Лабораторна робота

Тема: Визначення структури даних у SQLite.

Мета: Створення та видалення таблиці. Прикріплення бази даних. Типи даних. Обмеження стовпців та таблиць. Зовнішні ключі FOREIGN KEY. Зміна таблиць та стовпців.

Хід роботи

Створення та видалення таблиці. Прикріплення бази даних

Для створення таблиць використовується команда **CREATE TABLE** . Загальний формальний синтаксис команди **CREATE TABLE** :

```
1   CREATE TABLE table_name
2   (column_name1 data_type column_attributes1,
3    column_name2 data_type column_attributes2,
4    .....................................
5    column_nameN data_type column_attributesN,
6    table_attributes
7   )
```

Після команди **CREATE TABLE** вказується назва таблиці. Ім'я таблиці виконує роль її ідентифікатора у базі даних, тому має бути унікальним. Крім того, воно не повинне починатися на "sqlite_", оскільки назви таблиць, які починаються на "sqlite_", зарезервовані для внутрішнього користування.

Потім після назви таблиці у дужках перераховуються назви стовпців, їх типи даних та атрибути. Наприкінці можна визначити атрибути для всієї таблиці. Атрибути стовпців та атрибути таблиці вказувати необов'язково.

Створимо найпростішу таблицю. Перед виконанням команди **CREATE TABLE** незалежно від того, що ми використовуємо - консольний клієнт sqlite3, графічний клієнт DB Browser for SQLite або якийсь інший клієнт, спочатку відкриємо базу даних, де ми хочемо створити таблицю.

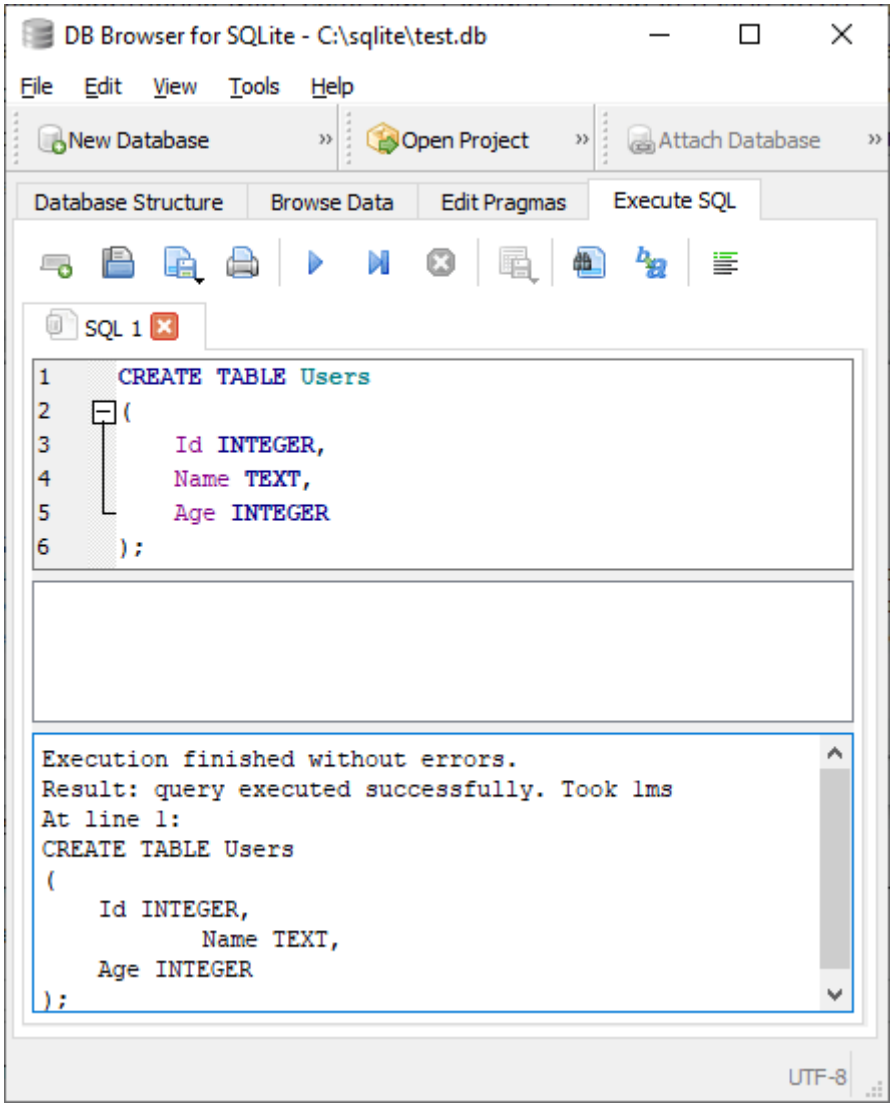
Для створення таблиці виконаємо наступний скрипт:

```
1   CREATE TABLE Users
```

```
2   (
3       Id INTEGER,
4       Name TEXT,
5       Age INTEGER
6   );
```

У цьому випадку таблиця називається "Users". У ній визначено три стовпці: Id, Age, Name. Перші два стовпці являють собою ідентифікатор користувача та його вік і мають тип **INTEGER**, тобто зберігатимуть числові значення. Стовпець "Name" представляє ім'я користувача та має тип **TEXT**, тобто представляє рядок. В даному випадку для кожного стовпця визначено ім'я та тип даних, при цьому атрибути стовпців та таблиці загалом відсутні.

І в результаті виконання цієї команди буде створено таблицю Users із трьома стовпцями.



Створення таблиці за її відсутності

Якщо ми повторно виконаємо вище певну sql-команду для створення таблиці Users, ми зіткнемося з помилкою - ми вже створили таблицю з такою назвою. Але можуть бути ситуації, коли ми можемо точно не знати або не впевнені, чи є в базі даних така таблиця (наприклад, коли ми пишемо додаток якоюсь мовою програмування і використовуємо базу даних, яка не нами створена). І щоб уникнути помилки, за допомогою виразу **IF NOT EXISTS** ми можемо встановити створення таблиці, якщо вона не існує:

```
1   CREATE TABLE IF NOT EXISTS Users
2   (
3       Id INTEGER,
4       Name TEXT,
5       Age INTEGER
6   );
```

Якщо таблиці Users немає, вона буде створена. Якщо вона є, то ніяких дій не робитиметься, і помилки не виникне.

Прикріплення бази даних

Також ми можемо прикріпити базу даних і потім у ній створити базу даних.

Для прикріплення бази даних застосовується команда **ATTACH DATABASE** :

```
1   ATTACH DATABASE 'C:\sqlite\test.db' AS test;
```

Після команди **ATTACH DATABASE** вказується шлях до файлу бази даних (в даному випадку це шлях "C:sqlite\test.db"). Потім після оператора **AS** йде псевдонім, на який проектуватиметься база даних. Тобто в коді для звернення до бази даних "C:sqlite\test.db" буде застосовуватись ім'я "text". При зверненні до таблиці з цієї бази даних спочатку вказується псевдонім бази даних і через точку назву таблиці:

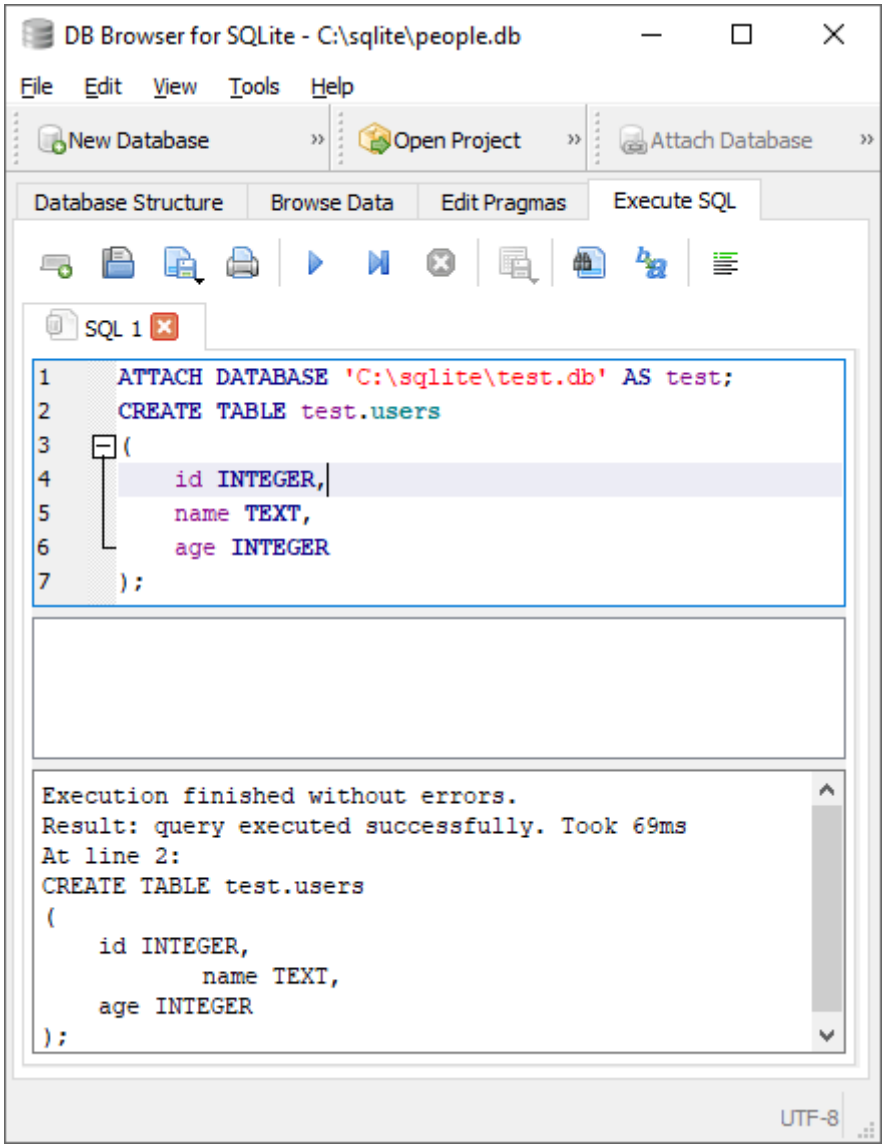
```
1   db.table
```

Наприклад, створимо таблицю у прикріпленій базі даних:

```
1   ATTACH DATABASE 'C:\sqlite\test.db' AS test;
2   CREATE TABLE test.users
3   (
4       id INTEGER,
5       name TEXT,
```

```
6       age INTEGER
7   );
```

Для створення таблиці users в бд test.db назва таблиці випереджається псевдонімом: test.users.



І після відкриття бази даних test.db у ній можна побачити таблицю users.

Видалення таблиць

Для видалення таблиці застосовується команда **DROP TABLE** , після якої вказується назва таблиці, що видаляється. Наприклад, видалімо таблицю users:

```
1   DROP TABLE users;
```

За аналогією до створення таблиці, якщо ми спробуємо видалити таблицю, яка не існує, то ми зіткнемося з помилкою. У цьому випадку знову ж таки за допомогою операторів **IF EXISTS** перевіряти наявність таблиці перед видаленням:

```
DROP TABLE IF EXISTS users;
```

Типи даних

При визначенні стовпців таблиці їм необхідно вказати тип даних. Кожен стовпець має певний тип даних. Для зберігання даних у SQLite застосовуються такі типи:

- **NULL** : вказує фактично відсутність значення
- **INTEGER** : представляє ціле число, яке може бути позитивним та негативним і в залежності від свого значення може займати 1, 2, 3, 4, 6 або 8 байт
- **REAL** : представляє число з точкою, що плаває, займає 8 байт в пам'яті
- **TEXT** : рядок тексту в одинарних лапках, що зберігається в кодуванні бази даних (UTF-8, UTF-16BE або UTF-16LE)
- **BLOB** : бінарні дані

Варто зазначити, що SQLite оперує концепцією **класів зберігання** або **storage class** . І насправді всі ці п'ять типів називаються класами зберігання. Концепція класів зберігання дещо ширша, ніж тип даних. Наприклад, клас **INTEGER** по суті поєднує 6 різних цілих типів даних різної довжини. Однак це більше стосується внутрішньої роботи SQLite. І зовні, наприклад, на рівні визначення таблиці та роботи з даними ми працюватимемо з типом **INTEGER** , а не з усіма реальними типами, що ховаються за цією назвою. Тому фактично класи зберігання асоціюються з типом даних. І вище представлені п'ять класів зберігання також називають типами даних і дані можна застосовувати при визначенні стовпців:

```
CREATE TABLE users
(
    id INTEGER,
    name TEXT,
    age INTEGER,
    weight REAL,
    image BLOB
);
```

Крім того, ми можемо використовувати ідентифікатор **NUMERIC** . Цей ідентифікатор не надає окремого типу даних. А фактично представляє стовпець, який може зберігати дані всіх п'яти вище перерахованих типів (у термінології SQLite NUMERIC ще називається **type affinity**). Наприклад:

```
1   CREATE TABLE users
2   (
3       id INTEGER,
4       name TEXT,
5       age NUMERIC
6   );
```

Обмеження стовпців та таблиць

При визначенні стовпців та таблиць для них можна встановити обмеження. Обмеження дозволяють налаштувати поведінку стовпців та таблиць. Обмеження шпальт вказуються після типу шпальти:

```
1   column_name column_type column_constraints
```

Обмеження таблиці зазначаються після визначення всіх стовпців.

Розглянемо, які обмеження шпальт ми можемо використовувати.

PRIMARY KEY

Атрибут **PRIMARY KEY** визначає первинний ключ таблиці. Первинний ключ унікально ідентифікує рядок таблиці. Наприклад:

```
1   CREATE TABLE users
2   (
3       id INTEGER PRIMARY KEY,
4       name TEXT,
5       age INTEGER
6   );
```

Тут стовпець id виступає як первинний ключ, він буде унікально ідентифікувати рядок і його значення має бути унікальним. Тобто у нас не може бути таблиці users більше одного рядка, де в стовпці id було б те саме значення.

Установка первинного ключа лише на рівні таблиці:

```
1   CREATE TABLE users
2   (
```

```
3       id INTEGER,
4       name TEXT,
5       age INTEGER,
6       PRIMARY KEY(id)
7   );
```

Первинний ключ може бути складним. Такий ключ використовувати відразу кілька стовпців, щоб унікально ідентифікувати рядок таблиці. Наприклад:

```
1   CREATE TABLE users
2   (
3       id INTEGER,
4       name TEXT,
5       age INTEGER,
6       PRIMARY KEY(id, name)
7   );
```

В даному випадку як первинний ключ виступає зв'язка стовпців id і name. Тобто в таблиці users не може бути двох рядків, де для обох з цих полів одночасно були б ті самі значення.

AUTOINCREMENT

Обмеження **AUTOINCREMENT** дозволяє вказати, що значення стовпця автоматично збільшуватиметься при додаванні нового рядка. Дане обмеження працює для стовпців, які представляють тип **INTEGER** з обмеженням **PRIMARY KEY** :

```
1   DROP TABLE users;
2   CREATE TABLE users
3   (
4       id INTEGER PRIMARY KEY AUTOINCREMENT,
5       name TEXT,
6       age INTEGER
7   );
```

У цьому випадку значення стовпця id кожного нового доданого рядка збільшуватиметься на одиницю.

UNIQUE

Обмеження **UNIQUE** вказує, що стовпець може зберігати лише унікальні значення.

```
1   CREATE TABLE users
2   (
```

```
3       id INTEGER PRIMARY KEY AUTOINCREMENT,
4       name TEXT,
5       age INTEGER,
6       email TEXT UNIQUE
7   );
```

У цьому випадку стовпець email, який представляє телефон користувача, може зберігати лише унікальні значення. І ми не зможемо додати до таблиці два рядки, які мають значення для цього стовпця співпадати.

Також ми можемо визначити це обмеження на рівні таблиці:

```
1   CREATE TABLE users
2   (
3       id INTEGER PRIMARY KEY AUTOINCREMENT,
4       name TEXT,
5       age INTEGER,
6       email TEXT,
7       UNIQUE (name, email)
8   );
```

У цьому випадку унікальність значень встановлено відразу для двох стовпців – name та email.

NULL та NOT NULL

За замовчуванням будь-який стовпець, якщо він не представляє первинний ключ, може набувати значення **NULL** , тобто фактично відсутність формального значення. Але якщо ми хочемо заборонити подібну поведінку та встановити, що стовпець обов'язково повинен мати якесь значення, то для нього слід встановити обмеження **NOT NULL** :

```
1   CREATE TABLE users
2   (
3       id INTEGER PRIMARY KEY,
4       name TEXT NOT NULL,
5       age INTEGER
6   );
```

У цьому випадку стовпець name не допускає значення NULL.

DEFAULT

Обмеження **DEFAULT** визначає значення за промовчанням для стовпця. Якщо при додаванні даних для стовпця не буде передбачено значення, то для нього використовуватиметься значення за замовчуванням.

```
1   CREATE TABLE users
2   (
3       id INTEGER PRIMARY KEY,
4       name TEXT,
5       age INTEGER DEFAULT 18
6   );
```

Тут стовпець age як значення за замовчуванням має 18.

CHECK

Обмеження **CHECK** визначає обмеження для діапазону значень, які можуть зберігатися в стовпці. Для цього після CHECK вказується в дужках умова, якій повинен відповідати стовпець або кілька стовпців. Наприклад, вік користувачів не може бути менше 0 або більше 100:

```
1   CREATE TABLE users
2   (
3       id INTEGER PRIMARY KEY,
4       name TEXT NOT NULL CHECK(name !=''),
5       age INTEGER NOT NULL CHECK(age >0 AND age < 100)
6   );
```

Крім перевірки віку, тут також перевіряється, що стовпець name не може мати порожній рядок як значення (порожній рядок не еквівалентний NULL).

Для з'єднання умов використовується ключове слово **AND** . Умови можна задати у вигляді операцій порівняння більше (>), менше (<), не дорівнює (! =).

Також CHECK можна використовувати на рівні таблиці:

```
1   CREATE TABLE users
2   (
3       id INTEGER PRIMARY KEY,
4       name TEXT NOT NULL,
5       age INTEGER NOT NULL,
6       CHECK ((age >0 AND age < 100) AND (name !=''))
7   );
```

Оператор CONSTRAINT. Встановлення імені обмежень

За допомогою оператора **CONSTRAINT** можна встановити ім'я для обмежень. Вони вказуються після ключового слова CONSTRAINT перед обмеженнями на рівні таблиці:

```
1    CREATE TABLE users
2    (
```

```
3        id INTEGER,
4        name TEXT NOT NULL,
5        email TEXT NOT NULL,
6        age INTEGER NOT NULL,
7        CONSTRAINT users_pk PRIMARY KEY(id),
8        CONSTRAINT user_email_uq UNIQUE(email),
9        CONSTRAINT user_age_chk CHECK(age >0 AND age < 100)
10   );
```

У разі обмеження для PRIMARY KEY називається users_pk, для UNIQUE - user_phone_uq, а CHECK - user_age_chk. Сенс встановлення імен обмежень полягає в тому, що згодом через ці імена ми зможемо керувати обмеженнями – видаляти чи змінювати їх.

Зовнішні ключі FOREIGN KEY

Зовнішні ключі дозволяють встановити зв'язок між таблицями. Зовнішній ключ встановлюється для стовпців із залежної, підлеглої таблиці, і вказує на один із стовпців із головної таблиці. Як правило, зовнішній ключ вказує на первинний ключ із пов'язаної головної таблиці.

Загальний синтаксис встановлення зовнішнього ключа на рівні таблиці:

```
1   [CONSTRAINT constraint_name]
2   FOREIGN KEY (column1, column2, ... columnN)
3   REFERENCES parent_table (parent_table_column1, parent_table_column2, ...
    parent_table_columnN)
4   [ON DELETE action]
5   [ON UPDATE action]
```

Для створення обмеження зовнішнього ключа після **FOREIGN KEY** вказується стовпець таблиці, який представляє зовнішній ключ. А після ключового слова **REFERENCES** вказується ім'я пов'язаної таблиці, а потім у дужках ім'я зв'язаного стовпця, на який вказуватиме зовнішній ключ. Після виразу **REFERENCES** йдуть вирази **ON DELETE** та **ON UPDATE**, які задають дію при видаленні та оновленні рядка з головної таблиці відповідно.

Наприклад, визначимо дві таблиці та зв'яжемо їх за допомогою зовнішнього ключа:

```
1    CREATE TABLE companies
```

```
2    (
3        id INTEGER PRIMARY KEY AUTOINCREMENT,
4        name TEXT NOT NULL
5    );
6    CREATE TABLE users
7    (
8        id INTEGER PRIMARY KEY AUTOINCREMENT,
9        name TEXT NOT NULL,
10       age INTEGER,
11       company_id INTEGER,
12       FOREIGN KEY (company_id)  REFERENCES companies (id)
13   );
```

У даному випадку визначено таблиці компаній і користувачів. companies є головною та представляє компанії, де може працювати користувач. users є залежною та представляє користувачів. Таблиця users через стовпець company_id пов'язана з таблицею компаній та її стовпцем id. Тобто стовпець company_id є **зовнішнім ключем** , який вказує стовпець id з таблиці companies.

За допомогою оператора **CONSTRAINT** можна встановити ім'я для обмеження зовнішнього ключа:

```
1   CREATE TABLE users
2   (
3       id INTEGER PRIMARY KEY AUTOINCREMENT,
4       name TEXT NOT NULL,
5       age INTEGER,
6       company_id INTEGER,
7       CONSTRAINT users_companies_fk
8       FOREIGN KEY (company_id)  REFERENCES companies (id)
9   );
```

ON DELETE та ON UPDATE

За допомогою виразів **ON DELETE** та **ON UPDATE** можна встановити дії, які виконуються відповідно при видаленні та зміні зв'язаного рядка з головної таблиці. Як дія можуть використовуватися такі опції:

- **CASCADE** : автоматично видаляє або змінює рядки із залежної таблиці під час видалення або зміни пов'язаних рядків у головній таблиці.

- **SET NULL : при видаленні або оновленні зв'язаного рядка з головної таблиці встановлює значення NULL** для стовпчика зовнішнього ключа . (У цьому випадку стовпець зовнішнього ключа має підтримувати встановлення NULL)
- **RESTRICT** : відхиляє видалення чи зміну рядків у головній таблиці за наявності пов'язаних рядків у залежній таблиці.
- **NO ACTION** : те саме, що і RESTRICT.
- **SET DEFAULT** : при видаленні зв'язаного рядка з головної таблиці встановлює для стовпчика зовнішнього ключа значення за промовчанням, яке задається за допомогою атрибута DEFAULT.

Каскадне видалення

Каскадне видалення дозволяє при видаленні рядка з головної таблиці автоматично видалити всі зв'язані рядки із залежної таблиці. Для цього застосовується опція **CASCADE** :

```
1   CREATE TABLE users
2   (
3       id INTEGER PRIMARY KEY AUTOINCREMENT,
4       name TEXT NOT NULL,
5       age INTEGER,
6       company_id INTEGER,
7       FOREIGN KEY (company_id)  REFERENCES companies (id) ON DELETE CASCADE
8   );
```

Подібним чином працює і вираз **ON UPDATE CASCADE** . При зміні первинного ключа автоматично зміниться значення пов'язаного з ним зовнішнього ключа. Однак оскільки первинні ключі змінюються дуже рідко, та й з принципу не рекомендується використовувати як первинні ключі стовпці зі змінними значеннями, то на практиці вираз ON UPDATE використовується рідко.

Встановлення NULL

У разі встановлення для зовнішнього ключа опції **SET NULL** необхідно, щоб стовпець зовнішнього ключа допускав значення NULL:

```
1   CREATE TABLE users
2   (
```

```
3       id INTEGER PRIMARY KEY AUTOINCREMENT,
4       name TEXT NOT NULL,
5       age INTEGER,
6       company_id INTEGER,
7       FOREIGN KEY (company_id)  REFERENCES companies (id) ON DELETE SET NULL
8   );
```

Зміна таблиць та стовпців

Якщо таблиця вже була створена, і її необхідно змінити, для цього застосовується команда **ALTER TABLE** . Вона підтримує різні опції та можливості. Розглянемо лише основні сценарії, з якими ми можемо зіткнутися.

Перейменування таблиці

Для перейменування таблиці застосовується оператор **RENAME TO** , після якого вказується нове ім'я таблиці:

```
1   ALTER TABLE users
2   RENAME TO people;
```

Тут таблиця користувачів перейменовується в "люди".

Додавання нового стовпця

Додамо до таблиці users новий стовпець email:

```
1   ALTER TABLE users
2   ADD COLUMN email TEXT NOT NULL;
```

В даному випадку стовпець email має тип TEXT і для нього визначено обмеження NOT NULL.

Перейменування стовпця

Перейменуємо стовпець email у login

```
1   ALTER TABLE users
2   RENAME COLUMN email TO login;
```

Видалення стовпця

Видалимо стовпець login з таблиці users:

```
1   ALTER TABLE users
2   DROP COLUMN login;
```