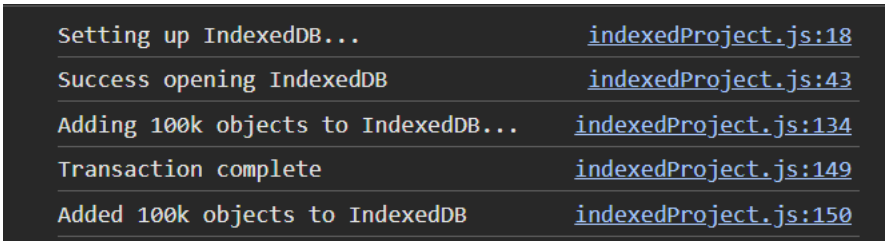
Project-1

Team: Sumeet Khedkar, Venkat Phani Krishna P

1) **Setting Up Indexed DB and adding 100k objects**
    a. We setup indexed dB and created project 1 with todo_list and todo_list_completed object store with index for status
    b. We wrote function to randomly generate 100 k objects
    c. We added the object each time and check if object already exists, we skip it

    Lessons learned: -

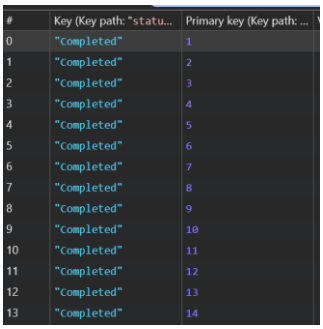
    We learned how to create and setup indexed dB and create indexed values. We learnt how to add huge data set in the database.



2) **Set 1000 objects to status "completed" and the remaining ones to status "progress"**
    a. We updated the data and by using "**.put(key, value)**" to update the first 1000 object status to 1000 and remaining object to in-progress. We used for loop for the first 1000 objects and cursor for remaining objects

    Lessons learned: -

    We learned how to update objects and use different ways to iterate over the db object store



#	Key (Key path: "statu...	Primary key (Key path: ...
0	"Completed"	1
1	"Completed"	2
2	"Completed"	3
3	"Completed"	4
4	"Completed"	5
5	"Completed"	6
6	"Completed"	7
7	"Completed"	8
8	"Completed"	9
9	"Completed"	10
10	"Completed"	11
11	"Completed"	12
12	"Completed"	13
13	"Completed"	14

1064	"In Progress"	1064
1065	"In Progress"	1065
1066	"In Progress"	1066
1067	"In Progress"	1067
1068	"In Progress"	1068
1069	"In Progress"	1069
1070	"In Progress"	1070
1071	"In Progress"	1071
1072	"In Progress"	1072
1073	"In Progress"	1073
1074	"In Progress"	1074
1075	"In Progress"	1075
1076	"In Progress"	1076

3) **Measure and display the time (in milliseconds) required to read all objects with `status` set to "completed" on the console or the browser**
    a. We required the data with check to filter only data which is completed we used both map and for loop to check it and check the performance for the same using performance.now()

Lesson learned:-

Learned how to use map and filter on object store value and also learned how to measure performance and display it in the table.

```
/**
 * Processes a database transaction on the specified object store and counts the number of records with a value of "Completed".
 * If a record with a value other than "Completed" is encountered, the provided callback function is called.
 *
 * @param {IDBDatabase} db - The IndexedDB database instance.
 * @param {string} storeName - The name of the object store to be accessed.
 * @param {function} callback - The callback function to be executed if a record with a value other than "Completed" is encountered.
 */
const f1 = (db, storeName, callback) => {
    let transaction = db.transaction(storeName, "readwrite");
    let objectStore = transaction.objectStore(storeName);
    let count = 0;

    let request = objectStore.openCursor();
    request.onsuccess = function (event) {
        let cursor = event.target.result;
        if (cursor.value === "Completed") {
            count++;
            cursor.continue();
        } else {
            callback(db);
        }
    };
```

4) **Apply a read-only flag to the object store and measure and display the time to read all completed tasks again on the console or the browser**
    a. We then used read-only filter to requery the data and check the performance for the same

    Lesson learned:-

    Learned how to set read-only objectstore and requery with it.

```
/**
 * Processes a read-only transaction on the specified object store and counts the entries with a value of "Completed".
 * If an entry does not have the value "Completed", the provided callback is invoked with the database instance.
 *
 * @param {IDBDatabase} db - The IndexedDB database instance.
 * @param {string} storeName - The name of the object store to be accessed.
 * @param {function} callback - The callback function to be called with the database instance if an entry does not have the value "Completed".
 */
const f2= (db, storeName, callback) => {
    let transaction = db.transaction(storeName, "readonly");
    let objectStore = transaction.objectStore(storeName);
    let count = 0;

    let request = objectStore.openCursor();
    request.onsuccess = function (event) {
        let cursor = event.target.result;
        if (cursor.value === "Completed") {
            count++;
            cursor.continue();
        } else {
            callback(db);
        }
    };
}
```

5) **Create an index on the `status` field, then measure and display the time to read all completed tasks on the console or the browser**
    a. **Used indexing on the object store to requery the data**

    Lesson learned:-

    Learned how to set indexing and how it affects the functionality.

```
/**
 * Retrieves the count of entries with the status "Completed" from the specified object store index.
 *
 * @param {IDBDatabase} db - The IndexedDB database instance.
 * @param {string} storeName - The name of the object store to query.
 * @param {function} callback - The callback function to execute once the count is retrieved.
 */
const f3 = (db, storeName, callback) => {
    let transaction = db.transaction(storeName, "readonly");
    let objectStore = transaction.objectStore(storeName);
    let index = objectStore.index("status");
    let count = 0;

    let request = index.openCursor();
    request.onsuccess = function (event) {
        let cursor = event.target.result;
        if (cursor.value === "Completed") {
            count++;
            cursor.continue();
        } else {
            callback(db);
        }
    };
}
```

6) **<u>Define a new object store called "TodoListCompleted", copy all completed tasks from "TodoList" to this new store, and measure and display the time required to read all completed tasks from "TodoListCompleted" on the console or the browser</u>**
    a. Created new object store and copied all the task from todo_list to todo_list_completed
    b. Took performance measurement for the same

<u>Lesson learned: -</u>

Learned how to open new object store and copy the data and how it affects the performance and time it takes

```
/**
 * Retrieves all completed tasks from the specified object store in the database.
 *
 * @param {IDBDatabase} db - The IndexedDB database instance.
 * @param {string} storeName - The name of the object store to query.
 * @returns {Array<Object>} An array of completed task objects.
 */
const getAllCompletedTasks = (db, storeName) => {
    let completedTasks = [];
    let transaction = db.transaction(storeName, "readonly");
    let objectStore = transaction.objectStore(storeName);
    let index = objectStore.index("status");
    let request = index.getAll();
    request.onsuccess = function (event) {
        let result = event.target.result;
        let copy_todo_list_completed = db.transaction("todo_list_completed", "readwrite").objectStore("todo_list_completed");
        for (let i = 0; i < result.length; i++) {
            if (result[i].status === "Completed") {
                copy_todo_list_completed.add(result[i]);
                completedTasks.push(result[i]);
            }
        }
    };
    return completedTasks;
}
```

<u>Challenges faced: -</u>

1) Biggest challenge we faced was to keep two objects store active
2) Learned how to navigate the callback functions
3) Learned to debug callback calls
4) Biggest issue was to handle creation of new object store and onUpgradeneeded called from the function

5) Showing graph on the html

Screenshot of the implementations