a controller.

HomeController:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace DBMSCourse.Controllers
{
    public class HomeController : Controller

```

```

{
    // GET: HomePage
    public ActionResult HomePage()
    {
        return View();
    }
}
}

```

QuizPageController:

```

using DBMSCourse.Repositories;
using System;
using System.Collections.Generic;
using System.Web.Mvc;
using DBMSCourse.Models;

namespace DBMSCourse.Controllers
{
    public class QuizPageController : Controller
    {
        private readonly QuizRepository _quizRepository;
        private readonly OverallReportRepository _reportRepository;

        public QuizPageController()
        {
            _quizRepository = new QuizRepository(new DBMSCourseContext());
            _reportRepository = new OverallReportRepository(new DBMSCourseContext());
        }

        // GET: QuizPage
        public ActionResult QuizPage(int? sectionId)
        {
            ViewBag.SectionId = sectionId;
            return View();
        }

        [HttpPost]
        public JsonResult CheckAnswer(int quizId, string selectedOption)
        {
            var quiz = _quizRepository.GetQuizById(quizId);
            if (quiz == null)
            {
                return Json(new { success = false, message = "Quiz not found." });
            }

            bool isCorrect = string.Equals(quiz.CorrectAnswer.ToLower(), selectedOption.ToLower());
            return Json(new { success = true, isCorrect });
        }

        [HttpPost]
        public JsonResult UpdateCorrectAnswerCount(int sectionId, int numberOfCorrectAnswer)
        {

```

```

try
{
    var report = _reportRepository.GetOverallReportBySectionId(sectionId);
    var quiz = _quizRepository.GetQuizzesBySection(sectionId);
    var quizCount = quiz.Count;

    double quizScore = (double)numberOfCorrectAnswer / quizCount * 100;
    if (report != null)
    {
        report.QuizScore = Convert.ToDecimal(quizScore);
        _reportRepository.UpdateOverallReportBySectionId(sectionId, report);

        return Json(new { success = true });
    }
    return Json(new { success = false, message = "Report not found." });
}
catch (Exception ex)
{
    return Json(new { success = false, message = ex.Message });
}
}
}

```

QAPage:

```

using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using System.Web.Mvc;
using Newtonsoft.Json;

namespace DBMSCourse.Controllers
{
    public class QAPageController : Controller
    {
        // GET: QAPage
        public ActionResult QAPage(int? sectionId)
        {
            ViewBag.SectionId = sectionId;
            return View();
        }

        [HttpPost]
        public async Task<JsonResult> SubmitQuestion(int sectionId, string question)
        {
            try
            {
                // Section bilgilerini veritabanından çek
                var sectionRepo = new DBMSCourse.Repositories.SectionRepository(new DBMSCourseContext());
                var section = sectionRepo.GetSectionById(sectionId);
            }
        }
    }
}

```

```

if (section == null)
{
    return Json(new { success = false, message = "Section not found." });
}

// Python API'sine gönderilecek JSON oluştur
var payload = new
{
    question = question,
    sectionInfo = section.DetailedInfo
};

var jsonPayload = JsonConvert.SerializeObject(payload);

using (var client = new HttpClient())
{
    client.BaseAddress = new Uri("http://127.0.0.1:5000/api/answer"); // Python API URL
    var content = new StringContent(jsonPayload, Encoding.UTF8, "application/json");

    // Python API'ye POST isteği gönder
    var response = await client.PostAsync("", content);

    if (!response.IsSuccessStatusCode)
    {
        return Json(new { success = false, message = "Failed to communicate with Python API." });
    }

    // Python API'den dönen cevabı al
    var result = await response.Content.ReadAsStringAsync();
    var apiResponse = JsonConvert.DeserializeObject<dynamic>(result);

    string answer = apiResponse.answer;
    bool success = apiResponse.success;
    string message = apiResponse.message;

    if (success)
    {
        return Json(new { success, answer });
    }
    else
    {
        return Json(new { success, message });
    }
}
}
catch (Exception ex)
{
    return Json(new { success = false, message = ex.Message });
}
}
}

```

CheckStudentInfoPageController:

```
using System;
using System.Web.Mvc;
using Newtonsoft.Json;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using DBMSCourse.Repositories;
using System.Runtime.Remoting.Contexts;

namespace DBMSCourse.Controllers
{
    public class CheckStudentInfoPageController : Controller
    {
        private readonly OverallReportRepository _overallReportRepository;

        public CheckStudentInfoPageController()
        {
            _overallReportRepository = new OverallReportRepository(new DBMSCourseContext());
        }

        // GET: CheckStudentInfoPage
        public ActionResult CheckStudentInfoPage(int? sectionId)
        {
            ViewBag.SectionId = sectionId;
            return View();
        }

        // POST: GetOverallScore
        [HttpPost]
        public async Task<JsonResult> GetOverallScore(EvaluationRequest request)
        {
            try
            {
                if (string.IsNullOrEmpty(request.StudentInput) || string.IsNullOrEmpty(request.SectionId))
                {
                    return Json(new { success = false, message = "Both student input and section information are required." });
                }

                var payload = new
                {
                    studentInput = request.StudentInput,
                    sectionId = request.SectionId
                };

                var jsonPayload = JsonConvert.SerializeObject(payload);

                using (var client = new HttpClient())
                {
                    client.BaseAddress = new Uri("http://127.0.0.1:5000/api/similarityScore");
                    var content = new StringContent(jsonPayload, Encoding.UTF8, "application/json");
```

```

        var respons = await client.PostAsync("", content);

        if (!respons.IsSuccessStatusCode)
        {
            return Json(new { success = false, message = "Failed to communicate with Python API." });
        }

        var result = await respons.Content.ReadAsStringAsync();
        var apiResponse = JsonConvert.DeserializeObject<dynamic>(result);

        string score = apiResponse.similarityScore;
        bool success = apiResponse.success;
        string message = apiResponse.message;

        if (success)
        {
            return Json(new { success, score });
        }
        else
        {
            return Json(new { success, message });
        }
    }
}
catch (Exception ex)
{
    return Json(new { success = false, error = ex.Message });
}
}

[HttpPost]
public JsonResult UpdateStudentScore(int sectionId, float score)
{
    try
    {
        var report = _overallReportRepository.GetOverallReportBySectionId(sectionId);
        if (report == null)
        {
            return Json(new { success = false, message = "No report found for the given sectionId." });
        }
        report.KnowledgeCheckScore = Convert.ToDecimal(score);
        _overallReportRepository.UpdateOverallReportBySectionId(sectionId, report);

        return Json(new { success = true, message = "Score updated successfully." });
    }
    catch (Exception ex)
    {
        return Json(new { success = false, message = $"Error: {ex.Message}" });
    }
}

public class EvaluationRequest
{
    [JsonProperty("studentInput")]

```



```
public string StudentInput { get; set; }

[JsonProperty("sectionId")]
public string SectionId { get; set; }
}
}
```

OverallReportPageController:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace DBMSCourse.Controllers
{
    public class OverallReportPageController : Controller
    {
        // GET: OverallReportPage
        public ActionResult OverallReportPage()
        {
            return View();
        }
    }
}
```

Repositories: Each database table has a corresponding repository file. In this file, we write methods for different operations, such as GetAllSections, GetQuizBySectionId, or UpdateOverallReportBySectionId. These methods make it easy to work with the data. This way, the code is more organized and simpler to use.

SectionRepository:

```
using DBMSCourse.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace DBMSCourse.Repositories
{
    public class SectionRepository
    {
        private readonly DBMSCourseContext _dbContext;

        public SectionRepository(DBMSCourseContext context)
        {
            _dbContext = context;
        }
    }
}
```

```

public List<Section> GetAllSections()
{
    return _dbContext.Sections.ToList();
}

public string GetSectionInfoById(int? sectionId)
{
    return _dbContext.Sections.FirstOrDefault(s => s.SectionId == sectionId).DetailedInfo;
}

public Section GetSectionById(int? sectionId)
{
    return _dbContext.Sections.FirstOrDefault(s => s.SectionId == sectionId);
}
}
}

```

QuizRepository:

```

using DBMSCourse;
using DBMSCourse.Models;
using System.Collections.Generic;
using System.Linq;

namespace DBMSCourse.Repositories
{
    public class QuizRepository
    {
        private readonly DBMSCourseContext _context;

        public QuizRepository(DBMSCourseContext context)
        {
            _context = context;
        }

        public List<Quiz> GetAllQuizzes()
        {
            return _context.Quizzes.ToList();
        }

        public List<Quiz> GetQuizzesBySection(int? sectionId)
        {
            return _context.Quizzes.Where(q => q.SectionId == sectionId).ToList();
        }

        public Quiz GetQuizById(int id)
        {
            return _context.Quizzes.FirstOrDefault(q => q.QuestionId == id);
        }
    }
}

```

OverallReportRepository:

```
using DBMSCourse.Models;
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;

namespace DBMSCourse.Repositories
{
    public class OverallReportRepository
    {
        private readonly DBMSCourseContext _dbContext;
        public OverallReportRepository(DBMSCourseContext dbContext)
        {
            _dbContext = dbContext;
        }

        public List<OverallReport> GetOverallReports()
        {
            return _dbContext.OverallReport.Include(r => r.Section).ToList();
        }

        public OverallReport GetOverallReportBySectionId(int? sectionId)
        {
            return _dbContext.OverallReport.FirstOrDefault(r => r.SectionId == sectionId);
        }

        public void UpdateOverallReportBySectionId(int? sectionId, OverallReport updatedReport)
        {
            var reportToUpdate = _dbContext.OverallReport.FirstOrDefault(s => s.SectionId == sectionId);

            reportToUpdate.QuizScore = updatedReport?.QuizScore ?? reportToUpdate.QuizScore;
            reportToUpdate.KnowledgeCheckScore = updatedReport?.KnowledgeCheckScore ??
reportToUpdate.KnowledgeCheckScore;

            decimal? overallScore = (reportToUpdate.QuizScore * Convert.ToDecimal(0.3)) +
(reportToUpdate.KnowledgeCheckScore * Convert.ToDecimal(0.7));
            reportToUpdate.SectionOverallScore = overallScore;

            _dbContext.SaveChanges();
        }
    }
}
```

DbContext: This class converts the models into database tables using Entity Framework's DbContext interface. The database context is named DatabaseNameContext.

DBMSCourseContext:

```
using DBMSCourse.Models;
using System.Data.Entity;

namespace DBMSCourse
{
    public class DBMSCourseContext:DbContext
    {
        public DbSet<Quiz> Quizzes { get; set; }
        public DbSet<Section> Sections { get; set; }
        public DbSet<OverallReport> OverallReport { get; set; }
    }
}
```

Configuration: Migration is added initially with this file.

Configuration.cs:

```
namespace DBMSCourse.Migrations
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Migrations;
    using System.Linq;

    internal sealed class Configuration : DbMigrationsConfiguration<DBMSCourse.DBMSCourseContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
        }

        protected override void Seed(DBMSCourse.DBMSCourseContext context)
        {
            // This method will be called after migrating to the latest version.

            // You can use the DbSet<T>.AddOrUpdate() helper extension method
            // to avoid creating duplicate seed data.
        }
    }
}
```

Migrations: Migrations in Entity Framework are a feature that helps update the database structure to match changes in the C# code (models). When you add or change something in your model, like a new property or table, you create a migration to apply these changes to the database.

Command to create migration

```
dotnet ef migrations add MigrationName
```

This generates a file that contains the necessary SQL commands to update the database. After that, you apply the migration with the command:

```
dotnet ef database update
```

This updates the database according to the changes defined in the migration file.

Migrations keep the database and code synchronized, making it easier to manage changes. They also allow you to roll back changes if needed. This feature helps keep the development process organized and reduces errors.

Example of Migration:

```
namespace DBMSCourse.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class NOCAremove : DbMigration
    {
        public override void Up()
        {
            DropColumn("dbo.Sections", "NumberOfCorrectAnswers");
        }

        public override void Down()
        {
            AddColumn("dbo.Sections", "NumberOfCorrectAnswers", c => c.Int(nullable: false));
        }
    }
}
```

2. LLM Part:

I have developed 3 modules for the LLM part:

Question Answering Module: This module generates answers for students based on section information. It uses a DistilBERT-based Question Answering Model to produce the responses.

The model uses the Hugging Face pipeline API for easy access to the pre-trained DistilBERT model fine-tuned on the SQuAD dataset. It takes two inputs: a question and a context then tokenizes the inputs internally (handled by pipeline), processes them through the Transformer model, and predicts the answer span based on token probabilities.

Source Code:

```
from transformers import pipeline
from sentence_transformers import SentenceTransformer, util

# Question Answering Model is loading.
qa_model = pipeline("question-answering", model="distilbert-base-cased-distilled-squad")

def generateAnswer(question, section_info):
    result = qa_model(question=question, context=section_info)
    answer = result['answer']
    return answer
```

Evaluation Module: This module evaluates the student's input by comparing it with predefined rules and generates a similarity score. It scores how closely the student's response matches the rules using sentence embeddings and cosine similarity.

The model loads predefined rules from a JSON file, converts both the student's input and the rules into dense vector representations using the SentenceTransformer model then computes cosine similarity between the vectors to find the most similar rule(s). Finally, adjusts the score based on the length of the student's input and excessive verbosity.

Source Code:

```
import json
from sentence_transformers import SentenceTransformer, util

def load_rules(input_file="rules.json"):
    """Load rules from JSON"""
    try:
        with open(input_file, "r") as file:
```

```

        rules = json.load(file)
    return rules
except FileNotFoundError:
    print("Rules file was not found.")
    return {}

def evaluate_student_input(sectionId, student_input, rules_file="rules.json"):
    # 1. Load the rules
    rules = load_rules(rules_file)
    if sectionId not in rules:
        print(f"Could not found rules for '{sectionId}'")
        return 0.0

    section_rules = rules[f"{sectionId}"]

    # 2. Sentence Transformers model
    similarity_model = SentenceTransformer('all-MiniLM-L6-v2')

    # 3. Embedding the student_input and rules
    student_embedding = similarity_model.encode(student_input, convert_to_tensor=True)
    rule_embeddings = similarity_model.encode(section_rules, convert_to_tensor=True)

    # 4. Calculating the similarity score.
    scores = util.cos_sim(student_embedding, rule_embeddings).max(dim=1)[0]

    # 5. Average of the scores
    average_score = scores.mean().item() * 100

    # 6. Add penalty based on length of student_input
    # Penalty for short inputs
    if len(student_input.split()) < 10: # Limits for short inputs
        average_score *= 0.5 # Halve the score

    # 7. Add penalty for too much information
    # If text length exceeds rules, apply penalty
    rule_terms = sum([len(rule.split()) for rule in section_rules])
    student_terms = len(student_input.split())

    if student_terms > rule_terms * 1.5: # If the student text is 1.5 times longer
        average_score -= 10 # Lower the score by 10 points

    # Limiting the score to not exceed 100%
    average_score = min(average_score, 100)
    return average_score

## Main function
if __name__ == "__main__":
    # student_input = "SQL is used for querying and manipulating data,there are four operations, they are
    select,delete,insert,update. delete is used for removing data,"
    # sectionId = "4"
    # score = evaluate_student_input(sectionId, student_input)
    # print(f"Değerlendirme sonucu: {score:.2f}%")

## Main function

```

```

# if __name__ == "__main__":
#     student_input = "SQL is used for querying and manipulating data, there are four operations, they are
select, delete, insert, update. SELECT is used for retrieving data, delete is used for removing data,"
#     sectionId = "4"
#     score = evaluate_student_input(sectionId, student_input)
#     print(f"Değerlendirme sonucu: {score:.2f}%")

```

rules.json:

```

{
  "2": [
    "A Database Management System (DBMS) is software designed to store, retrieve, manage, and manipulate data in a structured manner",
    "It replaces traditional file systems by addressing issues such as data redundancy, inconsistency, and lack of scalability",
    "A DBMS provides several key features, including data abstraction, data independence, and support for multiple users",
    "It consists of components like the database engine, which processes queries, and the storage manager, which handles the organization of data on physical storage",
    "Real-world applications include inventory management systems, customer relationship management (CRM) tools, and banking systems",
    "By understanding the role and components of a DBMS, one can appreciate its importance in managing large-scale data efficiently"
  ],
  "3": [
    "The relational model represents data in the form of tables, known as relations, which consist of rows (tuples) and columns (attributes)",
    "Each table has a primary key that uniquely identifies its rows, and foreign keys establish relationships between different tables",
    "This model ensures data consistency and eliminates redundancy by organizing information into logically related tables",
    "Operations such as selection, projection, and join allow for efficient data retrieval and manipulation",
    "The relational model also enforces constraints like entity integrity, which ensures no null values in primary keys, and referential integrity, which maintains valid relationships between tables",
    "Understanding the relational model is fundamental to designing structured and scalable databases"
  ],
  "4": [
    "Structured Query Language (SQL) is the standard language used to interact with relational databases",
    "It provides commands for data definition, manipulation, and querying",
    "For example, the CREATE TABLE statement defines the structure of a database table, while INSERT, UPDATE, and DELETE allow for modifying data",
    "The SELECT statement is used to query data, with additional clauses like WHERE for filtering, GROUP BY for aggregation, and ORDER BY for sorting",
    "SQL also supports joins, enabling the combination of data from multiple tables",
    "Mastery of SQL is essential for working with relational databases, as it allows users to extract meaningful insights and manage data effectively"
  ],
  "5": [
    "Normalization is the process of organizing database tables to reduce redundancy and improve data integrity",
    "It involves dividing a database into smaller, related tables and defining relationships between them",
    "The process is guided by normal forms, such as the First Normal Form (1NF), which eliminates duplicate columns; the Second Normal Form (2NF), which removes partial dependencies; and the Third Normal Form (3NF), which eliminates transitive dependencies",
    "By normalizing a database, one can ensure that it is free from update, delete, and insert anomalies"
  ]
}

```



```

    "Although normalization improves efficiency and consistency, it may sometimes require denormalization for performance optimization in certain applications"
  ],
  "6": [
    "Transactions are sequences of database operations treated as a single unit of work",
    "They adhere to the ACID properties: Atomicity ensures all operations are completed or none are applied;
\n Consistency guarantees the database remains in a valid state; Isolation prevents concurrent transactions from interfering with each other;
\n and Durability ensures changes are permanent even in case of a failure",
    "Concurrency control mechanisms, such as locking and timestamps, are used to manage simultaneous access to the database",
    "Common issues include dirty reads, where a transaction reads uncommitted data, and deadlocks,
\n where two transactions wait indefinitely for resources held by each other",
    "Effective transaction management ensures data integrity and system reliability in multi-user environments"
  ]
}

```

Main Program: This module manages API communication using the Flask server library. It receives data from ASP.NET through the API and processes it using either the Question-Answering Module or the Evaluation Module. Finally, it sends the results back to ASP.NET via the API gateway.

Source Code:

```

from flask import Flask, request, jsonify

from question_answering import generateAnswer
from similarity import calculateSimilarityScore
from evolution_with_rules import evaluate_student_input

app = Flask(__name__)

# REST API for QA
@app.route('/api/answer', methods=['POST'])
def provide_answer():
    try:
        # ASP.NET API'den gelen JSON verilerini al
        data = request.get_json()

        question = data.get('question')
        section_info = data.get('sectionInfo')

        if not question or not section_info:
            return jsonify({"success": False, "message": "Invalid input data."}), 400

        # QA modelini kullanarak soruyu yanıtla
        answer = generateAnswer(question=question, section_info=section_info)

        return jsonify({"success": True, "answer": answer})

    except Exception as e:

```

```
        return jsonify({"success": False, "message": str(e)}), 500

# REST API for TS
import traceback

@app.route('/api/similarityScore', methods=['POST'])
def calculate_similarity():
    try:
        # ASP.NET API'den gelen JSON verilerini al
        data = request.get_json()

        studentInput = data.get('studentInput')
        sectionId = data.get('sectionId')

        if not studentInput or not sectionId:
            return jsonify({"success": False, "message": "Invalid input data."}), 400

        # Modeli Kullanarak benzerliği hesapla

        #score = calculateSimilarityScore(sectionInfo,studentInput)
        score = evaluate_student_input(sectionId=sectionId,student_input=studentInput)
        score = round(score, 2)

        return jsonify({"success": True, "similarityScore": score})

    except Exception as e:
        print("Error occurred:", traceback.format_exc()) # Hata detaylarını loglayın
        return jsonify({"success": False, "message": str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

Describe how your program runs including the libraries required for it to run

For the websites, the program uses ASP.NET with MVS architecture to display the websites. User requests (which page the user wanted to display, some operations such as GetOverallScore, checkQuizAnswer, UpdateStudentScore) are sent with HTTP protocols to the controller of each page. Then the controller sends back a response for each request in json format. We use controllers to navigate the user between pages too with action methods. To access the database, entity framework was used. We have a repository script for each table in the database. We can access the data with this repository scripts easily.

For the communication between the AI part of the program, RESTful API is used with Flask library. It is required installing flask library with “pip install flask” to use API gateway. ASP.NET sends the data to the python program with RESTful API, then the python program gets the data with “post” method. Two model uses same way to get the data from ASP.NET but they use different api address. For the Question-Answering model “/api/answer” is used to send the answer. For the text Similarity model “/api/similarityScore” is used to send the score.

The AI part of the program is written in Python, and it works with LLM models. Two types of models are used: Question-Answering and Text Similarity.

For the Question-Answering model, the program uses the Hugging Face transformers library. It uses a pre-trained model called distilbert-base-cased-distilled-squad to find answers for given questions and context (section information). To use this, you need to install the transformers library with the command: pip install transformers.

For the Text Similarity model, the program uses the Hugging Face sentence-transformers library. It uses the all-MiniLM-L6-v2 model to measure cosine similarity. To use this model, you need to install the sentence-transformers library with the command: pip install sentence-transformers. This library also needs another library, torch, for tensor operations.

The APIs receive input and return output in JSON format. This format makes it easy to integrate the program with other systems or applications.

The program also uses some predefined rules to check student inputs. Without these rules, the similarity model might give wrong scores, especially if the input is too short. To fix this, the program adds penalties for very short or very long inputs. This way, the scores are more accurate and useful.

API Examples:

- **For Question-Answering:**

Input:

```
{  
  "question": "What is Python?",  
  "sectionInfo": "Python is a programming language that lets you work quickly."  
}
```

Output:

```
{  
  "success": true,  
  "answer": "a programming language"  
}
```

- **For Text Similarity:**

Input:

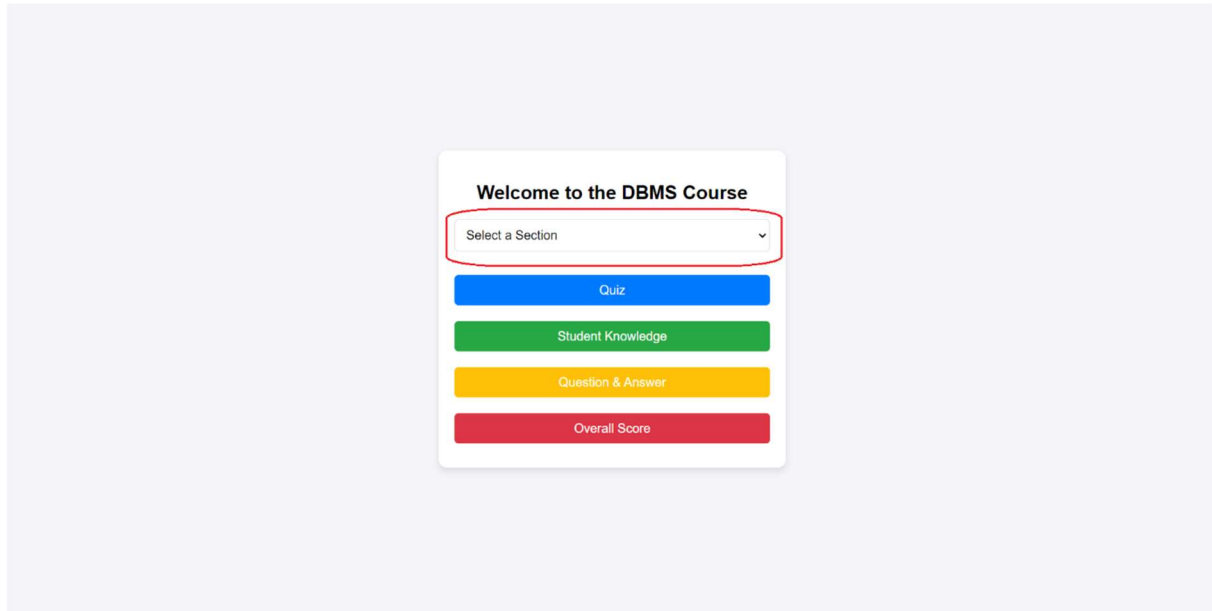
```
{  
  "studentInput": "Python is a fast and easy programming language.",  
  "sectionId": "2"  
}
```

Output:

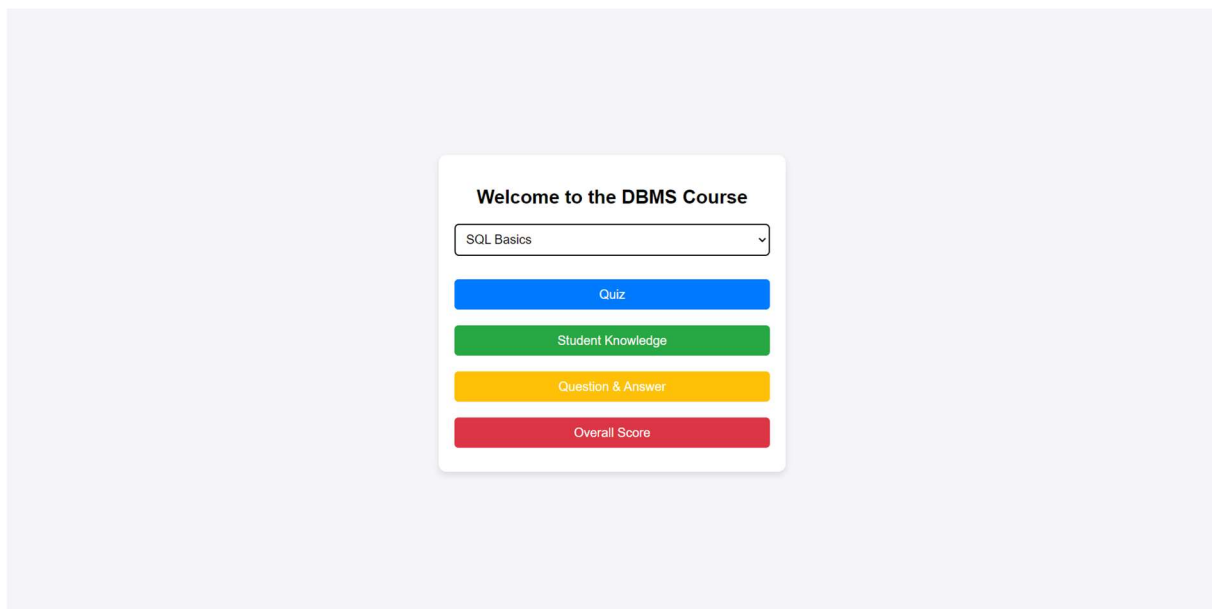
```
{  
  "success": true,  
  "similarityScore": 85.75  
}
```

Write a user manual how a user uses your program

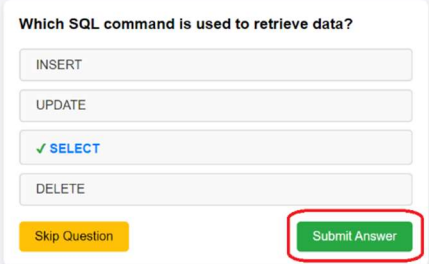
This website is for learning about Database Management Systems. The user can learn Database Management Systems section by section. To select a section, click the “Select a Section” and select a section to learn.



After a section is selected, you can solve quizzes, ask question about the section and measure the knowledge score for the section.



In quiz parts, there are 5 questions for each section. To solve the questions read the question and click the answer you think it is true and click the “Submit Answer” button.



Which SQL command is used to retrieve data?

INSERT

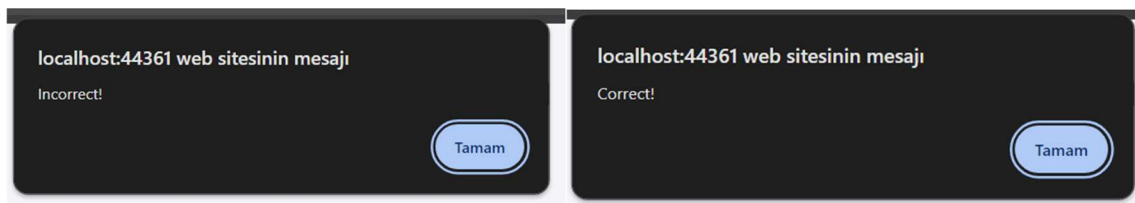
UPDATE

✓ SELECT

DELETE

Skip Question Submit Answer

After submitting the answer, you will get “Correct” or “Incorrect” messages.



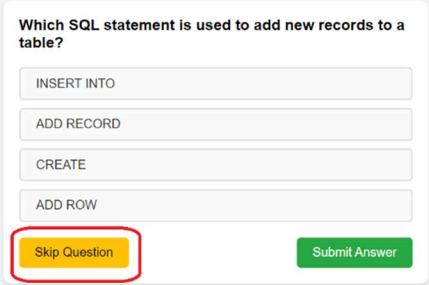
localhost:44361 web sitesinin mesaji

Incorrect! Tamam

localhost:44361 web sitesinin mesaji

Correct! Tamam

Or you can skip the question without answering by clicking the “Skip Question” button.



Which SQL statement is used to add new records to a table?

INSERT INTO

ADD RECORD

CREATE

ADD ROW

Skip Question Submit Answer

In Student Knowledge part, you can measure your knowledge. To do this, you need to write your knowledge to the right part and enter the “Submit” button.

SQL Basics

Structured Query Language (SQL) is the standard language used to interact with relational databases. It provides commands for data definition, manipulation, and querying. For example, the CREATE TABLE statement defines the structure of a database table, while INSERT, UPDATE, and DELETE allow for modifying data. The SELECT statement is used to query data, with additional clauses like WHERE for filtering, GROUP BY for aggregation, and ORDER BY for sorting. SQL also supports joins, enabling the combination of data from multiple tables. Mastery of SQL is essential for working with relational databases, as it allows users to extract meaningful insights and manage data effectively.

Enter Your Knowledge

Type what you know about this section...

Submit

Evaluation Result:
Your result will appear here after evaluation.

Also, you can see the content of section in the left side.

SQL Basics

Structured Query Language (SQL) is the standard language used to interact with relational databases. It provides commands for data definition, manipulation, and querying. For example, the CREATE TABLE statement defines the structure of a database table, while INSERT, UPDATE, and DELETE allow for modifying data. The SELECT statement is used to query data, with additional clauses like WHERE for filtering, GROUP BY for aggregation, and ORDER BY for sorting. SQL also supports joins, enabling the combination of data from multiple tables. Mastery of SQL is essential for working with relational databases, as it allows users to extract meaningful insights and manage data effectively.

Enter Your Knowledge

Delete is used for removing data

Submit

Evaluation Result:
Score: 22.95/100

After entering your knowledge, you can see your knowledge score in the marked area bellow.

SQL Basics

Structured Query Language (SQL) is the standard language used to interact with relational databases. It provides commands for data definition, manipulation, and querying. For example, the CREATE TABLE statement defines the structure of a database table, while INSERT, UPDATE, and DELETE allow for modifying data. The SELECT statement is used to query data, with additional clauses like WHERE for filtering, GROUP BY for aggregation, and ORDER BY for sorting. SQL also supports joins, enabling the combination of data from multiple tables. Mastery of SQL is essential for working with relational databases, as it allows users to extract meaningful insights and manage data effectively.

Enter Your Knowledge

Delete is used for removing data

Submit

Evaluation Result:

Score: 22.95/100

In the Question & Answer part, you can ask questions about the section. Enter your question and click the “Submit” button.

SQL Basics

Ask Your Question

Type your question here...

Submit

Answer:
The response will appear here.

After entering your question, the answer will appear in the marked area below.

SQL Basics

Ask Your Question

What is used for retrieving data?

Submit

The SELECT statement

In the Overall Score part, you can see all scores that is produced from quizzes and knowledge measurement part. Also, there is an Overall Score that is produced from quiz score and knowledge score.

Overall Report			
Section	Quiz Score	Knowledge Check Score	Overall Score
Introduction to DBMS	40,00	73,46	63,42
Relational Model	0,00	0,00	0,00
SQL Basics	20,00	22,95	22,06
Normalization	0,00	0,00	0,00
Transactions and Concurrency	0,00	0,00	0,00

Back to Home