**New Example: Adding a "String Functions" Question Set**

**Step 1: Create the New Question Set File**

1. **Navigate to the questionBank Folder**:
   * Inside your project directory, locate the questionBank folder where all question sets are stored.
2. **Create a New JavaScript File**:
   * Create a new file named stringFunctions.js inside the questionBank folder.
3. **Define the New Question Set**:
   * In the newly created file, define the question set with appropriate tables, questions, answers, and tips.

**Example: stringFunctions.js**

javascript

Copy code

export const stringFunctions = [

{

id: 1,

difficulty: "easy",

tables: [

{

name: "employees",

columns: ["id", "name", "department", "salary"],

data: [

[1, "John Doe", "Sales", 50000],

[2, "Jane Smith", "Marketing", 60000],

[3, "Bob Johnson", "IT", 70000],

[4, "Alice Brown", "Sales", 55000],

[5, "David Wilson", "Sales", 45000]

]

}

],

question: "Write a SQL query to extract the first name from the 'name' column.",

answer: `

SELECT SUBSTRING(name, 1, INSTR(name, ' ') - 1) as first\_name

FROM employees

`,

resultComparisonOptions: {

ignoreColumnNames: true,

approximateComparison: true

},

tips: [

"Use the SUBSTRING() function to extract the first part of the name.",

"Use the INSTR() function to find the position of the space in the name."

]

},

// Additional questions can be added here

];

**Step 2: Update questionSets.js**

1. **Open questionSets.js**:
   * Open the questionSets.js file in your project directory.
2. **Import the New Question Set**:
   * Add an import statement for the new question set at the top of the file.

**Updated questionSets.js**

javascript

Copy code

import { basics } from './questionBank/basics.js';

import { joins } from './questionBank/joins.js';

import { advanced } from './questionBank/advanced.js';

import { subqueries } from './questionBank/subqueries.js';

import { windowFunctions } from './questionBank/windowFunctions.js';

import { aggregateFunctions } from './questionBank/aggregateFunctions.js';

import { stringFunctions } from './questionBank/stringFunctions.js'; // Import the new question set

export const questionSets = {

basics,

joins,

advanced,

subqueries,

windowFunctions,

aggregateFunctions,

stringFunctions // Include the new question set

};

1. **Save the File**:
   * Save the questionSets.js file after adding the import statement and updating the questionSets object.

**Step 3: Update progressTracker.js**

1. **Open progressTracker.js**:
   * Open the progressTracker.js file.
2. **Add the New Question Set to the progressTracker Object**:
   * Ensure the new question set is represented in the progressTracker object.

**Updated progressTracker.js**

javascript

Copy code

import { questionSets } from './questionSets.js';

let progressTracker = {

basics: new Set(),

joins: new Set(),

advanced: new Set(),

subqueries: new Set(),

windowFunctions: new Set(),

aggregateFunctions: new Set(),

stringFunctions: new Set() // Add the new question set

};

export function updateProgress() {

const totalQuestions = Object.keys(questionSets).reduce((total, set) => total + questionSets[set].length, 0);

const completedQuestions = Object.values(progressTracker).reduce((total, set) => total + set.size, 0);

document.getElementById('completed-questions').textContent = completedQuestions;

document.getElementById('total-questions').textContent = totalQuestions;

// Update progress for each set

Object.keys(questionSets).forEach(set => {

const setProgress = progressTracker[set].size;

const setTotal = questionSets[set].length;

document.getElementById(`progress-${set}`).textContent = `${setProgress}/${setTotal}`;

});

}

export function checkAnswer(questionId, userResult, correctResult, options, currentSet) {

if (compareResults(userResult, correctResult, options)) {

if (!progressTracker[currentSet].has(questionId)) {

progressTracker[currentSet].add(questionId);

updateProgress();

}

return "Correct! Well done.";

} else {

return "Incorrect. The results don't match the expected output.";

}

}

function compareResults(userResult, correctResult, options = {}) {

if (userResult.length !== correctResult.length) {

return false;

}

for (let i = 0; i < userResult.length; i++) {

const userValues = userResult[i].values;

const correctValues = correctResult[i].values;

if (userValues.length !== correctValues.length) {

return false;

}

for (let j = 0; j < userValues.length; j++) {

const userRow = userValues[j];

const correctRow = correctValues[j];

if (userRow.length !== correctRow.length) {

return false;

}

for (let k = 0; k < userRow.length; k++) {

if (options.approximateComparison && typeof userRow[k] === 'number' && typeof correctRow[k] === 'number') {

if (Math.abs(userRow[k] - correctRow[k]) > 0.001) {

return false;

}

} else if (userRow[k] !== correctRow[k]) {

return false;

}

}

}

}

return true;

}

1. **Save the File**:
   * Save the progressTracker.js file after updating the progressTracker object.

**Step 4: Update index.html**

1. **Open index.html**:
   * Open your index.html file.
2. **Add the Progress Tracker for the New Question Set**:
   * Add a new <p> element under the "Your Progress" section for tracking progress on the new question set.

**Updated index.html**

html

Copy code

<h3>Your Progress:</h3>

<p>Total completed questions: <span id="completed-questions">0</span></p>

<p>Total questions: <span id="total-questions">0</span></p>

<p>SQL Basics progress: <span id="progress-basics">0/0</span></p>

<p>SQL Joins progress: <span id="progress-joins">0/0</span></p>

<p>Advanced SQL progress: <span id="progress-advanced">0/0</span></p>

<p>Subqueries progress: <span id="progress-subqueries">0/0</span></p>

<p>Window Functions progress: <span id="progress-windowFunctions">0/0</span></p>

<p>Aggregate Functions progress: <span id="progress-aggregateFunctions">0/0</span></p>

<p>String Functions progress: <span id="progress-stringFunctions">0/0</span></p> <!-- New progress tracker -->

1. **Save the File**:
   * Save the index.html file after adding the new progress tracker.

**Step 5: Test the New Question Set**

1. **Run the Application**:
   * Start or refresh your application to see the new question set automatically added as a button in the UI.
2. **Verify Button Generation**:
   * Check that a new button labeled "STRING FUNCTIONS" (based on the new file name) appears under "Choose a question set:".
3. **Select the New Question Set**:
   * Click on the button for the new question set to load the questions.
4. **Check the Progress Tracker**:
   * Ensure that the progress tracker updates correctly as you answer questions from the new set.
5. **Debug if Necessary**:
   * If anything doesn’t work as expected, open the browser’s developer console to check for errors and resolve them as needed.

**Step 6: Commit the Changes**

1. **Review Your Changes**:
   * Double-check all the changes you made, including the new question set file, updates to questionSets.js, progressTracker.js, and index.html.
2. **Commit to Version Control**:
   * Commit your changes to your version control system with a clear and descriptive message.

**Example Commit Message**:

vbnet

Copy code

Added new question set for String Functions and updated platform to support it.

**Summary of Steps**

1. **Create the New Question Set File**:
   * Define questions, tables, answers, and tips within the file.
2. **Update questionSets.js**:
   * Import the new question set and add it to the questionSets object.
3. **Update progressTracker.js**:
   * Add the new question set to the progressTracker object.
4. **Update index.html**:
   * Add a progress tracker entry for the new question set.
5. **Test the New Question Set**:
   * Run your application and verify that everything is functioning correctly.
6. **Commit the Changes**:
   * Review and commit your work to version control.

This process ensures that each new question set is correctly integrated into your SQL Learning Platform, and the platform remains well-organized and easy to maintain as you continue to expand its content.