**Пошаговое руководство по работе с IndexedDB:**

**Шаг 1. Открытие (или создание) базы данных**

Это первый и самый важный шаг. Если база данных не существует, она будет создана. Если база данных существует, она будет открыта.

javascript

const DB\_NAME = 'MyProductDB'; *// Имя вашей базы данных*

const DB\_VERSION = 1; *// Версия базы данных (целое число, >= 1)*

const STORE\_NAME = 'products'; *// Имя вашего объектного хранилища*

let db; *// Переменная для хранения объекта базы данных*

function openDatabase() {

return new Promise((*resolve*, *reject*) => {

*// 1. Открываем базу данных*

const request = indexedDB.open(DB\_NAME, DB\_VERSION);

*// 2. Обработчик ошибок открытия/создания БД*

request.onerror = (*event*) => {

console.error('Ошибка открытия IndexedDB:', event.target.errorCode);

reject('Ошибка открытия IndexedDB');

};

*// 3. Обработчик успешного открытия БД*

request.onsuccess = (*event*) => {

db = event.target.result; *// Получаем объект базы данных*

console.log('IndexedDB успешно открыта');

resolve(db);

};

*// 4. Обработчик обновления структуры БД (вызывается при создании или увеличении версии)*

request.onupgradeneeded = (*event*) => {

db = event.target.result; *// Получаем объект базы данных в контексте обновления*

*// Проверяем, существует ли уже объектное хранилище*

if (!db.objectStoreNames.contains(STORE\_NAME)) {

*// Создаем объектное хранилище 'products'*

*// keyPath: 'id' - указывает, что свойство 'id' каждого объекта будет использоваться как ключ*

*// autoIncrement: true - автоматически генерировать уникальные ID для новых объектов*

const objectStore = db.createObjectStore(STORE\_NAME, { keyPath: 'id', autoIncrement: true });

*// Создаем индекс для свойства 'category'*

*// 'category' - имя индекса*

*// 'category' - свойство объекта, по которому будет строиться индекс*

*// { unique: false } - значения в индексе могут повторяться*

objectStore.createIndex('category', 'category', { unique: false });

console.log(`Объектное хранилище "${STORE\_NAME}" и индекс "category" созданы.`);

}

};

});

}

**Когда вызывается onupgradeneeded?**

* Когда вы открываете базу данных, которой ещё не существует.
* Когда вы открываете базу данных с версией, которая выше текущей версии базы данных. Это позволяет вам изменять структуру базы данных (добавлять/удалять хранилища объектов, индексы).

**Шаг 2. Добавление данных (запись)**

Чтобы добавить данные, необходимо создать транзакцию.

javascript

async function addProduct(*product*) {

*// 1. Создаем транзакцию*

*// [STORE\_NAME] - массив имен объектных хранилищ, с которыми будет работать транзакция*

*// 'readwrite' - режим транзакции (для записи)*

const transaction = db.transaction([STORE\_NAME], 'readwrite');

*// 2. Получаем доступ к объектному хранилищу*

const objectStore = transaction.objectStore(STORE\_NAME);

*// 3. Добавляем объект в хранилище*

*// product - это JavaScript-объект, который мы хотим сохранить*

const request = objectStore.add(product);

*// 4. Обработчик успешного добавления*

request.onsuccess = () => {

console.log('Продукт успешно добавлен:', product);

*// Здесь можно вызвать функцию для обновления отображения списка продуктов*

};

*// 5. Обработчик ошибок добавления*

request.onerror = (*event*) => {

console.error('Ошибка добавления продукта:', event.target.errorCode);

};

*// 6. Обработчик завершения транзакции (успешно или с ошибкой)*

transaction.oncomplete = () => {

console.log('Транзакция добавления завершена.');

};

transaction.onerror = (*event*) => {

console.error('Транзакция добавления завершилась ошибкой:', event.target.errorCode);

};

}

*// Пример использования:*

*// const newProduct = { name: 'Молоко', quantity: 1, category: 'молочные' };*

*// addProduct(newProduct);*

**Шаг 3. Получение всех данных (чтение)**

Для получения всех данных из объектного хранилища используется курсор.

javascript

async function getAllProducts() {

return new Promise((*resolve*, *reject*) => {

*// 1. Создаем транзакцию в режиме 'readonly' (только для чтения)*

const transaction = db.transaction([STORE\_NAME], 'readonly');

const objectStore = transaction.objectStore(STORE\_NAME);

const products = [];

*// 2. Открываем курсор для итерации по всем записям*

const request = objectStore.openCursor();

*// 3. Обработчик для каждой записи, найденной курсором*

request.onsuccess = (*event*) => {

const cursor = event.target.result;

if (cursor) {

*// Если курсор указывает на запись, добавляем ее в массив*

products.push(cursor.value);

cursor.continue(); *// Переходим к следующей записи*

} else {

*// Курсор прошел все записи, возвращаем массив*

console.log('Все продукты получены:', products);

resolve(products);

}

};

*// 4. Обработчик ошибок*

request.onerror = (*event*) => {

console.error('Ошибка получения продуктов:', event.target.errorCode);

reject('Ошибка получения продуктов');

};

});

}

*// Пример использования:*

*// getAllProducts().then(products => {*

*// // Делаем что-то с полученными продуктами*

*// console.log('Продукты для отображения:', products);*

*// });*

**Шаг 4. Получение данных по индексу (фильтрация)**

Если вы хотите получить данные, отфильтрованные по какому-либо признаку (например, по категории), используйте индекс.

javascript

async function getProductsByCategory(*category*) {

return new Promise((*resolve*, *reject*) => {

const transaction = db.transaction([STORE\_NAME], 'readonly');

const objectStore = transaction.objectStore(STORE\_NAME);

*// 1. Получаем доступ к индексу 'category'*

const categoryIndex = objectStore.index('category');

const products = [];

*// 2. Открываем курсор по индексу, используя IDBKeyRange.only() для точного совпадения*

const request = categoryIndex.openCursor(IDBKeyRange.only(category));

request.onsuccess = (*event*) => {

const cursor = event.target.result;

if (cursor) {

products.push(cursor.value);

cursor.continue();

} else {

console.log(`Продукты категории "${category}" получены:`, products);

resolve(products);

}

};

request.onerror = (*event*) => {

console.error(`Ошибка получения продуктов по категории "${category}":`, event.target.errorCode);

reject(`Ошибка получения продуктов по категории "${category}"`);

};

});

}

*// Пример использования:*

*// getProductsByCategory('фрукты').then(fruits => {*

*// console.log('Только фрукты:', fruits);*

*// });*

**Шаг 5. Удаление данных**

Удаление данных также происходит в рамках транзакции.

javascript

async function deleteProduct(*id*) {

const transaction = db.transaction([STORE\_NAME], 'readwrite');

const objectStore = transaction.objectStore(STORE\_NAME);

*// 1. Удаляем запись по ее ключу (id)*

const request = objectStore.delete(id);

request.onsuccess = () => {

console.log('Продукт успешно удален, ID:', id);

*// Здесь можно вызвать функцию для обновления отображения списка продуктов*

};

request.onerror = (*event*) => {

console.error('Ошибка удаления продукта:', event.target.errorCode);

};

}

*// Пример использования:*

*// deleteProduct(1); // Удалить продукт с ID 1*

**Шаг 6. Обновление данных**

Чтобы обновить данные, вы получаете объект по ключу, изменяете его свойства, а затем с помощью метода **put()**. **put()** либо добавляете новый объект, либо обновляете существующий, если ключ уже есть.

javascript

async function updateProduct(*updatedProduct*) {

const transaction = db.transaction([STORE\_NAME], 'readwrite');

const objectStore = transaction.objectStore(STORE\_NAME);

*// 1. Используем put() для обновления. Если объект с таким id существует, он будет обновлен.*

*// Если нет, будет добавлен новый.*

const request = objectStore.put(updatedProduct);

request.onsuccess = () => {

console.log('Продукт успешно обновлен:', updatedProduct);

*// Здесь можно вызвать функцию для обновления отображения списка продуктов*

};

request.onerror = (*event*) => {

console.error('Ошибка обновления продукта:', event.target.errorCode);

};

}

*// Пример использования:*

*// const productToUpdate = { id: 1, name: 'Молоко 2.5%', quantity: 2, category: 'молочные' };*

*// updateProduct(productToUpdate);*

**Интеграция в ваше приложение (script.js):**

В предоставленном ранее **script.js** эти шаги уже реализованы.

* **openDatabase()**: Открывает базу данных и создаёт хранилище/индексы.
* **addProduct()**: Добавляет новый продукт.
* **getProducts()**: Получает продукты (все или по категориям, используя индекс).
* **deleteProduct()**: Удаляет продукт.
* **displayProducts()**: Использует **getProducts()** для получения данных, а затем динамически создаёт HTML-элементы для их отображения.

**Как это работает вместе:**

1. При загрузке страницы вызывается **openDatabase()**.
2. После успешного открытия базы данных вызывается функция **displayProducts()**, которая получает все продукты и отображает их.
3. Когда пользователь отправляет форму, вызывается функция **addProduct()**, которая сохраняет данные в IndexedDB, а затем снова вызывает функцию **displayProducts()** для обновления списка.
4. При нажатии кнопки «Удалить» вызывается **deleteProduct()**, а затем **displayProducts()**.
5. При изменении фильтра категории вызывается **displayProducts()** с новым параметром фильтра.

Надеюсь, это подробное объяснение поможет вам лучше понять и использовать IndexedDB на экзамене!