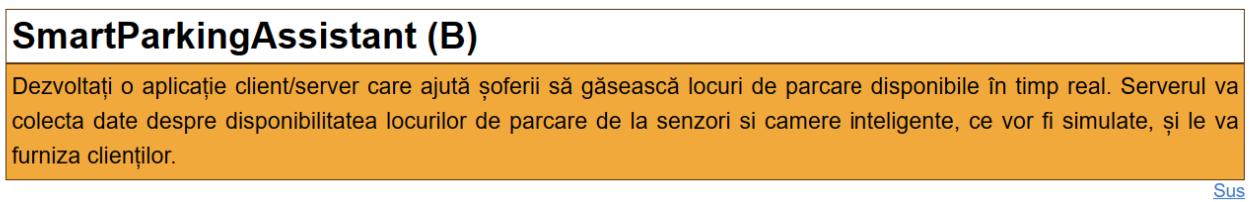


# SmartParkingAssistant - Raport tehnic

Chirilă Gabriela-Valentina

16 Ianuarie 2024



SmartParkingAssistant (B)
Dezvoltați o aplicație client/server care ajută șoferii să găsească locuri de parcare disponibile în timp real. Serverul va colecta date despre disponibilitatea locurilor de parcare de la senzori si camere inteligente, ce vor fi simulate, și le va furniza clienților.

Figure 1: Cerinta proiectului

## 1 Introducere

SmartParkingAssistant este o aplicatie de tip client-server în care un client se poate înregistra, autentifica și obține informații despre locurile de parcare dintr-o locație aleasă de el. Serverul suportă mai mulți clienți, iar informațiile despre orașe/zone și locuri de parcare sunt stocate într-o bază de date SQLite.

## 2 Tehnologii utilizate

### 2.1 TCP/IP

Pentru realizarea conexiunii client-server am ales să utilizez modelul TCP concurent deoarece acesta permite conectarea mai multor clienți la server. Totodată, acest model asigură transmiterea datelor în pachete complete și în ordine. Protocolul TCP ("Transmission Control Protocol") este un protocol orientat conexiune și garantează livrarea corectă a datelor la destinatar, fără duplicări sau pierderi. Dacă aplicația nu ar avea la bază o comunicare sigură, ar exista riscul ca datele trimise să fie alterate și în acest mod clientul să primească informații eronate despre locurile de parcare dintr-o anumită zonă de exemplu. Dacă aș fi ales protocolul UDP, acesta ar fi fost într-adevăr mai rapid în ceea ce privește transmiterea datelor, dar fiind un protocol neorientat conexiune, ar exista mari riscuri în ceea ce privește pierderea de informații și nu ne-ar fi garantat că datele ajung sau nu la destinatar.

### 2.2 SQLite

Pentru stocarea datelor (username, parole, orașe, zone de parcare, rezervări etc.) am folosit baza de date SQLite. Folosind interogări ce țin de inserarea/căutarea datelor putem obține informații

relevante în contextul aplicației noastre.

## 3 Arhitectura aplicației

Realizarea conexiunii dintre client și server se realizează cu ajutorul protocolului TCP multi-threading prin care se creează câte un fir de execuție pentru fiecare client, fiind astfel posibilă servirea mai multor clienți în mod concurent. Comunicarea dintre server și client se realizează prin socket-uri, pentru a se transmite corespunzător fluxul de mesaje și informații. În cadrul aplicației, utilizatorii au la dispoziție mai multe comenzi.

### 3.1 Înainte de autentificare

- **Register** : Utilizatorul alege un username și o parolă, iar dacă acestea sunt valide, acestea sunt introduse în baza de date și se poate autentifica cu respectivele
- **Login** : Utilizatorul introduce un username și o parolă, iar dacă acestea sunt valide, va avea la dispoziție un alt meniu din care poate alege să vadă locurile disponibile de parcare, să rezerve un loc sau să-și vadă istoricul parcărilor
- **Exit** : Această opțiune reprezintă ieșirea din aplicație

### 3.2 După autentificare

- **See available parking spots** : Utilizatorul alege un oraș dintr-o listă de orașe, apoi o anumită zonă din orașul ales, iar apoi primește numărul de locuri de parcare disponibile din acea parcare. Dacă sunt locuri disponibile, acesta este întrebat dacă dorește să rezerve un loc în acea parcare, iar dacă răspunsul este afirmativ, rezervarea va fi salvată în istoricul lui de rezervări.
- **See parking history** : Utilizatorul poate să își vadă istoricul rezervărilor de-a lungul timpului: orașele și zonele în care acesta a rezervat locuri de parcare precum și data și ora rezervării
- **Logout** : Prin această comandă, utilizatorul se deconectează și este trimis la meniul de înainte de autentificare

### 3.3 Diagrama Use Case

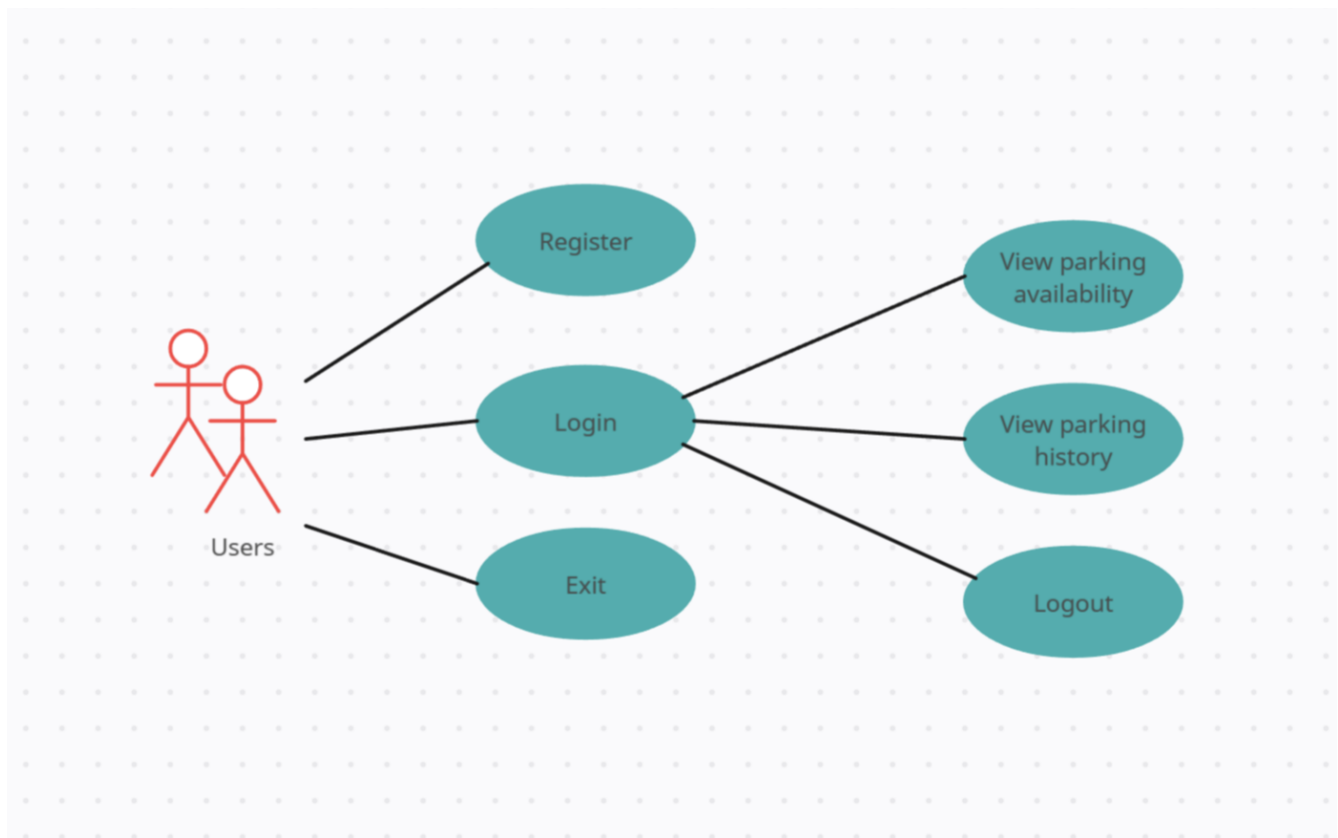


Figure 2: Diagrama Use Case

### 3.4 Diagrama generală a aplicației

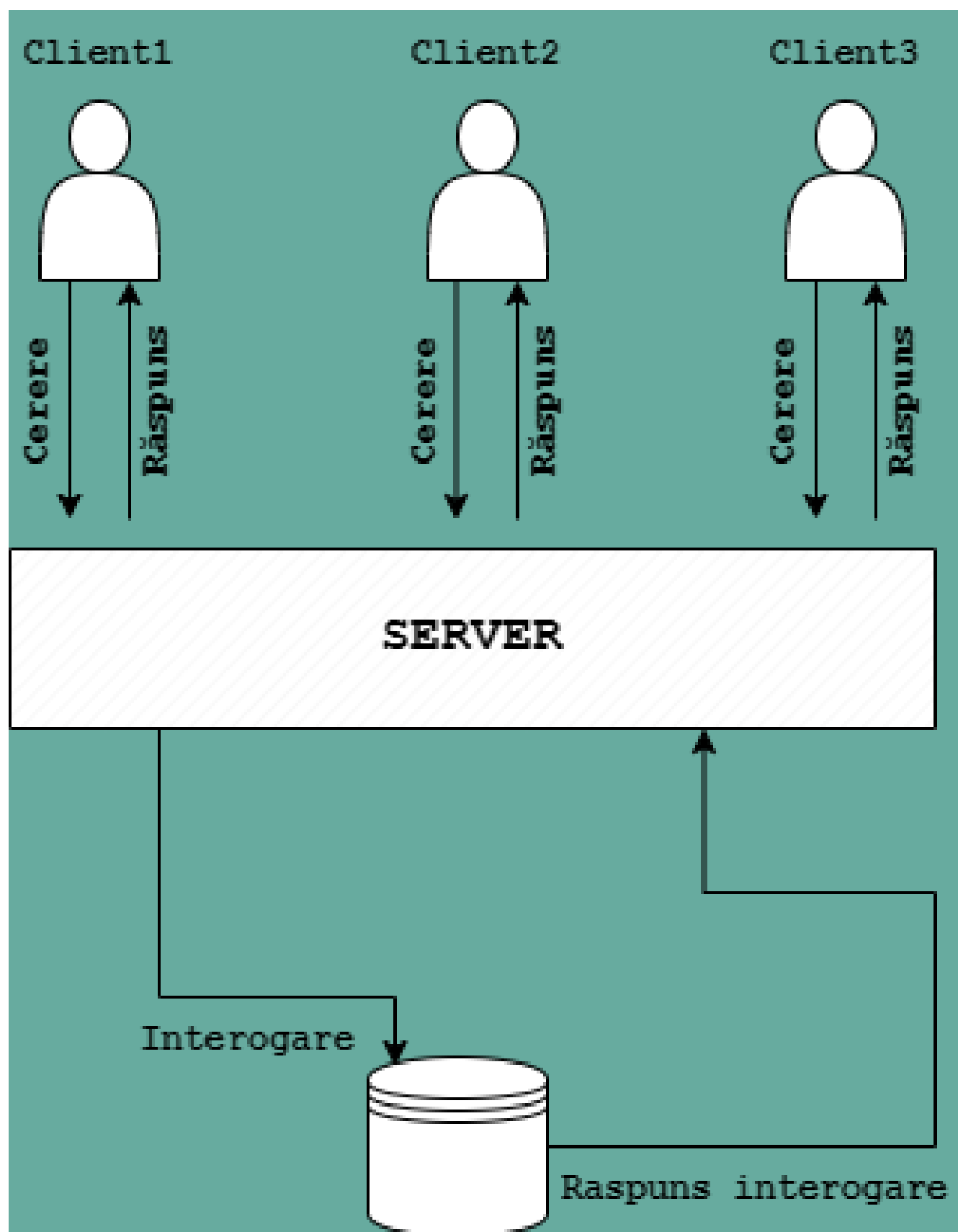


Figure 3: Diagrama generală a aplicației

### 3.5 Diagrama detaliată a aplicației

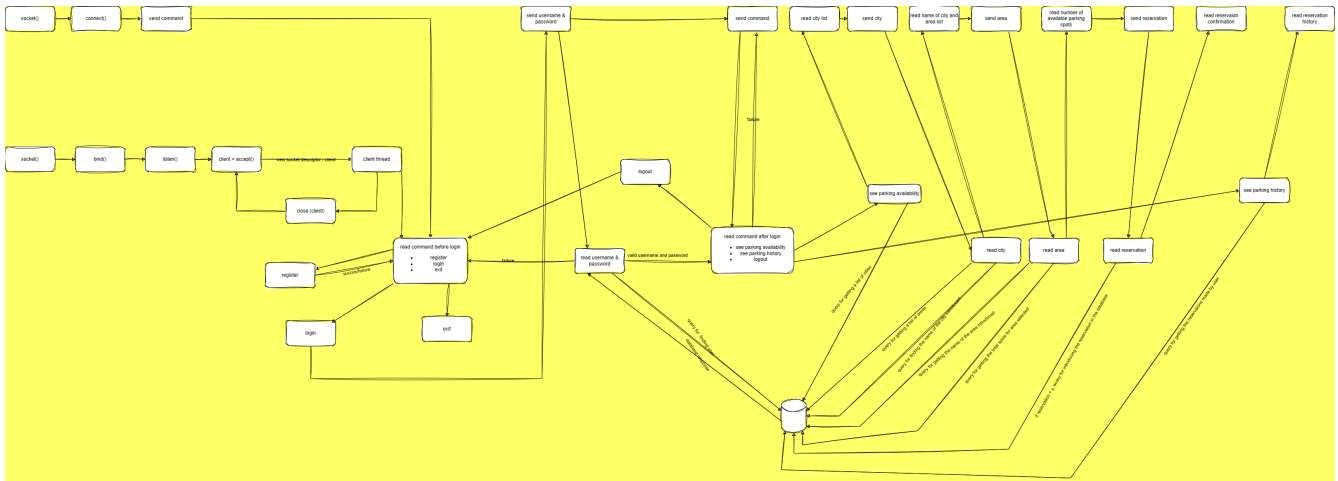


Figure 4: Diagrama detaliată a aplicației

## 4 Detalii de implementare

Comunicarea între server și client se realizează în primul rând cu ajutorul unui socket. Utilizarea unui socket este mult mai avantajoasă datorită bidirecționalității sale, în contrast cu pipe-urile/fifo, care sunt unidirecționale. Această caracteristică este benefică întrucât permite schimbul de informații în ambele direcții între server și client, spre deosebire de pipe-uri sau fifo unde fluxul de date este unidirecțional.

Avantajul utilizării socket-urilor devine evident și în contextul gestionării conexiunilor, deoarece nu este necesară implementarea închiderii capetelor canalului de transmisie. Spre deosebire de alte mecanisme de comunicare, socket-urile permit o manipulare mai flexibilă a comunicației bidirecționale, facilitând schimbul de informații în ambele sensuri fără a necesita operațiuni suplimentare pentru închiderea și deschiderea canalului de comunicație.

- **Structura thData:** este destinată să stocheze informații specifice firului de execuție, cum ar fi identificatorul de fir, descriptorul de client și identificatorul de utilizator.

```

4  typedef struct {
5      int idThread;
6      int cl; // descriptorul intors de accept
7      int userId;
8  } thData;
9

```

Figure 5: Structura thData

- **Structura Response:** structură care să încapsuleze rezultatul operațiunilor, facilitând transmiterea atât a stării de succes/eșec, cât și a mesajelor transmise de la server către client.

```

6   typedef struct {
7       int success;
8       char message[MAX_MESSAGE_LENGTH];
9   } Response;
10

```

Figure 6: Structura Response

- **Funcții care operează cu baza de date:** deschidere conexiune la baza de date SQLite și închiderea acesteia

```

30  sqlite3 *open_database_connection() {
31      sqlite3 *db;
32      int rc;
33
34      rc = sqlite3_open("proiect-RC.db", &db);
35
36      if (rc) {
37          fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
38          exit(1);
39      } else {
40          fprintf(stderr, "Opened database successfully\n");
41      }
42
43      return db;
44  }
45
46  void close_database_connection(sqlite3 *db) {
47      sqlite3_close(db);
48  }

```

Figure 7: Funcții pentru deschiderea/închiderea conexiunii la baza de date

- **Crearea unui socket:**
  - **AF\_INET:** Familia de adrese, în acest caz, IPv4.
  - **SOCK\_STREAM:** Tipul de socket, care indică o transmisie fiabilă (TCP)

- **0**: Protocolul. În acest caz, este 0, indicând faptul că sistemul de operare ar trebui să aleagă protocolul adecvat pe baza familiei de adrese și tipului de socket specificate.

```
67     if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
68     {
69         perror ("[server]Eroare la socket().\n");
70         return errno;
71     }
72
```

Figure 8: Enter Caption

- **Server-side: Funcția de login :**

```

162 int login_user(thData *tdL, sqlite3 *db) {
163     int ok = 0;
164
165     char username[MAX_USERNAME_LENGTH];
166     char password[MAX_PASSWORD_LENGTH];
167     bzero(username, MAX_USERNAME_LENGTH);
168     bzero(password, MAX_PASSWORD_LENGTH);
169
170     if (read(tdL->cl, username, sizeof(username)) <= 0) {
171         printf("[Thread %d] Error reading username from client. \n", tdL->idThread);
172     }
173
174     if (read(tdL->cl, password, sizeof(password)) <= 0) {
175         printf("[Thread %d] Error reading password from client. \n", tdL->idThread);
176     }
177
178     Response response;
179     response.success = 0;
180
181     if (username == NULL || password == NULL || username[0] == '\n' || password[0] == '\n') {
182         printf("Complete with valid username and/or password!\n");
183         snprintf(response.message, sizeof(response.message), "Complete with valid username and/or password!\n");
184     } else {
185         char *ErrMsg = 0;
186         sqlite3_stmt *res;
187         int rc;
188
189         char select[200] = "SELECT id from users WHERE username='";
190         strcat(select, username);
191         strcat(select, "' and Password='");
192         strcat(select, password);
193         strcat(select, "';");
194
195         rc = sqlite3_prepare_v2(db, select, -1, &res, 0);
196
197         if (rc != SQLITE_OK) {
198             fprintf(stderr, "[server]SQL error: %s\n", sqlite3_errmsg(db));
199             snprintf(response.message, sizeof(response.message), "Db error...Try again!\n");
200         }
201         else {
202             int step = sqlite3_step(res);
203
204             if (step == SQLITE_ROW) {
205                 response.success = 1;
206                 tdL->userId = get_user_id_from_database(username, db);
207                 printf("tdL.userId = %d \n", tdL->userId);
208                 snprintf(response.message, sizeof(response.message), "Authentification successful! \n");
209                 ok = 1;
210             }
211             else {
212                 snprintf(response.message, sizeof(response.message), "Username or password incorrect... Try again!\n");
213             }
214         }
215     }
216
217     if (write(tdL->cl, &response, sizeof(Response)) <= 0) {
218         perror("[Thread] Error writing to client.\n");
219     }
220
221     if (response.success) {
222         printf("%s\n", response.message);
223     }
224
225     if (ok == 1)
226         return 1;
227     else
228         return 0;
229 }
230

```

Figure 9: Funcția de login

- Threads:



```

121  static void *treat(void * arg)
122  {
123      thData tdL = *((thData*)arg);
124      printf ("[Thread %d] - Asteptam mesajul...\n", tdL.idThread);
125      fflush (stdout);
126      pthread_detach(pthread_self());
127      raspunde((struct thData*)arg);
128      /* am terminat cu acest client, inchidem conexiunea */
129      close ((intptr_t)arg);
130      return(NULL);
131  }
132  };

```

Figure 10: Funcția executată de fiecare thread ce realizează comunicarea cu clienții

## 5 Concluzii

Implementarea SmartParkingAssistant folosește cu succes TCP/IP pentru comunicarea client-server și utilizează SQLite pentru stocarea eficientă a datelor. Utilizarea multi-threading-ului îmbunătățește capacitatea aplicației de a servi mai mulți clienți simultan.

Evoluțiile viitoare pot include rafinarea interfeței cu utilizatorul și eventual extinderea aplicației pentru a suporta mai multe funcții. Actualizările și îmbunătățirile regulate pot îmbunătăți și mai mult gradul de utilizare a aplicației și pot răspunde oricăror cerințe.

În concluzie, SmartParkingAssistant demonstrează o bază solidă pentru o aplicație client-server cu manipulare eficientă a datelor și interacțiune cu utilizatorul. Deși există anumite provocări și zone de îmbunătățire, proiectul oferă un cadru solid pentru dezvoltarea și rafinarea continuă.

## 6 Bibliografie

- <https://profs.info.uaic.ro/computernetworks/cursullaboratorul.php>
- <https://profs.info.uaic.ro/computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh.c>
- <https://profs.info.uaic.ro/computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c>
- <https://profs.info.uaic.ro/computernetworks/files/NetEx/S12/ServerConcThread/cliTcpNr.c>
- <https://www.sqlite.org/cintro.html>
- <https://zetcode.com/db/sqlitec/>