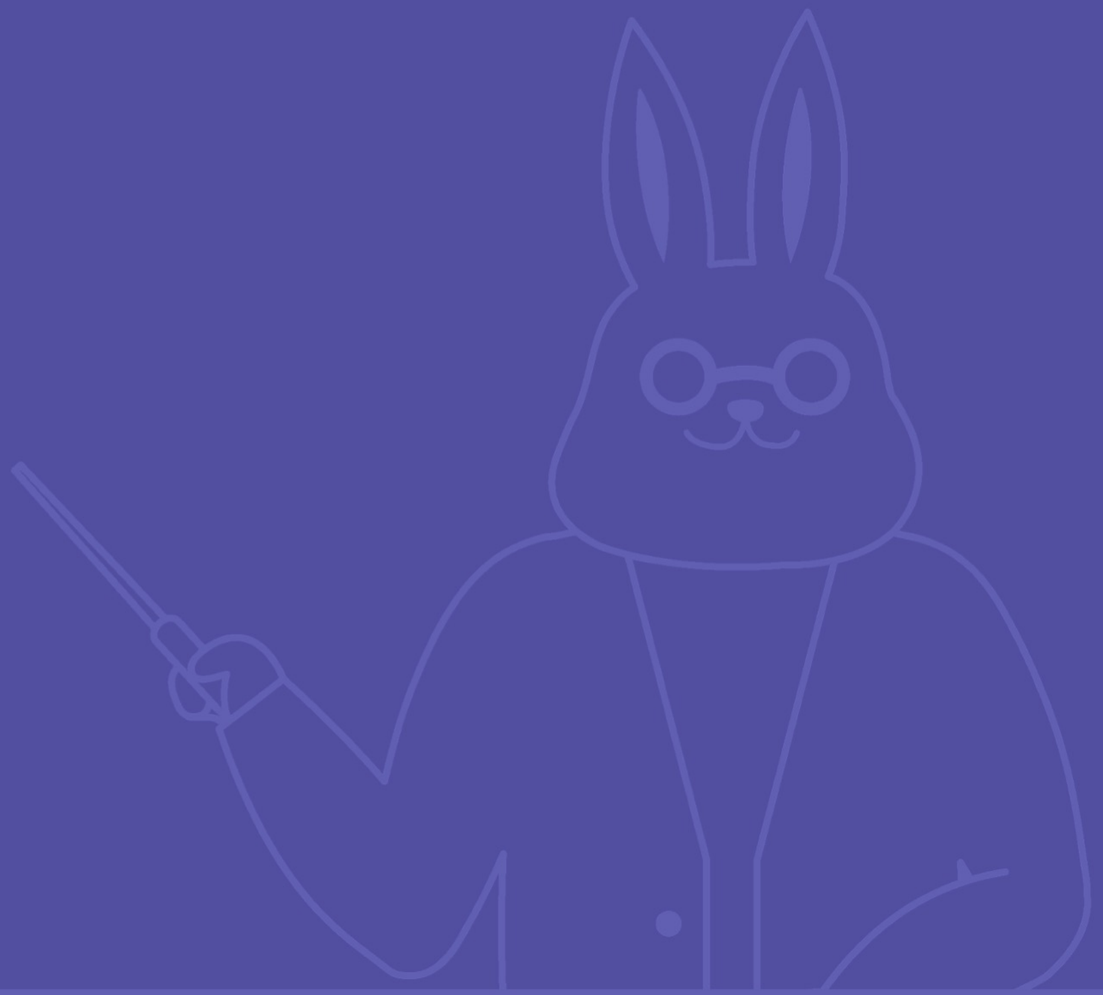
IndexedDB

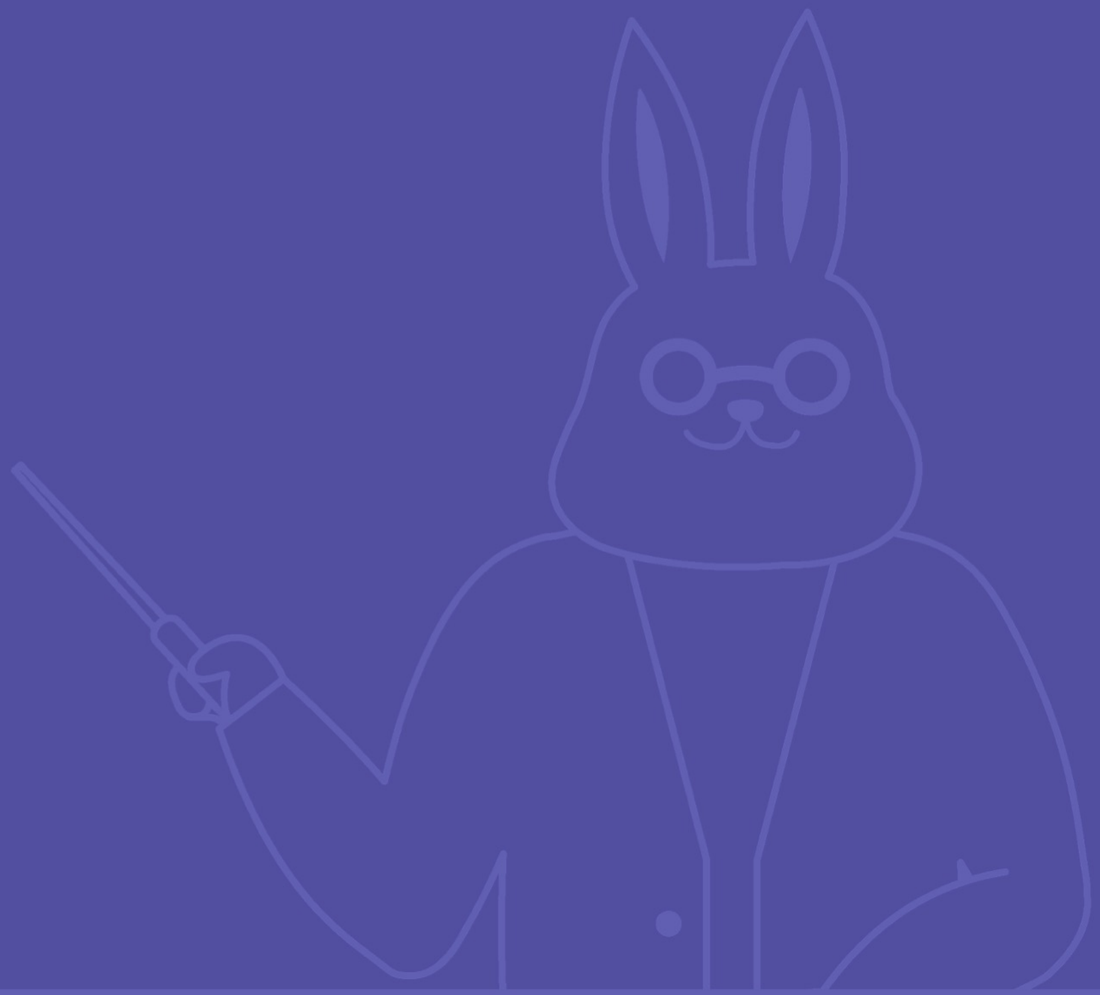
인스타그램 클론코딩



/* elice */

목차

IndexedDB 란?



IndexedDB란?

IndexedDB

≠ 관계형 데이터베이스

SQL 사용

= 브라우저의 객체 저장 가능

자바스크립트 객체, 파일, blobs 등을 저장

Transaction을 지원함

자바스크립트 기반의 객체 지향 데이터베이스 -> 자료형과 객체 저장 가능!

클라이언트(사용자) 측 브라우저에 저장하기 위한 API

1. 쿠키 (Cookies)	문서 내부에 간단 문자열 데이터를 저장
2. 로컬 저장소(Local Storage)	Json 데이터를 문자열로 변환하여 저장
3. 세션 저장소 (Session Storage)	Json 데이터를 오직 탭 세션에 저장
4. IndexedDB	Key를 이용해 Index되는 구조화된 데이터를 쉽게 저장

- IndexedDB는 **브라우저 기반 내장 데이터베이스**이므로 JS 관련 애플리케이션을 만들 때 사용 가능

  - 키/값 저장 가능 (Key-Value 형태의 데이터베이스)
  - Transaction 기능을 지원
  - Key 범위의 쿼리와 **Index를 지원**
  - localStorage에 비해 훨씬 많은 데이터 저장 가능

Index를 지원하기 때문에
많은 양의 구조화된 데이터를 다룰 때 적합하다

IndexedDB 장단점

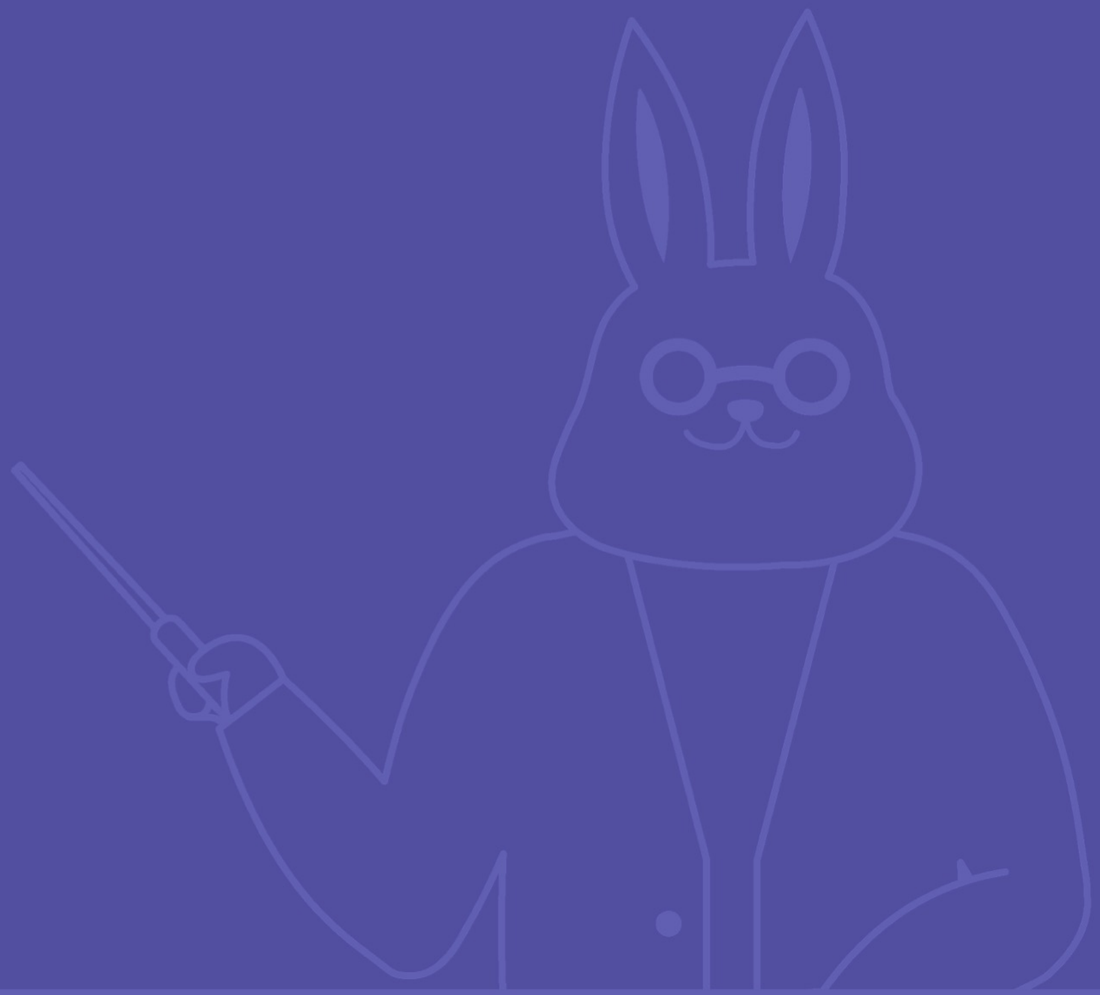
장점

I. 더 복잡하고 구조적인 데이터를 브라우저에서 다룰 수 있다
II. 더 많은 양의 데이터를 저장할수 있다
III. 상호작용시에 더 많은 제어를 할 수 있다
IV. 여러개의 데이터베이스, 그리고 각 DB 내부에 여러 개의 테이블을 가질 수 있다

단점 - 웹 저장소 API 보다 사용법이 더 복잡하다

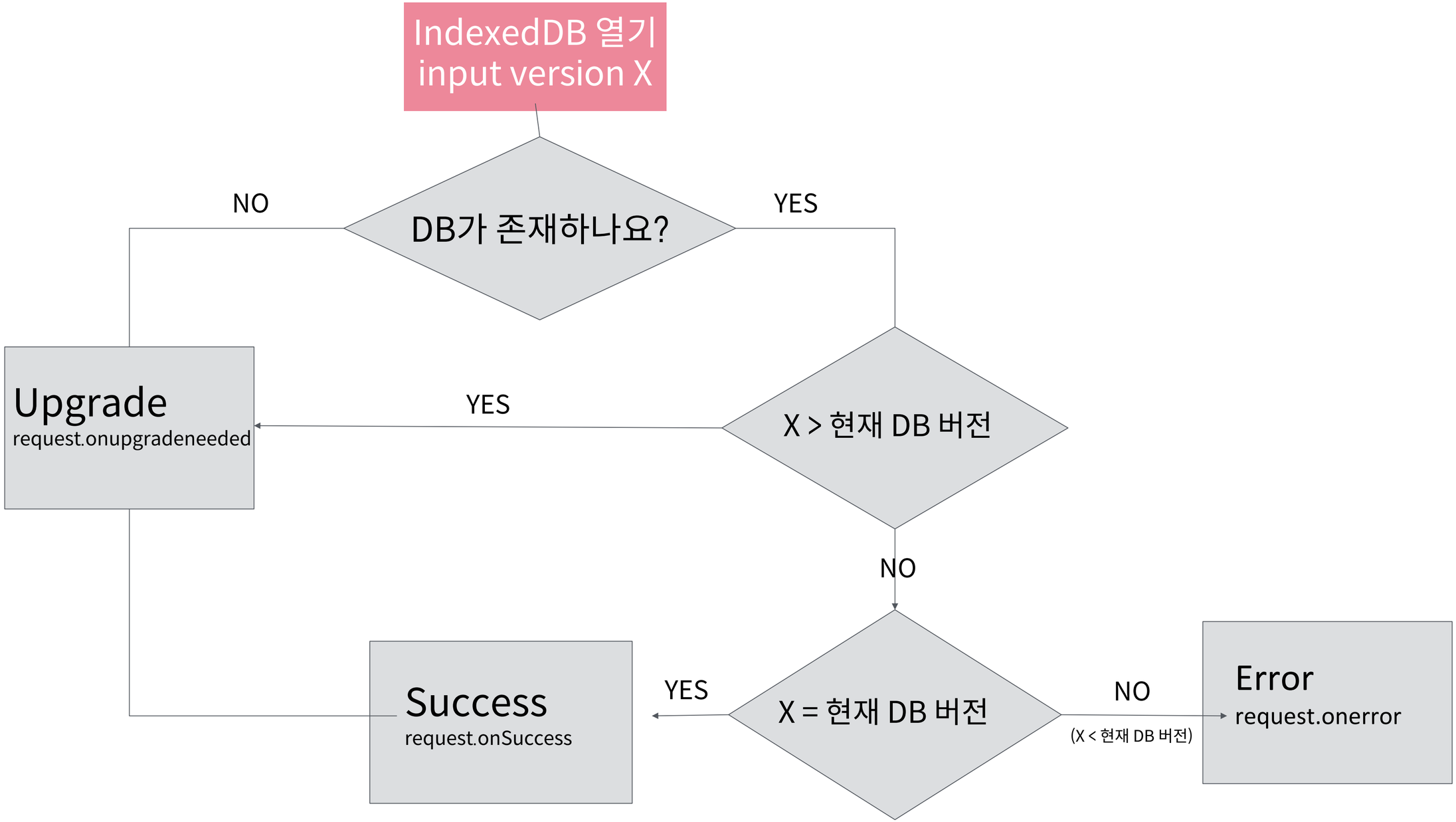
02

IndexedDB 작업 순서



IndexedDB 작업 순서

1. 데이터베이스 열기
2. 데이터베이스에 ObjectStore 생성
3. Transaction 시작 - 데이터 추가, 검색 작업 요청
4. index 마다 DOM 이벤트 수신해서 작업
5. 결과 수행


IndexedDB 열기
input version X
DB가 존재하나요?
NO
YES
Upgrade
request.onupgradeneeded
YES
X > 현재 DB 버전
NO
Success
request.onSuccess
YES
X = 현재 DB 버전
NO
(X < 현재 DB 버전)
Error
request.onerror

데이터베이스 열기

```
let request = indexedDB.open( name, version);
```

- name – 문자열, 데이터베이스 이름.
- version – 기본적으로 양의 정수 버전 1

첫번째 버전 데이터베이스 게시

```
let request = indexedDB.open( ‘myDB’, 1);
```

IndexedDB 데이터베이스 수정시 Version을 수정해야함

데이터베이스 열기

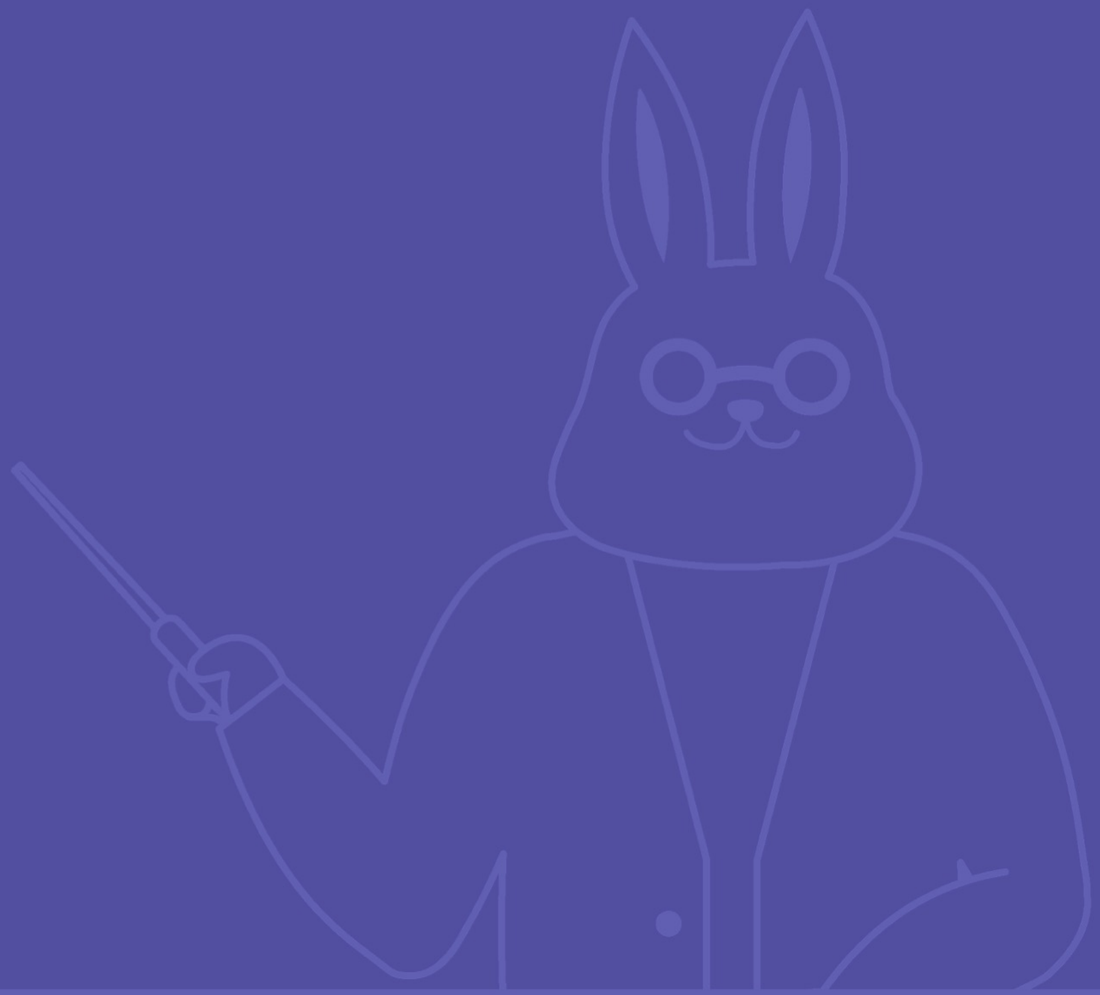
```
let onRequest = indexedDB.open( ‘myDB’, 1);
    onRequest.onsuccess = () => {
        console.log('Success creating or accessing db')
    }
    onRequest.onupgradeneeded = () => {
      const database = onRequest.result
    }
    onRequest.onerror = () => {
     console.log('Error creating or accessing db')
    }
```

로컬 데이터베이스 버전이 지정된 버전(1)보다 낮으면 **onupgradeneeded**가 트리거되고 데이터베이스 버전을 업그레이드

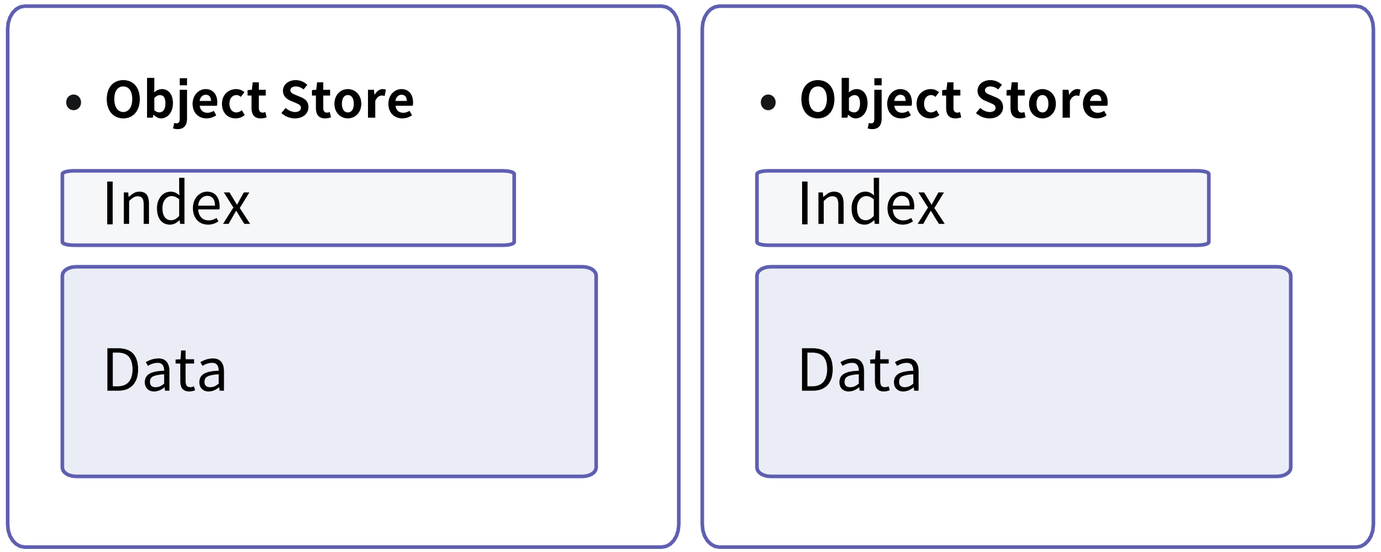
이벤트는 데이터베이스가 아직 존재하지 않을 때도 트리거되므로 초기화를 수행 가능

03

ObjectStore 란?



Database



ObjectStore

- 데이터를 담든 공간이며 여러개의 Key-Value 값으로 형성
- ObjectStore의 이름은 고유해야함

ObjectStore 생성하기

```
onRequest.onupgradeneeded = () => {
    const database = onRequest.result
    const objectStore = database.createObjectStore('store name',
    {keyPath: 'id'})
}
```

.onupgradeneeded()으로 데이터베이스가 업그레이드 될 때

ObjectStore 생성 또는 수정 가능

ObjectStore 생성하기

```
const usersStore= database.createObjectStore(‘users’, {keyPath: ‘id’})
```

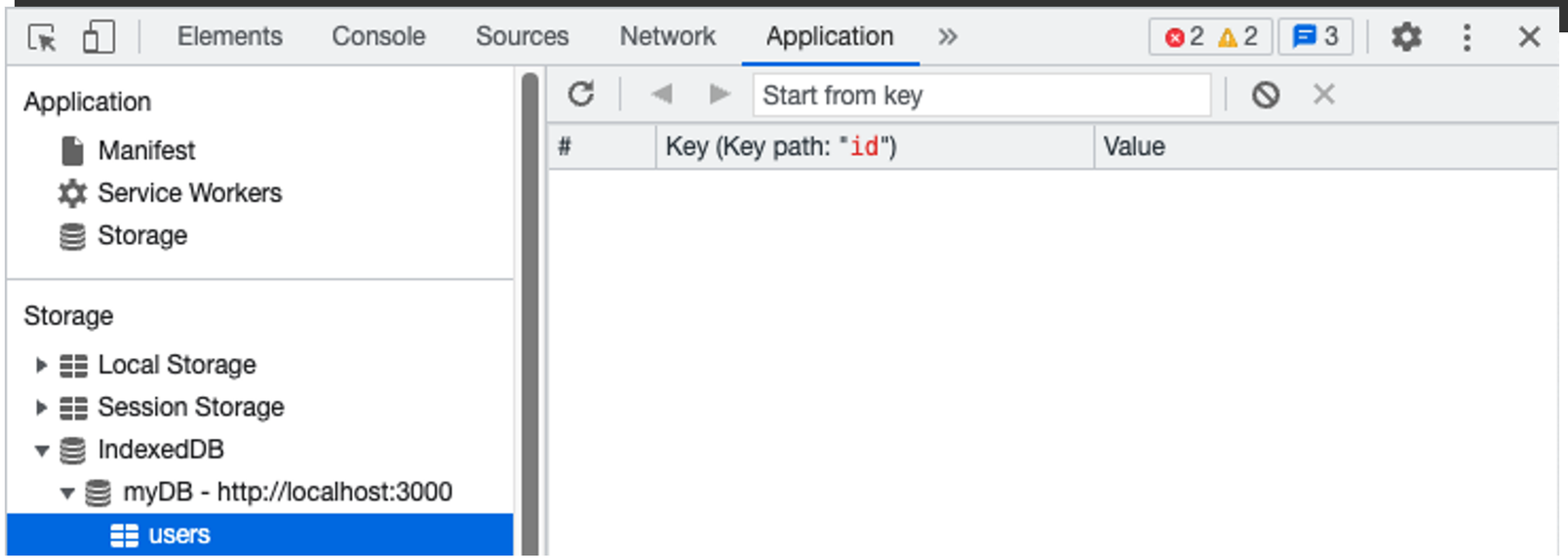
● **createObjectStore(tableName)** 함수를 이용해서 usersStore라는 이름으로 ObjectStore를 생성하고 users 라는 테이블을 만듦

=> usersStore 저장소에 **users**라는 테이블이 생성됨

● **{keyPath: ‘id’}** 로 "id"를 제공하는 데 필요한 인덱스 필드의 이름을 지정

ObjectStore 생성하기

```
const initializeDb = indexedDB.open('myDB', 1)
initializeDb.onupgradeneeded = () => {
    const database = initializeDb.result
    database.createObjectStore('users', {keyPath: ‘id’})
}
```



04

Transaction – 데이터 추가, 읽기, 수정, 제거

Transaction 시작하기

```
const transaction = database.transaction(“objectStore Name”,
'Transaction Mode')
```

모든 데이터 읽기 및 쓰기는 Transaction 내에서 수행됨

- objectStore Name: 객체 저장소 이름
- Transaction Mode: readonly, readwrite, versionchange

Transaction 시작하기

```
// users 테이블에서 transaction을 instance화
const transaction = database.transaction('users', 'readwrite').objectStore('users')
```

1. 트랜잭션은 데이터베이스 객체 단위로 작동하므로 트랜잭션을 사용할 객체 저장소를 지정해줘야 함
2. ‘readwrite’ 모드 사용 : 이미 존재하는 objectStore의 데이터를 읽고 다시 쓸 때

ObjetStore 에 데이터 추가하기

```
const todos= database.createObjectStore(‘todos’, {autoIncrement: true})
function addTodos() {
          const todo = {
                    title: “todo1” ,
                    text: “no.1 thing to do”
                  }
      // todos ObjectStore에 readwrite(읽기, 쓰기) 권한으로 Transaction 시작하기
        const transaction = database.transaction(“todos”, 'readwrite')
      // objectStore()함수로 todos 테이블 선택
        const todos = transaction.objectStore(“todos”)
      // 원하는 객체 (todo)를 테이블에 추가
        todos.add(todo)
    }
```

사용자 데이터 예시

```
var users = [ {
      id: 1,
      name: “Pete”,
      age: 35,
      },
      {
      id: 2,
      name: “Mark”,
      age: 23,
      }];
```

사용자 데이터 예시

```
// Add User Data
dbPromise.then(function(db) {
  const transaction = db.transaction(‘users’, 'readwrite')
  const store = transaction.objectStore(‘users’,{keyPath: ‘id’})
  store.add( { id: 3, name: ‘Frank’ , age: 23} )
  return transaction.complete
}
```

사용자 데이터 예시

```
var users = [ {
      id: 1,
      name: “Pete”,
      age: 35 },
      {
      id: 2,
      name: “Mark”,
      age: 23 },
      {
      id: 3,
      name: “Frank”,
      age: 23 }
      ];
```

사용자 데이터 읽기

objectStore.get(key);

```
  // Get Data
  dbPromise.then(function(db) {
    const transaction = db.transaction([‘users’], 'readwrite')
    const store = transaction.objectStore(‘users’)
       return store.get(‘Frank’) //{ id: 3, name: ‘Frank’ , age:
23}
```

objectStore.getAll();

```
// Get all the Data
    return store.getAll()  // { id: 1, name: “Pete”, age: 35 }, {id: 2, name:
“Mark”, age: 23 }, { id: 3, name: “Frank”, age: 23}
```

사용자 데이터 수정

```
// Update User Data

 dbPromise.then(function(db) {
   const transaction =
      db.transaction([‘users’],
      'readwrite')
   const store =
      transaction.objectStore(‘users’)
      return store.put(‘Frank’)
 }
```

사용자 데이터 삭제

```
// Delete Data

 dbPromise.then(function(db) {
   const transaction = db.transaction([‘users’], 'readwrite')
   const store = transaction.objectStore(‘users’)
      store.delete(‘Frank’)
       return transaction.complete;
 }
```

크레딧

/* elice */

코스 매니저
이재성

콘텐츠 제작자
안조현

강사
안조현

감수자
이재성

디자이너
강혜정

연락처

TEL

070-4633-2015

WEB

https://elice.io

E-MAIL

contact@elice.io