**Модуль 6: Реляційні бази даних**

**Заняття 1: Основи SQL**

**Основні терміни**

*База даних* (БД) — це набір з однієї та більше таблиць з даними. Кожна таблиця містить інформацію в 1 та більше стовпцях (полях). Один елемент даних (запис) — це один рядок у таблиці.

*Ключ* — унікальне поле, яке однозначно ідентифікує запис.

*Primary key* — унікальний ключ, який не повторюється у таблиці.

*Foreign key* — посилання на унікальний ключ, який НЕ ПОВТОРЮЄТЬСЯ у своїй таблиці.

**Взаємовідносини:**

* *один до одного* — таблиці пов'язані один до одного тоді, коли одному рядку (запису) таблиці A відповідає один рядок таблиці B, і одному запису таблиці B відповідає один запис таблиці A;
* *один до багатьох* — цей зв'язок у реляційних базах даних реалізується тоді, коли одному рядку таблиці A може належати або відповідати кілька записів у таблиці B, але записи з таблиці B можуть відповідати тільки одному запису таблиці А. Приклад: користувач може мати кілька телефонних номерів;
* *багато до багатьох* — реалізується у тому випадку, коли кільком записам з таблиці A може відповідати кілька записів з таблиці B, і водночас кільком записам таблиці B відповідає кілька записів з таблиці A;
* *багато до одного* — зворотна ситуація один до багатьох, тільки тепер таблиці A і B потрібно поміняти місцями;

**Нормалізація**— процес приведення структури БД до виду, що забезпечує мінімальну логічну надмірність, і не має на меті зменшення або збільшення продуктивності роботи, або зменшення або збільшення фізичного обсягу бази даних. Кінцевою метою нормалізації є зменшення потенційної суперечливості інформації, що зберігається в базі даних.

Основний інструмент спілкування з реляційними БД — SQL (Structured Query Language).

**SQL** — мова структурованих запитів.

**Основні оператори SQL**[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/intro#%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%96-%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%B8-sql)

1. Оператори визначення даних (Data Definition Language, DDL):

* CREATE створює об'єкт БД (саму базу, таблицю, представлення, користувача тощо),
* ALTER змінює об'єкт,
* DROP видаляє об'єкт;

2.Оператори маніпуляції даними (Data Manipulation Language, DML):

* SELECT вибирає дані, що задовольняють заданим умовам,
* INSERT додає нові дані,
* UPDATE змінює існуючі дані,
* DELETE видаляє дані;

3.Оператори визначення доступу до даних (Data Control Language, DCL):

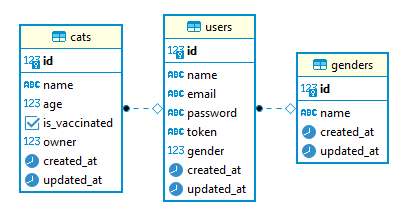
* GRANT надає користувачеві (групі) дозволи на певні операції з об'єктом,
* REVOKE відкликає раніше видані дозволи,

4.Оператори управління транзакціями (Transaction Control Language, TCL):

* COMMIT застосовує транзакцію,
* ROLLBACK відкочує всі зміни, зроблені в контексті поточної транзакції,
* SAVEPOINT ділить транзакцію на дрібніші ділянки.

**ER (Entity–relationship) - діаграми**[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/intro#er-entityrelationship---%D0%B4%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B8)

ER-модель — модель даних, що дозволяє описувати концептуальні схеми предметної галузі. ER-модель використовується при високорівневому (концептуальному) проектуванні бази даних. З її допомогою можна виділити ключові сутності та позначити зв'язки, які можуть встановлюватися між цими сутностями.



**Підключення до БД**

Програм для робіт з SQL базами достатньо багато, але рекомендувати у процесі навчання ми будемо дві: [HeidiSQL](https://www.heidisql.com/) (https://www.edu.goit.global/uk/learn/15249151/10926565/10926607/training)  та [DBeaver](https://dbeaver.io/) (https://dbeaver.io/).

В якості пояснень щодо підключення ми будемо використовувати **DBeaver**, який працює з більшістю відомих баз даних.

Завантажуємо з сайту розробників <https://dbeaver.io/download/>

**Підключення до PostgreSQL**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/connect#%D0%BF%D1%96%D0%B4%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B4%D0%BE-postgresql)

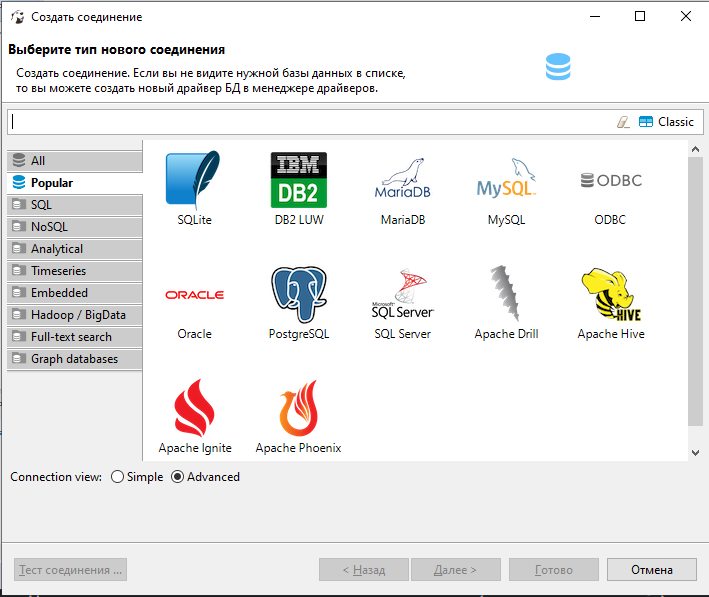
Для запуску бази даних [postgres](https://hub.docker.com/_/postgres) (https://hub.docker.com/\_/postgres) ми використовуємо Docker. У командному рядку необхідно виконати таку команду:

docker run --name some-postgres -p 5432:5432 –e

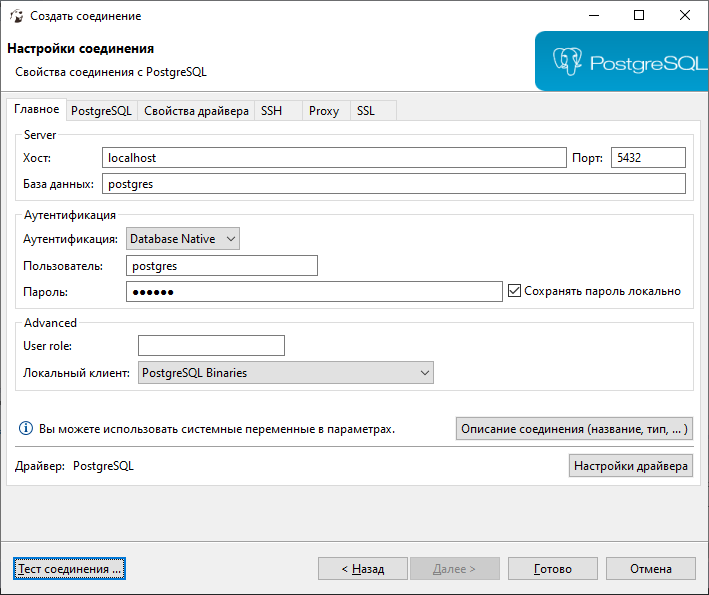
POSTGRES\_PASSWORD=mysecretpassword -d postgres

Замість some-postgres виберіть свою назву контейнера, а замість mysecretpassword придумайте свій пароль для підключення до бази даних

Тепер нам потрібно підключитись до нашої бази.

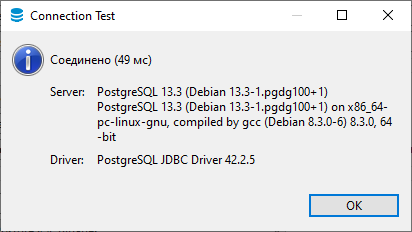


Початкові налаштування вже повинні бути правильними, в полі password впишіть пароль, який ви встановили.

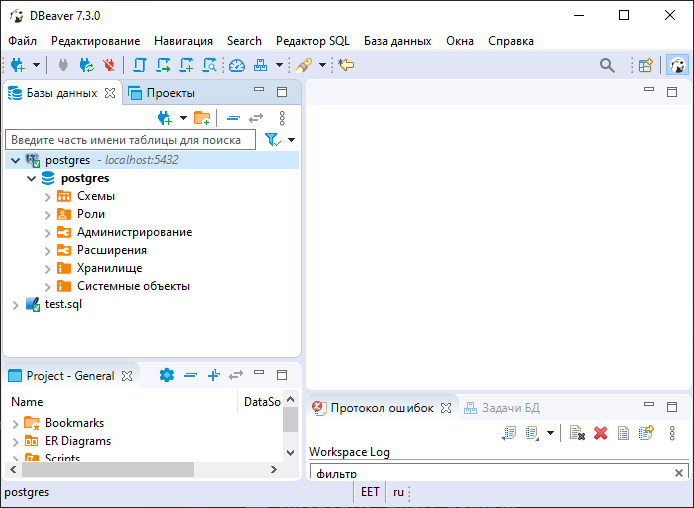


Натискаємо

Тест з'єднання..., повинно бути наступне повідомлення.



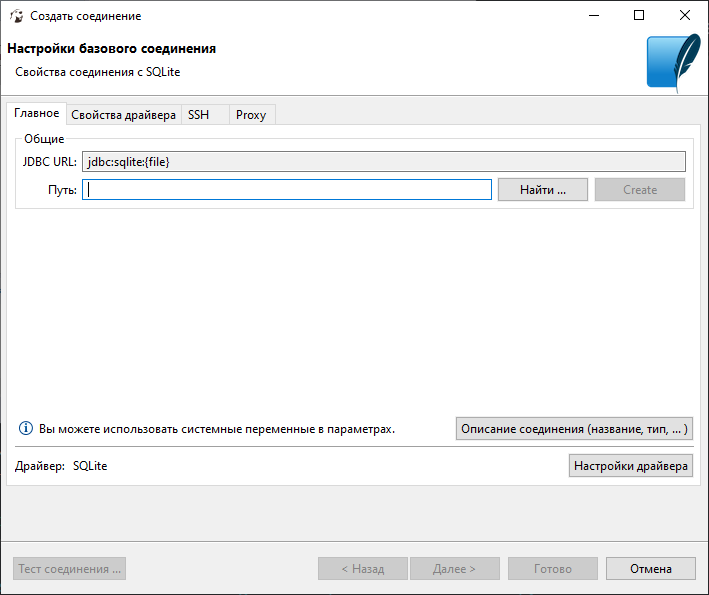
Якщо все добре, у вас у списку з'явиться сервер, відкрийте його, у вас буде як на скріні нижче



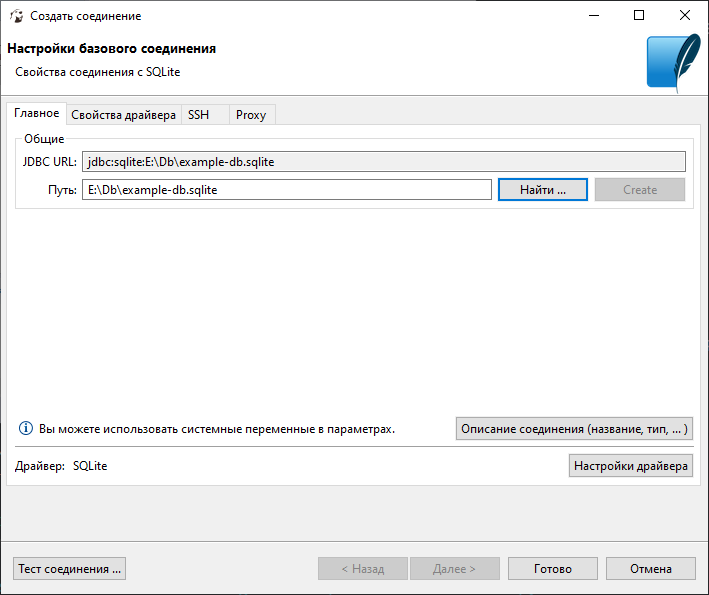
І вже є готова база даних з ім'ям: postgres

**Підключення до SQLite**[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/connect#%D0%BF%D1%96%D0%B4%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B4%D0%BE-sqlite)

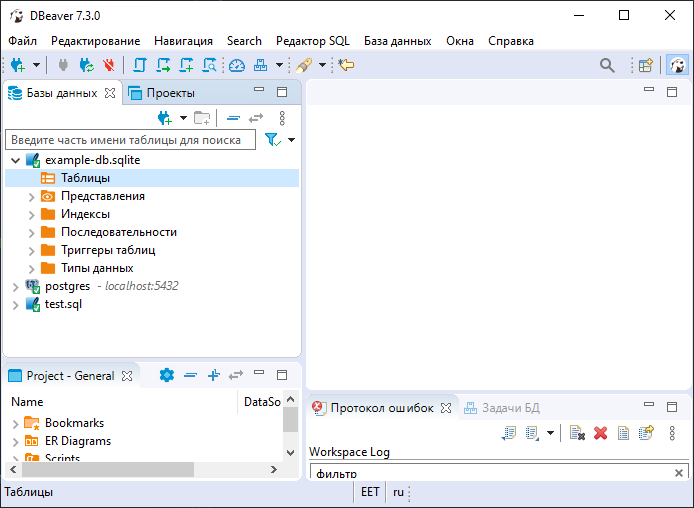
Підключення до SQLite бази даних відбувається набагато простіше. По суті, це локальний файл, який необхідно створити.



Пропишіть шлях та ім'я файлу, де ви хочете, щоб зберігався наш файл з базою даних



Якщо все добре, у вас у списку з'явиться база даних SQLite, відкрийте її, у вас буде як на скріні нижче



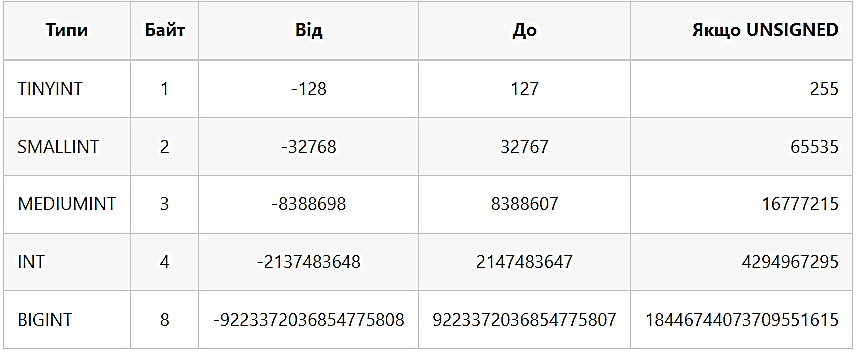
**Типи даних**

**Числові типи даних**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/type#%D1%87%D0%B8%D1%81%D0%BB%D0%BE%D0%B2%D1%96-%D1%82%D0%B8%D0%BF%D0%B8-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)Числа діляться на цілі та дійсні (число з крапкою)

**Цілі числа**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/type#%D1%86%D1%96%D0%BB%D1%96-%D1%87%D0%B8%D1%81%D0%BB%D0%B0)



Також цілі числа поділяють на типи:

* SIGNED – знакові (один біт йде на знак плюс або мінус)
* UNSIGNED – беззнакові, якщо значення лише додатні, наприклад, відстань, площа тощо. Це збільшить додатне значення для типу вдвічі, тобто TINYINT UNSIGNED буде від 0 до 255

**Дробові числа**

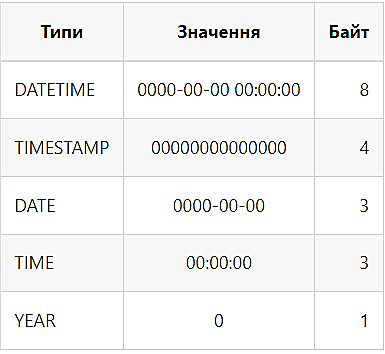
[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/type#%D0%B4%D1%80%D0%BE%D0%B1%D0%BE%D0%B2%D1%96-%D1%87%D0%B8%D1%81%D0%BB%D0%B0)Усі дійсні числа в SQL суворо обумовленої точності

Псевдоніми типу:

* REAL(10,2)
* DECIMAL(10,2)
* FIXED(10,2)
* FLOAT(10,2)

Ці всі 4 функції — псевдоніми, і принципової різниці в SQL між ними немає. Щодо реалізацій краще уточнити в документації.

**Типи даних: дата та час**[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/type#%D1%82%D0%B8%D0%BF%D0%B8-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85-%D0%B4%D0%B0%D1%82%D0%B0-%D1%82%D0%B0-%D1%87%D0%B0%D1%81)



Детальніше:

* **DATETIME** – для повноцінної дати та часу, великої за обсягом, приблизно 4-8 байт. Записується у вигляді рядка, і завжди записується від великого до меншого: рік, місяць, день, година, хвилина, секунда. Підтримуваний діапазон від ‘1000-01-01 00:00:00’ до ‘9999-12-31 23:59:59’. Якщо потрібно нижче, просто пишемо з мінусом ‘-500-01-09 00:00:00’.
* **TIMESTAMP** – часова мітка, діапазон епохи Юнікса 1970-01-01 00:00:00 — 2038-12-31 00:00:00
* **DATE** – це тільки дата без часу
* **TIME** – тільки час
* **YEAR** – тільки рік

**Символьні типи даних**

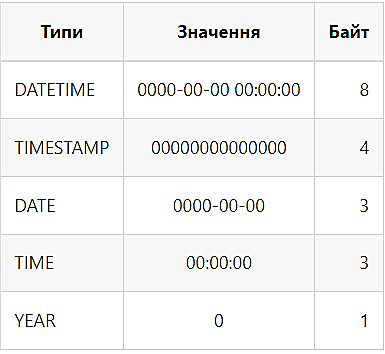
[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/type#%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D1%8C%D0%BD%D1%96-%D1%82%D0%B8%D0%BF%D0%B8-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)**Типи CHAR та VARCHAR**[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/type#%D1%82%D0%B8%D0%BF%D0%B8-char-%D1%82%D0%B0-varchar)

Ці типи даних зберігають рядок певної довжини.

**CHAR** – зберігає символи завжди однакової довжини, так для CHAR(10) завжди буде записано 10 символів (не байтів). Якщо задана межа типу 10, то завжди записано 10. Якщо у рядку не вистачає символів - буде доповнено пробілами, якщо символів багато – вони будуть обрізані.

**VARCHAR** – це змінний CHAR, у нього є додатковий байт, що символізує кінець, так би мовити маркер кінця. Таким чином, символи вводитимуться рівно стільки, скільки ми ввели плюс 1 байт. Тобто при CHAR(4), порожній рядок важить 4 байти, а при VARCHAR(4) важить 1 байт, вага тільки маркера. При переповненні рядок також обрізається.

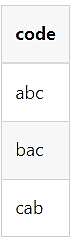
Порівняльна таблиця



Причина тоді використовувати CHAR, якщо VARCHAR явно зручніше. Але проблема не в економії місця, 1 байт на мільйонних таблицях дасть багато місця.

Головне, що пошук за CHAR працює набагато швидше. Якщо ми маємо явне значення, яке більше бути не може, як номер телефону, паспорта тощо, немає сенсу робити це типом VARCHAR, ви сповільните базу та втратите по байту на запис.

Припустимо у нас рядки CHAR(3) запишемо їх до бази.



Але в базі воно зберігатиметься приблизно так: **abc bac cab**

І якщо ми хочемо отримати третій запис, то база точно знає, що кількість символів у базі по 3, і вона просто почне читання з 7-го символу. У випадку, якби це був VARCHAR(3), їй у цій же структурі довелося б переглянути всі записи для пошуку їх маркерів кінця запису, вона вже не може перегорнути 6 символів, тому що рядки можуть бути не повними та різної довжини.

**Типи даних TEXT**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/type#%D1%82%D0%B8%D0%BF%D0%B8-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85-text)Це типи для зберігання великого тексту, крім TINYTEXT — це синонім VARCHAR. Якщо нам потрібно зберігати опис, відгуки або навіть книги, то це той тип, який потрібний.

* **TEXT**- використовується найчастіше з цих типів.

**CREATE TABLE: Створюємо таблицю**

Для створення таблиць в SQL використовується вираз CREATE TABLE. Він приймає як параметри імена стовпців, які ми хочемо внести, а також їх типи даних.

Створимо три таблиці з назвою "genders", "users" та "contacts",

В таблиці "genders" буде три стовпці:

* **id**— порядковий номер гендера, тип INT, це унікальний ключ
* **name** — назва гендера, тип VARCHAR(30)
* **created\_at** — час створення запису, тип TIMESTAMP, значення за замовчуванням — поточний час та дата

Код для створення:

CREATE TABLE genders (

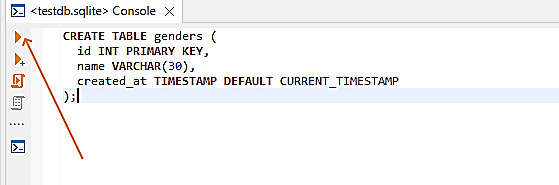
id INT PRIMARY KEY,

name VARCHAR(30),

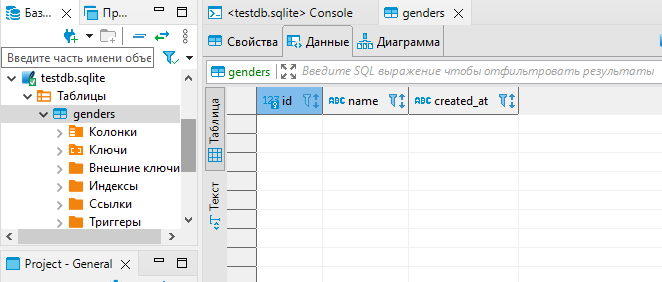
created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

Щоб виконати код для нашої БД, клацніть правою кнопкою миші по назві БД зліва в меню DBeaver та виберіть 'Редактор SQL' > 'Open SQL Console'. Вставте скрипт і натисніть виконати команду



Після виконання у вас з'явиться потрібна таблиця БД. Щоб її побачити, можливо потрібно буде виконати команду оновлення для БД (F5).



Ці операції потрібно проводити для всіх команд, які розглядаються далі.

В таблиці "users" у нас буде сім стовпців:

* **id** — порядковий номер контакту, типINT, це унікальний ключ
* **name** — ім'я користувача, тип VARCHAR(30)
* **email** — електронна адреса користувача, тип VARCHAR(30)
* **password** — пароль користувача, тип VARCHAR(30)
* **age** — вік користувача, тип TINYINT UNSIGNED
* **gender\_id** — це foreign key, який пов'язує таблицю "users" та "genders" зв'язком один до багатьох. Один гендер може бути у багатьох користувачів.
* **created\_at** — час створення запису, тип TIMESTAMP, значення за замовчуванням — поточний час та дата

Код для створення:

CREATE TABLE users (

id INT PRIMARY KEY,

name VARCHAR(30),

email VARCHAR(30),

password VARCHAR(30),

age TINYINT UNSIGNED,

gender\_id INT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (gender\_id) REFERENCES genders (id)

ON DELETE SET NULL

ON UPDATE CASCADE

);

Тут у нас з'явилася інструкція:

FOREIGN KEY (gender\_id) REFERENCES genders (id)

ON DELETE SET NULL

ON UPDATE CASCADE

Вона каже, що для стовпця gender\_id в таблиці users необхідно створити референ посилання на таблицю genders, причому значення стовпця співпадатиме зі значенням стовпця id в таблиці genders для конкретного запису. Запис ON DELETE SET NULL каже, що якщо буде видалено запис у таблиці genders, ми повинні значення для стовпця gender в таблиці users встановити в значення NULL. Запис ON UPDATE CASCADE каже, що якщо буде змінено значення поля id в таблиці genders, то значення для стовпця gender\_id в таблиці users також автоматично буде змінено.

В таблиці "contacts" буде сім стовпців:

* **id** — порядковий номер контакту, тип INT, це унікальний ключ
* **name** — назва контакту, тип VARCHAR(30)
* **email** — електронна адреса контакту, тип VARCHAR(30)
* **phone**— телефон контакту, тип VARCHAR(30)
* **favorite** — контакт знаходиться в обраному або ні, логічного типу
* **user\_id**— це foreign key, який пов'язує таблицю "contacts" та "users" зв'язком один до багатьох. Один користувач може мати багато контактів.
* **created\_at** — час створення запису, тип TIMESTAMP, значення за замовчуванням — поточний час та дата

Код виглядатиме ось так:

CREATE TABLE contacts (

id INT PRIMARY KEY,

name VARCHAR(30),

email VARCHAR(30),

phone VARCHAR(30),

favorite BOOLEAN,

user\_id INT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

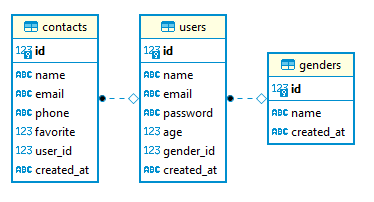
FOREIGN KEY (user\_id) REFERENCES users (id)

ON DELETE CASCADE

ON UPDATE CASCADE

);

Після виконання цих команд наша база даних матиме наступну ER-діаграму



**INSERT: Введення даних**

Тепер давайте заповнимо наші таблиці. Зробити це можна за допомогою команди INSERT. Формат команди такий, що перед введенням даних ми вказуємо назви колонок. У випадку, якщо ми не вкажемо один із стовпців, на його місце буде записано NULL або задане значення за замовчуванням.

Вставимо значення в таблицю genders:

INSERT INTO genders (id, name)

VALUES (1, 'male'), (2, 'female');

Ми не вказали під час вставлення значення для поля created\_at, але, завдяки інструкції DEFAULT CURRENT\_TIMESTAMP, значення буде підставлено автоматично.

Вставимо значення в таблицю users:

INSERT INTO users (id, name, email, password, age, gender\_id)

VALUES (1, 'Boris', 'boris@test.com', 'password', 23, 1),

(2, 'Alina', 'alina@test.com', 'password', 32, 2),

(3, 'Maksim', 'maksim@test.com', 'password', 40, 1);

Вставимо значення в таблицю contacts:

INSERT INTO contacts (id, name, email, phone, favorite, user\_id)

VALUES (1, 'Allen Raymond', 'nulla.ante@vestibul.co.uk', '(992) 914-3792', 0, 1),

(2, 'Chaim Lewis', 'dui.in@egetlacus.ca', '(294) 840-6685', 1, 1),

(3, 'Kennedy Lane', 'mattis.Cras@nonenimMauris.net', '(542) 451-7038', 1, 2),

(4, 'Wylie Pope', 'est@utquamvel.net', '(692) 802-2949', 0, 2),

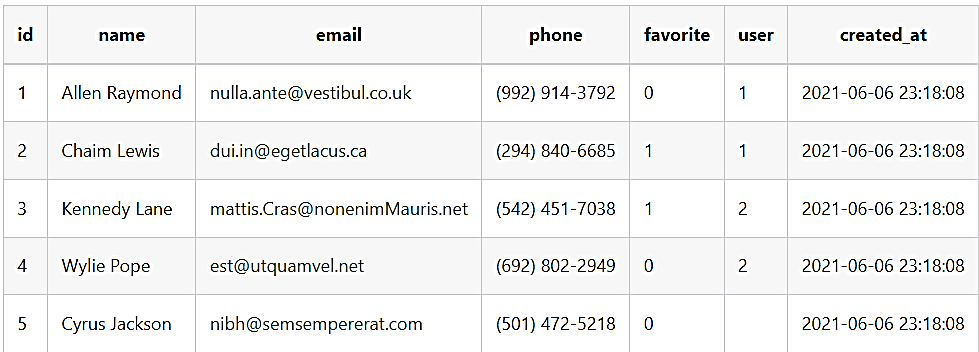
(5, 'Cyrus Jackson', 'nibh@semsempererat.com', '(501) 472-5218', 0, null);

**SELECT: Отримання даних**

Цей запит використовується у випадку, якщо нам потрібно показати дані у таблиці. Напевно, найпростішим прикладом використання SELECT буде наступний запит:

SELECT \* FROM contacts

Результатом цього запиту буде таблиця з усіма даними в таблиці contacts.



Знак зірочки \* означає те, що ми хочемо показати всі стовпці з таблиці без винятків. Оскільки у базі даних зазвичай більше однієї таблиці, нам необхідно вказувати назву таблиці, дані з якої хочемо подивитися. Це робиться, використовуючи ключове слово FROM.

Коли вам потрібні лише деякі стовпці з таблиці, то ви можете вказати їхні імена через кому, замість зірочки.

SELECT name, email FROM contacts ORDER BY name

Також іноді нам потрібно відсортувати дані, що виводяться. Для цього ми використовуємо ORDER BY "назва стовпця". ORDER BY має два модифікатори: ASC — сортувати за зростанням, значення за замовчуванням та DESC — сортувати за спаданням



Якщо необхідно включити до виведення лише деякі конкретні записи за умовою, для цього використовується ключове слово WHERE. Воно дозволяє фільтрувати дані за певною умовою.

У наступному запиті ми виведемо лише вибрані контакти.

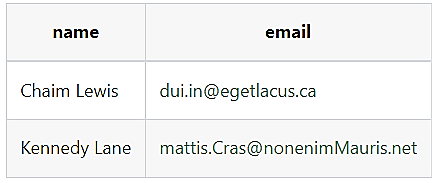
SELECT name, email

FROM contacts

WHERE favorite = true

ORDER BY name

Виведення:



Умови в WHERE можуть бути написані з використанням логічних операторів AND та OR, а також математичних операторів порівняння (=, <, >, <=, >=, <>).

Умови в WHERE можуть бути записані з використанням ще кількох команд, якими є:

* IN — порівнює значення в стовпці з кількома можливими значеннями та повертає true, якщо значення збігається хоча б із одним значенням

SELECT name, email

FROM users

WHERE age IN(20, 30, 40)

ORDER BY name

Виведення:



* BETWEEN — перевіряє, чи є значення в якомусь проміжку

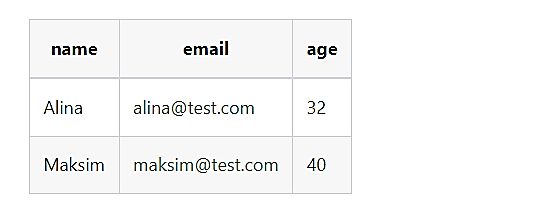
SELECT name, email, age

FROM users

WHERE age BETWEEN 30 AND 40

ORDER BY name

Виведення:



* LIKE — шукає за шаблоном

Також, якщо ми хочемо вивести всі контакти, в імені яких є літера 'L', ми можемо використовувати наступний запис:

SELECT name, email

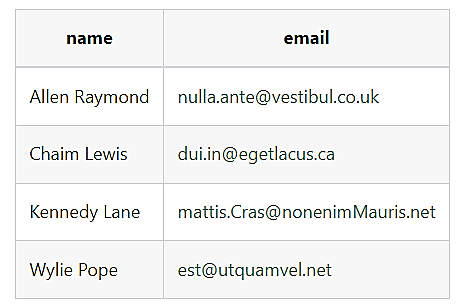
FROM contacts

WHERE name LIKE '%L%'

ORDER BY name

Знак % означає будь-яку послідовність символів (0 символів також вважається послідовністю).

Виведення:



SQL також має інверсію. Для цього потрібно написати NOT перед будь-яким логічним виразом в умові (NOT BETWEEN тощо).

SELECT name, email, age

FROM users

WHERE age NOT BETWEEN 30 AND 40

ORDER BY name

Виведення:



**Функції агрегації**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/select#%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%97-%D0%B0%D0%B3%D1%80%D0%B5%D0%B3%D0%B0%D1%86%D1%96%D1%97)SQL має безліч вбудованих функцій для виконання різних операцій. Ми розглянемо ті, що найчастіше використовуються:

* COUNT() — повертає кількість рядків
* SUM() — повертає суму всіх полів з числовими значеннями в них
* AVG() — повертає середнє значення серед рядків
* MIN()/MAX() — повертає мінімальне/максимальне значення серед рядків

Знайти мінімальний вік серед користувачів

SELECT min(age) as minAge

FROM users

Виведення:

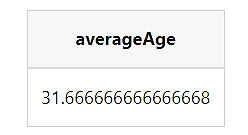


Знайти середній вік користувачів:

SELECT avg(age) as averageAge

FROM users

Виведення:



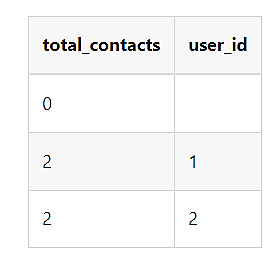
Давайте знайдемо кількість контактів кожного користувача за допомогою функції COUNT. Але в операторі SELECT нам потрібно додати рядок GROUP BY user\_id. Оскільки функція є агрегованою за полем user\_id, і необхідно групувати значення за ім'ям.

SELECT COUNT(user\_id) as total\_contacts, user\_id

FROM contacts

GROUP BY user\_id

Виведення:



Як і очікувалося, у кожного користувача ми додали по два контакти. І є контакт без користувача.

**Вкладені SELECT**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/select#%D0%B2%D0%BA%D0%BB%D0%B0%D0%B4%D0%B5%D0%BD%D1%96-select)Поки що ми розглядали лише прості запити. Але часто необхідно використовувати так звані вкладені запити або підзапити.

Нехай необхідно вивести контакти для користувачів віком менше 30 років.

Перший запит — знайти id користувачів молодше 30 років

SELECT id

FROM users

WHERE age < 30

Потім ми об'єднуємо запити за допомогою WHERE:

SELECT \*

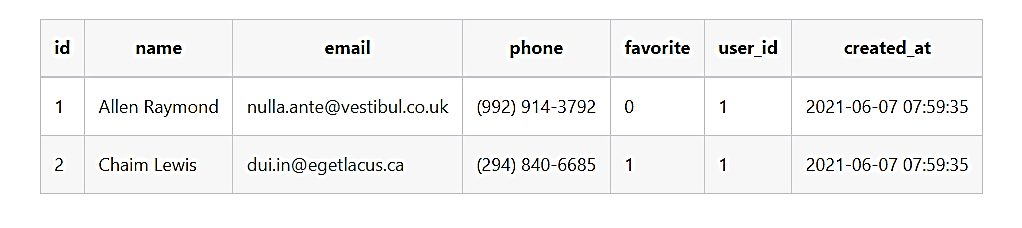
FROM contacts

WHERE user\_id IN (SELECT id

FROM users

WHERE age < 30)

Виведення:



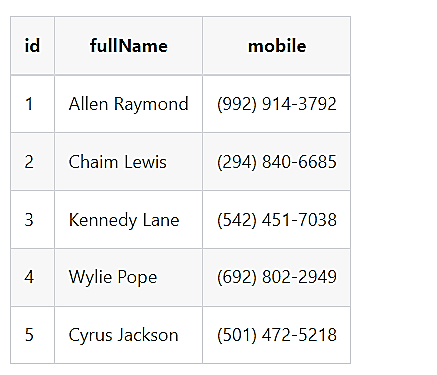
**Псевдоніми**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/select#%D0%BF%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D0%BD%D1%96%D0%BC%D0%B8)У попередніх прикладах ми вже використали псевдоніми. Щоб присвоїти стовпцю псевдонім, можна використовувати ключове слово AS:

SELECT id, name as fullName, phone as mobile

FROM contacts

Виведення:



Псевдоніми часто використовують, коли починають працювати з пов'язаними таблицями.

**JOIN: Приєднання таблиць**

В базах даних найчастіше таблиці пов'язані між собою. Наприклад, у нас таблиці users та genders пов'язані між собою за полем gender\_id, а таблиці contacts та users — за полем user\_id.

Для приєднання таблиць використовують оператор JOIN. Розглянемо конкретні приклади.

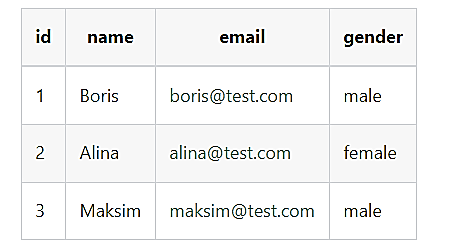
Як ми вже говорили, в таблиці users є стовпець gender\_id, у цьому випадку він є так званим foreign key і сполучною ланкою між двома таблицями. Якщо ми хочемо вивести всю інформацію про користувача, включаючи інформацію про його гендер, нам необхідно підключити другу таблицю genders. Щоб це зробити, можна використовувати INNER JOIN, де умова об'єднання задається за допомогою ON:

SELECT u.id, u.name, u.email, g.name AS gender

FROM users AS u

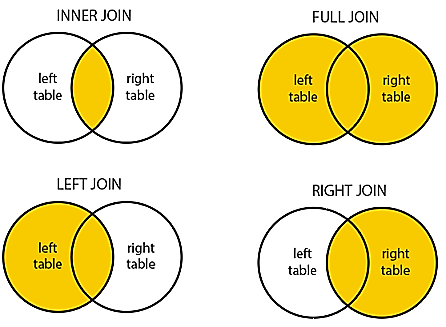
INNER JOIN genders AS g ON g.id = u.gender\_id

Виведення:



Це простий приклад використання JOIN. Є ще кілька варіантів його використання:

* (INNER) JOIN: Повертає записи, значення яких збігаються в обох таблицях.
* LEFT (OUTER) JOIN: Повертає всі записи з лівої таблиці та відповідні записи із правої таблиці.
* RIGHT (OUTER) JOIN: Повертає всі записи з правої таблиці та відповідні записи із лівої таблиці.
* FULL (OUTER) JOIN: Повертає всі записи, якщо є збіги у лівій або правій таблиці



У круглих дужках слово можна за замовчуванням не писати, тобто записи INNER JOIN та JOIN - еквівалентні.

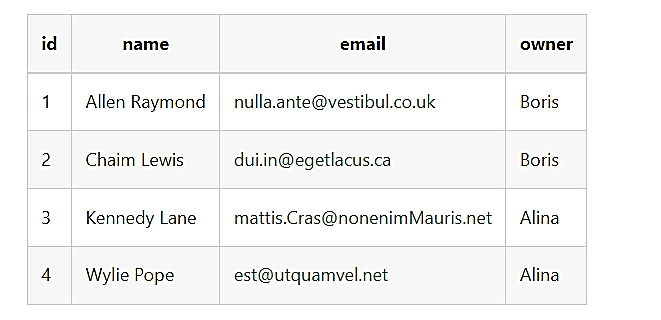
Розглянемо конкретний приклад, щоб зрозуміти різницю між INNER JOIN та LEFT JOIN.

SELECT c.id, c.name, c.email, u.name AS owner

FROM contacts AS c

JOIN users AS u ON u.id = c.user\_id

Виведення:



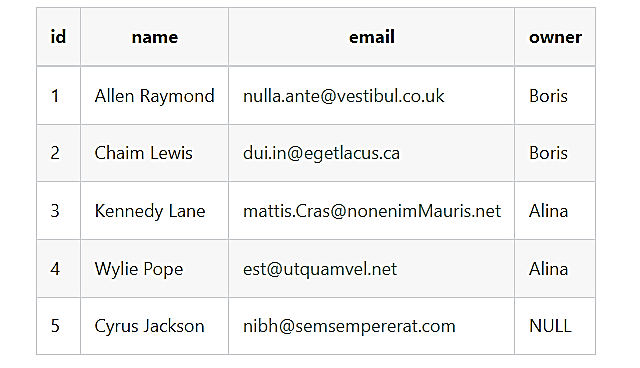
У цьому випадку у вибірці ми не отримуємо контакту без користувача, оскільки використовувався перетин таблиць. Щоб отримати всі контакти, навіть якщо вони не мають власників з таблиці users, нам потрібно використовувати LEFT JOIN.

SELECT c.id, c.name, c.email, u.name AS owner

FROM contacts AS c

LEFT JOIN users AS u ON u.id = c.user\_id

Виведення:



Цей запит приєднав дані таблиці contacts, яким не знайшлося відповідності у таблиці users.

Для більшого розуміння побудови запитів рекомендуємо прочитати статтю [*SQL: мені тільки спитати. Як писати скрипти для отримання вибірок*](https://dou.ua/forums/topic/34981/)(<https://dou.ua/forums/topic/34981/> Прим. стаття RU)

**Зміна та видалення даних у таблиці**

**UPDATE: Зміна даних**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/update#update-%D0%B7%D0%BC%D1%96%D0%BD%D0%B0-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)Зміна даних у таблиці SQL робиться за допомогою команди UPDATE.

Використання UPDATE включає в себе: перше — вибір таблиці, в якій знаходиться поле, яке ми хочемо змінити, друге — встановлення в запис нового значення за допомогою оператора SET і третє — це використання WHERE, щоб позначити конкретне місце у таблиці.

В таблиці contacts у нас є запис з id = 5, у якого немає значення поля user\_id.

Давайте встановимо, що власником цього контакту буде користувач Maksim з id = 3 в таблиці users.

UPDATE contacts SET user\_id = 3 WHERE id = 5;

**DELETE: Видалення записів з таблиці**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/update#delete-%D0%B2%D0%B8%D0%B4%D0%B0%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%96%D0%B2-%D0%B7-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%96)Видалення запису з таблиці через SQL — теж проста операція. Головне позначити за допомогою WHERE, що саме ми хочемо видалити. Інакше ми видалимо всі записи з таблиці, чого б хотілося уникнути.

DELETE FROM contacts WHERE id = 4;

**Видалення таблиць**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/update#%D0%B2%D0%B8%D0%B4%D0%B0%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%8C)Якщо ми хочемо видалити всі дані з таблиці, але при цьому саму таблицю залишити, варто використовувати команду TRUNCATE:

TRUNCATE TABLE contacts;

У випадку, якщо ми хочемо видалити власне саму таблицю, то нам слід використати команду DROP:

DROP TABLE contacts;

**Модифікація таблиць**

**Оператор ALTER**[**​**](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/alter#%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80-alter)

Найчастіше потрібно змінити початкову схему таблиці. Ви створили таблицю, вона живе своїм життям, але потім потрібно або змінити тип колонки, або додати нову колонку, видалити колонку тощо. Тут нам потрібний оператор ALTER. В наступному розділі ми познайомимося з інструментом alembic, який автоматизує ці операції за допомогою міграцій. Але поки що давайте розглянемо роботу самого оператора:

Розберемо основні типи запитів.

Перейменувати колонку column\_a у колонку column\_b і задати їй тип INTEGER:

ALTER TABLE table\_name CHANGE column\_a column\_b INTEGER;

Змінити тільки тип колонки column\_b в таблиці table\_name:

ALTER TABLE table\_name MODIFY column\_b BIGINT NOT NULL;

Додати колонку column\_c з типом FLOAT в таблицю table\_name:

ALTER TABLE table\_name ADD column\_c FLOAT;

Додати колонку column\_d з типом VARCHAR після column\_c в таблицю table\_name:

ALTER TABLE table\_name ADD column\_d VARCHAR(30) AFTER column\_c;

Додати колонку column\_f з типом CHAR і зробити її першою:

ALTER TABLE table\_name ADD column\_f CHAR(10) FIRST;

Видалити колонку column\_c в таблиці table\_name:

ALTER TABLE table\_name DROP COLUMN column\_c;

Додати індекс з ім'ям index\_name\_ix для колонки column\_b

ALTER TABLE table\_name ADD INDEX index\_name\_ix (column\_b);

**Індекси**[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/alter#%D1%96%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%B8)

Бази даних спрямовані на операції модифікації, а особливо оператор INSERT. Тому процес вибірки відбувається довго. І це проблема баз даних на будь-якому сервері. Щоб вирішити цю проблему, вигадали індекси, індекси - це спроба уникнути простого перебору записів у базі даних.

У нас у базі в таблиці users записані різні користувачі в колонку fullname і вони не відсортовані



Якщо ми намагатимемося знайти Steve, то це буде перебирання і шуканий користувач буде четвертим, додамо індекс на поле fullname

CREATE INDEX fullname\_ix ON users (fullname);

Тепер вибірка даних прискорюється в рази за полем fullname, а ось запис уповільнюється, оскільки додаванні запису потрібно перераховувати дерево індексу та відсортувати дані. Також потрібно враховувати, що розмір бази також значно збільшиться і на один запит до таблиці працює лише один індекс.

Індекси потрібно додавати розумно, не варто на всі поля додавати індекси. Сам індекс створюється трьома шляхами.

Перший під час створення таблиці.

CREATE TABLE table1

(

id INT UNSIGNED NOT NULL AUTO\_INCREMENT COMMENT 'Код товару',

code char(4) NOT NULL DEFAULT 'AAAA',

name VARCHAR(50) NOT NULL DEFAULT '' COMMENT 'Назва товару',

description TEXT NOT NULL DEFAULT '' COMMENT 'Опис товару',

price FLOAT NOT NULL DEFAULT 0 COMMENT 'Ціна товару',

CONSTRAINT table1\_id\_pk PRIMARY KEY (id),

INDEX table1\_name\_ix(name),

INDEX table1\_price\_ix(price)

) COMMENT 'Таблиця товарів з ключами та індексами';

У цьому запиті ми створили індекс з ім'ям table1\_name\_ix на колонку name та індекс table1\_price\_ix на колонку price.

Другий, змінюючи таблицю через оператор ALTER:

ALTER TABLE table1 ADD INDEX table1\_name\_ix(name), ADD INDEX table1\_price\_ix(price);

І останній третій - це додати індекс через оператор CREATE:

CREATE INDEX table1\_name\_ix ON table1 (name);

Також ми можемо створювати складені індекси

CREATE INDEX table1\_name\_price\_ix ON table1 (name, price);

Складені індекси потрібні, якщо ми часто шукаємо за кількома полями, наприклад, у нас інтернет магазин і ми помічаємо, що часто йде пошук за ім'ям товару, у якого ціна більша 5000, тоді ми робимо складений індекс на ім'я товару та ціну, і тепер вибірка на такий запит буде майже моментальна.

Щоб видалити індекс, потрібно виконати оператор DROP:

DROP INDEX table1\_price\_ix ON table1;

INFO

Є невеликі правила іменуванням, вони не особливо суворі і у різних компаній можуть відрізнятися.

Приписка ix може бути на початку або наприкінці імені індексу. Приклад у верблюжій нотації

ixPrice или priceIx

Також може бути через нижнє підкреслення

ix\_price или price\_ix

Хорошим тоном вважається вказувати в імені індексу таблицю, для MySQL це не критично, для таких баз як PostgreSQL є обов'язковим.

ix\_table1\_price или table1\_price\_ix

Зазначення таблиці в імені індексу спрощує розуміння при оптимізації складних запитів, з якої таблиці цей індекс.

**Обмеження**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/alter#%D0%BE%D0%B1%D0%BC%D0%B5%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F)Ми вже стикалися з обмеженнями, настав час познайомитись з ними детальніше. Це насправді індекси. У більшості випадків, роблячи обмеження, сервер сам створює індекс без нашого втручання.

Розберемо, які основні обмеження є:

* **PRIMARY KEY**– первинний ключ (pk).
* **UNIQUE KEY**– унікальний ключ, має всі властивості первинного, але, при цьому, первинним не є.
* **FOREIGN KEY** – зовнішній ключ (fk). Для нас це зв'язок з іншою таблицею, а для сервера – це обмеження.

Розберемо їх окремо

**PRIMARY KEY**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/alter#primary-key)Коли ми створювали таблиці, використовували PRIMART KEY, це був індекс, як і всі обмеження. Коли ми говоримо серверу PRIMART KEY, ми ставимо обмеження первинного ключа, тобто ми пояснюємо серверу, що обов'язково перевіряй на унікальність.

NOTE

Як і всі індекси, може бути складним, але зазвичай це не зручно.

Первинний ключ – це колонка, за якою ми завжди можемо знайти запис. Наприклад, якщо це автоінкрементне, то це дає гарантію, що якщо запис має id рівне 10, то більше записів з id 10 точно не буде. Основне правило – первинний ключ повинен бути *унікальним* та вічним.

Зазвичай він створюється під час опису таблиці. Але якщо ми хочемо додати первинний ключ таблиці, де його немає з якоїсь причини, то:

ALTER TABLE table1 ADD CONSTRAINT table1\_id\_pk PRIMARY KEY (id);

Якщо ми хочемо видалити первинний ключ:

ALTER TABLE table1 DROP PRIMARY KEY;

NOTE

Якщо поле AUTO\_INCREMENT, то видалити первинний ключ не вийде. В імені PRIMARY KEY повинна бути приписка pk: table1\_id\_pk або pk\_table1\_id

UNIQUE KEY​

Це унікальний ключ, має всі властивості первинного, але, при цьому, первинним не є. Допомагає не записувати дублі.

Припустимо, що у нас є таблиця користувачів, де є первинний ключ, але email користувача теж унікальний, розділяти в різні таблиці неправильно.

З'являється можливість того, що зловмисник зареєструє фейк. Щоб такого не було, ми просто робимо email теж унікальним.

Створення при описі структури

CREATE TABLE persons (

id INT PRIMARY KEY,

email CHAR(50) NOT NULL,

fullName varchar(100) NULL,

CONSTRAINT persons\_email\_un UNIQUE KEY (email)

);

Або додати його за допомогою ALTER, якщо у структурі таблиці його немає:

ALTER TABLE persons ADD CONSTRAINT persons\_email\_un UNIQUE KEY (email);

Тепер не можна записати однакове значення у полі email.

NOTE

В імені UNIQUE KEY повинна бути

приписка un або uq: persons\_email\_un або un\_persons\_email

**FOREIGN KEY**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-06/module-06-01/alter#foreign-key)Зовнішній ключ. Ми його вже використовували і він потрібний для зв'язку двох таблиць.

Створюємо батьківську таблицю

CREATE TABLE genders (

id INT PRIMARY KEY,

name VARCHAR(30),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

Та дочірню

CREATE TABLE users (

id INT PRIMARY KEY,

name VARCHAR(30),

email VARCHAR(30),

password VARCHAR(30),

age TINYINT UNSIGNED,

gender\_id INT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (gender\_id) REFERENCES genders (id)

ON DELETE SET NULL

ON UPDATE CASCADE

);

Потрібно запам'ятати основні принципи:

1. Завжди під час створення FOREIGN KEY потрібно спиратися на первинний ключ pk або унікальний ключ.
2. Колонка, яка відповідає за зв'язок, повинна бути з тим самим типом, що і колонка батька. У нашому випадку і gender\_id, і id із таблиці genders мають однаковий тип INT

Якщо написати ON DELETE CASCADE, це означає, що якщо в genders видалити запис, то й у users видаляться всі пов'язані записи. Є необов'язковим. Якщо ми не хочемо, щоб поле видалялося з батьківським, задаємо ON DELETE SET NULL, тоді при видаленні ми встановимо в полі gender\_id значення NULL. Можна взагалі не вказувати, але тоді при видаленні ми отримаємо помилку, потрібно буде спочатку видалити записи з дочірньої таблиці, а потім уже з батьківської, іноді потрібна саме така поведінка.

Можна також задати ON UPDATE CASCADE, це означатиме, що при зміні ключа у батьківської він зміниться і у дочірньої таблиці, операція вкрай рідкісна і її часто просто пропускають при визначенні ключа.

Зовнішній ключ потрібен, щоб ми точно знали, яка таблиця використовується.

Якщо ми спробуємо додати первинний ключ з іншої таблиці, в колонку для зв'язку таблиць, якого там не повинно бути, то без обмежень ми навіть не дізнаємося, що зробили помилку.

NOTE

В імені FOREIGN KEY повинна бути приписка

fk: children\_parentId\_fk або fk\_children\_parentId

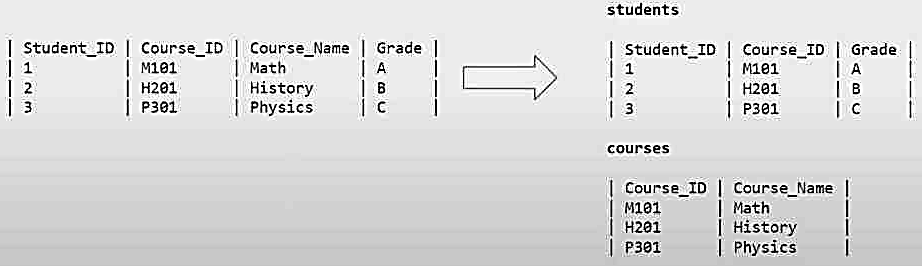
**Перша нормальна форма**

Перша нормальна форма (1ИР) - це стадія нормалізації таблиць у базі даних, у якій гарантується, що всі значення в стовпчиках є атомарними, тобто не можуть бути розділені на більш дрібні частини. Це означає, що в кожному стовпчику таблиці має зберігатися тільки одне значення, а не набір значень.



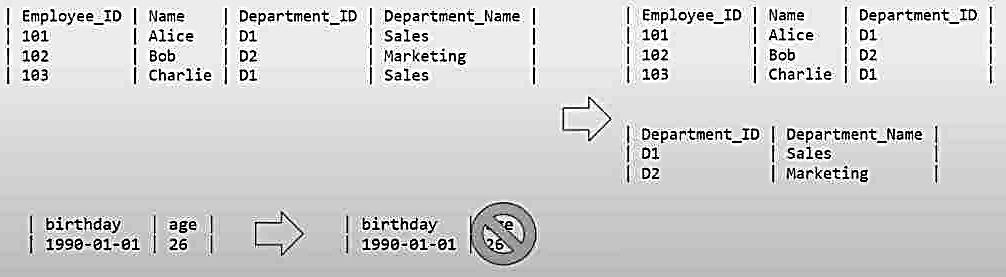
**Друга нормальна форма**

Друга нормальна форма припускає, що таблиця вже перебуває в першій нормальній формі та всі її не ключові стовпці залежать від первинного ключа



**Третя нормальна форма**

Третя нормальна форма припускає, що таблиця вже перебуває в другій нормальній формі і кожен її не ключовий стовпець нетранзитивно залежить від первинного ключа



**Використання оператора LIKE для пошуку даних**

Оператор LIKE використовується для пошуку даних за шаблоном.

Синтаксис:

SELECT column1, column2, ...

FROM table\_name

WHERE column\_name LIKE pattern;

**Патерни оператора LIKE**

1. *%-* відповідає будь-якому рядку символів, навіть порожньому рядку

2. \_ - відповідає будь-якому одиночному символу

3.[abc] - відповідає будь-якому символу в дужках (у цьому випадку ‘а’, ‘b’ або ‘с’)

4. [^аbс] - відповідає будь-якому символу, окрім тих, що перераховані в дужках (у даному випадку будь-якому символу, окрім ‘а’, ‘b’ і ‘с’)

5. [a-z] - відповідає будь-якому символу в діапазоні від ‘а’ до ‘z’

6. [^a-z] - не відповідає будь-якому символу в діапазоні від ‘а’ до ‘z’

Синтаксис:

SELECT \* FROM table\_name WHERE column1 LIKE ‘а\_[^е-о]%’