```
// idb.js: IndexedDB Wrapper Library

/**
 * Initializes or retrieves the IndexedDB instance.
 *
 * @param {string} dbName - The name of the database.
 * @param {string} storeName - The name of the object store.
 * @returns {Promise<IDBDatabase>} A promise that resolves with the database instance.
 */
export const defineIdb = (dbName, storeName) => {
  return new Promise((resolve, reject) => {
    const request = indexedDB.open(dbName, 1);

    // Handle database initialization
    request.onupgradeneeded = (event) => {
      const db = event.target.result;
      if (!db.objectStoreNames.contains(storeName)) {
        db.createObjectStore(storeName, { keyPath: "id", autoIncrement: true });
      }
    };

    request.onsuccess = () => resolve(request.result);
    request.onerror = () => reject(request.error);
  });
};

/**
 * Performs a transaction on the IndexedDB store.
 *
 * @param {IDBDatabase} db - The database instance.
 * @param {string} storeName - The name of the object store.
 * @param {"readonly"|"readwrite"} mode - The transaction mode.
 * @param {Function} callback - The callback function to perform an operation on the store.
 * @returns {Promise<any>} A promise that resolves with the result of the operation.
 */
export const performTransaction = (db, storeName, mode, callback) => {
  return new Promise((resolve, reject) => {
    const transaction = db.transaction(storeName, mode);
    const store = transaction.objectStore(storeName);
    const request = callback(store);

    request.onsuccess = () => resolve(request.result);
    request.onerror = () => reject(request.error);
  });
};

/**
 * Adds a cost item to the database.
 *
 * @param {string} dbName - The name of the database.
 * @param {string} storeName - The name of the object store.
 * @param {Object} costItem - The cost item to add, containing details like `sum`,
 *  `category`, `description`, and `date`.
 * @returns {Promise<number>} A promise that resolves with the ID of the added item.
 */
```

```
export const addingCost = async (dbName, storeName, costItem) => {
  const db = await defineIdb(dbName, storeName);
  return performTransaction(db, storeName, "readwrite", (store) =>
    store.add(costItem)
  );
};

/**
 * Retrieves costs for a specific month and year.
 *
 * @param {string} dbName - The name of the database.
 * @param {string} storeName - The name of the object store.
 * @param {number} month - The month to filter costs (1-12).
 * @param {number} year - The year to filter costs.
 * @returns {Promise<Object[]>} A promise that resolves with an array of cost items for
 *  the specified month and year.
 */
export const getMonthlyCosts = async (dbName, storeName, month, year) => {
  const db = await defineIdb(dbName, storeName);

  return new Promise((resolve, reject) => {
    const transaction = db.transaction(storeName, "readonly");
    const store = transaction.objectStore(storeName);
    const request = store.getAll();

    request.onsuccess = () => {
      const allItems = request.result;
      const filteredItems = allItems.filter((item) => {
        const itemDate = new Date(item.date);
        return (
          // getMonth returns 0-based month
          itemDate.getMonth() + 1 === month && itemDate.getFullYear() === year
        );
      });
      resolve(filteredItems);
    };

    request.onerror = () => reject(request.error);
  });
};

/**
 * Retrieves the total costs grouped by category for a specific month and year.
 *
 * @param {string} dbName - The name of the database.
 * @param {string} storeName - The name of the object store.
 * @param {number} month - The month to filter costs (1-12).
 * @param {number} year - The year to filter costs.
 * @returns {Promise<Object>} A promise that resolves with an object where keys are categories
 * and values are the total sums.
 */
export const getCostsByCategory = async (dbName, storeName, month, year) => {
  const costs = await getMonthlyCosts(dbName, storeName, month, year);
  return costs.reduce((acc, cost) => {
    acc[cost.category] = (acc[cost.category] || 0) + cost.sum;
    return acc;
  }, {});

```

```
        };
```
