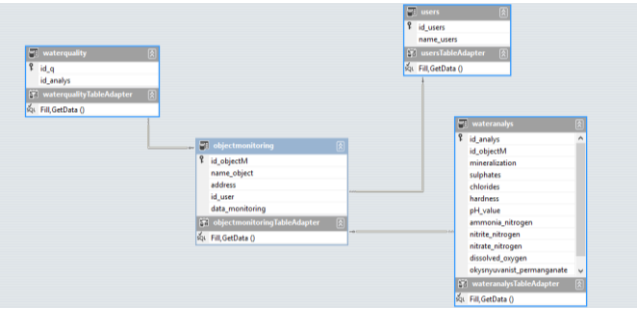
оперативно-стратегічному та оперативному рівнях державного управління.

Функціонування системи санітарно-епідеміологічної нагляду пов'язане зі значними потоками інформації, що вимагає наявність інформаційної системи на базі системи управління базами даних для їх обробки. В якості такої обрана СУБД Microsoft Access. Головним плюсом Access є те, що СУБД може працювати як з локальними додатками, а також її легко інтегрувати в Інтернеті. Схема бази даних наведена нижче.



При розробці таблиць бази даних була проведена їх нормалізація, що істотно скоротило надмірність даних. Інтерфейс користувача системи побудовано на базі MDI-інтерфейса, який передбачає використання віконного інтерфейсу. Для обміну інформацією з користувачами розроблені відповідні форми і звіти.

ПРАКТИЧНЕ ВИКОРИСТАННЯ ЗАМИКАННЯ В JAVASCRIPT

Одегова Є.О.
Науковий керівник – ***Погребняк Б.І., канд. техн. наук, доцент***
(Харківський національний університет радіоелектроніки)

При створенні додатків доволі часто виникає необхідність підрахунку кількість викликів деякої функції. Найпростішим рішенням цієї проблеми є, наприклад, такий код:

```
function counter() {
  let count = 0;

  return ++count;
}

console.log(counter()); // 1
```

```
console.log(counter()); // 1
console.log(counter()); // 1
```

Але, скільки б разів ми не викликали функцію `counter()`, кожного разу на виході отримуємо одне і теж число – 1. Так відбувається тому, що при виклику функції `counter()` спочатку створюється локальна змінна `count` та їй присвоюється значенням 0. Потім вона збільшується на 1 та повертається як результат виконання функції. На цьому функція `counter()` завершує свою роботу і перестають існувати всі її локальні змінні та параметри. При наступному виклику функції все відбувається з самого початку.

Одним із можливих способів збереження значення змінної `count` від одного виклику функції до іншого є винесення її за межі тіла функції `counter()`, наприклад, так:

```
let count = 0;

function counter() {
  return ++count;
}

console.log(counter()); // 1
console.log(counter()); // 2
console.log(counter()); // 3
```

Тепер все працює як треба. Але, це рішення не є надійним – за межами нашої функції `counter()` будь-яка інша функція ненароком, для своєї потреби, може використати таке ж саме ім'я (`count`) та змінити значення нашої змінної. Тому необхідно забезпечити, щоб окрім нашої функції `counter()` весь інший код взагалі «не бачив» змінну `count`.

В мові програмування JavaScript для таких випадків є спеціальна властивість, яка називається замикання [1]. Тоді, з її урахуванням, наш код буде виглядати наступним чином:

```
function makeCounter() {
  let count = 0;

  return () => ++count;
}
```

```
const counter1 = makeCounter();
const counter2 = makeCounter();

console.log(counter1()); // 1
console.log(counter2()); // 1
console.log(counter2()); // 2
console.log(counter1()); // 2
```

Приклад показує, що далі ми можемо створювати необмежену кількість лічильників, які будуть незалежно і надійно виконуватися один від одного.

В JavaScript у кожної виконуваної функції, блоку коду чи скрипта є пов'язаний з ними внутрішній (прихований) об'єкт, який називається лексичним оточенням (англ. Lexical Environment). Цей об'єкт складається з двох частин [1]:

1. об'єкт, в якому як властивості зберігаються всі локальні змінні, а також параметри та деяка інша інформація, така, наприклад, як значення this;
2. посилання на зовнішнє лексичне оточення – тобто те, яке відповідає коду зовні (зовні від поточних фігурних дужок).

Література

1. Флэнаган Д. JavaScript. Подробное руководство, 6-е издание. – Пер. с англ. – СПб: СимволПлюс, 2012. – 1080 с., ил.

ПРАКТИКА ДЕСТРУКТУРИЗАЦІЇ МАСИВІВ В JAVASCRIPT

Одегова Є.О.
Науковий керівник – ***Григорьев О.В., канд. техн. наук, доцент***
(Харківський національний університет радіоелектроніки)

Нехай маємо наступний масив `arr1 = [1, 2, 3, 4, 5]`, елементи якого необхідно присвоїти окремим змінним, наприклад, `a, b, c, d` та `e`. Найочевиднішим вирішенням цієї проблеми буде, наприклад, такий код:

```
const arr1 = [1, 2, 3, 4, 5];

const a = arr1[0];
const b = arr1[1];
const c = arr1[2];
const d = arr1[3];
const e = arr1[4];
```