**Фільтрація даних**

Напишіть SQL-запит для виведення всіх записів про замовлення, вартість яких не перевищує $250.

SELECT \* FROM Orders WHERE Cost <= 250;

Напишіть SQL-запит, щоб вивести всі записи про замовлення, вартість яких становить від $50 до $750.

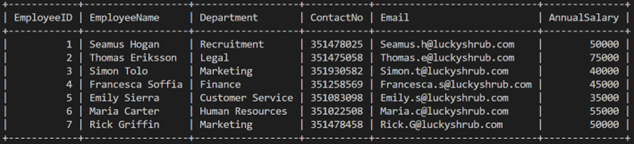
SELECT \* FROM Orders WHERE Cost > 50 AND Cost < 750;

Напишіть SQL-запит, щоб вивести всі записи про замовлення, які були зроблені клієнтом з ідентифікатором Cl3 і де вартість замовлення більше $100.

SELECT \* FROM Orders WHERE ClientID = "Cl3" and Cost > 100;

Напишіть SQL-запит, щоб вивести всі записи про замовлення, які мають ідентифікатор товару p1 або p2 і кількість замовлення більше 2.

SELECT \* FROM Orders WHERE ProductID = "P1" OR ProductID = "P2" AND Quantity > 2;



За допомогою оператора AND знайдіть співробітників, які отримують річну зарплату $50,000 або більше і працюють у відділі маркетингу.

SELECT \* FROM employees WHERE AnnualSalary >= 50000 AND Department = 'Marketing';

Використовуйте оператор NOT, щоб знайти співробітників, які не заробляють більше $50,000 у всіх відділах.

SELECT \* FROM employees WHERE NOT AnnualSalary > 50000;

Використовуйте оператор IN, щоб знайти співробітників маркетингового, фінансового та юридичного відділів, чия річна зарплата не перевищує $50 000.

SELECT \* FROM employees WHERE AnnualSalary < 50000 AND Department IN('Marketing',

'Finance', 'Legal');

Використовуйте оператор BETWEEN, щоб знайти співробітників, які отримують річну зарплату від $10 000 до $50 000.

SELECT \* FROM employees WHERE AnnualSalary BETWEEN 10000 AND 50000;

Використовуйте оператор LIKE, щоб знайти співробітників, чиї імена починаються на 'S' і мають довжину не менше 4 символів.

SELECT \* FROM employees WHERE EmployeeName LIKE 'S\_\_\_%';

Використовуйте речення WHERE в операторах SQL SELECT, UPDATE і DELETE для фільтрації даних.

У реченні WHERE можна вказати одну або кілька умов фільтрації, щоб визначити, як обмежити або звузити вибірку даних із запитуваних таблиць.

При формулюванні умов фільтрації можна також використовувати логічні та деякі інші оператори.

Логічні оператори І та АБО використовуються як **кон'юнктивні** оператори для об'єднання декількох умов у реченні WHERE. Логічний оператор NOT **заперечує** результат обчислення заданої умови (або умов) і відповідно фільтрує дані.

Оператор IN може бути альтернативою оператору OR. Оператор IN може перевіряти декілька значень, що еквівалентно використанню логічного оператора OR для вказівки різних умов.

Оператор BETWEEN - це зручний спосіб виконати перевірку діапазону для числових і датових значень. Його можна використовувати як альтернативу операторам порівняння більше або дорівнює (>=) і менше або дорівнює (<=).

Використовуйте оператор LIKE для порівняння і фільтрації даних на основі цих шаблонів. Для формулювання шаблонів використовуються символи підстановки, відсоток (%) і підкреслення (\_). Символ процента (%) відповідає будь-якому рядку з нуля або більше символів. Символ підкреслення (\_) відповідає будь-якому одному символу.

**Об’єднання таблиць**

**Що таке об'єднання?**

Об'єднання в базі даних пов'язує записи даних між однією або декількома таблицями на основі спільного стовпця між ними.

**Навіщо потрібно використовувати JOIN?**

Іноді вам потрібно знайти інформацію про певну діяльність або об'єкт в базі даних, де відповідна інформація існує в більш ніж одній таблиці. У цій ситуації ви можете використовувати речення SQL JOIN для запиту необхідних даних з декількох таблиць.

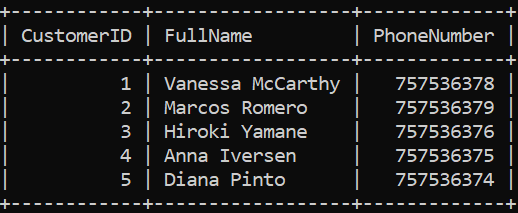
У MySQL підтримується чотири різних типи об'єднань, які розглядаються в цьому уроці.

* ВНУТРІШНЄ ОБ'ЄДНАННЯ
* LEFT JOIN
* ПРАВЕ ОБ'ЄДНАННЯ
* САМОЗ'ЄДНАННЯ

Щоб пояснити різницю між цими типами з'єднань, розглянемо базу даних ресторану Little Lemon, яка складається з двох таблиць.

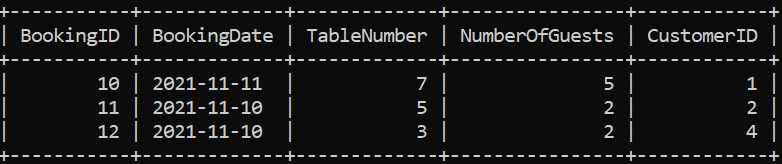
Перша - це таблиця Customers з наступними стовпчиками:

* CustomerID,
* FullName
* і стовпцями PhoneNumber, як показано нижче:



Друга таблиця - це таблиця бронювань зі стовпчиками

* BookingID,
* BookingDate,
* TableNumber,
* NumberOfGuests
* та CustomerID.



Ви могли помітити, що обидві таблиці містять стовпець Customer ID, який є спільним стовпцем.

**ВНУТРІШНЄ** ОБ'ЄДНАННЯ

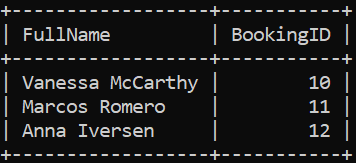
Цей тип з'єднання повертає записи даних, які мають однакові значення в таблицях, що об'єднуються. Наприклад, припустимо, що ви хочете повернути повне ім'я та ідентифікатор бронювання клієнтів, які здійснили бронювання. У цій ситуації ви можете використати речення INNER JOIN, щоб витягти записи даних з таблиць Customers і Bookings на основі відповідного значення ID клієнта, як показано нижче.

 SELECT Customers.FullName, Bookings.BookingID

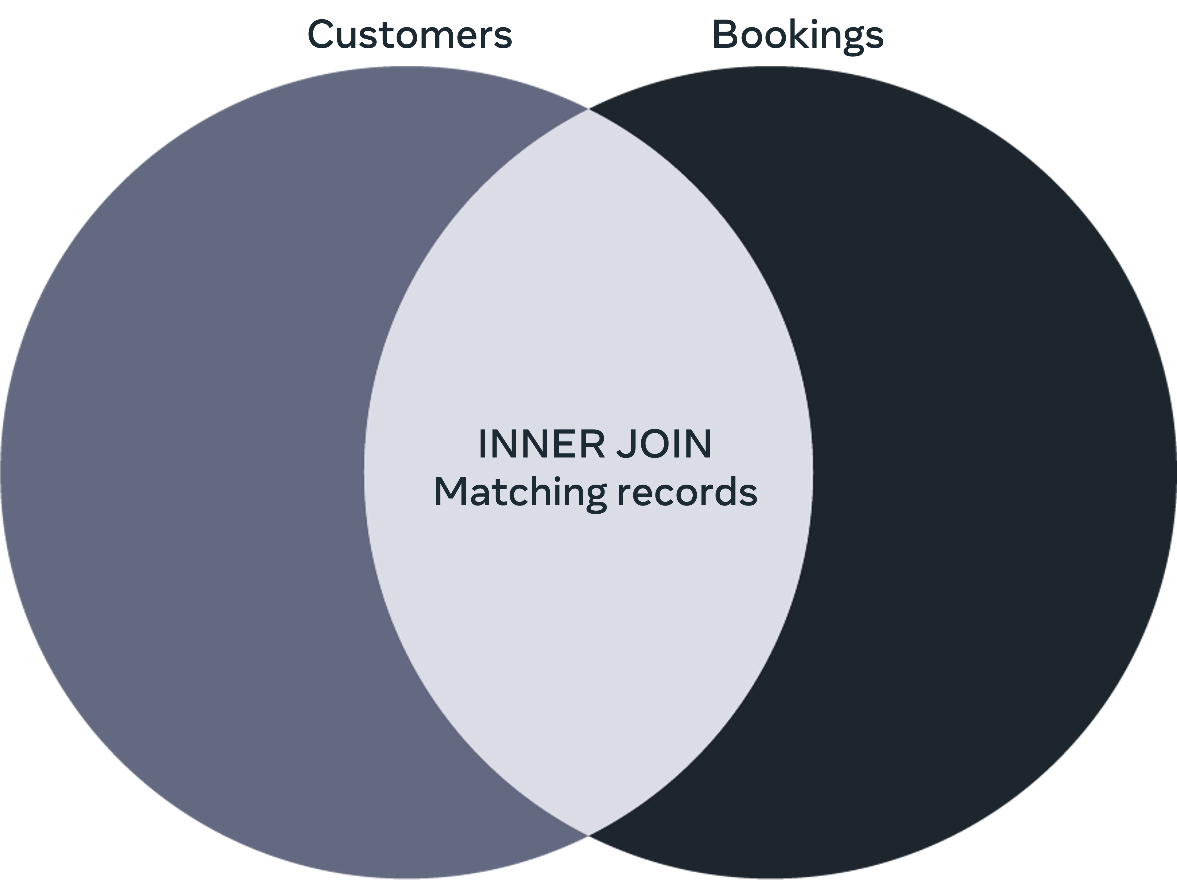
 FROM Customers INNER JOIN Bookings

 ON Customers.CustomerID = Bookings.CustomerID;

Вихідний результат показано нижче



INNER JOIN проілюстровано на наступній діаграмі Венна.



**ЛІВЕ** З'ЄДНАННЯ

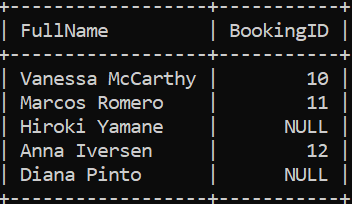
Ви можете використовувати речення LEFT JOIN для вилучення повних імен та ідентифікаторів бронювань з таблиць Customers і Bookings, як показано нижче:

SELECT Customers.FullName, Bookings.BookingID

FROM Customers LEFT JOIN Bookings

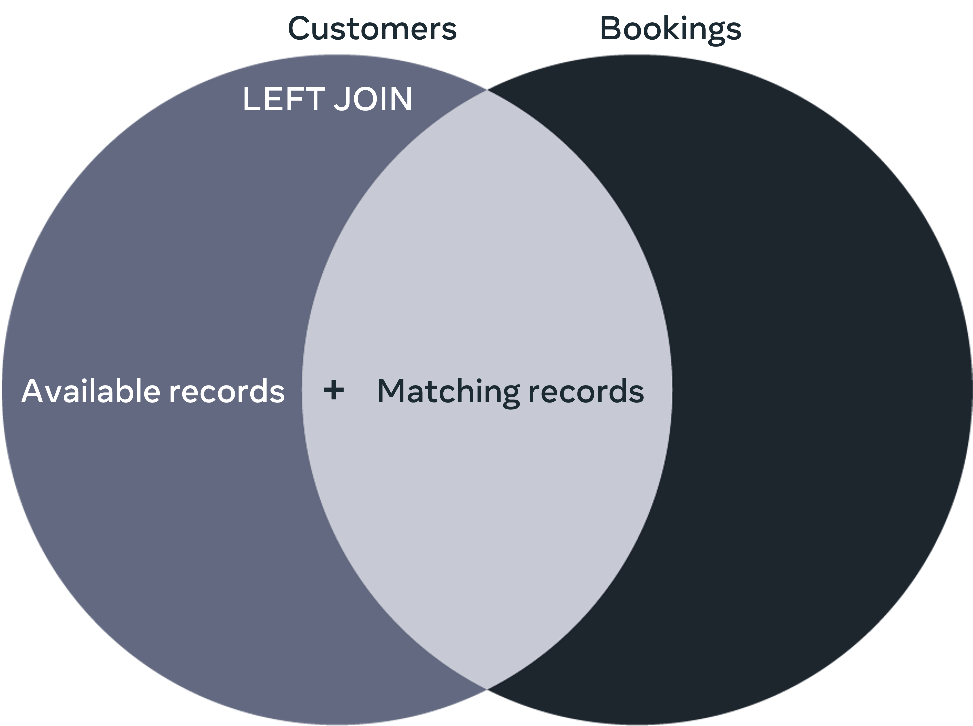
ON Customers.CustomerID =  Bookings.CustomerID;

Результати цього запиту наведені нижче:



LEFT JOIN повертає всі спільні записи так само, як і INNER JOIN, плюс всі запитувані записи з лівої таблиці, незалежно від того, чи є збіг у правій таблиці чи ні. Якщо в правій таблиці немає відповідних записів, то для ідентифікаторів бронювань будуть вставлені нульові значення.

ЛІВЕ З'ЄДНАННЯ проілюстровано на наступній діаграмі Венна.



**ПРАВЕ** З'ЄДНАННЯ

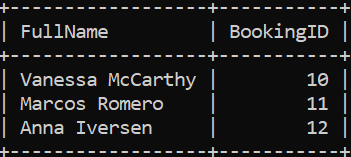
Ви можете використовувати речення RIGHT JOIN для вилучення повних імен та ідентифікаторів бронювань з таблиць Клієнти і Бронювання наступним чином:

SELECT Customers.FullName, Bookings.BookingID

FROM Customers RIGHT JOIN Bookings

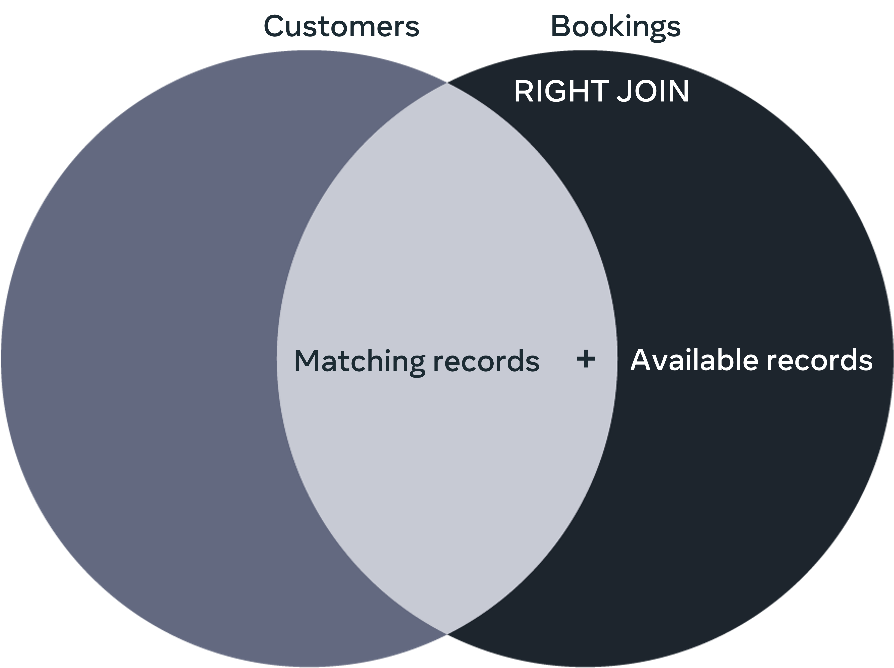
ON Customers.CustomerID = Bookings.CustomerID;

Результат цього запиту буде наступним:



RIGHT JOIN повертає всі спільні записи так само, як і INNER JOIN, плюс всі запитувані записи з правої таблиці, незалежно від того, чи є збіг у лівій таблиці чи ні. Якщо в лівій таблиці немає відповідних записів, то для повних імен клієнтів будуть вставлені нульові значення.

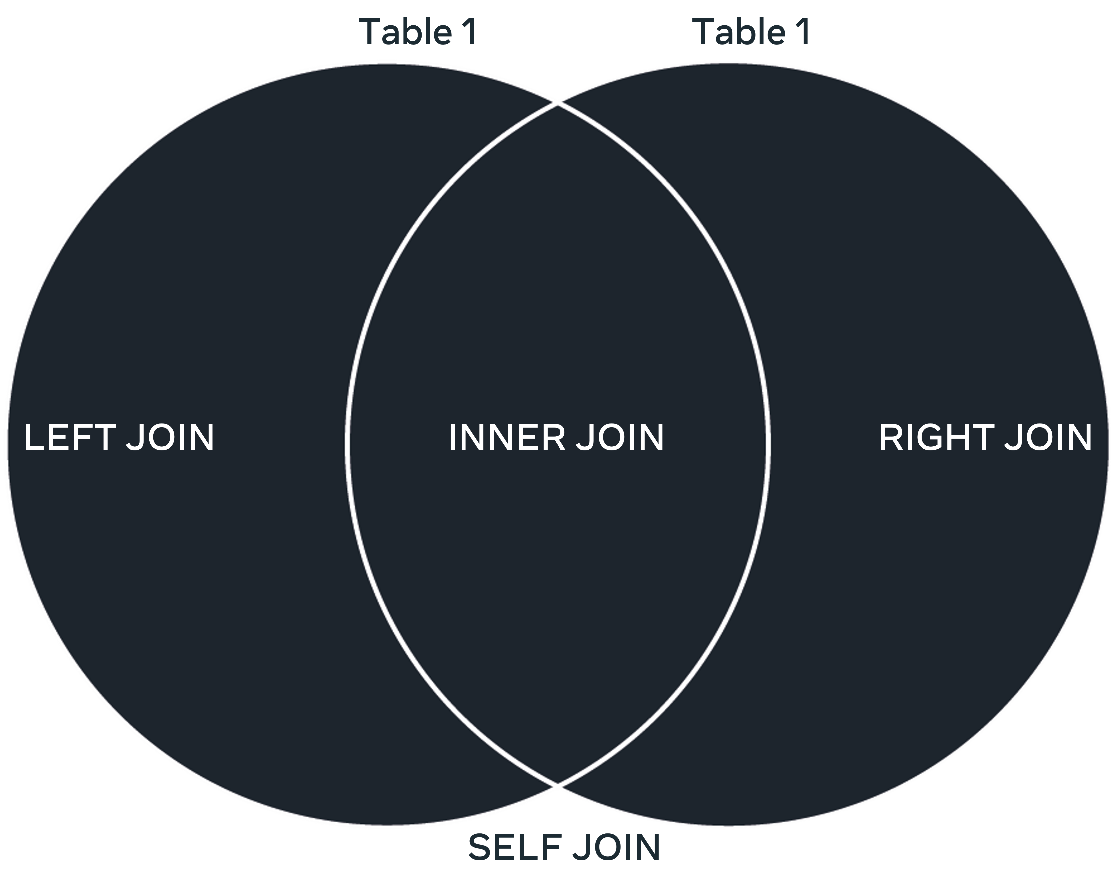
ПРАВЕ З'ЄДНАННЯ проілюстровано на наступній діаграмі Венна.



**САМООБ**'ЄДНАННЯ

Це особливий випадок, коли вам потрібно об'єднати таблицю з самою собою, щоб отримати певну інформацію, яка міститься в тій самій таблиці. У цьому випадку ви можете вибрати INNER JOIN, LEFT JOIN або RIGHT JOIN, представлені раніше, щоб запитати необхідні дані.

Самооб'єднання проілюстровано на наступній діаграмі Венна.



**Висновок**

JOIN з'єднує таблиці в базі даних на основі спільного стовпця між ними. У цьому уроці ви дізнаєтеся про чотири різні типи з'єднань, що підтримуються в MySQL, які можна використовувати для надання користувачам інформації про пов'язані записи даних. Важливо розуміти, як працює кожен з цих типів, щоб отримати відповідний тип інформації.

**Групування даних**

Після того, як ви виконаєте ці завдання, перевірте і порівняйте свої відповіді з наведеними нижче рішеннями:

Напишіть інструкцію SQL SELECT, щоб згрупувати всі записи, які мають однакову дату замовлення.

SELECT OrderDate FROM Orders GROUP BY OrderDate;

Напишіть інструкцію SQL SELECT для отримання кількості замовлень, розміщених в один день.

SELECT OrderDate,COUNT(OrderID) FROM Orders GROUP BY OrderDate;

Напишіть інструкцію SQL SELECT, щоб отримати загальну кількість замовлень, розміщених кожним відділом.

SELECT Department, SUM(OrderQty) FROM Orders GROUP BY Department;

Напишіть інструкцію SQL SELECT для отримання кількості замовлень, розміщених в один день між наступними датами: 1 червня 2022 року та 30 червня 2022 року.

SELECT OrderDate,COUNT(OrderID) FROM Orders GROUP BY OrderDate HAVING OrderDate BETWEEN '2022-06-01' AND '2022-06-30';

**GROUP BY**

Використовуйте речення GROUP BY в операторі SELECT, щоб згрупувати рядки в таблиці (таблицях) на основі заданого стовпця (стовпців) у підсумкові рядки або підгрупи.

Воно розміщується після речення FROM. Якщо в операторі SELECT є інструкція WHERE, її слід розміщувати після інструкції WHERE. Після ключового слова GROUP BY розмістіть список імен стовпців, розділених комами, за якими ви хочете згрупувати дані.

**HAVING**

Якщо ви також хочете відфільтрувати згруповані дані, використовуйте речення HAVING. Слід пам'ятати, що речення WHERE не може фільтрувати згруповані дані. Речення HAVING має стояти після речення GROUP BY. У реченні HAVING ви можете вказати умови фільтрації, які потрібно застосувати до згрупованих даних.

**Оператор ANY**

Оператор ANY дозволяє виконати порівняння між значенням одного стовпчика і діапазоном інших значень. Діапазон значень отримується в результаті виконання підзапиту.

Синтаксис оператора ANY наступний:

SELECT column\_name(s)

FROM table\_name

WHERE column\_name comparison operator ANY

  (SELECT column\_name

  FROM table\_name

  WHERE condition);

Оператор ANY повертає логічне значення після операції порівняння. Він повертає значення ІСТИНА, якщо БУДЬ-ЯКІ значення підзапиту відповідають заданій умові. Іншими словами, умова буде ІСТИНА, якщо операція істинна для будь-якого значення в діапазоні.

У цьому синтаксисі оператору ANY має передувати ім'я стовпця та оператор порівняння, який порівнює ім'я стовпця з набором значень.

Тут можна використовувати стандартні оператори порівняння, такі як =, <>, !=, >, >=, < або <=.

Оператор **ALL**

Оператор ALL використовується з тією ж метою, що й оператор ANY. Однак, спосіб його роботи дещо відрізняється. Він повертає логічне значення в результаті виконання операції порівняння. Він повертає значення TRUE тільки в тому випадку, якщо ВСІ значення підзапиту відповідають заданій умові. Іншими словами, умова буде ІСТИНА тільки в тому випадку, якщо операція істинна для всіх значень в діапазоні.

Синтаксис оператора ALL наступний:

SELECT column\_name(s)

FROM table\_name

WHERE column\_name operator ALL

  (SELECT column\_name FROM table\_name WHERE condition);

Синтаксис інтерпретується так само, як і для ANY, і з ним можна використовувати стандартні оператори порівняння.

**Оновлення та вставка даних**

## Що таке оператор REPLACE в MySQL?

Оператор REPLACE в MySQL - це альтернативний спосіб вставки і оновлення даних в таблиці бази даних. Це розширення до стандарту SQL, яке вставляє або оновлює дані в таблиці. Ви використовуєте його для дещо інших цілей, ніж стандартні INSERT INTO і UPDATE. (Це буде з'ясовано пізніше в цій статті, коли буде пояснено, як працює кожен з трьох операторів) Ви можете використовувати два типи синтаксису для вставки або оновлення даних за допомогою оператора REPLACE. Перший синтаксис дуже подібний до стандартного оператора SQL INSERT INTO, де замість команди INSERT використовується команда REPLACE, як показано нижче:

REPLACE INTO table\_name (column1name, column2name, ...)

VALUES (value1, value2, ...);

Другий синтаксис подібний до стандартного оператора SQL UPDATE, де ви використовуєте команду REPLACE з ключовим словом SET для оновлення даних у таблиці ось так.

REPLACE INTO table\_name SET column1name = value, column2name = value, ... ;

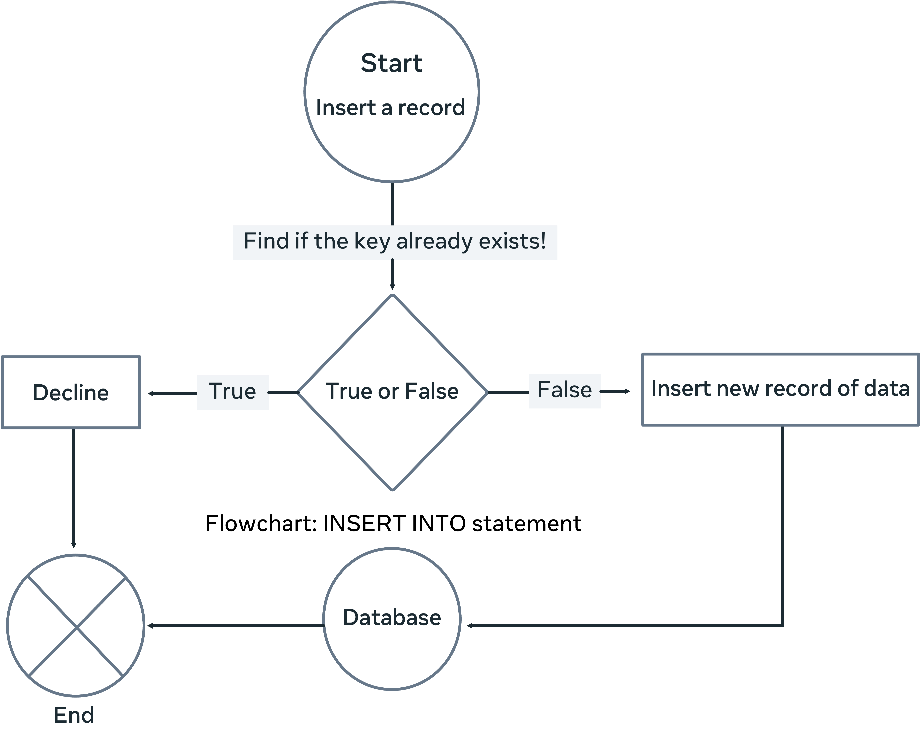
Оскільки інструкція INSERT INTO вставляє дані, а інструкція UPDATE оновлює дані, навіщо потрібна інструкція REPLACE?

Щоб відповісти на це питання, потрібно зрозуміти, як працює кожен з трьох операторів.

## Як працює інструкція INSERT INTO

Оператор INSERT INTO намагається вставити новий запис даних. Він перевіряє, чи унікальний ключ вже існує в таблиці. Якщо значення YES або TRUE, процес вставки відхиляється, і MySQL генерує повідомлення про помилку.

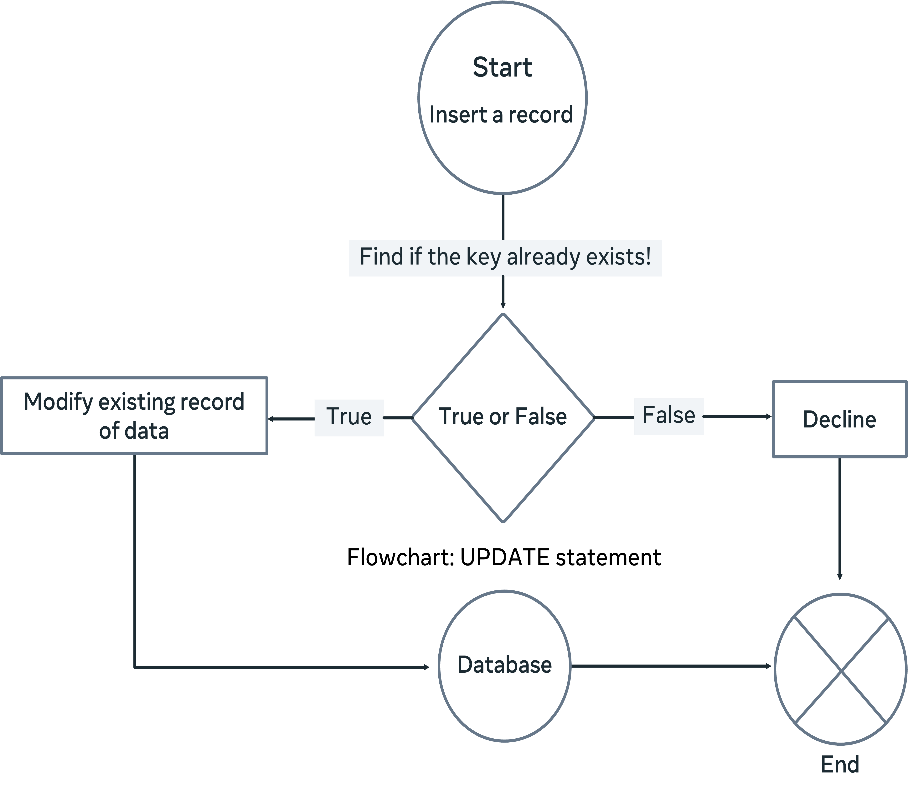
Припустимо, що повертається значення NO або FALSE. У цьому випадку процес вставки буде завершено, і новий запис буде додано до бази даних. Нижче наведено блок-схему, яка демонструє, як працює інструкція INSERT INTO.



## Як працює інструкція UPDATE

Оператор update намагається змінити існуючий запис новими даними. Він перевіряє, чи унікальний ключ вже існує в таблиці. Припустимо, що повертається значення NO або FALSE. У цьому випадку процес оновлення буде відхилено, і MySQL видасть повідомлення про помилку.

Припустимо, що повертається значення YES або TRUE. У цьому випадку процес оновлення завершується, а існуючий запис даних модифікується новими даними. Нижче наведено блок-схему, що демонструє роботу оператора UPDATE.



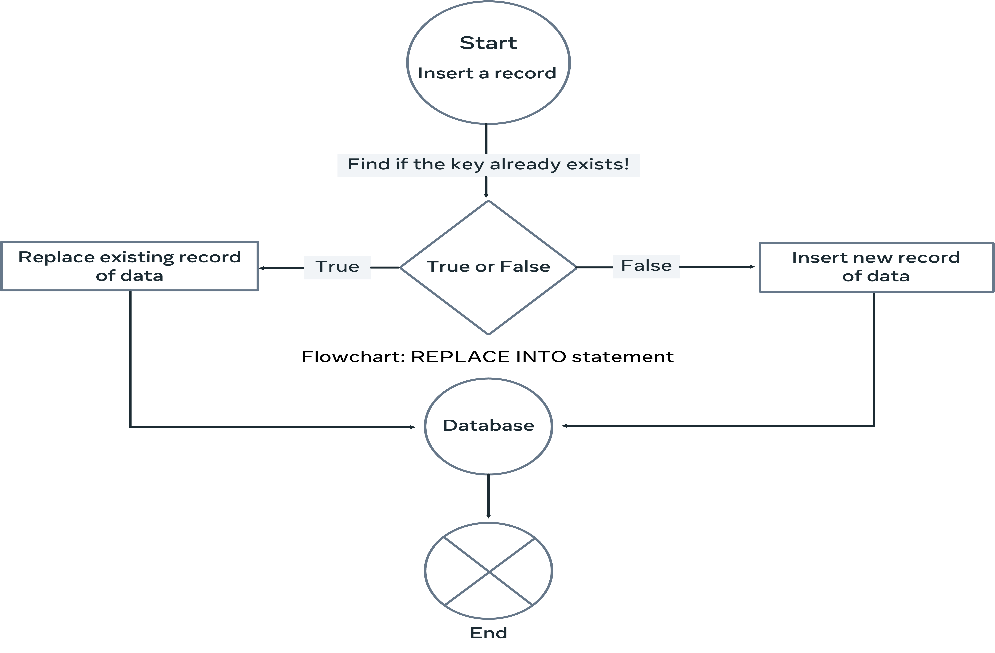
# Як працює інструкція REPLACE INTO

Оператор REPLACE перевіряє, чи унікальне значення ключа потрібного запису даних вже існує в таблиці, перш ніж вставити його як новий запис або оновити його.

Оператор REPLACE INTO намагається вставити новий запис або змінити існуючий. В обох випадках перевіряється, чи унікальний ключ запропонованого запису вже існує у таблиці. Припустимо, що повертається значення NO або FALSE. У цьому випадку оператор REPLACE вставляє запис подібно до оператора INSERT INTO.

Припустимо, що значення ключа вже існує в таблиці (іншими словами, дублікат ключа). У цьому випадку оператор REPLACE видаляє існуючий запис даних і замінює його новим записом даних. Це відбувається незалежно від того, чи ви використовуєте перший або другий синтаксис оператора REPLACE.

Нижче наведено блок-схему роботи оператора REPLACE INTO.



Щойно інструкція REPLACE INTO використовується для вставки або модифікації даних, вона спочатку визначає, чи існує новий запис даних у таблиці. Перевіряється, чи збігається ПЕРВИННИЙ або УНІКАЛЬНИЙ КЛЮЧ з одним з існуючих записів.

Якщо збігу немає, REPLACE працює як звичайний оператор INSERT. В іншому випадку він видаляє існуючий запис і замінює його новим. Це вважається своєрідною модифікацією або оновленням існуючого запису. Однак, тут краще бути обережним. Припустимо, ви не вказали значення для стовпця в реченні SET. У цьому випадку оператор REPLACE використовує значення за замовчуванням (якщо значення за замовчуванням було встановлено). В іншому випадку, воно встановлюється як NULL.

## Висновок

Оператор REPLACE в MySQL - це альтернативний спосіб вставки і оновлення даних в таблиці бази даних. Він корисний у багатьох випадках, особливо коли ви хочете видалити або замінити існуючі записи новими.

**Обмеження**

## Що таке обмеження в MySQL?

Обмеження в базах даних MySQL використовуються для визначення правил, які підтримують дані в таблицях, щоб забезпечити їхню валідність, точність, узгодженість і надійність.

Застосування обмежень до даних у реляційній базі даних контролює тип даних, що зберігаються в таблиці. Це змусить MySQL відхиляти процеси, які порушують зазначені правила.

Наприклад, у багатьох країнах ви повинні досягти 18 років, щоб подати заявку на отримання водійських прав. У цьому випадку ви можете застосувати обмеження SQL CHECK до відповідного стовпця у відповідній таблиці в базі даних, щоб обмежити реєстрацію будь-якої особи, яка не досягла 18 років.

Іншим прикладом є те, що в більшості університетів сьогодні кожен студент реєструється з адресою електронної пошти, що містить доменне ім'я університету. Ви можете застосувати обмеження UNIQUE, щоб гарантувати, що університет не надасть двом студентам однакові адреси електронної пошти.

У цих випадках, якщо виникає конфлікт між обмеженнями і процесом вставки або оновлення даних, процес, що порушує правило обмеження, буде перерваний.

## Типи обмежень

Існує три основних типи обмежень, які можна застосувати в MySQL.

## Обмеження на ключі

У реляційній базі даних існують різні типи ключів. Наприклад, кожна таблиця повинна мати первинний ключ, який підтримує цілісність таблиці. Первинний ключ гарантує відсутність дублікатів записів в одній таблиці. Крім того, він дозволяє ідентифікувати кожен запис даних за допомогою значення первинного ключа. Тому він повинен бути унікальним у кожному рядку таблиці і не повинен містити нульових значень.

Наприклад, кожен громадянин, який проживає в Данії, повинен мати унікальний персональний номер, який можна використовувати для доступу до різних видів державних послуг.

## Доменні обмеження

Доменні обмеження - це спеціальні правила, визначені для значень, які можуть зберігатися для певного стовпця. Щоб їх застосувати, ви повинні вказати, які значення даних дозволені, а які - ні.

Наприклад, ви можете визначити допустимий діапазон, в якому користувачі можуть оцінювати потоковий сервіс, що пропонує широкий вибір телепередач і фільмів. Цей діапазон може бути числом від 3 до 10, і в цьому випадку користувач не зможе вставити значення, яке є більшим за 10 і меншим за 3.

## Обмеження посилальної цілісності

У реляційній базі даних таблиці з'єднуються за допомогою зовнішнього ключа в одній таблиці, пов'язаного з первинним ключем (або унікальним ключем) в іншій таблиці.

Це означає, що значення стовпця зовнішнього ключа у "посилальній" таблиці також повинно існувати у таблиці, на яку посилаються. В іншому випадку ви зіткнетеся з проблемою, оскільки "зв'язок" між записами таблиць припиниться.

Тому підтримка посилальної цілісності вимагає, щоб значення зовнішнього ключа мало відповідне значення первинного ключа для зв'язування записів різних пов'язаних таблиць.

## Висновок

Дуже важливо застосовувати обмеження на дані в реляційній базі даних для збереження даних в таблицях, щоб забезпечити валідність, точність, узгодженість і надійність даних.

**Зміна структури таблиці**

Напишіть SQL-запит, який створює таблицю Staff з наступними стовпцями:

* StaffID: INT
* Повне ім'я: VARCHAR(100)
* PhoneNumber: VARCHAR(10)

CREATE TABLE Staff (StaffID INT, FullName VARCHAR(100), PhoneNumber VARCHAR(10));

Напишіть SQL-запит для застосування наступних обмежень до таблиці Staff:

* StaffID: INT NOT NULL та PRIMARY KEY
* ПІБ: VARCHAR(100) і NOT NULL
* PhoneNumber: INT NOT NULL

ALTER TABLE Staff MODIFY StaffID INT PRIMARY KEY, MODIFY FullName VARCHAR(100) NOT NULL, MODIFY PhoneNumber INT NOT NULL;

Напишіть SQL-запит, який додає до таблиці Staff новий стовпець з назвою Role з наступним обмеженням:

* Role: VARCHAR(50) і NOT NULL

ALTER TABLE Staff ADD COLUMN Role VARCHAR(50) NOT NULL;

Напишіть оператор SQL, який видаляє стовпець Phone Number з таблиці Staff.

ALTER TABLE Staff DROP COLUMN PhoneNumber;

**Підзапити**

Використовуйте запит SQL SELECT, щоб знайти всі бронювання, які повинні бути зроблені після бронювання гостя "Ванесса Маккарті":

SELECT \* FROM Bookings WHERE BookingSlot > (SELECT BookingSlot FROM Bookings WHERE GuestFirstName = 'Vanessa' AND GuestLastName = 'McCarthy');

Використовуйте запит SQL SELECT, щоб знайти пункти меню, які коштують дорожче, ніж всі типи пунктів меню "Закуски" і "Десерти":

SELECT \* FROM MenuItems WHERE Price > ALL (SELECT Price FROM MenuItems WHERE Type IN ('Starters', 'Desserts'));

За допомогою запиту SQL SELECT знайти пункти меню, які коштують стільки ж, скільки і пункти меню "Закуски", що відносяться до італійської кухні:

SELECT \* FROM MenuItems WHERE Price = (SELECT Price FROM Menus, MenuItems WHERE Menus.ItemID = MenuItems.ItemID AND MenuItems.Type = 'Starters' AND Cuisine = 'Italian');

За допомогою запиту SQL SELECT знайти позиції меню, які не були замовлені гостями, що зробили бронювання:

SELECT \* FROM MenuItems

WHERE NOT EXISTS (SELECT \* FROM TableOrders, Menus WHERE TableOrders.MenuID = Menus.MenuID AND Menus.ItemID = MenuItems.ItemID);

**Віртуальні таблиці**

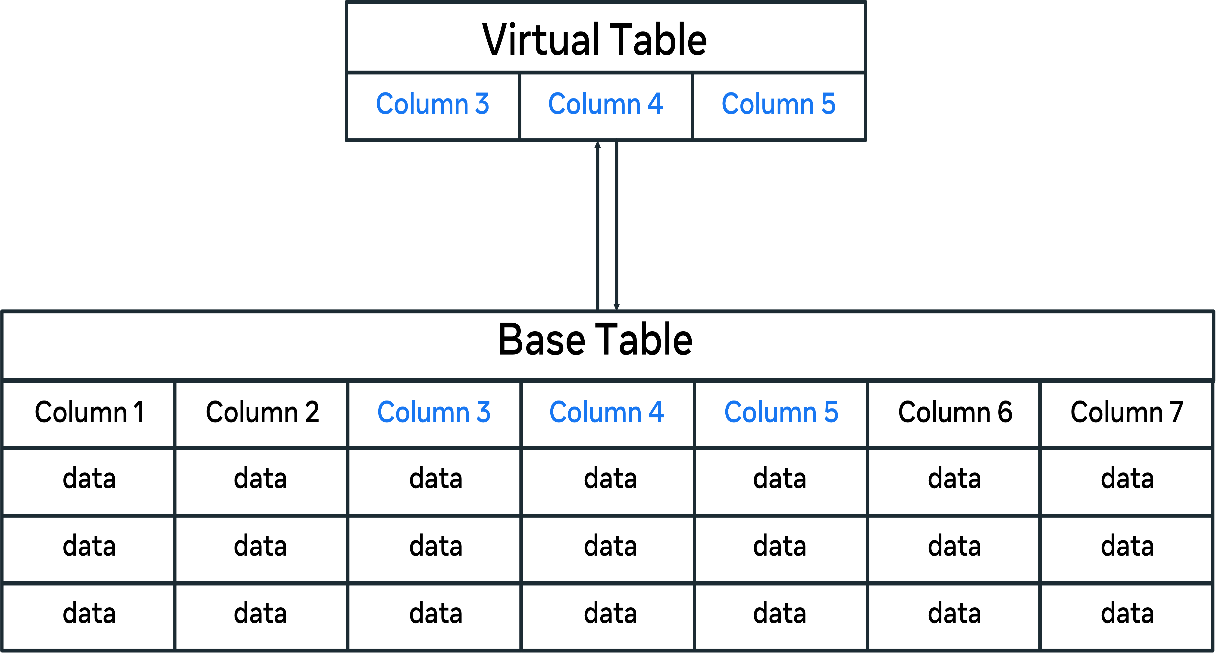
**Що таке представлення в SQL?**

У SQL подання діє як віртуальна таблиця, яка використовує дані, що зберігаються в існуючих таблицях бази даних. Віртуальна таблиця сама по собі не зберігає ніяких даних. Замість цього вона діє як інтерфейс, який забезпечує доступ до існуючих даних.

**Навіщо потрібно використовувати віртуальні таблиці?**

У вас є база даних з базовою таблицею з 7 стовпчиками з назвами: стовпчик 1, стовпчик 2, стовпчик 3, стовпчик 4, стовпчик 5, стовпчик 6 і стовпчик 7.

Однак вас цікавить лише перегляд та аналіз інформації у стовпчиках 3, 4 та 5. У цьому випадку ви можете створити віртуальну таблицю, яка містить три необхідні стовпці. Ця віртуальна таблиця використовує дані, що містяться у відповідних стовпчиках базової таблиці, як показано на наступній ілюстрації.



У цьому випадку всі дані в базовій таблиці можна розглядати як частину віртуальної таблиці. Хоча фізично вони зберігаються в базовій таблиці, а не у віртуальній.

Таким чином, використання віртуальних таблиць замість базових має кілька переваг. Вони включають спрощення доступу до даних, маніпулювання даними та забезпечення безпеки.

Давайте розглянемо ці переваги більш детально.

**Спрощення доступу до даних**

Ви можете використовувати віртуальні таблиці, щоб спростити доступ користувачів до даних у базі даних. Наприклад, Lucky Shrub зберігає в своїй базі даних інформацію про клієнтів, співробітників, продукти та замовлення.

Менеджери Lucky Shrub хочуть відобразити ідентифікатор клієнта, його ім'я та місцезнаходження з таблиці Клієнти, а також розміщені ними замовлення, включаючи ідентифікатор замовлення, кількість і вартість з таблиці Замовлення.

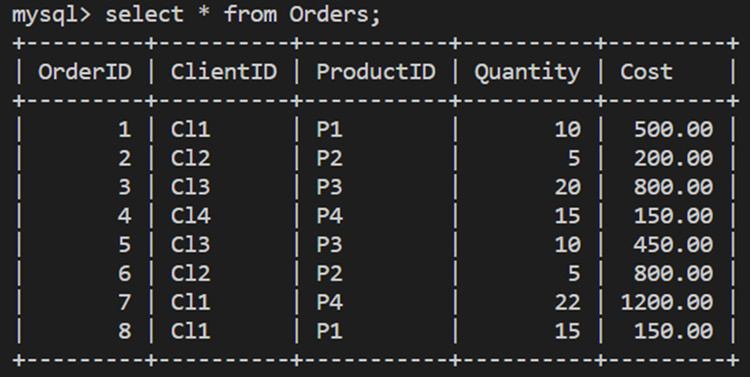
Менеджери хочуть відображати всі ці дані в одному поданні без необхідності вводити складні запити кожного разу, коли вони звертаються до цих даних.

Щоб допомогти менеджерам, ви можете створити віртуальну таблицю з двох таблиць, клієнтів і замовлень. Для цього ви можете об'єднати ідентифікатор клієнта, його ім'я та місцезнаходження з таблиці Клієнти з ідентифікатором замовлення, кількістю та вартістю з таблиці Замовлення.

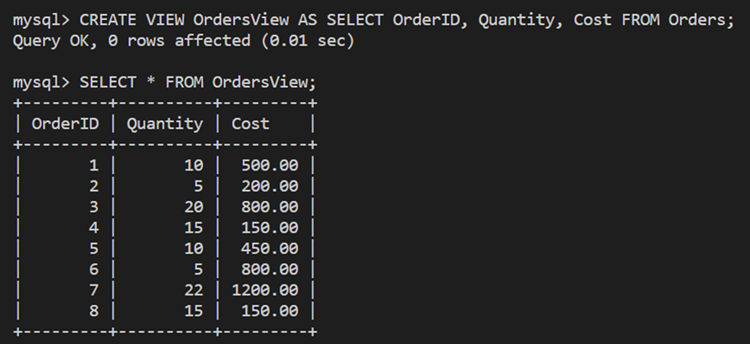
**Маніпулювання даними в базовій таблиці**

Віртуальна таблиця дозволяє маніпулювати, фільтрувати і навіть оновлювати дані в базовій таблиці, якщо це необхідно.

Наприклад, таблиця Замовлення нижче показує інформацію про ідентифікатор замовлення, ідентифікатор клієнта, ідентифікатор товару, кількість і вартість:



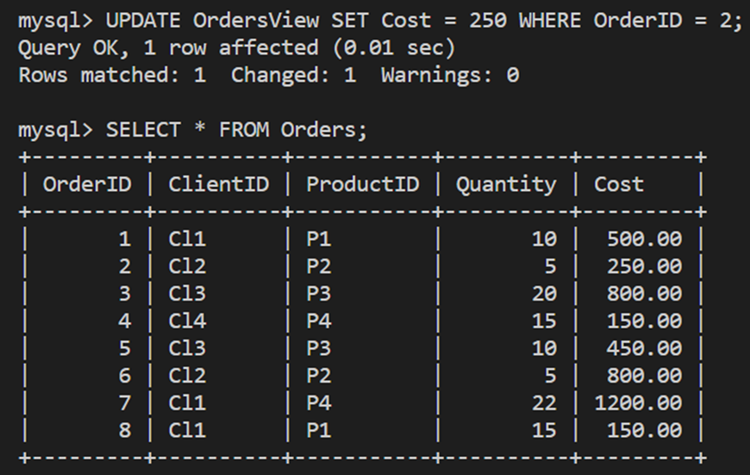
Наступна віртуальна таблиця, "OrdersView", була створена на основі таблиці Orders з використанням лише трьох необхідних стовпців.



Компанії Lucky Shrub потрібно змінити вартість другого запису (ідентифікатор замовлення = 2) на 250. Ви можете зробити цю зміну безпосередньо у віртуальній таблиці. Просто скористайтеся наступним оператором SQL UPDATE:

UPDATE OrdersView SET Cost = 250 WHERE OrderID = 2;

Він оновить вартість для замовлення id = 2 в базовій таблиці Orders, як показано нижче.



**Віртуальні таблиці підтримують безпеку бази даних**

Коли ви проектуєте віртуальні таблиці, ви створюєте інтерфейси для доступу користувачів до відповідних даних у вашій базі даних, подібно до базових таблиць. Однак у віртуальних таблицях ви можете показувати лише потрібні дані і приховувати непотрібні (включно з даними, до яких ви не хочете, щоб користувачі мали доступ).

У попередньому прикладі ви створили віртуальну таблицю з назвою "OrdersView" на основі таблиці Orders з інформацією лише про ідентифікатор замовлення, кількість і вартість. Це обмежує користувачів від перегляду певних стовпців. Вони бачитимуть лише те, що ви хочете, щоб вони бачили. У цьому випадку користувачі не зможуть прочитати інформацію про ідентифікатори клієнтів і товарів. Це може бути конфіденційна інформація, яку ви хочете приховати від інших користувачів бази даних.

**Висновок**

Віртуальні таблиці використовують дані, що зберігаються в існуючих таблицях бази даних, щоб зосередитися на певних типах інформації або приховати дані, які ви не хочете, щоб користувачі бачили. Крім того, ви можете використовувати віртуальні таблиці, щоб отримати всебічний персоналізований перегляд існуючих даних у вашій базі даних. Однак для цього вам потрібно створити подання з декількох таблиць.

**Функції в MySQL**

Напишіть SQL-запит SELECT, використовуючи відповідні рядкові функції MySQL, щоб вивести список товарів, кількість і статус замовлення в наступному форматі: Назва товару-кількість-статус замовлення

SELECT CONCAT(LCASE(Name),'-',Quantity,'-', UCASE(OrderStatus))

FROM item,mg\_orders

WHERE item.ItemID = mg\_orders.ItemID;

Напишіть запит SQL SELECT, використовуючи відповідну функцію дати та рядок формату, щоб знайти назву дня тижня, коли замовлення компанії M&G будуть доставлені.

SELECT DATE\_FORMAT(DeliveryDate,'%W') FROM mg\_orders;

Напишіть запит SQL SELECT, який обчислює вартість обробки кожного замовлення. Вона становить 5% від загальної вартості замовлення. Використовуйте відповідну математичну функцію, щоб округлити це значення до 2 знаків після коми.

SELECT OrderID, ROUND((Cost \* 5 / 100),2) AS HandlingCost FROM mg\_orders;

Перегляньте запит, який ви написали у другому завданні. Використовуйте відповідну функцію порівняння, щоб відфільтрувати записи, які не мають значення NULL у стовпчику дата доставки.

SELECT DATE\_FORMAT(DeliveryDate,'%W') FROM mg\_orders WHERE !ISNULL(DeliveryDate);

**Процедури**

**Що таке збережена процедура?**

Збережена процедура в MySQL - це набір інструкцій SQL, загорнутих в інструкцію CREATE PROCEDURE для досягнення певної мети.

Наприклад:

CREATE PROCEDURE GetItalianCustomers() SELECT \* FROM Customers WHERE Country = “Italy”;

Ця збережена процедура повертає всіх клієнтів з Італії, просто викликавши ідентифікатор процедури наступним чином:

CALL GetItalianCustomers();

**Навіщо використовувати збережені процедури?**

Основна ідея створення збережених процедур полягає у створенні багаторазового коду, який можна ефективно викликати і виконувати.

Таким чином, замість того, щоб вводити один і той самий код багаторазово, ви можете зберегти свої блоки коду в збереженій процедурі у вигляді готового запиту, який ви можете використовувати, коли це необхідно.

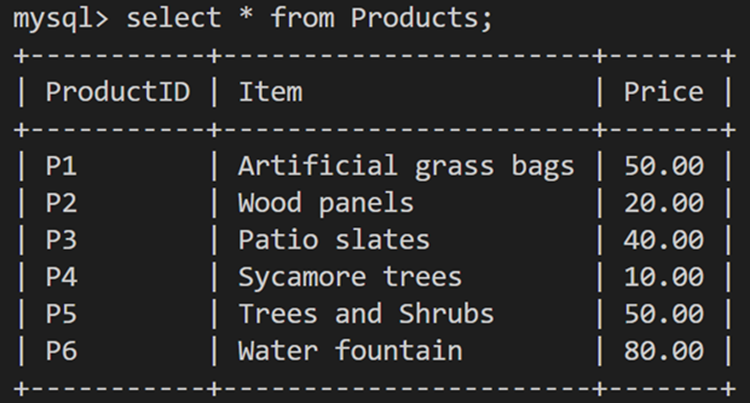
Ви можете створити стільки процедур, скільки вам потрібно. Кожна процедура може мати унікальне ім'я, нуль або кілька параметрів і містити блоки коду, включаючи оператори SQL, змінні та умови.

Збережені процедури роблять код більш послідовним, придатним для багаторазового використання і простішим у використанні та підтримці.

**Коли створювати збережену процедуру з параметрами?**

Кожна процедура повинна досягати чіткої мети. Наприклад, припустимо, ви очікуєте, що при виклику процедури користувачі будуть вводити значення. У такому випадку потрібно вказати параметр з відповідним типом даних. Інакше процедура буде створена з нульовим параметром.

Наприклад, таблиця Products з бази даних Lucky Shrub містить 3 стовпці: ідентифікатор товару, назва товару і ціна товару. Таблиця заповнюється відповідними даними, як показано нижче.



Ви можете створити базову збережену процедуру з порожніми (або нульовими) параметрами і одним єдиним SQL-оператором, набравши наступний запит.

CREATE PROCEDURE GetProductsBasedOnPrice (inputPrice INT)

Ця процедура повертає всі записи з таблиці Products. Не очікується, що процедурі буде передано жодного аргументу (значення, введеного користувачем) для її виконання. Саме тому вона не має параметра в круглих дужках.

В інших випадках вам може знадобитися створити збережену процедуру з одним або кількома параметрами для обробки запиту на основі значення, вказаного користувачем. Наприклад, ви можете створити збережену процедуру для отримання даних з таблиці products на основі введеної користувачем конкретної ціни. Для цього ви можете написати процедуру з одним параметром наступним чином:

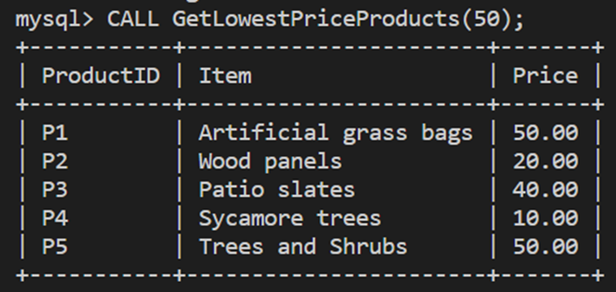
SELECT \* FROM Products WHERE Price <= inputPrice;

У цьому операторі ви оголошуєте параметр з типом даних INT, щоб передати ціле значення в збережену процедуру і змусити запит діяти на основі цього значення.

У цьому випадку ви можете повернути всі дані з таблиці Products, де ціна товару менша або дорівнює введеному користувачем значенню. Ось приклад коду:

CALL GetProductsBasedOnPrice (50);

Виводиться список усіх товарів, ціна яких менша або дорівнює $50, як показано на екрані нижче.



**Висновок**

Збережені процедури - це інструкції SQL-коду, інкапсульовані в інструкції CREATE PROCEDURE для досягнення певної мети. Ви виконуєте процедуру, викликаючи її ім'я і передаючи параметри (якщо вони є!). Їх легко підтримувати, можна повторно використовувати так часто, як потрібно, вони є послідовними і навіть допомагають захистити вашу базу даних, запобігаючи SQL-ін'єкціям.