Web SQL Database

➔ Introducción

Web SQL Database es una API web para almacenar datos del lado del cliente en bases de datos que pueden ser consultadas usando una variante de SQL. La API es soportada por Google Chrome, Opera, Safari y el navegador de Android.

Aunque comenzó a formar parte de W3C públicamente en 2009, el Grupo de Trabajo de Aplicaciones Web de W3C dejó de trabajar en la especificación en noviembre de 2010, citando una falta de implementaciones independientes (como sería el uso algún sistema de bases de datos distinto de SQLite tomando el rol en el backend) como la razón de que la especificación no pudiera seguir en el proceso de convertirse en una recomendación de W3C.

Mozilla Corporation fue una de las principales voces tras la rotura de negocios y el hecho de que el estándar quedase obsoleto, mientras al mismo tiempo eran los principales impulsores detrás de un estándar de “almacenamiento alternativo”, IndexedDB.

➔ Ventajas e inconvenientes

La principal ventaja de WebSQL es precisamente su implementación en el cliente, basada sobre SQLite. Esto permite proporcionar una base de datos verdaderamente relacional. Además, se trata de una API asíncrona, por lo que la programación funcional, concretamente el uso de funciones *lambda*, resulta especialmente útil en WebSQL.

Sin embargo, el hecho de que la especificación esté obsoleta ya es el primer inconveniente, pues indica que que el W3C ya no lo soporta. La especificación cesó sus actualizaciones debido a la dependencia constante de SQLite, lo que hacía difícil sostener a WebSQL como una API sólida que se mantuviese por sí sola.

Otro obstáculo a tener en cuenta es la necesidad de transformar todos los objetos de JavaScript en esquemas relacionales, además de requerir el aprendizaje del lenguaje SQL. Esto se debe también a que WebSQL no está orientado a objetos, un paradigma de programación que aún está muy extendido hoy en día.

Debido a los problemas que presenta, WebSQL ha sido ampliamente sustituido por IndexedDB. Aunque puede presentar problemas para los programadores acostumbrados a las bases de datos relacionales, posee una serie de ventajas sobre WebSQL.

➔ Funcionamiento

Como buena API de SQL, funciona a base de tablas que se relacionan entre sí, y contienen columnas y filas. Cada tabla contiene una serie de atributos que las definen. Su mecanismo de consulta es SQL.

El método `openDatabase()` se usa para acceder a la base de datos. Si no existe, el método la crea y luego la abre. Sus argumentos son el nombre real de la base, la versión, el nombre con el que se mostrará, su tamaño estimado en bytes y, opcionalmente, la función que se usará como manejador de eventos.

```
var db;
var version = 1.0;
var dbName = "tizendb";
var dbDisplayName = "tizen_test_db";
var dbSize = 2 * 1024 * 1024;
try
{
   db = openDatabase(dbName, version, dbDisplayName, dbSize, function(database)
   {
      alert("database creation callback");
   });
}
```

Las sentencias SQL pueden ejecutarse de forma asíncrona, encapsulándolas mediante el método `transaction()` o `readTransaction()`. Hay que tener en cuenta que `readTransaction()` sólo se puede usar con operaciones de lectura, no creación, borrado ni modificación.

```
db.transaction(function(t)
{
   /* Place SQL statements here */
}, function()
{
   alert("SQL statements were executed successfully.");
});
```

Para ejecutarlo, usamos el método `executeSql()`.

```
t.executeSql("CREATE TABLE tizenTable (id INTEGER PRIMARY KEY, title TEXT, content
TEXT, insertDay DATETIME)",
             [], function(sqlTransaction, sqlResultSet)
{
   alert("Table has been created.");
}, function(sqlTransaction, sqlError)
{
   /* Error handling */
});
```

En cuanto a manejar errores, se hace a través del objeto `sqlError`.

```
sqlTransaction.executeSql("SELECT * FROM notExistingTable", [],
                          function(sqlTransaction, sqlResultSet) {},
                          function(sqlTransaction, sqlError)
{
   switch (sqlError.code)
   {
      case sqlError.SYNTAX_ERR:
         alert("Syntax error has occurred. " + sqlError.message);
         break;
      default:
       alert("Other error");
   }
});
```

Se accede a los resultados de la sentencia mediante el objeto `sqlResultSet`.

```
sqlTransaction.executeSql("INSERT INTO tizenTable(title, content, insertDay) VALUES
(?, ?, ?)",
                                    [title, context, day], function(sqlTransaction,
sqlResultSet)
{
   alert("The 'id' of the new record  is " + sqlResultSet.insertId);
});
```

También es posible realizar sentencias SQL de forma síncrona, mediante `openDatabaseSync()`.

```
database.transaction(function(sqlTransactionSync)
{
   /* Place SQL statements here */
}, function()
{
   alert("SQL statements were executed successfully.");
});
```

Para acceder a los resultados de forma síncrona también se usa el objeto `sqlResultSet`, aunque de forma ligeramente distinta.

```
var sqlResultSet = sqlTransactionSync.executeSql("INSERT INTO books (id, title,
author) VALUES(NULL, ?, ?)",
                                                 ["Ulysses", "James Joyce"]);
alert("The 'id' of the new record  is " + sqlResultSet.insertId);
```

En este caso, si se quiere manejar los posibles errores se debe usar el objeto `sqlException`.

```
try
{
   databaseSync.transaction(function(sqlTransactionSync)
   {
      var sqlResultSet = sqlTransactionSync.executeSql("DELETE FROM books WHERE
id=?", [id]);
   });
   /* Instructions if the above SQL statement is executed successfully */
}
catch (sqlException)
{
   postMessage("An error has occurred during deleting the book from the table!
               Error code: " + sqlException.code + " (" + sqlException.message +
").");
}
```