Project Part 1: Data and Scalability

Team ID: Minghao_Chihao

Team Member: Minghao Sun, Chihao Tu

```
Upgrading...
Object store 'TodoList' created with 'status' index.
Object store 'TodoListCompleted' created.
Database opened successfully!
All 100,000 tasks have been added to the TodoList.
Copied 1000 completed tasks to 'TodoListCompleted'.
Time taken to read all 'completed' tasks from 'TodoListCompleted': 17 milliseconds
Total completed tasks read from 'TodoListCompleted': 1000
```

```
// Open (or create) the IndexedDB
const openRequest = indexedDB.open("TodoDB", 1);

// Handle the onupgradeneeded event to create the object stores
openRequest.onupgradeneeded = function (event) {
  console.log("Upgrading...");
  let db = event.target.result;

  // Create the object store named "TodoList" with an auto-incrementing id
  if (!db.objectStoreNames.contains("TodoList")) {
    let objectStore = db.createObjectStore("TodoList", {
      keyPath: "id",
      autoIncrement: true,
    });
    objectStore.createIndex("status", "status", { unique: false });
    console.log("Object store 'TodoList' created with 'status' index.");
  }

  // Create the object store named "TodoListCompleted"
  if (!db.objectStoreNames.contains("TodoListCompleted")) {
    db.createObjectStore("TodoListCompleted", { keyPath: "id" });
    console.log("Object store 'TodoListCompleted' created.");
  }
};
```

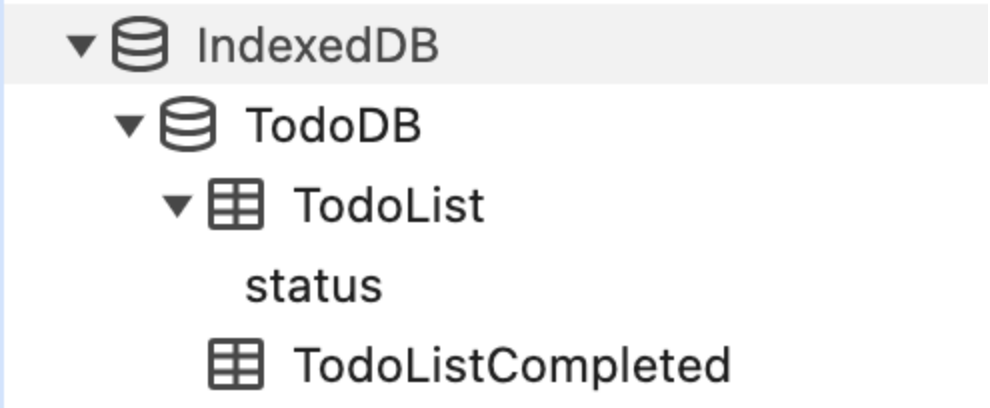
```
openRequest.onsuccess = function (event) {
  console.log("Database opened successfully!");
  let db = event.target.result;

  // Start a new transaction to add data
  let transaction = db.transaction("TodoList", "readwrite");
  let objectStore = transaction.objectStore("TodoList");

  // Function to generate random tasks with specified statuses
  function generateRandomTask(id) {
    const tasks = [
      "Finish the monthly report",
      "Update website content",
      "Prepare for client presentation",
      "Review team feedback",
      "Organize office meeting",
      "Plan the next sprint",
      "Refactor codebase",
      "Perform code review",
      "Analyze marketing data",
      "Test new software release",
    ];
    const randomTask = tasks[Math.floor(Math.random() * tasks.length)];

    // Generate a random future due date
    const dueDate = new Date(
      Date.now() + Math.floor(Math.random() * 30) * 24 * 60 * 60 * 1000
```

```
  )
    .toISOString()
    .split("T")[0];

  // Set status based on the id: first 1000 are 'completed', the rest 'in progress'
  const status = id <= 1000 ? "completed" : "in progress";

  return {
    id: id,
    task: randomTask,
    status: status,
    dueDate: dueDate,
  };
}

// Insert 100,000 randomly generated tasks into the database
for (let i = 1; i <= 100000; i++) {
  let newTask = generateRandomTask(i);
  objectStore.add(newTask);
}

transaction.oncomplete = function () {
  console.log("All 100,000 tasks have been added to the TodoList.");

  // Start a new transaction to copy completed tasks
  let copyTransaction = db.transaction(
    ["TodoList", "TodoListCompleted"],
    "readwrite"
```

```
);
~/Desktop/2024 Fall Semester/Database Management System/project
let todoStore = copyTransaction.objectStore("TodoList");
let completedStore = copyTransaction.objectStore("TodoListCompleted");

// Read all completed tasks and copy them to TodoListCompleted
let completedTasks = [];
let request = todoStore
  .index("status")
  .openCursor(IDBKeyRange.only("completed"));

request.onsuccess = function (event) {
  let cursor = event.target.result;
  if (cursor) {
    // Add the completed task to the completed store
    completedStore.add(cursor.value);
    completedTasks.push(cursor.value); // Store the completed task
    cursor.continue();
  } else {
    console.log(
      `Copied ${completedTasks.length} completed tasks to 'TodoListCompleted'.`
    );

    // Now let's measure the time to read tasks from 'TodoListCompleted'
    let readTransaction = db.transaction("TodoListCompleted", "readonly");
    let readStore = readTransaction.objectStore("TodoListCompleted");

    // Start time measurement
    const startTime = performance.now();
```

```
      let readCompletedTasks = [];
      let readRequest = readSt (local function)(event: any): void
      readRequest.onsuccess = function (event) {
        let readCursor = event.target.result;
        if (readCursor) {
          readCompletedTasks.push(readCursor.value); // Store the read task
          readCursor.continue();
        } else {
          // End time measurement
          const endTime = performance.now();
          const duration = endTime - startTime; // Calculate duration
          console.log(
            `Time taken to read all 'completed' tasks from 'TodoListCompleted': ${duration} milliseconds`
          );

          // Optionally display the number of completed tasks
          console.log(
            `Total completed tasks read from 'TodoListCompleted': ${readCompletedTasks.length}`
          );
        }
      };

      readTransaction.onerror = function (event) {
        console.error("Read transaction failed: ", event.target.error);
      };
    }
  };
```

```
    copyTransaction.onerror = function (event) {
      console.error("Copy transaction failed: ", event.target.error);
    };
  };

  transaction.onerror = function (event) {
    console.error("Transaction failed: ", event.target.error);
  };
};

openRequest.onerror = function (event) {
  console.error("Database failed to open: ", event.target.error);
};
```



#	Key (Key ...	Value
0	1	▶ {id: 1, task: 'Update website content', status: 'completed', dueDate: '2024-10-16'}
1	2	▶ {id: 2, task: 'Analyze marketing data', status: 'completed', dueDate: '2024-10-19'}
2	3	▶ {id: 3, task: 'Perform code review', status: 'completed', dueDate: '2024-10-07'}
3	4	▶ {id: 4, task: 'Plan the next sprint', status: 'completed', dueDate: '2024-10-06'}
4	5	▶ {id: 5, task: 'Perform code review', status: 'completed', dueDate: '2024-10-05'}
5	6	▶ {id: 6, task: 'Finish the monthly report', status: 'completed', dueDate: '2024-10-15'}
6	7	▶ {id: 7, task: 'Prepare for client presentation', status: 'completed', dueDate: '2024-10-19'}
7	8	▶ {id: 8, task: 'Perform code review', status: 'completed', dueDate: '2024-10-27'}
8	9	▶ {id: 9, task: 'Plan the next sprint', status: 'completed', dueDate: '2024-10-24'}
9	10	▶ {id: 10, task: 'Finish the monthly report', status: 'completed', dueDate: '2024-10-06'}
10	11	▶ {id: 11, task: 'Analyze marketing data', status: 'completed', dueDate: '2024-10-16'}
11	12	▶ {id: 12, task: 'Test new software release', status: 'completed', dueDate: '2024-10-06'}
12	13	▶ {id: 13, task: 'Review team feedback', status: 'completed', dueDate: '2024-10-22'}
13	14	▶ {id: 14, task: 'Analyze marketing data', status: 'completed', dueDate: '2024-10-14'}
14	15	▶ {id: 15, task: 'Review team feedback', status: 'completed', dueDate: '2024-10-21'}
15	16	▶ {id: 16, task: 'Finish the monthly report', status: 'completed', dueDate: '2024-10-09'}
16	17	▶ {id: 17, task: 'Review team feedback', status: 'completed', dueDate: '2024-10-03'}
17	18	▶ {id: 18, task: 'Perform code review', status: 'completed', dueDate: '2024-10-20'}
18	19	▶ {id: 19, task: 'Plan the next sprint', status: 'completed', dueDate: '2024-10-04'}
19	20	▶ {id: 20, task: 'Analyze marketing data', status: 'completed', dueDate: '2024-10-11'}

From this project, I learned valuable lessons about handling large datasets and improving the efficiency of data queries in NoSQL databases. Using IndexedDB, I gained hands-on experience defining object stores and working with key performance optimization techniques such as applying a read-only flag, creating indexes, and managing dedicated object stores for specific data subsets. By simulating a "Todo List" domain with 100,000 objects, I practiced measuring query performance and exploring how different strategies can impact read times, particularly when querying completed tasks.

This project also emphasized the importance of structuring databases effectively to enhance query performance and reduce overhead. Implementing an index on the `status` field and using a dedicated store for completed tasks demonstrated how optimizations can significantly improve retrieval times. Finally, the project helped reinforce teamwork skills, as collaborating with others to meet project objectives is essential in real-world scenarios.