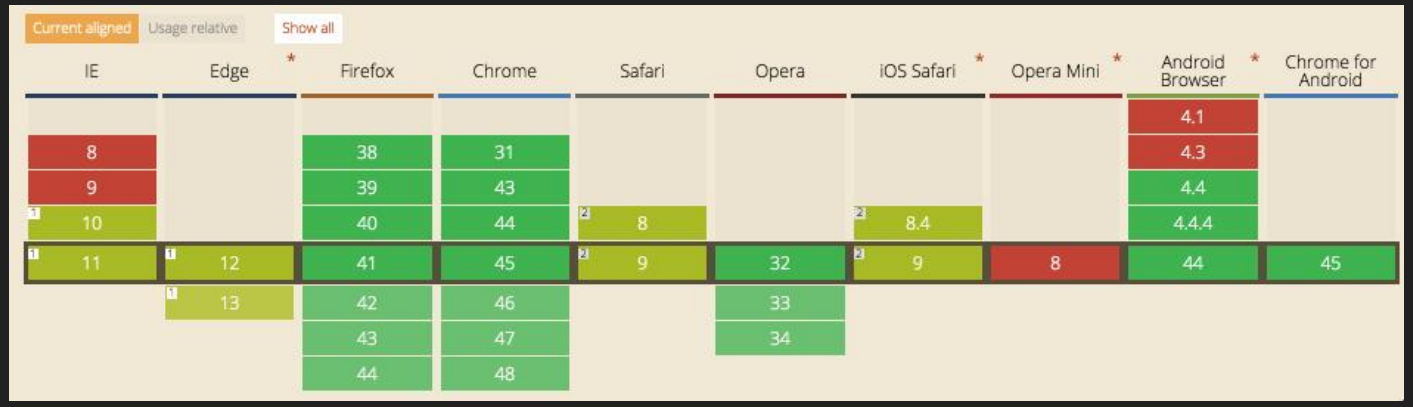
JavaScript

indexedDB – 김근형 강사

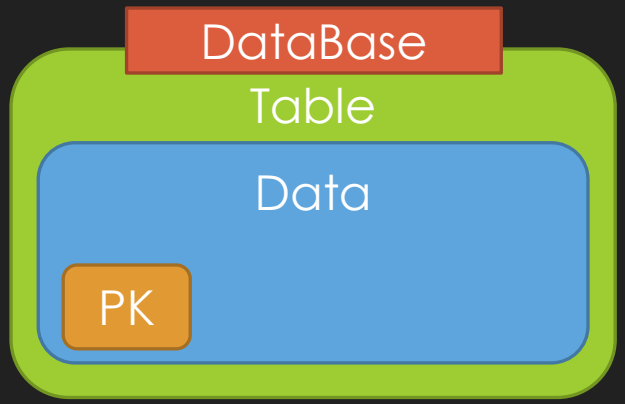
IndexedDB API

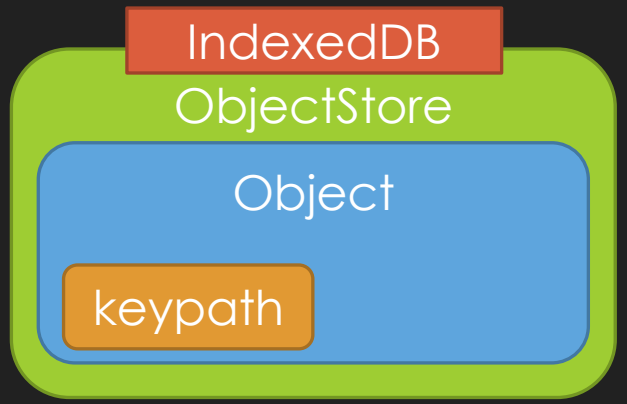
- IndexedDB API
  - 웹 스토리지는 효율적인 정보 저장의 수단이기는 하지만 일반적으로 소량의 단순한 데이터를 처리하기에 적합하다.
  - 반면, 인덱스드디비(IndexedDB)는 대량의 구조적인 데이터를 웹 브라우저에서 처리하기 위한 API다.
  - 기존의 웹 환경에서는 데이터가 모두 서버 영역에 저장되어야 하므로 온라인 작업이 필수였지만 인덱스드디비를 이용하면 클라이언트의 브라우저에 구조적 데이터를 저장할 수 있어서 오프라인 작업도 가능하다.
    - 기존에 websql 이라는 기술이 있었지만 W3C에서 지원을 중단하며 표준에서 탈락



IndexedDB API

- IndexedDB API
  - IndexedDB API 는 우리가 흔히 사용하는 RDB와 매우 유사한 구조를 갖는다.
  - 인덱스드 디비도 데이터베이스 생성 후 이를 통해 오브젝트 스토어(ObjectStore)를 생성하고 데이터를 저장할 준비를 한다.
  - 오브젝트스토어에 저장되는 데이터는 이름 그대로 자바스크립트의 객체(Object)들이며 이들은 keypath로 구분될 수 있다.
  - 하지만 기본 구조가 유사할 뿐 인덱스드디비는 RDB와는 달리 객체 기반으로 동작한다.





IndexedDB API

- IndexedDB 특징
  - 키-값 쌍으로 데이터를 저장한다
    - 값은 복합적인 속성을 갖는 객체이며 키는 그 객체의 속성 중 하나이다. 이 키들을 이용해 인덱스(Index)를 만들 수 있으며 객체 조회를 위해 사용된다.
  - 트랜잭션 데이터베이스 모델에 기반을 둔다
    - IndexedDB API 는 데이터베이스의 인덱스, 테이블, 커서 등에 해당하는 객체들을 제공하며 이들은 모두 특정 트랜잭션 내에서 실행되어야 한다. 따라서 트랜잭션이 종료된 후 관련 작업을 수행하면 예외를 발생시킨다. IndexedDB 트랜잭션의 경우는 자동 커밋(commit)을 지원하며 수동으로는 커밋할 수 없다.
  - 비동기 방식으로 동작한다.
    - 비동기 방식의 API는 함수 호출 시 결과를 반환하지 않는 대신 함수를 호출할 때 함수 종료 시 동작할 콜백 함수를 전달한다. 요청한 동작이 종료되면 대부분 success 또는 error 와 같은 DOM 이벤트가 발생하는데 이것을 받아서 다음 동작을 진행한다.

IndexedDB API

- IndexedDB 특징
  - SQL을 사용하지 않는다
    - 함수 개반으로 데이터 저장, 수정, 검색, 삭제를 수행하며 특히 검색을 위해서는 인덱스를 통해 생성한 커서(Cursor)를 사용한다.
  - 동일근원 정책(same-origin policy)를 따른다
    - 동일근원 정책은 스크립트가 실행되는 어플리케이션 레이어 프로토콜(http), 도메인(www.poo.bar), 포트번호(9090)가 같은 경우는 IndexedDB를 공유하고 다를 경우 공유되지 않는다는 것이다.

IndexedDB API

- 비동기 동작방식과 IDBRequest
  - 인덱스드디비의 동작 방식은 비동기 방식이다. 그리고 작업의 결과로 IDBRequest 타입의 객체를 DOM 이벤트 처리 방식으로 돌려준다.
  - 대부분의 함수는 호출 즉시 IDBRequest 객체를 반환하지만 이 객체를 바로 쓸 수는 없다. success와 error 이벤트에 대한 리스너를 등록하고 이벤트를 수신했을 때 관련 속성들을 사용할 수 있다.

IndexedDB API

- 비동기 동작방식과 IDBRequest

```
var openResult = indexedDB.open("hello");
openResult.addEventListener("success", function(){
    console.log("콜백 내부 : "+openResult.result);
});

console.log("전역 레벨 : "+openResult.result);
```

```
var openResult = indexedDB.open("hello");
openResult.addEventListener("success", function(){
    console.log("콜백 내부 : "+openResult.result);
});

console.log("전역 레벨 : "+openResult.result);
```

```
Uncaught DOMException: Failed to read the 'result' property from
'IDBRequest': The request has not finished.
    at file:///C:/Users/oklin/OneDrive/%EC%88%98%EC%97%85/%EC%88%
JQuery/2.0/source/90/01_async.html:16:36

콜백 내부 : [object IDBDatabase]
```

IndexedDB API

- IDBRequest 이벤트 & 속성
  - 이벤트

이벤트 이름	설명
success	요청이 성공했을 때 동작하는 이벤트로 필요한 리스너를 등록해서 사용한다.
error	요청이 실패햇을 때 동작하는 이벤트로 필요한 리스너를 등록해서 사용한다.

  - 속성

속성 이름	설명
result	요청에 대한 결과를 가지는 속성. 만약 요청이 실패해서 사용할 수 없을 경우 invalidStateError 가 발생한다.
error	일반적으로 error 이벤트 시 오류의 원인을 나타낸다.
source	요청에 대한 소스. IDBIndex나 IDBObjectStore 등이며 요청에 따라 달라진다. 소스가 없을 경우 null이 반환된다.
transaction	요청이 동작하는 트랜잭션 객체이다. 처음 데이터베이스를 열어서 아직 트랜잭션에 포함되지 않았을 경우 null이 반환된다.

IndexedDB API

- 데이터베이스 생성
  - 자바스크립트의 전역 객체인 window는 indexedDB 속성을 제공한다.
  - 이 속성은 IDBFactory 인터페이스의 구현체로 어플리케이션에서 비동기적으로 인덱스드디비를 여는 기능을 제공한다.
  - 또한, IDBRequest 를 상속받은 IDBOpenDBRequest 타입의 반환 값을 통해 실제 데이터베이스를 사용할 수 있게 한다.
  - IDBFactory의 함수는 아래와 같다.

함수 이름	설명
open(DB_NAME [, version])	데이터베이스와의 연결을 요청하는 함수다. 호출 결과로 IDBOpenDBRequest 타입의 객체를 반환한다. DB_NAME과 동일한 이름의 데이터베이스가 존재하는 경우 그 데이터베이스를 사용하고 존재하지 않는 경우 새로운 데이터베이스를 연결한다. 새로 생성되는 데이터베이스의 버전은 1이다. version 정보를 기존 DB 정보보다 더 높게 줄 경우 데이터베이스는 upgradeneeded 이벤트를 발생시켜 구조를 변경할 수 있게 한다.
deleteDatabase(DB_NAME)	데이터베이스 삭제를 요청하는 함수다. 호출 결과로 open과 마찬가지로 IDBOpenDBReqeust 타입의 객체를 반환한다.

IndexedDB API

- 데이터베이스 생성
  - IDBOpenDBRequest 는 IDBRequest를 상속받았으므로 기본 속성은 IDBRequest와 동일하다.
  - 이 객체 역시 바로 사용은 불가능 하며 반환값에 이벤트 리스너를 등록해서 그 안에서 사용해야 한다.
  - 이때 발생하는 이벤트는 success, error, upgradedneeded, blocked가 있다. 데이터베이스가 처음 생성되었거나 기존에 있는 버전보다 요청하는 버전이 높을 경우 upgradeneeded 이벤트가 발생한다.
  - 데이터 베이스를 성공적으로 오픈할 경우 IDBRequest 객체의 result 속성에는 데이터베이스 객체가 할당 된다.
  - IDBOpenDBRequest 의 이벤트는 아래와 같다.

이벤트 이름	설명
blocked	데이터베이스의 업그레이드가 필요한 상황에서 다른 사용자가 이미 기존 버전으로 사용하고 있을 때 새로운 사용자에게 발생하는 이벤트
error	요청 처리에 실패했을 때 발생하는 이벤트
source	요청 처리에 성공했을 때 발생하는 이벤트
transaction	업그레이드가 필요한 경우 발생하는 이벤트, 기존의 데이터베이스 버전보다 높은 버전을 파라미터로 open() 함수를 호출할 때 발생한다.

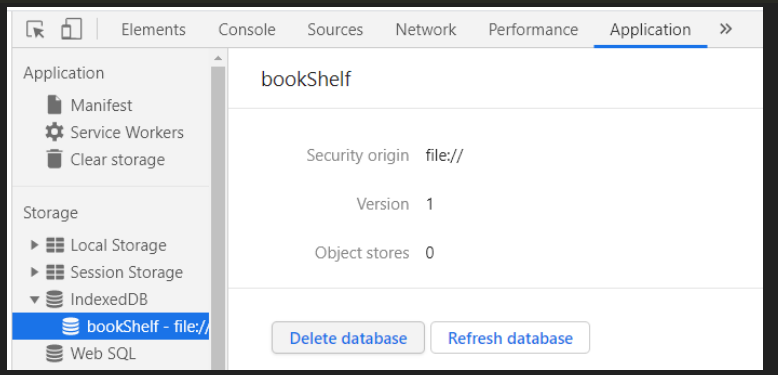
IndexedDB API

- 데이터베이스 생성 예제

```
var bookShelfDB;

function openDB() {
    var openResult = window.indexedDB.open("bookShelf", 1);
    openResult.addEventListener("success", function() {
        console.log("데이터베이스 오픈 성공 : " + this.result);
        bookShelfDB = this.result;
    });
    openResult.addEventListener("error", function() {
        console.log("데이터베이스 오픈 실패");
    });
    openResult.addEventListener("upgradeneeded", function() {
        console.log("데이터베이스 업그레이드 진행");
    });
}
openDB();
```

```
데이터베이스 업그레이드 진행
데이터베이스 오픈 성공 : [object IDBDatabase]
```

```
<fieldset>
    <legend>도서 정보</legend>
    <label for="title">제목</label>
    <input id="title" type="text">
    <label for="isbn">ISBN</label>
    <input id="isbn" type="text">
    <br>
    <label for="year">출판년도</label>
    <input id="year" type="number">
    <label for="category">분야</label>
    <select id="category">
        <option>문학
        <option>경제
        <option>기술
        <option>시사
    </select> <br>
    <button id="registerB">등록</button>
</fieldset>
<fieldset id="control">
    <label for="datas">자료 목록</label>
    <select id="datas"></select>
    <button id="searchB">조회</button>
    <button id="delB">삭제</button>
    <button id="getAllB">전체조회</button>
</fieldset>
<div id="dashboard"></div>
```



IndexedDB API

- 데이터베이스 구조 생성
  - IDBDatabase 타입의 객체는 데이터베이스 자체를 나타내며 관리를 위한 다양한 속성과 기능을 가지며 이벤트도 등록이 가능하다.

속성 이름	설명
name	연결된 데이터베이스의 이름을 나타낸다
version	연결된 데이터베이스의 버전을 나타낸다
objectStoreNames	연결된 데이터베이스에 속한 ObjectStore 의 이름을 리스트 형태로 반환한다.

함수 이름	설명
createObjectStore(store_name [, optionalParameter])	연결된 데이터베이스에서 store_name 으로 IDBObjectStore 타입의 객체를 생성한다. optionalParameter에서는 다음과 같은 두개의 값이 들어있는 객체를 지정한다. - keypath : 일종의 기본키(primary key)이다. 오브젝트 스토어에 저장되는 모든 객체는 keyPath에 설정된 속성을 가져야 한다. - autoIncrement : 불리언 값이며 true 로 설정된 경우 키 생성기(key generator)를 통해 키가 자동으로 공급되고 저장되는 객체들은 별도로 keypath 값을 입력할 필요 없다. false 일 경우 키 생성은 제공되지 않으며 반드시 keyPath를 입력해야 한다.
deleteObjectStore(store_name)	store_name 으로 등록된 오브젝트 스토어 객체를 삭제한다.
close()	연결된 데이터베이스외의 연결을 즉시 종료한다.
transaction(store_name_arr, prop)	IDBTransaction 타입의 트랜잭션 객체를 반환한다. store_name_arr 은 트랜잭션을 적용할 오브젝트 스토어 이름의 배열이다. prop은 트랜잭션을 사용할 모드이다.

IndexedDB API

- 데이터베이스 구조 생성
  - IDBDatabase 의 이벤트

이벤트 이름	설명
abort	연결된 데이터베이스에의 접근이 실패한 경우 이벤트가 발생한다.
error	데이터베이스 접근 시 오류가 발생한 경우 이벤트가 발생한다.
versionchange	데이터베이스 구조가 변경된 경우 이벤트가 발생한다.

IndexedDB API

- 데이터베이스 구조 생성 예제

```
데이터베이스 업그레이드 진행                          03_bookshelf_objects…e_1_keypath.html:57
데이터베이스 오픈 성공 : [object IDBDatabase]          03_bookshelf_objects…e_1_keypath.html:49
이름: bookShelf 버전: 2                              03_bookshelf_objects…e_1_keypath.html:71
ObjectStore 개수: 1                                  03_bookshelf_objects…e_1_keypath.html:73
```

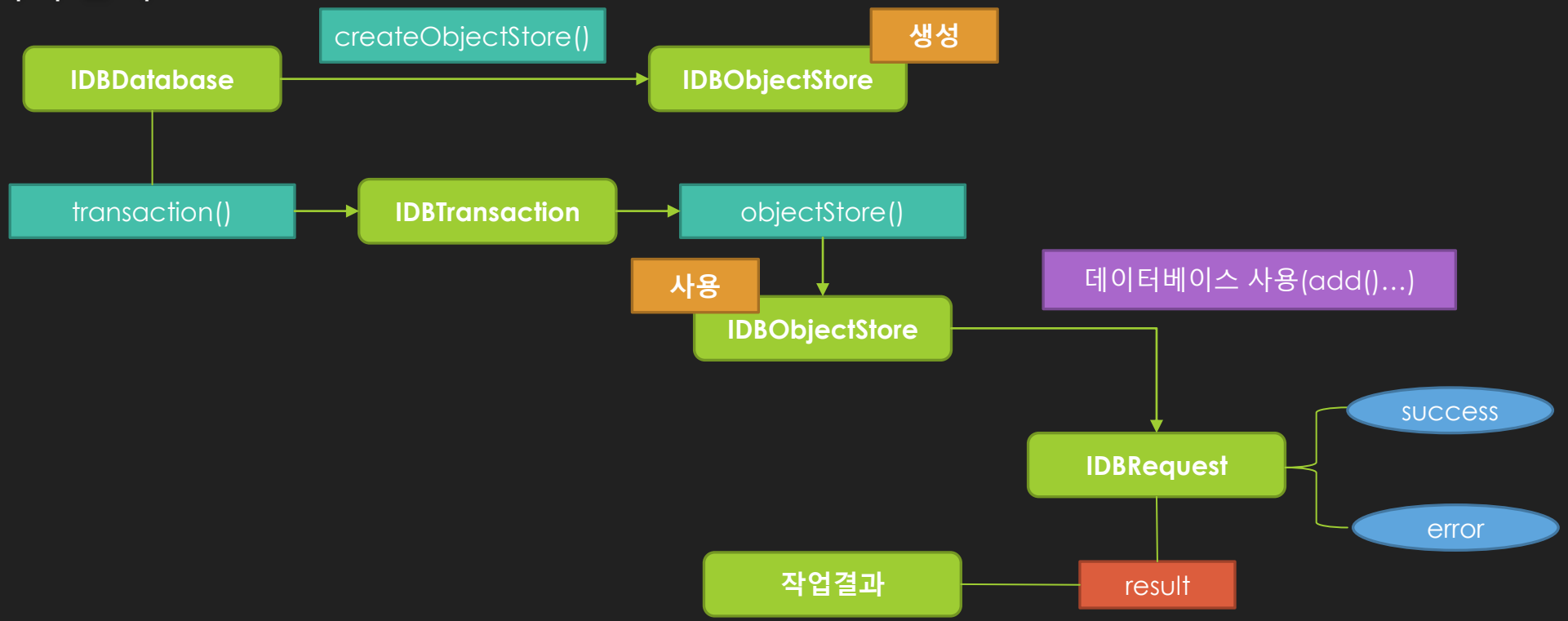
```
var bookShelfDB;

function openDB() {
    var openResult = window.indexedDB.open("bookShelf", 2);
    openResult.addEventListener("success", function() {
        console.log("데이터베이스 오픈 성공 : " + this.result);
        bookShelfDB = this.result;
        showDatabaseInfo(bookShelfDB);
    });
    openResult.addEventListener("error", function() {
        console.log("데이터베이스 오픈 실패");
    });
    openResult.addEventListener("upgradeneeded", function() {
        console.log("데이터베이스 업그레이드 진행");
        bookShelfDB = this.result;
        if (bookShelfDB.objectStoreNames.contains("books")) {
            bookShelfDB.deleteObjectStore("books");
        }
        bookShelfDB.createObjectStore("books", {
            keyPath: "isbn",
            autoIncrement: true
        });
    });
}
openDB();

function showDatabaseInfo(db) {
    console.log("이름: " + db.name, "버전: " + db.version);

    console.log("ObjectStore 개수: " + db.objectStoreNames.length);
    for (var i = 0; i < db.objectStoreNames; i++) {
        console.log(i + " : " + db.objectStoreNames[i]);
    }
}
```

IndexedDB 데이터 관리

- 데이터 관리
  - IndexedDB 를 이용한 데이터 관리를 위해 IDBIndexedDB, IDBTransactionm IDBObjectStore, IDBRequest 등이 필요하다.
  - IndexedDB 에서 데이터를 관리하기 위해서는 다음의 절차를 따른다.
    - 데이터베이스의 createObjectStore() 함수를 이용해 오브젝트 스토어를 생성한다.
    - 데이터베이스의 transaction() 함수를 통해 특정 오브젝트 스토어에서 사용할 트랜잭션을 얻는다.
    - 트랜잭션의 objectStore() 함수를 이용해 트랜잭션에서 사용할 오브젝트 스토어를 획득한다.
    - 오브젝트 스토어를 통해 add(), remove() 등의 함수로 데이터베이스를 사용한다.
    - 이 함수들은 IDBRequest 를 반환하는 비동기 방식으로 동작하며 성공 여부에 따라 success나 error 이벤트를 발생시킨다.
    - 이벤트 내에서 IDBRequest 객체가 갖는 result 속성에서 상황에 따른 결과를 확인한다.

IndexedDB 데이터 관리

- 데이터 관리



IndexedDB 데이터 관리

- IDBObjectStore 인터페이스
  - IDBObjectStore 는 RDB에서의 테이블과 같은 역할을 한다. 즉 다수의 데이터를 저장하는 곳이 바로 오브젝트스토어이고, 다양한 역할을 수행할 수 있다.
  - 다음은 오브젝트 스토어를 통해 처리하는 작업들이다
    - 저장되는 데이터의 구조를 결정할 수 있다 : 오브젝트스토어를 생성할 때 입력했던 keypath 는 RDB의 기본키와 같은 역할을 한다. 또한, 오브젝트스토어를 통해서 IDBIndex 객체를 얻어 keypath 외에 추가적인 컬럼을 정의할 수 있다.
    - 데이터에 대한 추가/수정/삭제/조회를 처리한다 : 각 작업은 IDBTransaction 을 이용해서 트랜잭션을 시작해 데이터의 무결성을 확보한 후 작업해야 한다. 또한, 전문적인 조회를 위해 IDBCursor 를 사용할 수 있다.

IndexedDB 데이터 관리

- IDBObjectStore 속성

속성 이름	설명
indexNames	ObjectStore 에 설정된 인덱스의 이름들을 리스트로 나타낸다.
keyPath	ObjectStore 에 설정된 keyPath를 나타낸다.
name	ObjectStore 에 설정된 name을 나타낸다.
transaction	ObjectStore 에 속해 있는 IDBTransaction 객체를 나타낸다.
autoIncrement	ObjectStore 에 설정된 auto Increment 속성의 값을 나타낸다.

IndexedDB 데이터 관리

- IDBObjectStore 함수

함수 이름	설명
add(Item [, optionalKey])	IDBRequest 를 반환하여 비동기적으로 동작한다. item의 복제본을 새로운 데이터로 오브젝트스토어에 저장한다. 이 기능은 추가 전용으로 오브젝트스토어에서 지정한 키로 이미 다른 객체가 저장되어 있는 경우 오류를 발생한다.
put(Item, optionalKey)	IDBRequest 를 반환하여 비동기적으로 동작한다. item의 복제본을 오브젝트스토어에 저장한다. 기존에 동일한 keyPath의 아이템이 등록되었다면 수정으로 동작하고, 처음이라면 추가로 동작한다. put()을 위해서는 트랜잭션 모드가 readwrite로 설정되어야 한다.
delete(recordKey)	IDBRequest 를 반환하며 비동기적으로 동작한다. 오브젝트 스토어에서 keypath 항목이 recordKey인 자료를 삭제한다.
get(key)	IDBRequest 를 반환하며 비동기적으로 동작한다. 주어진 key로 오브젝트스토어에 등록된 자료를 조회한다. 조회 결과는 IDBRequest의 result 속성에서 확인할 수 있다. 자료가 정상적으로 조회된 경우 등록된 객체의 복제본이 반환된다.
getAll([query, count])	IDBRequest 를 반환하며 비동기적으로 동작한다. 파라미터로 줄 수 있는 query는 IDBKeyRange 타입의 객체로 조회 결과를 필터링할 때 사용된다. count는 조회된 결과를 몇 개까지 반환할 것인가를 나타낸다.
clear()	IDBRequest 를 반환하며 비동기적으로 동작한다. 오브젝트스토어에 저장된 모든 자료를 삭제한다
getAllKeys([query, count])	IDBRequest 를 반환하며 비동기적으로 동작한다. 오브젝트스토어에 저장된 모든 객체들의 키들을 반환한다. 파라미터는 getAll()의 것과 동일하다.
createIndex(name, property, optionsObj)	IDBIndex 타입을 생성해서 즉시 반환한다. name은 생성하는 인덱스의 이름이다. property는 인덱스로 사용할 객체의 속성으로 오브젝트스토어에 저장되는 모든 객체들은 반드시 property에 해당하는 속성을 가져야 한다. optionsObj는 인덱스가 가질 속성을 설정하는데 unique라는 키로 Boolean 타입의 값을 갖는다. true가 설정되면 값은 중복될 수 없고 false면 중복을 허용한다. 이 함수는 versionChange 트랜잭션 모드에서 호출되어야 한다. 따라서 일반적으로 upgradeneeded 이벤트에서 호출된다.

IndexedDB 데이터 관리

- IDBObjectStore 함수

함수 이름	설명
deleteIndex(name)	name으로 등록된 인덱스를 삭제한다. createIndex와 마찬가지로 주로 upgradeneeded 이벤트에서 호출된다.
index(name)	name으로 등록된 인덱스를 반환한다.
openCursor([keyRange, direction])	IDBRequest 를 반환하여 비동기적으로 동작한다. IDBRequest의 result 에서 IDBCursorWithValue 객체를 얻을 수 있다. 처음 파라미터는 검색 조건을 나타내고 두 번째 파라미터는 데이터의 정렬 조건을 나타낸다.
count([keyRange])	IDBRequest 를 반환하여 비동기적으로 동작한다. IDBKeyRange 의 범위에 적합한 데이터의 개수가 반환되거나 파라미터가 없을 경우 전체 데이터의 개수를 반환한다.

IndexedDB 데이터 관리

- IDBTransaction 인터페이스
  - 데이터의 무결성을 유지하기 위해 오브젝트 스토어에서 사용되는 함수들은 트랜잭션을 통해서 수행된다.
  - 일반적인 RDB의 트랜잭션과 달리 IndexedDB 의 트랜잭션은 오토커밋 만을 지원한다. 즉 별도의 커밋을 입력하지 않아도 하나의 기능이 수행 완료되면 알아서 커밋하고 수동 조절이 불가능하다.
  - 트랜잭션은 IDBDatabase 의 transaction(store_name_arr, prop) 함수를 이용해서 얻을 수 있다.
  - store_name_arr 속성에는 해당 트랜잭션에서 사용하려는 오브젝트 스토어의 이름들을 배열 형태로 입력한다. 두번째 속성인 prop에는 트랜잭션의 모드를 문자열 형태로 입력하는 데 사용하는 트랜잭션 모드는 아래와 같다.

모드	설명
readonly	데이터에 대한 읽기 작업만을 지원한다.
readwrite	데이터에 대한 읽기는 물론 쓰기, 수정, 삭제를 지원한다.
versionchange	readwrite와 같은 기능을 할 수 있지만, 데이터베이스 버전을 업데이트하는 데에서만 사용할 수 있다.

IndexedDB 데이터 관리

- IDBTransaction 인터페이스 속성/함수/이벤트

속성 이름	설명
db	현재 트랜잭션이 연결된 데이터베이스 객체를 나타낸다.
mode	현재 트랜잭션에 설정된 모드를 나타낸다. 기본값은 readonly 이다.
objectStoreNames	현재 트랜잭션에서 사용할 수 있는 오브젝트스토어의 이름을 리스트 형태로 반환한다.

함수 이름	설명
objectStore(store_name)	현재 트랜잭션 영역에 등록된 오브젝트 스토어 중 store_name에 해당하는 객체를 반환한다.
abort()	현재의 트랜잭션 아래에서 수행했던 모든 작업들을 되돌린다.(rollback)

이벤트 이름	설명
abort	트랜잭션에서 abort가 호출되었을 때 동작한다.
complete	트랜잭션이 성공적으로 마무리되었을 때 동작한다.
error	예외 발생으로 트랜잭션이 실패했을 때 동작한다.

IndexedDB 데이터 관리

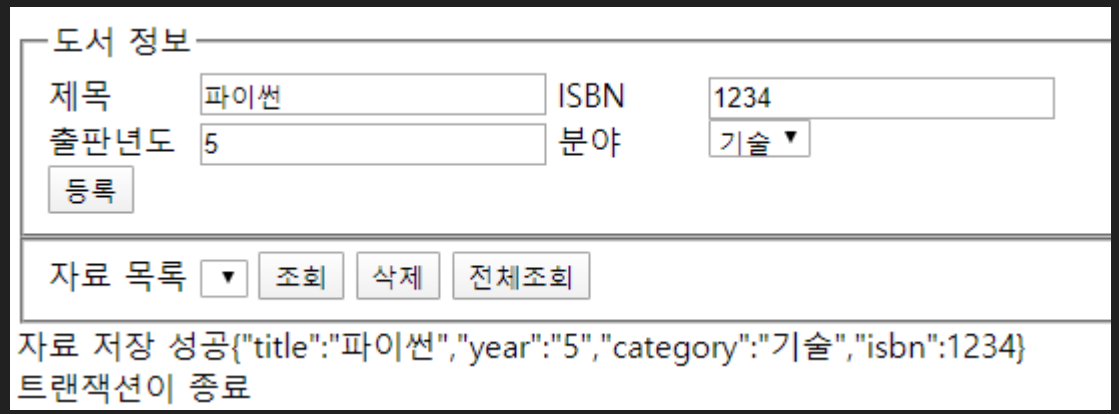
- IDBTransaction 예제 – 1(자료추가)

```
var bookShelfDB;

function openDB() {
    var openResult = window.indexedDB.open("bookShelf", 2);
    openResult.addEventListener("success", function() {
        console.log("데이터베이스 오픈 성공 : " + this.result);
        bookShelfDB = this.result;
        showDatabaseInfo(bookShelfDB);
    });
    openResult.addEventListener("error", function() {
        console.log("데이터베이스 오픈 실패");
        console.log(this.result);
    });
    openResult.addEventListener("upgradeneeded", function() {
        console.log("데이터베이스 업그레이드 진행");
        bookShelfDB = this.result;
        if (bookShelfDB.objectStoreNames.contains("books")) {
            bookShelfDB.deleteObjectStore("books");
        }
        var objStore = bookShelfDB.createObjectStore("books", {
            keyPath: "isbn",
            autoIncrement: true
        });
    });
}
openDB();

function showDatabaseInfo(db) {
    console.log("데이터베이스 이름: " + db.name, "버전: " + db.version);

    console.log("ObjectStore 개수: " + db.objectStoreNames.length);
    for (var i = 0; i < db.objectStoreNames; i++) {
        console.log(i + " : " + db.objectStoreNames[i]);
    }
}
// 자료 추가
var title = document.getElementById("title");
var year = document.getElementById("year");
var category = document.getElementById("category");
var isbn = document.getElementById("isbn");
var dashboard = document.getElementById("dashboard");
```

IndexedDB 데이터 관리

- IDBTransaction 예제 - 1(자료추가)

```
function showTransactionInfo(tx) {
    console.log("Transaction 정보 " + "소속 DB :" + tx.db + ", 모드 : " + tx.mode);
    console.log("연동되는 ObjectStore 정보", tx.objectStoreNames);
    console.log("Transaction 정보 출력 종료------------------------------");
}

function showObjectStoreInfo(os) {
    console.log("ObjectStore 정보", "이름: " + os.name);
    console.log("자동완성 컬럼 보유 여부 : " + os.autoIncrement);
    console.log("키패스 : ", os.keyPath);
    console.log("인덱스 정보: ", os.indexNames);
    console.log("ObjectStore 정보 출력 종료------------------------------");
}
```




```
document.getElementById("registerB").addEventListener("click", function() {
    dashboard.innerHTML = "";
    var data = {
        title: title.value,
        year: year.value,
        category: category.value
    };
    if (isbn.value) {
        data.isbn = Number.parseInt(isbn.value);
    }
    var tx = bookShelfDB.transaction(["books"], "readwrite");
    showTransactionInfo(tx);

    tx.addEventListener("complete", function() {
        dashboard.innerHTML += "트랜잭션이 종료<br>";
    });
    tx.addEventListener("abort", function() {
        dashboard.innerHTML += "트랜잭션이 취소<br>";
    });
    tx.addEventListener("error", function() {
        dashboard.innerHTML += "트랜잭션이 실패<br>";
    });
    var objStore = tx.objectStore("books");
    showObjectStoreInfo(objStore);
    var request = objStore.put(data);
    request.addEventListener("success", function() {
        dashboard.innerHTML += "자료 저장 성공" + JSON.stringify(data) + "<br>";
    });
    request.addEventListener("error", function() {
        dashboard.innerHTML += "자료 저장 실패<br>";
    });
    //tx.abort();
});
```

IndexedDB 데이터 관리

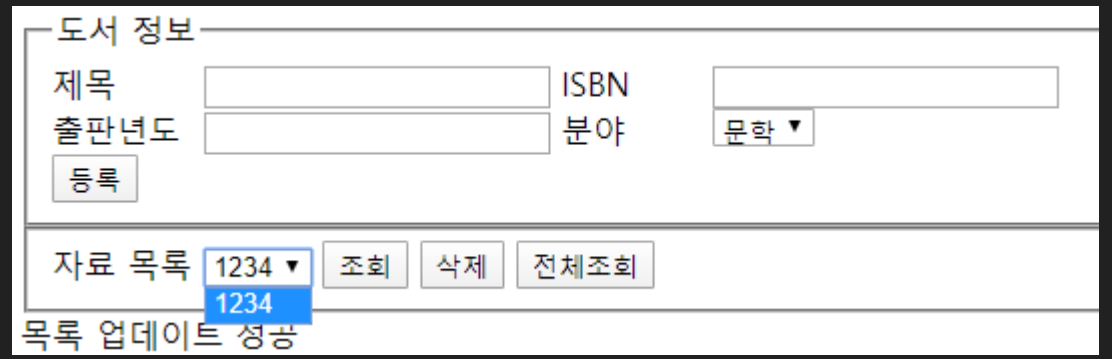
- IDBTransaction 예제 - 1(자료추가)

```
데이터베이스 오픈 성공 : [object IDBDatabase]
데이터베이스 이름: bookShelf 버전: 2
ObjectStore 개수: 1
Transaction 정보 소속 DB :[object IDBDatabase], 모드 : readwrite
연동되는 ObjectStore 정보 ▶DOMStringList
Transaction 정보 출력 종료----------
ObjectStore 정보 이름: books
자동완성 컬럼 보유 여부 : true
키패스 : isbn
인덱스 정보: ▶DOMStringList
ObjectStore 정보 출력 종료----------
```

IndexedDB 데이터 관리

- IDBTransaction 예제 - 2(업데이트)

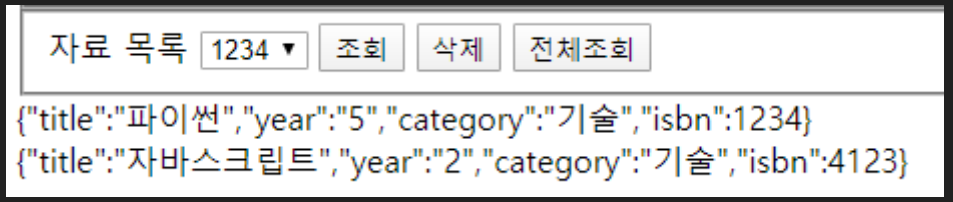
```
// 자료 목록 업데이트
var datas = document.getElementById("datas");

function updateISBNList() {
    dashboard.innerHTML = "";
    var tx = bookShelfDB.transaction(["books"], "readonly");

    var objStore = tx.objectStore("books");
    var request = objStore.getAllKeys();
    request.addEventListener("success", function() {
        datas.innerHTML = "";
        for (var i = 0; i < request.result.length; i++) {
            datas.innerHTML += "<option>" + request.result[i];
        }
        dashboard.innerHTML += "목록 업데이트 성공";
    });
    request.addEventListener("error", function() {
        dashboard.innerHTML += "목록 업데이트 실패<br>";
    });
}
```



IndexedDB 데이터 관리

- IDBTransaction 예제 - 3(전체조회)

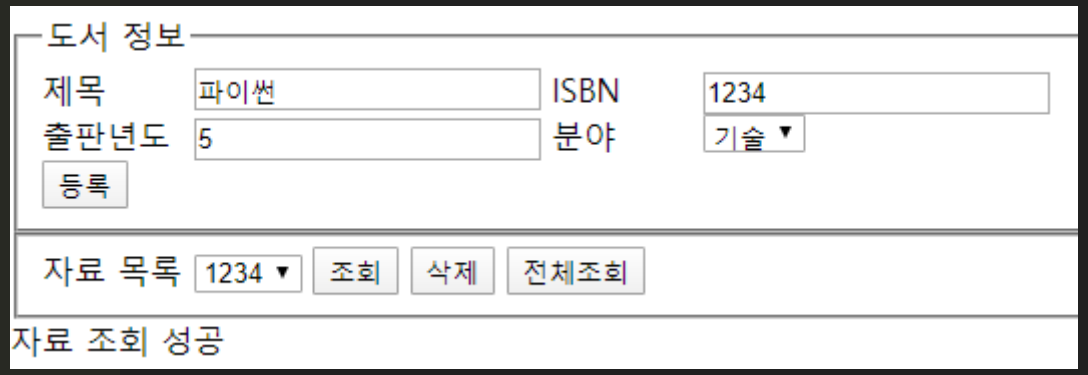
```
document.getElementById("getAllB").addEventListener("click", function() {
    dashboard.innerHTML = "";
    var tx = bookShelfDB.transaction(["books"], "readonly");

    var objStore = tx.objectStore("books");
    var request = objStore.getAll();
    request.addEventListener("success", function() {
        dashboard.innerHTML = "";
        for (var i = 0; i < request.result.length; i++) {
            dashboard.innerHTML += JSON.stringify(request.result[i]) + "<br>";
        }
    });
    request.addEventListener("error", function() {
        dashboard.innerHTML += "자료 조회 실패<br>";
    });
});
```



IndexedDB 데이터 관리

- IDBTransaction 예제 - 4(개별조회)

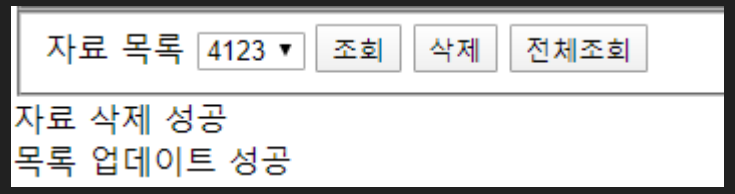
```
document.getElementById("searchB").addEventListener("click", function() {
    var tx = bookShelfDB.transaction(["books"], "readonly");

    var objStore = tx.objectStore("books");
    var request = objStore.get(Number.parseInt(datas.value));
    request.addEventListener("success", function() {
        dashboard.innerHTML = "자료 조회 성공";
        var data = this.result;
        title.value = data.title;
        isbn.value = data.isbn;
        year.value = data.year;
        category.value = data.category;
    });
    request.addEventListener("error", function() {
        dashboard.innerHTML += "자료 조회 실패<br>";
    });
});
```



IndexedDB 데이터 관리

- IDBTransaction 예제 - 5(개별삭제)

```
document.getElementById("delB").addEventListener("click", function() {
    var tx = bookShelfDB.transaction(["books"], "readwrite");

    var objStore = tx.objectStore("books");
    var request = objStore.delete(Number.parseInt(datas.value));
    request.addEventListener("success", function() {
        updateISBNList();
        dashboard.innerHTML = "자료 삭제 성공<br>";
    });
    request.addEventListener("error", function() {
        dashboard.innerHTML += "자료 삭제 실패<br>";
    });
});
```



IndexedDB 고급 조회

- 고급조회
  - 고급 조회 시 필요한 객체는 IDBIndex, IDBCursor, IDBKeyRange 객체가 필요하다.

IndexedDB 고급 조회

- KeyRange
  - 키레인지(KeyRange)는 이름 그대로 검색하려는 키의 범위를 제한하는 역할을 한다.
  - 이를 통해 검색 조건의 최소, 최대 범위 등을 함수를 통해 설정할 수 있다.
  - HTML5 에서는 IDBKeyRange 인터페이스가 이 역할을 수행한다. 아래는 해당 객체의 속성과 함수이다.

속성 이름	설명
lower	키레인지의 최소값(하안선)으로 설정되지 않은 경우 undefined 이다.
upper	키레인지의 최대값(상한선)으로 설정되지 않은 경우 undefined 이다.
lowerOpen	최소값에 등호가 포함되면 true, 아니면 false 이다.
upperOpen	최대값에 등호가 포함되면 true, 아니면 false 이다.

IndexedDB 고급 조회

- KeyRange

함수 이름	설명
lowerBound(value, lowerOpen)	조회하려는 최소값을 지정한다. lowerOpen은 등호를 포함할지 설정하는 Boolean 값이다.
upperBound(value, upperOpen)	조회하려는 최대값을 지정한다. ipperOpen은 등호를 포함할지 설정하는 Boolean 값이다.
bound(lower, upper, lowerOpen, upperOpen)	조회하려는 최대값과 최소값을 지정한다. 등호 포함 여부는 각각 lowerOpen, upperOpen 으로 설정한다.
only(value)	범위가 아닌 등호(=)로 값을 조회한다.
includes(value)	value 가 지정된 키레인지의 범위에 포함되는지를 Boolean 값으로 반환한다.

IndexedDB 고급 조회

- KeyRange 예제 - 1

```
<fieldset id="search">
    <legend>정보 조회</legend>
    <button id="getAllB">전체조회</button>
    <button id="getByIsbnRangeB">ISBN범위 조회</button>
    (
    <input type="number" id="lowerISBN" placeholder="최소"
        class="smallInput">
    ~
    <input type="number" id="upperISBN" placeholder="최대"
        class="smallInput">
    )<br>
    <button id="titleIdxB">타이틀인덱스</button>
    <input type="text" id="startChar" placeholder="시작" class="smallInput">
    부터
    <input type="text" id="endChar" placeholder="종료" class="smallInput">
    까지의 제목 <br>
    <button id="yearIdxB">출판년도인덱스</button>
    <input type="text" id="fromYear" placeholder="년도" class="smallInput">
    부터
    <input type="text" id="toYear" placeholder="년도" class="smallInput">
    까지 출판된 도서 (정렬:
    <input type="radio" id="next" name="orderBy" checked="checked">
    <label for="next">오름차순</label>
    <input type="radio" id="prev" name="orderBy">
    <label for="prev">내림차순</label>
    )
</fieldset>
<div id="dashboard"></div>
```

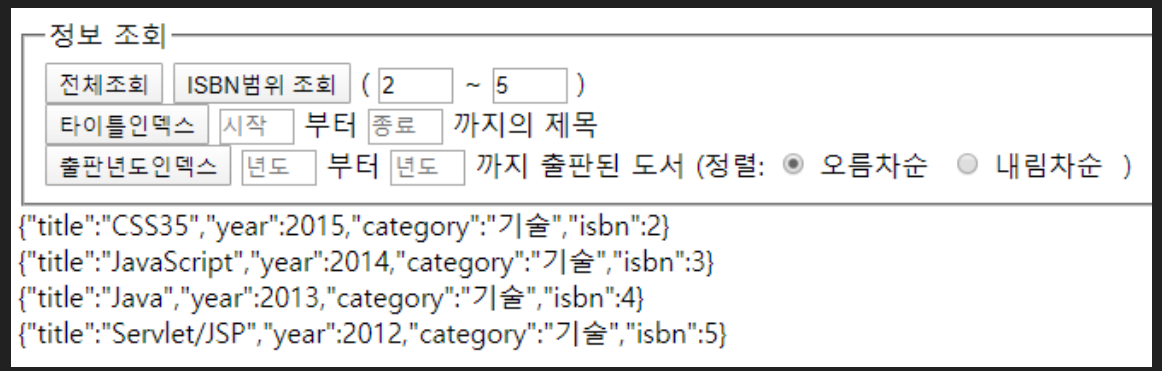
IndexedDB 고급 조회

- KeyRange 예제 - 1

```
var dashboard = document.getElementById("dashboard");
var bookShelfDB;

function openDB() {
    indexedDB.deleteDatabase("bookShelf");
    var openResult = window.indexedDB.open("bookShelf", 1);
    openResult.addEventListener("success", function() {
        bookShelfDB = this.result;
        initData();
    });
    openResult.addEventListener("upgradeneeded", function() {
        bookShelfDB = this.result;
        if (bookShelfDB.objectStoreNames.contains("books")) {
            bookShelfDB.deleteObjectStore("books");
        }
        var objStore = bookShelfDB.createObjectStore("books", {
            keyPath: "isbn",
            autoIncrement: true
        });
    });
}
openDB();

function initData() {
    var tx = bookShelfDB.transaction(["books"], "readwrite");

    var objStore = tx.objectStore("books");
    objStore.put({title: "HTML5",year: 2016,category: "기술"});
    objStore.put({title: "CSS35",year: 2015,category: "기술"});
    objStore.put({title: "JavaScript",year: 2014,category: "기술"});
    objStore.put({title: "Java",year: 2013,category: "기술"});
    objStore.put({title: "Servlet/JSP",year: 2012,category: "기술"});
    objStore.put({title: "JQuery",year: 2011,category: "기술"});
    objStore.put({title: "SQL",year: 2010,category: "기술"});
    objStore.put({title: "Spring",year: 2009,category: "기술"});
    objStore.put({title: "MyBatis",year: 2008,category: "기술"});
    objStore.put({title: "XML",year: 2007,category: "기술"});
}
```

IndexedDB 고급 조회

- KeyRange 예제 - 1



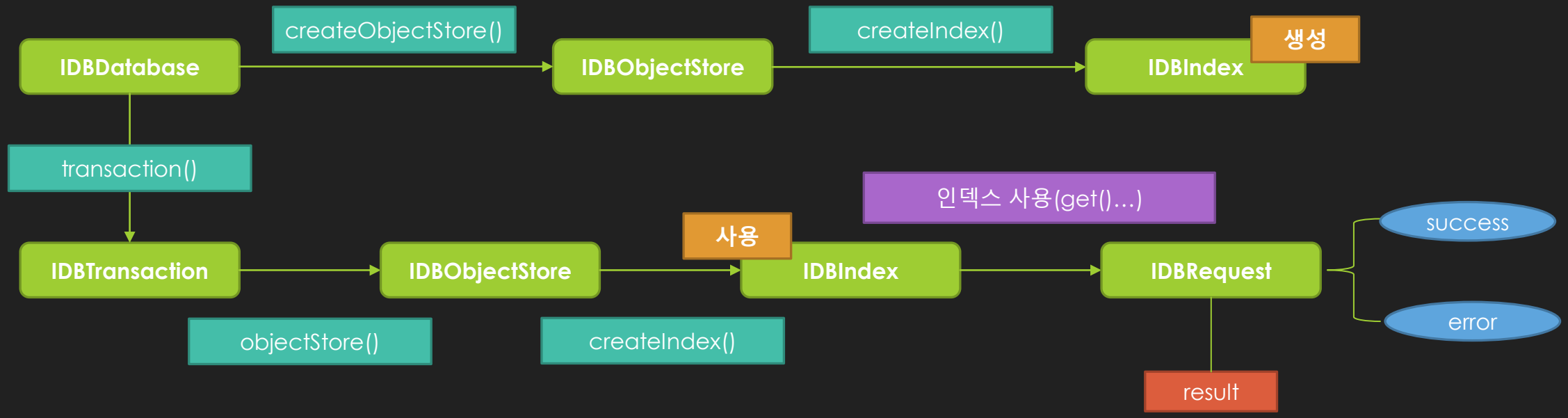

```
document.getElementById("getAllB").addEventListener("click", function() {
    dashboard.innerHTML = "";
    var tx = bookShelfDB.transaction(["books"], "readonly");

    var objStore = tx.objectStore("books");
    var request = objStore.getAll();
    request.addEventListener("success", function() {
        dashboard.innerHTML = "";
        for (var i = 0; i < request.result.length; i++) {
            dashboard.innerHTML += JSON.stringify(request.result[i]) + "<br>";
        }
    });
});

var lowerIsbn = document.getElementById("lowerISBN");
var upperIsbn = document.getElementById("upperISBN");

document.getElementById("getByIsbnRangeB").addEventListener("click",function() {
        var keyRange;
        var lowerValue = Number.parseInt(lowerIsbn.value);
        var upperValue = Number.parseInt(upperIsbn.value);

        if (!(lowerValue || upperValue)) {
            dashboard.innerHTML = "최소값과 최대값이 필요합니다.";
            return;
        } else {
            keyRange = IDBKeyRange.bound(lowerValue, upperValue, false,
                false);
        }
        var tx = bookShelfDB.transaction(["books"], "readonly");

        var objStore = tx.objectStore("books");
        var request = objStore.getAll(keyRange);
        request.addEventListener("success", function() {
            dashboard.innerHTML = "";
            for (var i = 0; i < request.result.length; i++) {
                dashboard.innerHTML += JSON.stringify(request.result[i]) +
                    "<br>";
            }
        });
    });
```

IndexedDB 고급 조회

- Index
  - 출판연도, 제목의 문자열 등을 기준으로도 검색하기 위해선 기존의 keypath 만으로 부족하다.
  - 이런 추가적인 검색을 위해 Index 를 사용한다.
  - HTML5 에서는 인덱스를 위해 IDBIndex 인터페이스를 제공한다.
  - 기본적으로 오브젝트스토어를 생성할 때 설정한 keyPath가 인덱스로 사용되는데 추가로 다른 인덱스를 생성하려는 경우 오브젝트 스토어에 있는 createIndex() 함수를 이용한다.
  - 주의할 점은 이 함수는 VersionChange 트랜잭션 모드에서 호출되어야 하므로 upgradeneeded 이벤트에서 호출된다.

IndexedDB 고급 조회

- Index



IndexedDB 고급 조회

- Index

속성 이름	설명
name	인덱스의 이름을 나타낸다
objectStore	인덱스가 속해 있는 오브젝트스토어를 나타낸다
keypath	인덱스가 사용하는 keypath 를 나타낸다.
unique	Boolean 형태로 true 인 경우 인덱스의 값은 중복될 수 없다.

함수 이름	설명
count()	IDBRequest 를 반환하며 비동기적으로 동작한다. IDBRequest 의 result 속성에는 인덱스를 이용해서 저장된 자료의 개수를 표시한다.
get(condition)	IDBRequest 를 반환하며 비동기적으로 동작한다. IDBRequest 의 result 에는 지정된 condition에 부합되는 자료가 할당된다. condition에 키레인지 객체를 할당해서 여러 개의 자료가 해당할 때는 처음 발견된 자료가 반환된다.
getAll([condition [, count]])	IDBRequest 를 반환하며 비동기적으로 동작한다. IDBRequest 의 result 에는 지정된 condition에 부합되는 자료들이 할당된다. 파라미터를 생략할 경우 모든 자료를 조회한다. condition에 키레인지 객체를 할당하면 부합되는 모든 자료를 조회할 수 있다. 결과의 개수를 재한하기 위해서는 count 변수를 사용한다.
openCursor(range, direction)	IDBRequest 를 반환하며 비동기적으로 동작한다. IDBRequest의 result에는 지정된 키레인지 객체를 사용하는 IDBCursor 타입의 객체가 할당된다. direction은 정렬 방법을 나타낸다.

IndexedDB 고급 조회

- Index 예제 - 1

```
function openDB() {
    indexedDB.deleteDatabase("bookShelf");
    var openResult = window.indexedDB.open("bookShelf", 1);
    openResult.addEventListener("success", function() {
        bookShelfDB = this.result;
        initData();
    });
    openResult.addEventListener("upgradeneeded", function() {
        bookShelfDB = this.result;
        if (bookShelfDB.objectStoreNames.contains("books")) {
            bookShelfDB.deleteObjectStore("books");
        }
        var objStore = bookShelfDB.createObjectStore("books", {
            keyPath: "isbn",
            autoIncrement: true
        });

        objStore.createIndex("yearIdx", "year", {
            unique: false
        });
        objStore.createIndex("titleIdx", "title", {
            unique: false
        });
    });
}
openDB();
```

```
function showIndexInfo(idx) {
    console.log("인덱스 정보");
    console.log("이름: " + idx.name, "소속 오브젝트스토어: " + idx.objectStore);
    console.log("키패스: " + idx.keyPath, "유일성: " + idx.unique);
}
var startChar = document.getElementById("startChar");
var endChar = document.getElementById("endChar");

document.getElementById("titleIdxB").addEventListener("click", function() {
    var keyRange;
    var startValue = startChar.value;
    var endValue = endChar.value;

    if (!(startValue || endValue)) {
        dashboard.innerHTML = "시작 문자와 끝 문자가 필요합니다.";
        return;
    } else {
        keyRange = IDBKeyRange.bound(startValue, endValue, false, false);
    }
    var tx = bookShelfDB.transaction(["books"], "readonly");

    var objStore = tx.objectStore("books");
    var titleIdx = objStore.index("titleIdx");
    showIndexInfo(titleIdx);
    var request = titleIdx.getAll(keyRange);
    request.addEventListener("success", function() {
        dashboard.innerHTML = "";
        for (var i = 0; i < request.result.length; i++) {
            dashboard.innerHTML += JSON.stringify(request.result[i]) + "<br>";
        }
    });
});
```

IndexedDB 고급 조회

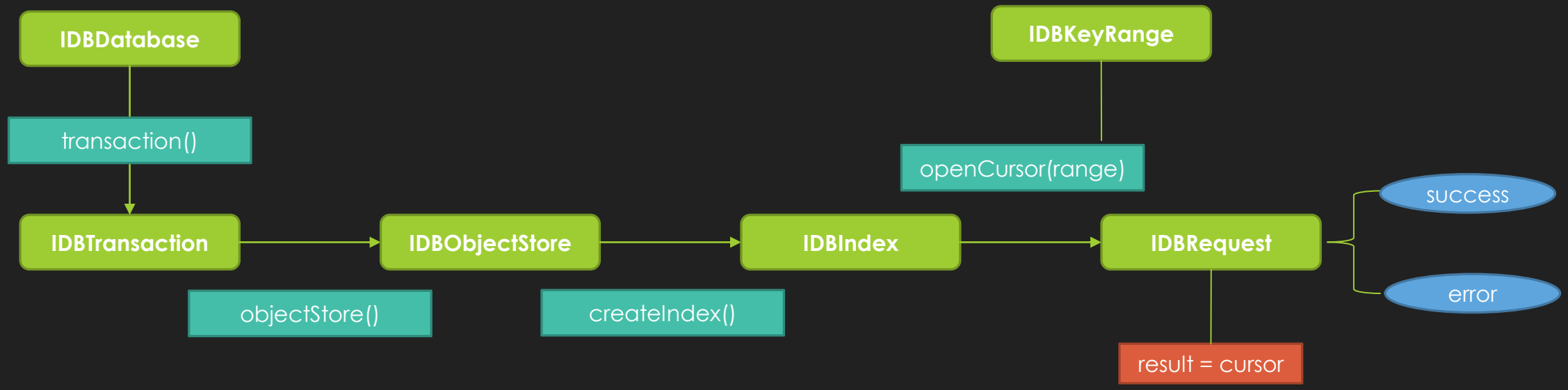
- Index 예제 - 1

IndexedDB 고급 조회

- Cursor
  - 데이터베이스에서의 커서란 select 와 같은 SQL 문장의 실행 결과를 임시로 저장할 수 있는 객체를 의미한다.
  - 포인터는 커서에 저장된 데이터 중 하나를 가리키는 개념으로 이해할 수 있다.
  - 인덱스드디비에서의 커서 역시 조회된 자료들을 관리하는 역할로 포인터를 이용해 인덱스를 통해서 조회된 개별 자료를 가리킬 수 있고 수정, 삭제, 정렬시킬 수 있다.
  - HTML5 에서는 커서를 위해 IDBCursor 인터페이스를 제공한다.
  - 커서는 오브젝트 스토어나 인덱스의 openCursor() 함수를 이용해서 생성할 수 있다.

IndexedDB 고급 조회

- Cursor



IndexedDB 고급 조회

- Cursor

속성 이름	설명
source	커서가 오픈된 오브젝트 스토어나 o니덱스 객체를 나타낸다.
direction	커서의 진행 방법으로 next, nextunique, prev, prevunique 중 하나의 값을 가지며 기본은 next이다.
key	커서에서 사용된 속성의 값을 나타낸다.
primaryKey	조회된 자료의 keypath 속성값을 나타낸다.
value	현재 포인터가 가리키는 객체를 반환한다.

함수 이름	설명
advance(count)	커서의 포인터를 count만큼 이동시킨다.
continue()	커서의 현재 포인터에서 다음 위치로 이동시키고 IDBRequest와 success 이벤트를 다시 발생시킨다. 포인터가 목록의 끝에 도달해도 역시 success 이벤트가 발생하지만 빈 객체가 반환되고 더는 조회할 자료가 없음을 알 수 있다.
delete()	IDBRequest 를 반환하여 비동기적으로 동작한다. 현재 커서 위치에 있는 객체를 삭제한다.
update(value)	IDBRequest 를 반환하여 비동기적으로 동작한다. 현재 커서 위치에 있는 객체의 내용을 value 값으로 수정한다.

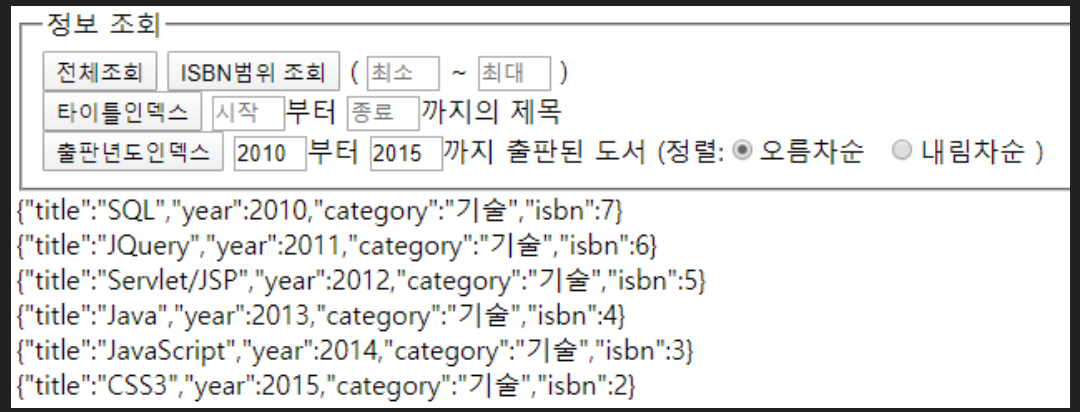
IndexedDB 고급 조회

- Cursor
  - 커서의 속성 중 direction 속성에 사용할 수 있는 값은 아래와 같다.

값	설명
next	조회한 조건에 대해 오름차순으로 정렬한다. 기본값
nextunique	조회한 조건에 대해 오름차순으로 정렬한다. 단 키가 중복되는 경우 처음 자료만 조회된다.
prev	조회한 조건에 대해 내림차순으로 정렬한다.
prevunique	조회한 조건에 대해 내림차순으로 정렬한다. 단 키가 중복되는 경우 처음 자료만 조회된다.

IndexedDB 고급 조회

- Cursor 예제




```
var fromYear = document.getElementById("fromYear");
var toYear = document.getElementById("toYear");

document.getElementById("yearIdxB").addEventListener("click", function() {
    var keyRange;
    var fromValue = Number.parseInt(fromYear.value);
    var toValue = Number.parseInt(toYear.value);

    if (!(fromValue || toValue)) {
        dashboard.innerHTML = "시작 년도와 끝 년도가 필요합니다.";
        return;
    } else {
        keyRange = IDBKeyRange.bound(fromValue, toValue, false, false);
    }
    var tx = bookShelfDB.transaction(["books"], "readonly");

    var objStore = tx.objectStore("books");
    var yearIdx = objStore.index("yearIdx");
    var dir = document.getElementById("prev").checked ? "prev" : "next";
    var request = yearIdx.openCursor(keyRange, dir);
    dashboard.innerHTML = ""
    request.addEventListener("success", function() {
        var cursor = request.result;
        if (cursor) {
            dashboard.innerHTML += JSON.stringify(cursor.value) + "<br>";
            cursor.continue();
        }
    });
});
```