МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра наноелектроніки та модифікації поверхні

«До захисту допущено»

Завідувач кафедри

_____ Олександр ПОГРЕБНЯК

_____ _____20___р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 153 «Мікро- та наносистемна техніка»,

освітньо-професійної програми «Нанотехнології та біомедичні системи»

на тему: «Проектування бази даних для системи управління складом та продажем товарів для роздрібної торгівлі»

Здобувачки групи ФЕ_91 Солонар Євгенії Сергіївни

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Євгенія СОЛОНАР

Керівник доцент кафедри наноелектроніки
та модифікації поверхні Ольга ЮЩЕНКО _____

Суми – 2023

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет ЕЛІТ Кафедра наноелектроніки та модифікації поверхні

Спеціальність 153 Мікро та наносистемна техніка

ЗАТВЕРДЖУЮ:

Зав. Кафедрою _____

_____

«____» _____20___р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

СОЛОНАР ЄВГЕНІЇ СЕРГІЇВНИ

1. Тема проекту (роботи) «Проектування бази даних для системи управління складом та продажем товарів для роздрібної торгівлі»

Затверджена наказом по університету від «_» _____20___р. №_____

2. Термін здачі студентом закінченого проекту (роботи) _____
3. Вхідні дані до проекту (роботи) база даних «Сирний Сомельє», програма DB Browser for SQL lite, мова програмування SQL, додаткова програма для створення скриншотів LightShot, товар представлений на сайті «Сирного Сомельє».
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) ER-діаграма, рисунок таблиці Cheese, рисунок таблиці Drink, рисунок запиту за сортуванням товару по таблиці Cheese, рисунок заповнення таблиці Акції.
6. Дата видачі завдання _____

Керівник _____

(підпис)

Завдання прийняв до виконання _____

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Інструктаж з техніки безпеки	1.05.2023	виконано
2	Робота з програмним забезпеченням	1.05.2023 – 10.05.2023	виконано
3	Дані та поняття баз даних	10.05.2023 – 13.05.2023	виконано
4	Поняття СУБД, модель баз даних	13.05.2023 – 15.05.2023	виконано
5	Типи БД. Реляційна модель	15.05.2023 – 17.05.2023	виконано
6	Нормалізація даних	17.05.2023 – 19.05.2023	виконано
7	Створення створення та заповнення таблиць даними в DB Browser for SQLite	19.05.2023 – 22.05.2023	виконано
8	Опис предметів функціональної області	22.05.2023 – 24.05.2023	виконано
9	Побудова ER-діаграма	24.05.2023 – 29.05.2023	виконано
10	Заповнення баз даних	29.05.2023 – 02.06.2023	виконано
11	Створення тригерів	02.06.2023 – 07.06.2023	виконано
12	Створення запитів до баз даних	07.06.2023 – 12.06.2023	виконано
13	Оформлення звіту	12.06.2023 – 15.06.2023	виконано

Студент-дипломник _____

(підпис)

Керівник проекту _____

(підпис)

АНОТАЦІЯ

Дана робота складається з реферативної доповіді, в якій викладається загальна інформація про існування баз даних, систем управління баз даних, мова програмування SQL, можливості для формування складних запитів і таблиць. Вступ має інформацію про програму, в якій виконані напрацювання для бази даних – Database Browser for SQLlite, представлена ER-діаграма, таблиці моделі роздрібнених магазинів та виконані запити в базі даних за допомогою мови програмування SQL, також наведені висновки та списки використаних джерел.

Мета роботи – створення складних і простих запитів для спрощення роботи в базі даних роздрібнених магазинів, щоб швидко та зручно отримувати потрібну інформацію.

Звіт містить 61 сторінку, 14 рисунків та 4 використаних джерел.

В цій роботі були використані бази даних роздрібнених магазинів «Сирний сомельє». Мережа магазинів містить дані про наявність товару, надає послуги з продажу голландських сирів та імпортного алкоголю. База даних магазину містить інформацію постійних клієнтів, накопичення знижок, прихід товару, яка потребує сортування та обробки.

КЛЮЧОВІ СЛОВА: БАЗА ДАНИХ, DATABASE BROWSER FOR SQLITE, ЗАПИТ, ER-ДІАГРАМА, МОВА SQL

ЗМІСТ

ВСТУП

В сучасному світі вся інформація все більше переноситься до цифрового світу. Майже всі підприємства зберігають свої дані в комп’ютерах і використовують різні бази даних для швидкого пошуку потрібної та важливої інформації.

В цій роботі ми розглянемо бази даних, їх типи, СУБД баз даних, можливі корисні запити для роздрібнених магазинів. Це стало необхідністю для промисловості, як спосіб більш ефективного використання ресурсів та охоплення більшої кількості потенційних клієнтів.

Бази даних використовуються майже скрізь - у банках, торгових точках і на веб-сайтах. Банки потребують баз даних, щоб відстежувати клієнтські рахунки, залишки та депозити. Роздрібні торговці використовують бази даних для введення інформації про ціни, персональні дані клієнтів, продажі та кількість товарів, доступних для продажу, а також інформацію про власників карток. Веб-сайти використовують їх для зберігання контенту, даних для входу, вподобань клієнтів, а також для зберігання даних користувачів. Бази даних використовуються там, де потрібне зберігання даних і швидкий пошук.

РОЗДІЛ 1 БАЗА ДАНИХ

База даних SQL або реляційна база даних — це набір структурованих таблиць, де кожен рядок відображає призначення даних, а кожен стовпець визначає певне інформаційне поле. Таким чином, SQL є базовою мовою програмування для всіх систем управління реляційними базами даних (RDBMS), таких як MySQL, Oracle і Sybase, та інш.

Сервер бази даних SQL зберігає та організовує дані в таблицях. У RDBMS таблиці є основними об'єктами бази даних, логічно розробленими для збору даних у форматі рядків і стовпців. Хоча рядки відображають сутності, стовпці визначають атрибути кожної сутності. Наприклад, у таблиці даних клієнта кожен рядок відображає запис для конкретного клієнта, а кожен стовпець таблиці містить відповідну інформацію про клієнта, як-от ім'я та адреса клієнта. Нижче наведено ключові елементи таблиці бази даних SQL:

Стовпці: кожен стовпець містить певну інформацію про атрибути, а властивості стовпця визначають тип даних (наприклад, числові або текстові дані) і діапазон, який він може приймати. Кожна таблиця має первинний ключ для унікальної ідентифікації сутності. Певний стовпець, наприклад, ідентифікатор клієнта в таблиці даних клієнта, може бути первинним ключем.

Рядки: користувачі бази даних можуть додавати дані до кожного рядка та виконувати SQL-запити для отримання даних. Для первинного ключа кожен рядок містить унікальне значення, яке також допомагає подолати проблеми дублювання даних.

Бази даних SQL служать основою для широкого спектру додатків і сервісів у різних галузях; бази даних SQL забезпечують широкий спектр обробки транзакцій, аналізу та бізнес-аналітики, необхідних для запуску критично важливих бізнес-додатків. Для полегшення операційних функцій компанії використовують сервери баз даних SQL для зберігання та отримання даних.

Реляційна база даних містить кілька таблиць з відповідними стовпчиками (атрибутами) і рядками (записами), а також унікальний первинний ключ. В час,

коли користувач робить запит, він може оновити або змінити дані в базі даних або отримати відповідні результати для конкретного запиту після перевірки обмежень.

Користувачі можуть використовувати бази даних SQL для отримання змістовної інформації, об'єднуючи різні таблиці, з метою краще зрозуміти контекст і взаємозв'язки даних SQL використовується для здійснення базових функцій управління даними і складних запитів для перетворення наявних необроблених даних в корисну і контекстну інформацію. Користувачі баз даних можуть використовувати стандартні мови SQL, такі як мова визначення даних (DDL) для створення структур баз даних і таблиць, а також мову маніпулювання даними (DML) для вставки, оновлення, видалення та вибору даних тощо.

В першому розділі ми дізнаємося о базах даних, їх існуючі типи, котрі програмні забезпечення краще використовувати, які є зв’язки між даними та що таке нормалізація даних.

1.1. Дані та поняття баз даних.

Насамперед дамо визначення, що таке інформація та дані.

Інформація - це дані, зібрані для того, щоб зробити значущі висновки відповідно до контекстуальних вимог. Інформацію обробляють, структурують, надають їй значення і представляють таким чином, щоб підвищити надійність отриманих даних. При цьому гарантується, що не залишиться жодної двозначності або небажаної інформації.

Дані - це сукупність деталей або інформації, яка залишається у вигляді діаграм, символів, описів або просто спостережень за об'єктом, подією чи річчю, або в письмовому вигляді, яку можна проаналізувати і зробити висновки. Це сирі дані, які необхідно відтворити, щоб отримати змістовну інформацію.

Бази даних призначені для зберігання наборів даних разом для якомога більшої кількості застосувань. Тому бази даних часто розглядають як сховища інформації, необхідної для виконання певних функцій у компанії чи організації. БД повинна бути сховищем даних, необхідних для обробки даних організації. Ці дані повинні бути точними, приватними і захищеними від пошкодження. Вони мають бути

коректними, щоб їх могли використовувати різні додатки з різними вимогами до даних. Різні прикладні програмісти та кінцеві користувачі мають різні погляди на дані, отримані зі спільної основної структури даних. Вони мають різні способи пошуку та доступу до даних.

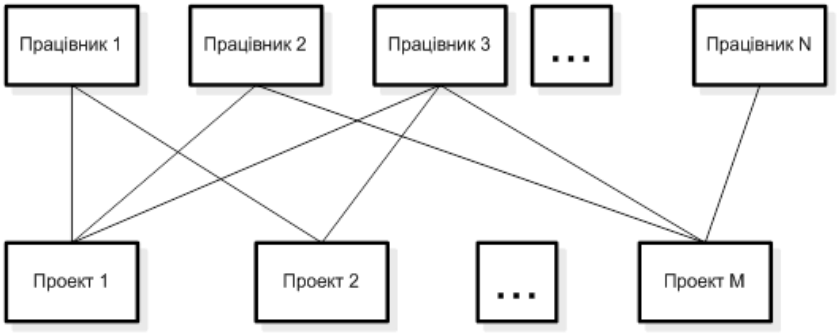

Рис.1.1. – Приклад спрощеної реляційної бази даних.

Переваги використання баз даних:

1) Бази даних можуть суттєво контролювати надмірність та неузгодженість даних;
2) Бази даних також можуть обмінюватися даними;
3) Бази даних забезпечують відповідність стандартам;
4) Бази даних можна використовувати для забезпечення безпеки і управління цілісністю даних.

Бази даних зазвичай реалізуються на трьох рівнях:

a) Внутрішній або фізичний рівень;
b) Концептуальний рівень;
c) Зовнішній або рівень представлення.

Типи даних SQL визначають види даних, які може містити стовпець або змінна в базі даних SQL. Ці типи даних включають числові, символьні, рядкові, дати і часу, двійкові, логічні, перераховані, масиви і JSON типи. Кожен тип даних має певний діапазон значень і способів використання. Вибір правильного типу даних для стовпців і змінних важливий для точного та ефективного зберігання і пошуку даних.

Зв'язки: Загалом, зв'язок - це таблиця, тобто дані, організовані в рядки і стовпці. Зв’язки мають наступні характеристики:

Всі елементи в стовпчику будь-якої таблиці можуть бути одного типу, в той час як елементи в різних стовпчиках можуть бути різних типів. Для рядків кожен стовпець повинен мати атомарне значення, а стовпець не може мати більше одного значення. Всі рядки відношення є різними. Порядок рядків у відношенні не має значення. товпці відношення мають різні імена і порядок цих стовпців не має значення.

Кортежі: рядки таблиці у відношенні часто називають кортежами.

Атрибути: стовпці та поля таблиці називаються атрибутами.

Ступінь: кількість атрибутів у відношенні визначає ступінь відношення; відношення з трьома атрибутами називається відношенням ступеня 3.

Суттєвість: кількість кортежів або рядків у відношенні називається суттєвістю.

1.2. Поняття СУБД, модель баз даних.

Модель бази даних представляє логічну структуру бази даних, включаючи взаємозв'язки та обмеження, які визначають, як зберігаються дані та доступ до них. Індивідуальні моделі баз даних розробляються на основі загальних правил і концепцій моделей даних, прийнятих розробником. Більшість моделей даних можна представити у вигляді супровідної схеми бази даних.

Модель даних - це абстрактне представлення системи баз даних. Вона використовується для проектування та реалізації бази даних або для опису схеми (структури та організації, в якій фізично зберігаються дані). Моделі даних створюються за допомогою техніки, відомої як концептуальне моделювання. Однак більшість моделей даних ґрунтуються принаймні на одній формальній моделі, наприклад, на моделюванні сутностей або зв'язків.

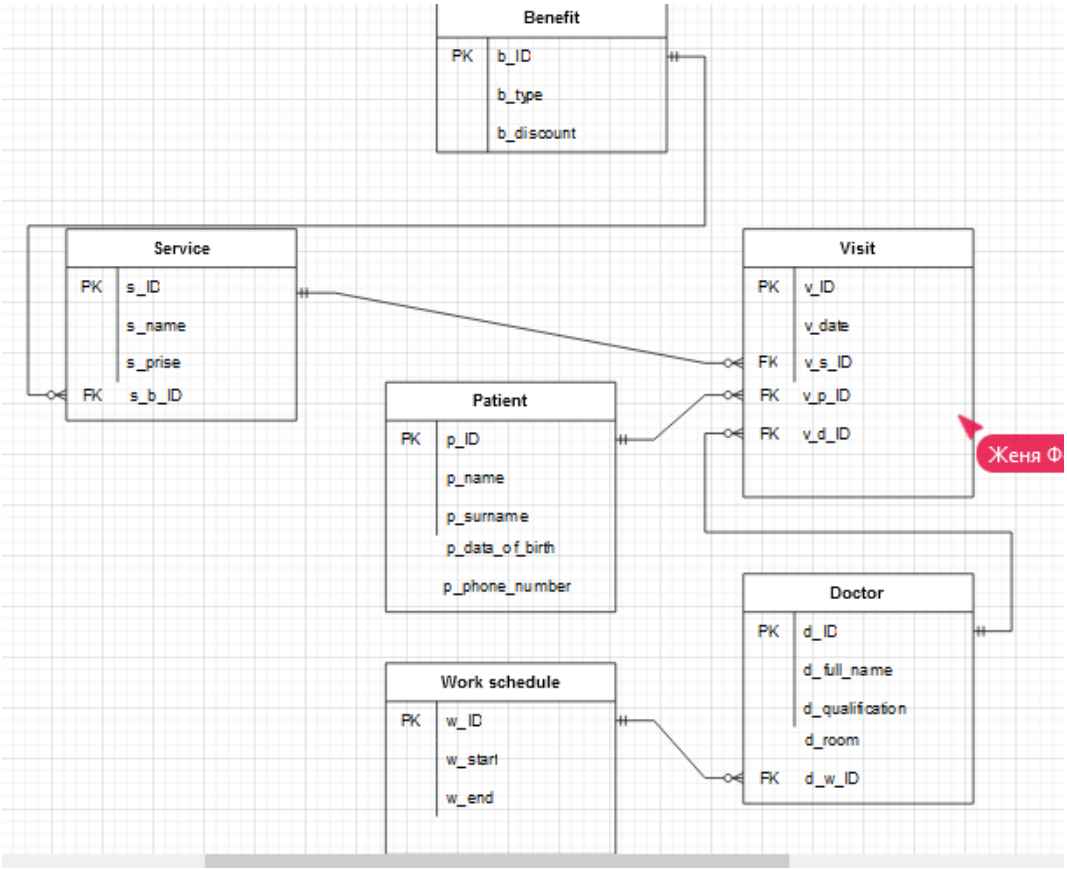

Рис.1.2. – Модель бази даних.

Існує багато типів моделей даних. Типові приклади включають:

- Ієрархічні моделі баз даних;
- Реляційні моделі;
- Мережеві моделі;
- Об'єктно-орієнтована модель бази даних;
- Модель "сутність-зв'язок;
- Модель документів;
- Модель "атрибут-значення;
- Зіркова діаграма;
- Модель "об'єкт-зв'язок.

Різні фактори можуть визначити базу даних як будь-яку з цих моделей. Найважливішим фактором є те, чи підтримує використовувана система керування базами даних певну модель. Більшість систем керування базами даних побудовані з урахуванням певної моделі даних і вимагають від користувачів прийняття цієї моделі, але деякі з них підтримують декілька моделей.

Крім того, різні моделі застосовуються на різних етапах процесу проектування бази даних. Високорівневі концептуальні моделі даних найкраще підходять для відображення взаємозв'язків між даними так, як їх сприймають люди. Логічні моделі на основі записів, з іншого боку, більш точно відображають спосіб зберігання даних на серверах. Вибір моделі даних також повинен відповідати сильним сторонам конкретної моделі та пріоритетам бази даних, таким як швидкість, зниження витрат і доступність. Давайте розглянемо найпоширеніші моделі баз даних.

Реляційна модель.

Найпоширеніша модель, реляційна модель, класифікує дані в таблиці (також відомі як відношення), кожна з яких складається зі стовпців і рядків. Кожен стовпець містить атрибути відповідної сутності, такі як ціна, поштовий індекс, дата народження тощо. Атрибути відношення в сукупності називаються доменами. Певний атрибут або комбінація атрибутів вибирається як первинний ключ, на який можна посилатися в інших таблицях, коли він називається зовнішнім ключем.

Кожен рядок, який також називається кортежем, містить дані про конкретний екземпляр відповідної сутності (наприклад, конкретного працівника). Модель також враховує тип зв'язку між цими таблицями (один-до-одного, один-до-багатьох, багато-до-багатьох).

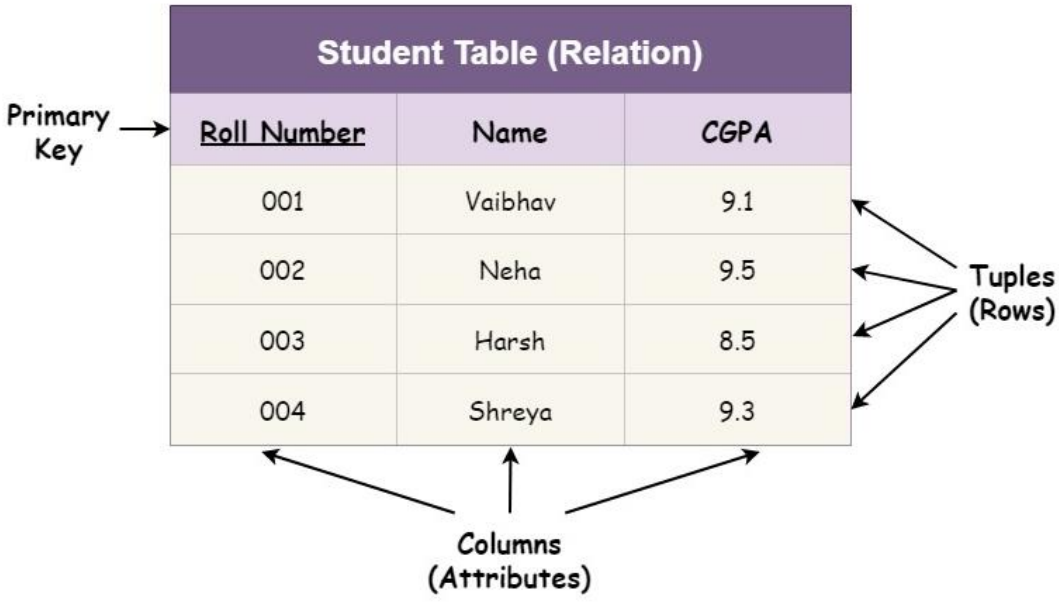

Рис.1.3. Приклад реляційної моделі бази даних.

У базах даних таблиці можуть бути нормалізовані або приведені у відповідність до правил нормалізації, щоб база даних стала гнучкою, адаптивною та масштабованою. Після нормалізації кожен фрагмент даних розбивається на атомарні або найменші корисні частини. Реляційні бази даних зазвичай пишуться мовою структурованих запитів (SQL). Ця модель була опублікована Е.Ф. Коддом у 1970 році.

Ієрархічні моделі.

Ієрархічні моделі організовують дані у вигляді деревовидної структури, де кожен запис має єдиного батька або корінь. Однорідні записи сортуються в певному порядку. Цей порядок використовується як фізичний порядок, в якому зберігається база даних. Ця модель підходить для опису багатьох реальних відносин. Ця модель в основному використовувалася в системах управління інформацією IBM в 60-х і 70-х роках, але зараз не часто зустрічається, частково через операційну неефективність.

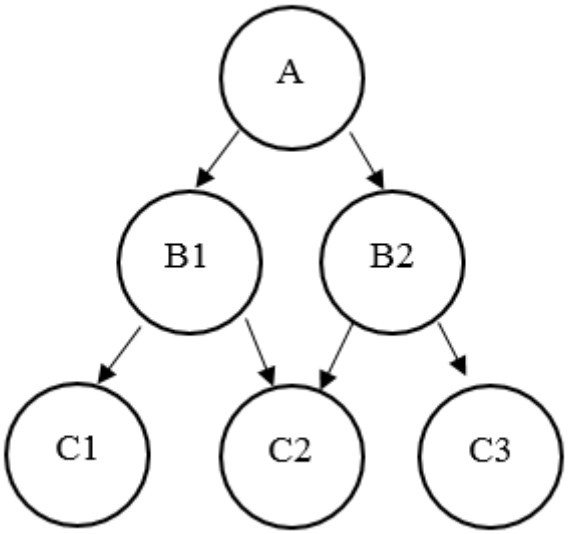

Рис.1.4. –Приклад ієрархічної моделі бази даних.

Мережева модель.

Мережева модель базується на ієрархічній моделі і допускає зв'язки "багато-до-багатьох" між пов'язаними записами, що передбачає існування більш ніж одного головного запису. Заснована на математичній теорії множин, модель побудована

на кластерах пов'язаних записів. Кожен кластер складається з власника або батьківського запису та одного або більше записів-членів або дочірніх записів; оскільки запис може бути членом або дочірнім записом більш ніж одного кластера, модель може передавати складні зв'язки. Найбільшої популярності вона набула в 1970-х роках після того, як була офіційно визначена на Конференції з мов систем даних (CODASYL).

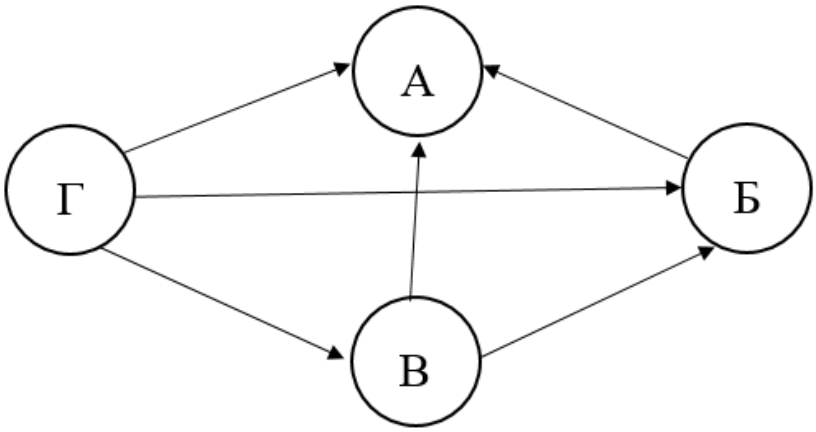

Рис. 1.5. – Приклад мережевої моделі баз даних.

Об'єктно-орієнтована модель.

Ця модель визначає базу даних як сукупність об'єктів або програмних елементів багаторазового використання з відповідними функціями та методами. Існують різні типи об'єктно-орієнтованих баз даних:

- Мультимедійні бази даних містять мультимедійні файли, такі як зображення, які не можуть зберігатися в реляційних базах даних.
- Гіпертекстові бази даних дозволяють пов'язати будь-який об'єкт з будь-яким іншим об'єктом. Це корисно для організації великих обсягів різних даних, але не підходить для числового аналізу.
- Об'єктно-орієнтовані моделі баз даних найбільш відомі як пост-реляційні моделі баз даних, оскільки вони включають таблиці, але не обмежуються ними. Такі моделі також називають гібридними моделями баз даних.

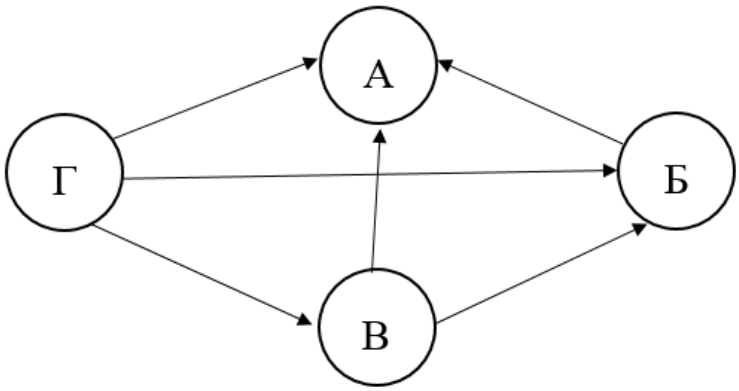

Рис. 1.6. Приклад об'єктно-орієнтованої моделі бази даних.

Об'єктно-реляційна модель.

Даний тип гібридної моделі бази даних поєднує в собі простоту реляційної моделі з деякими розширеними можливостями об'єктно-орієнтованої моделі бази даних. По суті, вона дозволяє дизайнерам вбудовувати об'єкти у звичні структури таблиць. Мови та інтерфейси виклику SQL3 включає мови постачальників, ODBC, JDBC і спеціалізовані інтерфейси виклику, які є розширеними версіями мов та інтерфейсів, що використовуються в реляційній моделі.

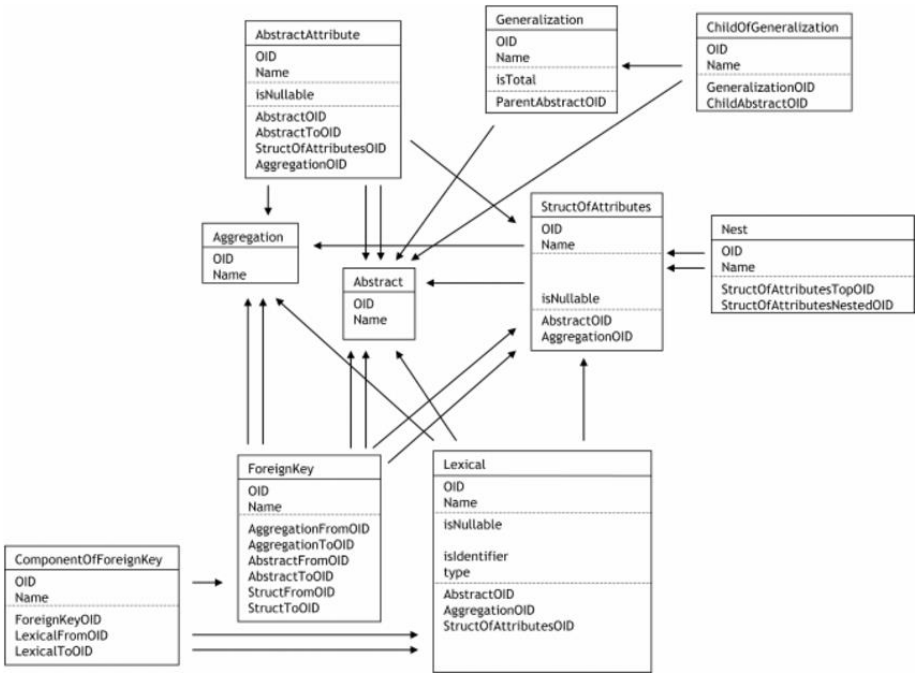

Рис. 1.7. - Приклад об'єктно-реляційної моделі бази даних.

Модель сутність-зв'язок.

Представлена модель відображає зв'язки між об'єктами реального світу так само, як і мережева модель, але не пов'язана безпосередньо з фізичною структурою бази даних. Натомість її часто використовують для концептуального проектування бази даних. Тут люди, місця та об'єкти, де зберігаються точки даних, називаються сутностями, кожна з яких має певні атрибути, які разом утворюють домен. Також визначаються суттєвість і зв'язки між сутностями.

Поширеною формою ER-діаграми є зіркоподібна діаграма, в якій центральна таблиця фактів з'єднана з кількома таблицями вимірів.

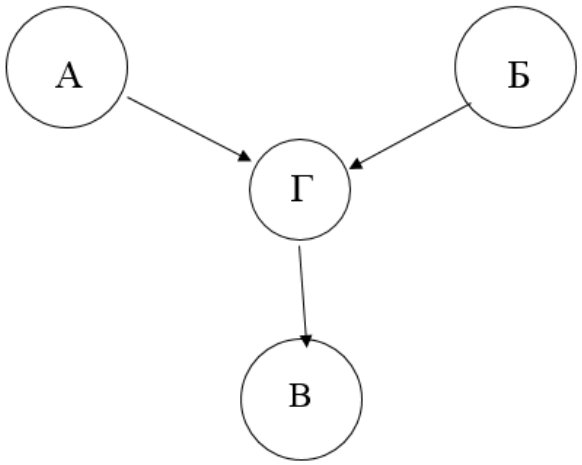

Рис. 1.8. – Приклад моделі бази даних сутність-зв’язок.

Системи управління базами даних (СУБД) організовують дані таким чином, щоб ви могли швидко та легко знайти потрібну інформацію, навіть якщо самі дані не розташовані в певному порядку. Ці системи є незамінними інструментами в сучасному технологічному суспільстві, оскільки вони дозволяють як великим, так і малим компаніям зберігати дані та отримувати їх за потреби.

Системи управління базами даних (СУБД) - це програмне забезпечення, яке надає можливість створювати великі обсяги даних, керувати ними та отримувати до них доступ. Ці програми допомагають автоматизувати такі процеси, як додавання нових записів, модифікація існуючих записів і видалення записів, коли це необхідно.

Системи управління базами даних також допомагають користувачам знаходити інформацію швидше та ефективніше. Користувачам більше не потрібно

витрачати час на перегляд сотень документів один за одним, вони можуть знайти те, що шукають, лише за кілька кліків.

Розглянемо як система управління базами даних працює.

Основна функція системи управління базами даних - надавати користувачам доступ до збережених даних. Тому СУБД повинна дозволяти користувачам додавати нову інформацію, змінювати існуючу інформацію та видаляти старі дані. Крім того, система повинна гарантувати, що тільки авторизовані користувачі можуть отримати доступ до будь-якої інформації; для того, щоб СУБД виконувала ці функції, вона повинна бути налаштована і організована таким чином, щоб користувачі могли отримувати певні типи даних на основі певних критеріїв. Наприклад, якщо користувач хоче отримати всі рахунки, пов'язані з клієнтом, який проживає в Нью-Йорку, він може просто ввести "New York" в поле пошуку, і всі відповідні записи будуть повернуті з таблиці. Після того, як базу даних організовано відповідно до конкретних потреб організації, розробники можуть починати створювати на її основі додатки.

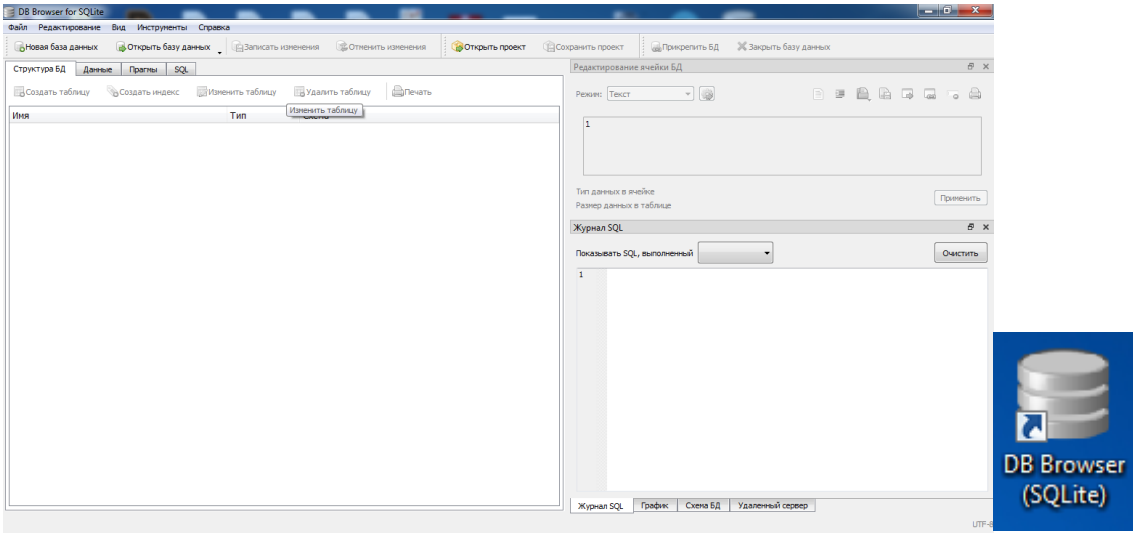

Рис. 1.9. – Простий приклад системи управління бази даних.

Основні концепції та функції СУБД включають моделі даних, мови запитів, організацію файлів та індексів, нормалізацію, ключі-кандидати та ключові поля.

1.3. Типи БД. Реляційна модель даних.

У більшості випадків ви побачите, що для різних завдань потрібні різні типи баз даних. Нижче наведено деякі поширені типи баз даних:

- Централізовані бази даних;
- Хмарні бази даних;
- Комерційні бази даних;
- Розподілені бази даних;
- Бази даних кінцевих користувачів;
- Графічні бази даних;
- NoSQL бази даних;
- Об'єктно-орієнтовані бази даних;
- Бази даних з відкритим кодом;
- Оперативні бази даних;
- Персональні бази даних;
- Реляційні бази даних.

Централізовані бази даних.

Централізована база даних - це база даних, яка працює повністю з одного місця. Централізовані бази даних часто використовуються у великих організаціях, таких як компанії та університети. Сама база даних знаходиться на центральному комп'ютері або системі баз даних. Користувачі можуть отримати доступ до бази даних через комп'ютерну мережу, але саме центральний комп'ютер запускає і підтримує базу даних.

Хмарні бази даних.

Хмарна база даних - це база даних, яка працює через Інтернет. Дані зберігаються на локальному жорсткому диску або сервері, але інформація доступна онлайн. Це означає, що до файлів можна легко отримати доступ з будь-якого місця, де є інтернет-з'єднання. Щоб користуватися хмарною базою даних, користувачі повинні або створити власну базу даних, або оплатити послугу зі зберігання своїх даних. Шифрування є важливим для хмарних баз даних, оскільки вся інформація, що передається онлайн, повинна бути захищена.

Комерційні бази даних.

Комерційні бази даних - це бази даних, розроблені комерційними підприємствами. Компанія розробляє багатофункціональну базу даних і продає її своїм клієнтам. Комерційні бази даних відрізняються за своєю структурою та технологією, яку вони використовують. На відміну від баз даних з відкритим вихідним кодом, вони характеризуються тим, що користувачі платять за їх використання.

Розподілені бази даних.

Розподілена база даних - це база даних, яка розподілена між декількома пристроями. Замість того, щоб зберігати всю інформацію на одному пристрої, як у випадку з іншими базами даних у цьому списку, розподілена база даних працює на декількох машинах, наприклад, на різних комп'ютерах в одному місці або в мережі. Перевагами розподіленої бази даних є вища швидкість, більша надійність і легкість розширення.

Бази даних для кінцевих користувачів.

Кінцевий користувач - це термін, який використовується при розробці продукту для позначення людей, які використовують продукт. Іншими словами, база даних кінцевого користувача - це база даних, яка в основному використовується однією людиною. Хорошим прикладом такого типу бази даних є електронна таблиця, що зберігається на локальному комп'ютері.

Графічні бази даних.

Графічні бази даних - це бази даних, які в рівній мірі зосереджені як на даних, так і на зв'язках між ними. У цій базі даних дані не обмежуються заздалегідь визначеним шаблоном. У більшості інших баз даних зв'язки між даними можна знайти під час пошуку. У графічній базі даних ці зв'язки зберігаються в базі даних разом з вихідними даними. Це робить бази даних ефективнішими та швидшими, якщо основною метою є управління зв'язками між даними.

Бази даних NoSQL.

NoSQL бази даних мають ієрархічну структуру, подібну до системи файлових папок, а дані в них не є структурованими або реляційними. Відсутність структури означає, що більші обсяги даних можуть оброблятися швидше і можуть

бути легко розширені в майбутньому. Хмарні обчислення регулярно використовують бази даних NoSQL.

Об'єктно-орієнтовані бази даних.

Об'єктно-орієнтована база даних - це база даних, в якій дані представлені у вигляді об'єктів або класів. Об'єкти - це елементи, такі як імена або телефонні номери, а класи - це групи об'єктів. Об'єктно-орієнтовані бази даних є різновидом реляційних баз даних. Якщо вам потрібно швидко обробляти великі обсяги складних даних, розгляньте можливість використання об'єктно-орієнтованої бази даних.

Бази даних з відкритим кодом.

Бази даних з відкритим вихідним кодом призначені для безкоштовного використання. На відміну від комерційних баз даних, користувачі можуть завантажувати та зберігати бази даних з відкритим кодом без оплати. Під "відкритим кодом" маються на увазі програми, в яких користувачі можуть бачити, як вони були написані і створені, і можуть вільно вносити зміни в програму. Бази даних з відкритим вихідним кодом зазвичай набагато дешевші за комерційні бази даних, але їм може бракувати деяких більш розширених функцій комерційних баз даних.

Оперативні бази даних.

Мета оперативної бази даних - дозволити користувачам вносити зміни в дані в режимі реального часу. Оперативні бази даних необхідні для бізнес-аналітики та зберігання даних. Залежно від потреб, вони можуть бути налаштовані як реляційні бази даних або NoSQL. Традиційні бази даних базуються на пакетній обробці, де команди виконуються групами. Оперативні бази даних, з іншого боку, дозволяють додавати, редагувати або видаляти дані в будь-який час.

Персональні бази даних.

Персональні бази даних - це бази даних, призначені для однієї людини. Зазвичай вони зберігаються на персональному комп'ютері і мають дуже просту структуру, що складається з декількох таблиць. Персональні бази даних, як

правило, не підходять для складних транзакцій, великих обсягів даних або бізнес-транзакцій.

Реляційні бази даних.

Це інший основний тип баз даних, який є протилежністю NoSQL. У реляційних базах даних інформація зберігається у структурованому вигляді відносно інших даних. Типовим прикладом реляційної бази даних є зв'язок між онлайн-покупцями та їхніми кошиками. Реляційним базам даних часто надають перевагу там, де важлива цілісність даних або де масштабованість не є особливо важливою.

Розглянемо реляційну модель бази даних на прикладі малого підприємства.

Ось простий приклад двох таблиць, які мале підприємство може використовувати для обробки замовлень на свою продукцію. Перша таблиця - це таблиця інформації про клієнта, де кожен запис містить ім'я клієнта, адресу, адресу доставки, адресу для виставлення рахунку, номер телефону та іншу контактну інформацію. Кожна частина інформації (кожен атрибут) знаходиться в окремому стовпчику, а кожному рядку база даних присвоює унікальний ідентифікатор (ключ). У другій таблиці, таблиці "Замовлення клієнтів", кожен запис містить ідентифікатор клієнта-замовника, замовлену продукцію, кількість, розмір, колір тощо, але не містить імені клієнта та його контактної інформації.

Єдине, що об'єднує ці дві таблиці, - це стовпчик ID (ключовий). Однак саме цей спільний стовпець дозволяє реляційній базі даних створити зв'язок між двома таблицями. Коли програма обробки замовлень компанії надсилає замовлення до бази даних, база даних може звернутися до таблиці замовлень клієнтів, отримати правильну інформацію про замовлення товару і використати ідентифікатор клієнта в цій таблиці для пошуку інформації про виставлення рахунків і доставку в таблиці інформації про клієнтів. Таким чином, склад отримує потрібний товар, клієнт отримує замовлення вчасно, а компанія отримує оплату.

Рис. 1.10. – Приклад реляційної бази даних.

Переваги реляційних систем керування базами даних.

Проста, але потужна реляційна модель використовується організаціями всіх типів і розмірів для задоволення широкого спектру інформаційних потреб. Реляційні бази даних використовуються для управління запасами, здійснення транзакцій електронної комерції, управління великими обсягами критично важливої інформації про клієнтів та багато іншого. Реляційні бази даних можуть задовольнити будь-які інформаційні потреби, де точки даних взаємопов'язані і потребують безпечного, заснованого на правилах і послідовного управління. Реляційні бази даних існують з 1970-х років. Сьогодні переваги реляційної моделі продовжують робити її найпоширенішою моделлю баз даних.

Реляційні бази даних мають справу з бізнес-правилами та політиками на дуже детальному рівні і мають суворі політики щодо зобов'язань (постійних змін у базі даних). Наприклад, розглянемо базу даних інвентаризації, в якій записано три частини, що завжди використовуються разом: коли одна частина вилучається з інвентаризації, дві інші також повинні бути вилучені; якщо одна з трьох частин недоступна, жодна з частин не повинна бути вилучена - якщо база даних всі три частини повинні бути доступними до того, як будуть зроблені будь-які зобов'язання. Реляційна база даних не зафіксує частину, не знаючи, що вона може зафіксувати всі три частини. Ця універсальна можливість фіксації називається атомарністю. **Атомарність** - це ключ до збереження точності даних в базі даних і забезпечення відповідності бізнес-правилам, нормам і політикам.

1.4. Нормалізація даних.

Нормалізація - це процес усунення надлишкових даних і забезпечення того, щоб зв'язки між різними записами в базі даних мали сенс. Нормалізація даних у базі даних вимагає, щоб пов'язані таблиці були розділені на кілька таблиць відповідно до бізнес-правил. Розділення пов'язаних таблиць забезпечує більш ефективне зберігання даних і узгодженість між декількома таблицями. Таке розділення також робить оновлення простішим і надійнішим. Нормалізація є важливою частиною проектування схеми бази даних, оскільки вона допомагає запобігти проблемам, яких можна було б уникнути, таким як надмірність даних і пошкодження бази даних. Це також полегшує оновлення бази даних і підвищує продуктивність запитів. Нормалізація бази даних або нормалізація SQL допомагає згрупувати пов'язані дані в одну таблицю. Атрибутивні та опосередковано пов'язані дані розміщуються в окремих таблицях, які пов'язані між собою логічними зв'язками між батьківською та дочірньою таблицями.

Перша нормальна форма (1НФ)

Вважається, що таблиця має першу нормальну форму, якщо її атомарність дорівнює 1. Тут атомарність означає, що комірка не може містити більше одного значення; комірка повинна містити лише один атрибут. У першій нормальній формі заборонено використовувати багатозначні атрибути, складені атрибути та комбінації атрибутів.

№	Атрибут	Пояснення
1	Акциз	Акцизна марка алкоголю
2	Молодий	Сир витриманий 2 місяці
3	Вартість	Вартість товару

Рис. 1.11. – Приклад таблиці за 1-ю нормальною формою.

У таблиці записів товарів видно, що стовпець COURSE має два значення. Отже, він не відповідає першій нормальній формі. Тепер, якщо ми використаємо першу нормальну форму для наведеної вище таблиці, ми отримаємо наступну таблицю.

За допомогою першої нормальної форми досягається атомарність і кожен стовпець має унікальне значення. Перш ніж перейти до другої нормальної форми, дізнайтеся про ключі-кандидати та батьківські ключі. **Ключ-кандидат** - це набір з одного або декількох стовпців, який однозначно ідентифікує запис у таблиці, кожен ключ-кандидат може бути використаний як первинний ключ. **Батьківський ключ** - це єдиний набір ключів, який однозначно ідентифікує запис у таблиці, тоді як первинний ключ є підмножиною батьківського ключа.

Друга нормальна форма (2НФ).

Першою умовою для того, щоб таблиця перебувала у другій нормальній формі, є те, що таблиця перебуває у першій нормальній формі. Таблиця не повинна мати часткових залежностей. Тут часткова залежність означає, що відповідна підмножина ключів-кандидатів повинна створювати атрибут, який не є головним. Вона вимагає, щоб дані, які зберігаються в таблицях зі складеними ключами, не залежали тільки від частини ключа. Схема бази даних повинна відповідати всім вимогам другої нормальної форми. Дані, які повторюються в деяких рядках, розміщуються в окремій таблиці.

Третя нормальна форма (3НФ).

Першою умовою для того, щоб таблиця була у третій нормальній формі, є те, щоб вона була у другій нормальній формі. Це означає, що непервинний атрибут (який не є частиною ключа-кандидата) не повинен залежати від іншого непервинного атрибута в таблиці. Таким чином, транзитивні залежності - це функціональні залежності, де вираз A → C (де A визначає C) визначається опосередковано через A → B і B → C (де B → A не визначається). Третя поширена форма - та, що зменшує дублювання даних. Вона також використовується для забезпечення цілісності даних.

Нормальна форма Бойса-Кодда (BCNF).

Нормальна форма Бойса-Кодда, також відома як 3.5NF, є вдосконаленням 3NF, розробленим Реймондом Ф. Бойсом та Едгаром Ф. Коддом для вирішення деяких аномалій, які 3NF не могла вирішити. Першою умовою для того, щоб таблиця була в нормальній формі Бойса-Кодда, є те, що вона має третю нормальну

форму. По-друге, кожен атрибут правої частини (RHS) функціональної залежності повинен залежати від батьківського ключа відповідної таблиці.

РОЗДІЛ 2 СТВОРЕННЯ ER-ДІАГРАМИ. СТВОРЕННЯ ТА ЗАПОНЕННЯ ТАБЛИЦЬ ДАНИМИ В DB BROWSER FOR SQL LITE

2.1 Опис предметів функціональної області.

Для проектування бази даних мережі магазинів, де продають сир та вино різних виробників і діє накопичувальна система знижок, можуть включати:

- Магазин
- Виробник
- Продукт
- Замовлення
- Деталі замовлення
- Клієнт
- Знижка

Ці сутності представляють основні складові системи мережі магазинів, де кожна з них має свої атрибути, що визначають їх властивості та взаємозв'язки між сутностями.

Розглянемо окремо ці сутності та їх взаємозв'язки:

1) Сутність "Магазин":
   - shop_id (ідентифікатор магазину);
   - назва (назва магазину);
   - адреса (адреса магазину);
   - контактні дані (телефон, електронна пошта тощо);

2) Сутність "Виробник":
   - producer_id (ідентифікатор виробника);
   - назва (назва виробника);
   - країна (країна походження виробника);
   - контактні дані (телефон, електронна пошта тощо);

3) Сутність "Продукт":
   - product_id (ідентифікатор продукту);
   - назва (назва продукту);
   - тип (сир або вино);
   - ціна (ціна продукту);
   - кількість на складі

4) Сутність "Замовлення":
   - order_id (ідентифікатор замовлення);
   - дата (дата замовлення);
   - загальна сума (загальна сума замовлення);
   - статус (статус замовлення, наприклад, "в обробці", "виконано" тощо);
   - shop_id (ідентифікатор магазину, з якого було зроблено замовлення);

5) Сутність "Деталі замовлення":
   - order_id (ідентифікатор замовлення);
   - product_id (ідентифікатор продукту);
   - кількість (кількість продукту у замовленні);
   - знижка (знижка на продукт у замовленні);

6) Сутність "Клієнт":
   - customer_id (ідентифікатор клієнта);
   - ім'я (ім'я клієнта);
   - прізвище (прізвище клієнта);
   - адреса (адреса клієнта);
   - контактні дані (телефон, електронна пошта тощо);

Останню сутність опишемо більш детально:

7) Сутність "Знижка" в базі даних мережі магазинів відображає інформацію про наявні знижки, які можуть застосовуватися до продуктів у замовленнях або покупках клієнтів. Описуючи цю сутність, можна включити наступні атрибути:

- discount_id (ідентифікатор знижки): унікальне значення, що ідентифікує конкретну знижку.
- назва (name): назва або опис знижки для ідентифікації та відображення клієнтам.
- тип (type): тип знижки, наприклад, відсоток, фіксована сума або акційна пропозиція.
- значення (value): значення знижки, яке може бути відсотком, фіксованою сумою або конкретною акційною пропозицією.
- діапазон дат (date_range): діапазон дат, протягом якого знижка діє або доступна для використання.
- умови (conditions): умови, за яких знижка може застосовуватися, наприклад, мінімальна сума покупки або конкретні категорії продуктів.
- активна (active): прапорець, що вказує, чи є знижка активною в даний момент часу.
- shop_id (ідентифікатор магазину): зв'язок з магазином, до якого відноситься знижка (якщо знижка є специфічною для певного магазину).

Ці атрибути дозволять зберігати та управляти інформацією про знижки, дозволяючи магазинам ефективно застосовувати їх до продуктів та замовлень, а також надавати відповідні знижки клієнтам у залежності від умов та правил, встановлених для кожної знижки.

2.2. ER-діаграма

Розберемо, що таке ER-діаграма.

ER-діаграма (Entity Relationship) - це тип блок-схеми, яка показує, як "сутності", такі як люди, об'єкти та концепції, пов'язані між собою в системі.

Ось приблизний приклад ER-діаграми для цієї бази даних, побудований за допомогою сайту draw.io.

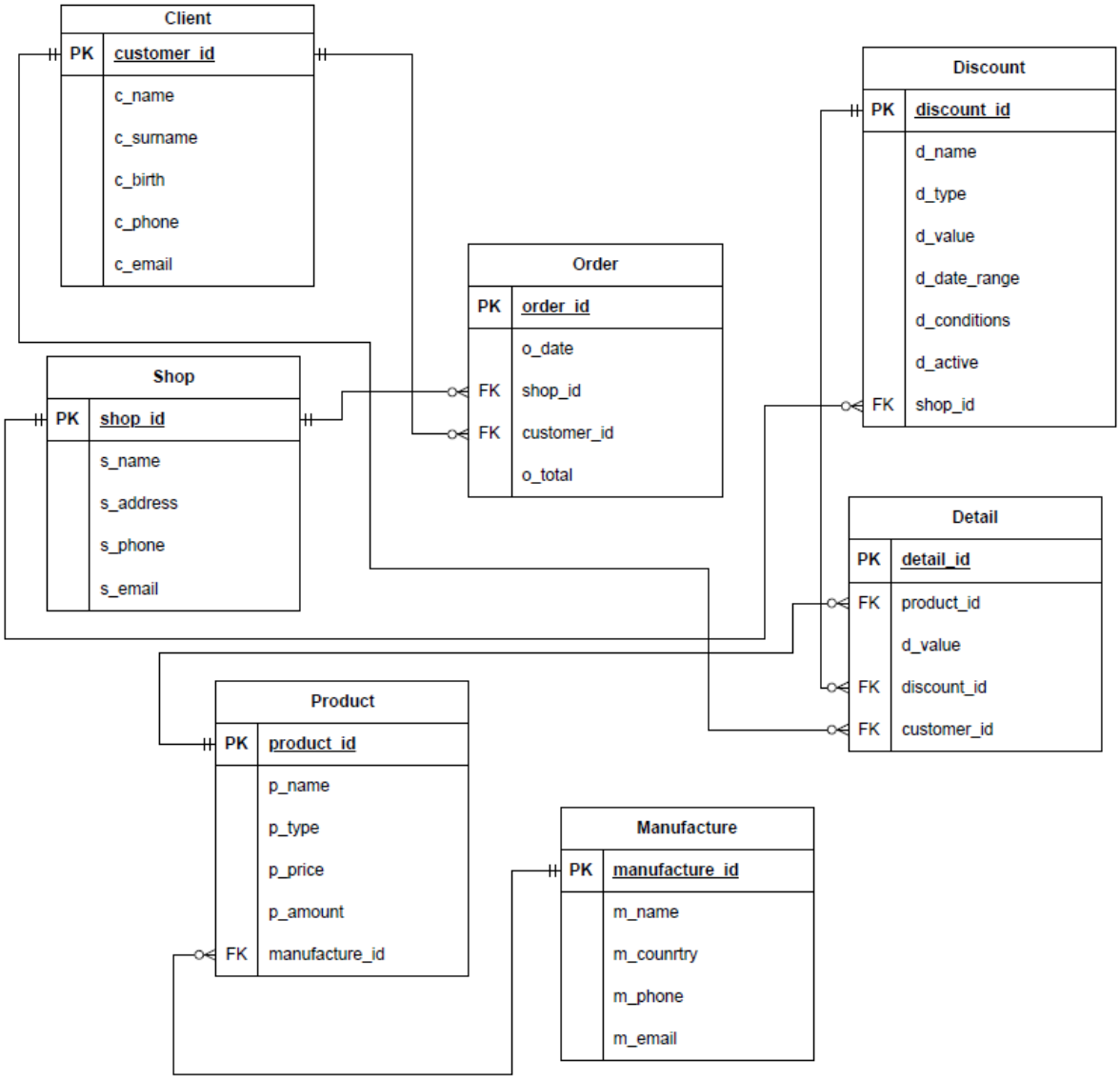

Рис. 2.1. – ER-діаграма бази даних

У цій ER-діаграмі кожна сутність представлена прямокутником, а зв'язки між ними показані за допомогою ліній та стрілок. Наприклад, існує зв'язок "магазин має продукти", представлений стрілкою від сутності "Магазин" до "Продукт". Атрибути кожної сутності вказані всередині прямокутника.

Ця ER-діаграма надає загальне уявлення про структуру бази даних та взаємозв'язки між сутностями, але може бути додатково розширена або налаштована відповідно до вимог конкретного проекту.

2.3. Заповнення бази даних

Таблиці бази даних створювалися у програмі DBbrowser for SQLite (Рис.2.2).

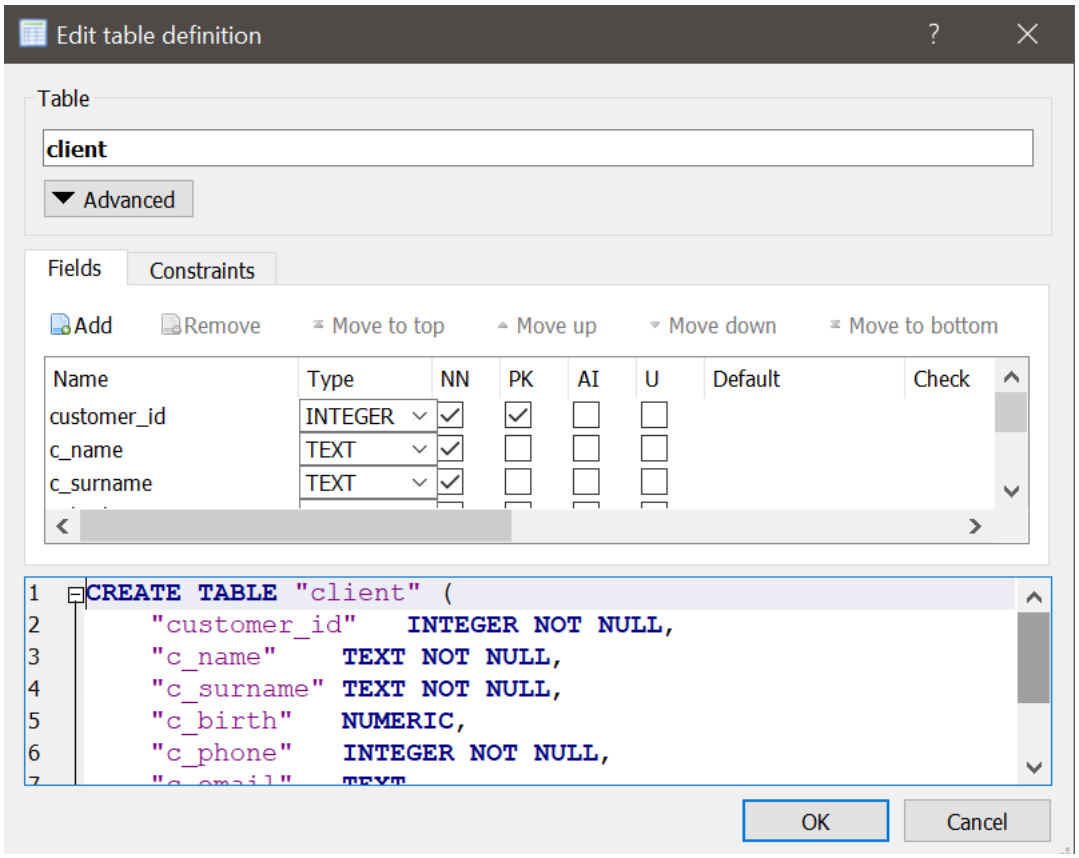

Рис.2.2. – Приклад створення таблиці «Клієнт»

Для заповнення бази даних були використані реальні найменування товарів для магазину «Сирний сомельє».

Для сиру за виробниками використовувалися наступні дані:

Holand cheese

- Young Chevret goat cheese
- Cheese Lutewinkel Royal
- Maasdam young cheese
- Old Amsterdam cheese 48%
- Mouton young sheep's cheese
- Pittoresque cheese with walnut
- Basiron Tricolor cheese
- Gouda cheese VI Old Rotterdam 36 weeks 48% fat.

Norway

- Gudbrandsdalen cheese by weight

Italy

- Parmesan

Turkey

- Sliced ham 100g

Germany

- Brie cheese 200 g COBURGER
- Dor Blue Grand Noir cheese (black) weight

Denmark

- Cream cheese PINEAPPLE 125 gr

France

- Cheese EXCELLENCE weight .Fromager d'Affinois
- Fourme d'Amber cheese 50% by weight

Для вина за виробниками використовувалися наступні дані:

Italy

- Sparkling white wine Pinot Grigio Brut, Canti 0.75 l
- Sparkling wine extra-dry white Prosecco Spumante Santero (Twist), Santero 0.75l

- Sparkling sweet white wine Asti DOCG Chianti Liberty, Canti 0.75 l
- Semi-sparkling sweet white wine Lambrusco del Emilia, Chiarli 0.75 l

Spain

- Brut white sparkling wine Cava Jaume Serra Brut, J.Garcia Carrion 0.75l
- Semi-sweet white non-alcoholic wine Muscat Natureo, Torres 0.75 l
- Semi-dry red wine Shiraz, Pete's Pure 0.75 l

Portugal

- Fortified wine port wine pink Pink Port, Offley 0.75 l
- Fortified red port wine Sandeman Tawny 10 years, Sogrape Vinhos 0.75l

France

- Semi-sweet white wine Muscat 0.75 l Baron d'Arignac

Australia

- Semi-dry red wine Shiraz, Pete's Pure 0.75 l

New Zealand

- Dry white wine Sauvignon Blanc Marlboro San, Saint Clair 0.75 l

Інші дані заповнювалися за допомогою генерації даних. Наприклад, адреси та контактні дані десяти магазинів були згенеровані наступним чином:

1) Магазин: Сирний сомельє
вул. Харківська, 10
Телефон: +38067-123-4567
Електронна пошта: sirsom_sumi1@example.com

2) Магазин: Сирний сомельє
вул. Шевченка, 25
Телефон: +38050-987-6543
Електронна пошта: sirsom_sumi2@example.com

3) Магазин: Сирний сомельє

просп. Незалежності, 42

Телефон: +38063-555-8888

Електронна пошта: sirsom_sumi3@example.com

4) Магазин: Сирний сомельє

просп. Курортний, 14

Телефон: +38066-111-2222

Електронна пошта: sirsom_sumi4@example.com

5) Магазин: Сирний сомельє

вул. Петропавлівська, 31

Телефон: +38068-444-9999

Електронна пошта: sirsom_sumi5@example.com

6) Магазин: Сирний сомельє

вул. Соборна, 56

Телефон: +38063-777-3333

Електронна пошта: sirsom_sumi6@example.com

7) Магазин: Сирний сомельє

вул. ЗСУ, 15

Телефон: +38067-222-5555

Електронна пошта: sirsom_sumi7@example.com

8) Магазин: Сирний сомельє

вул. Лебединська, 3

Телефон: +38068-666-7777

Електронна пошта: sirsom_sumi8@example.com

9) Магазин: Сирний сомельє

вул. Роменська, 13

Телефон: +38050-999-1111

Електронна пошта: sirsom_sumi9@example.com

10) Магазин: Сирний сомельє

просп. Миру, 72

Телефон: +38066-888-3333

Електронна пошта: sirsom_sumi10@example.com

Ці контактні дані є випадковими та створені для прикладу і не відображають реальні контакти магазинів "Сирний сомельє" у місті Суми.

Аналогічно випадковим чином були створені контактні дані клієнтів магазину (це уявлені дані і не відображають реальних контактів клієнтів магазину):

Ось 20 прикладів контактних даних (ім'я, прізвище, телефон, електронна пошта) для клієнтів магазину:

Ім'я: Іван

Прізвище: Петров

Телефон: +38067-123-4567

Електронна пошта: ivan.petrov@example.com

Ім'я: Олена

Прізвище: Сидоренко

Телефон: +38050-987-6543

Електронна пошта: olena.sydorenko@example.com

Ім'я: Михайло

Прізвище: Коваленко

Телефон: +38063-555-8888
Електронна пошта: m.kovalenko@example.com

Ім'я: Наталія
Прізвище: Іванова
Телефон: +38066-111-2222
Електронна пошта: natalia.ivanova@example.com

Ім'я: Петро
Прізвище: Семенов
Телефон: +38068-444-9999
Електронна пошта: petro.semenov@example.com

Ім'я: Лілія
Прізвище: Григоренко
Телефон: +38063-777-3333
Електронна пошта: liliya.grigorenko@example.com

Ім'я: Сергій
Прізвище: Мельник
Телефон: +38067-222-5555
Електронна пошта: sergiy.melnik@example.com

Ім'я: Анна
Прізвище: Волкова
Телефон: +38068-666-7777
Електронна пошта: anna.volkova@example.com

Ім'я: Олексій
Прізвище: Козлов

Телефон: +38050-999-1111
Електронна пошта: o.kozlov@example.com

Ім'я: Марія
Прізвище: Лисенко
Телефон: +38066-888-3333
Електронна пошта: maria.lisenko@example.com

Ім'я: Віктор
Прізвище: Жуков
Телефон: +38067-777-8888
Електронна пошта: v.zhukov@example.com

Ім'я: Оксана
Прізвище: Кравченко
Телефон: +38063-333-2222
Електронна пошта: oksana.kravchenko@example.com

Ім'я: Дмитро
Прізвище: Іванов
Телефон: +38068-444-5555
Електронна пошта: dmitro.ivanov@example.com

Ім'я: Юлія
Прізвище: Соловйова
Телефон: +38066-222-8888
Електронна пошта: yulia.solovyova@example.com

Ім'я: Андрій
Прізвище: Павленко

Телефон: +38067-111-9999

Електронна пошта: andriy.pavlenko@example.com

Ім'я: Ніна

Прізвище: Грищук

Телефон: +38050-777-2222

Електронна пошта: nina.grishchuk@example.com

Ім'я: Григорій

Прізвище: Савченко

Телефон: +38063-444-6666

Електронна пошта: g.savchenko@example.com

Ім'я: Євгенія

Прізвище: Ковальова

Телефон: +38068-999-3333

Електронна пошта: yevgeniya.kovalova@example.com

Ім'я: Артем

Прізвище: Морозов

Телефон: +38066-555-7777

Електронна пошта: artem.morozov@example.com

Ім'я: Катерина

Прізвище: Литвиненко

Телефон: +38067-888-2222

Електронна пошта: katerina.litvinenko@example.com

2.4. Створення тригерів

Інші дані заповнювалися, використовуючи інтерфейс програми DBbrowser for SQLite (Рис.2.3) або тригери.

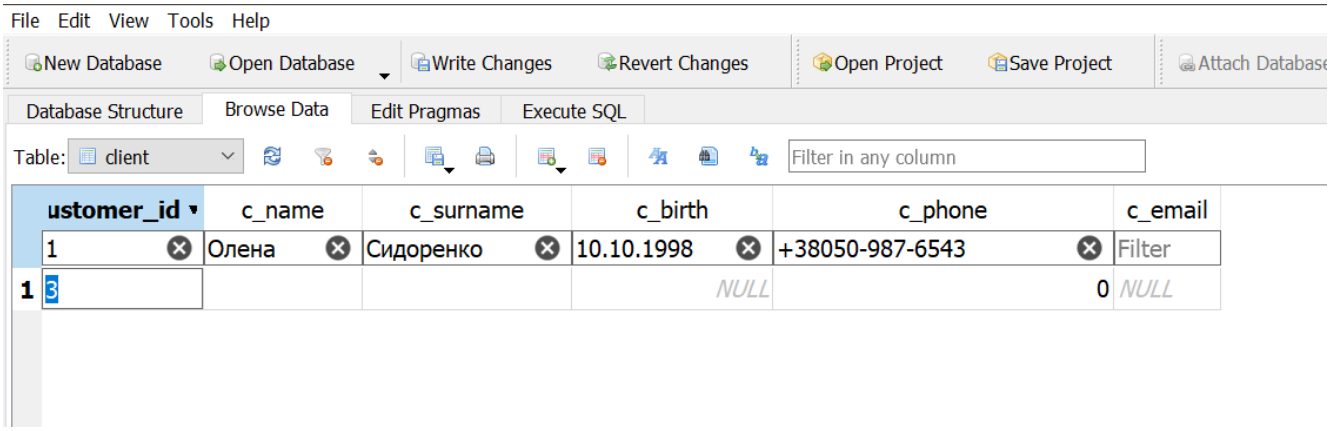

Рис. 2.3. – Приклад заповнення таблиці з даними клієнтів.

Для деяких комірок використання тригерів було необхідним для врахування особливостей підрахунку загальної суми покупки, або умов використання знижки.

Наприклад, для реалізації зміни відсотка знижки в залежності від загальної суми покупки можна використати тригер в базі даних. Нижче наведено приклад SQL-запиту для створення тригера, який автоматично оновлює відсоток знижки в сутності "Знижка" на основі загальної суми покупки:

```
CREATE TRIGGER update_discount_percentage
AFTER INSERT OR UPDATE ON Замовлення
FOR EACH ROW
BEGIN
    DECLARE total_amount DECIMAL(10, 2);
    DECLARE discount_percent DECIMAL(5, 2);
    -- Отримати загальну суму покупки для даного замовлення
    SELECT SUM(Продукт.ціна * Деталі_замовлення.кількість)
    INTO total_amount
    FROM Замовлення
    INNER JOIN Деталі_замовлення ON Замовлення.order_id =
Деталі_замовлення.order_id
```

```
    INNER JOIN Продукт ON Деталі_замовлення.product_id = Продукт.product_id
    WHERE Замовлення.order_id = NEW.order_id;
    -- Оновити відсоток знижки в залежності від загальної суми покупки
    IF total_amount >= 1000 THEN
      SET discount_percent = 10.00;
    ELSE
      SET discount_percent = 5.00;
    END IF;
    -- Оновити відсоток знижки в записі знижки
    UPDATE Знижка
    SET Знижка.відсоток = discount_percent
    WHERE Знижка.discount_id = NEW.discount_id;
  END;
```

У цьому прикладі тригер update_discount_percentage викликається після вставки або оновлення запису в сутності "Замовлення". Він обчислює загальну суму покупки для даного замовлення та на основі цієї суми встановлює відповідний відсоток знижки. Потім тригер оновлює відсоток знижки в записі сутності "Знижка" за допомогою операції UPDATE.

Важливо врахувати, що цей приклад базується на загальному уявленні про сутності та відповідні зв'язки у базі даних. Назви таблиць та полів можуть відрізнятися в залежності від конкретної реалізації бази даних.

Для створення тригера в таблиці "замовлення", який автоматично розраховує суму замовлення на основі ваги сиру та його ціни, можна використати наступний SQL-запит:

```
CREATE TRIGGER calculate_order_total
BEFORE INSERT ON замовлення
```

```
FOR EACH ROW
BEGIN
   DECLARE total DECIMAL(10, 2);
   SET total = 0;
      -- Розрахунок суми замовлення на основі ваги сиру та його ціни
   SELECT SUM(товари.вага * товари.ціна) INTO total
   FROM товари
   WHERE товари.id_товару IN (
      SELECT id_товару
      FROM замовлення_товари
      WHERE id_замовлення = NEW.id_замовлення
   );
   -- Оновлення поля "сума" в таблиці "замовлення"
   SET NEW.сума = total;
END;
```

У цьому тригері calculate_order_total виконується до вставки (INSERT) нового запису в таблицю "замовлення". Для кожного нового запису, він обчислює суму замовлення, перебираючи всі товари, пов'язані з цим замовленням. Вага товару помножується на його ціну, і всі ці значення додаються до змінної total. На кінці тригера, поле "сума" в таблиці "замовлення" оновлюється знайденою сумою.

Основуючись на вказаній раніше базі даних, вигляд тригера для таблиці "замовлення", де знижка залежить від дати замовлення, може бути наступним:

```
CREATE TRIGGER calculate_order_discount
BEFORE INSERT ON замовлення
FOR EACH ROW
BEGIN
```

```
    DECLARE discount DECIMAL(5, 2);
    SET discount = 0;
     -- Отримання знижки на основі дати замовлення
    SELECT знижки.відсоток INTO discount
    FROM знижки
    WHERE NEW.дата_замовлення BETWEEN знижки.початок_дії AND знижки.кінець_дії;
     -- Розрахунок суми замовлення з урахуванням знижки
    SET NEW.сума = NEW.сума * (1 - discount/100);
END;
```

У цьому тригері calculate_order_discount виконується до вставки (INSERT) нового запису в таблицю "замовлення". Для кожного нового запису, він отримує знижку на основі дати замовлення, шукавши відповідну знижку у таблиці "знижки". Потім розраховує суму замовлення, помножуючи її на (1 - discount/100), де discount - отримана знижка.

Ось приклад тригера, який рахує загальну суму рядків для певного клієнта в таблиці "деталі замовлення" і заповнює комірку "total" в таблиці "замовлення":

```
CREATE TRIGGER calculate_total_amount
AFTER INSERT OR UPDATE ON деталі_замовлення
FOR EACH ROW
BEGIN
    DECLARE total_amount DECIMAL(10, 2);
    -- Рахуємо загальну суму рядків для даного клієнта
    SELECT SUM(price * quantity)
    INTO total_amount
    FROM деталі_замовлення
    WHERE customer_id = NEW.customer_id;
    -- Оновлюємо комірку "total" в таблиці "замовлення" для відповідного замовлення
```

```
    UPDATE замовлення
    SET total = total_amount
    WHERE order_id = NEW.order_id;
END;
```

У цьому тригері ми використовуємо подію AFTER INSERT OR UPDATE, що означає, що тригер буде спрацьовувати після вставки або оновлення запису в таблиці "деталі замовлення". Кожен раз, коли відбувається вставка або оновлення, тригер буде обраховувати загальну суму рядків для даного клієнта і оновлювати комірку "total" в таблиці "замовлення" для відповідного замовлення.

2.5. Створення запитів до бази даних

Нижче наведено приклади типових запитів до бази даних мережі магазинів з продажу сиру та вина з накопичувальною системою знижок, які дозволять отримати різноманітну інформацію з бази даних, спрямовану на аналіз продажів, знижок, клієнтів та інших аспектів мережі магазинів.

- Показати список всіх магазинів у базі даних.
- Вивести всіх виробників сиру та вина з їхніми контактними даними.
- Отримати список усіх продуктів, які є в наявності у всіх магазинах.
- Знайти всі замовлення, зроблені певним клієнтом.
- Визначити загальну суму всіх замовлень для кожного магазину.
- Вивести список клієнтів, які мають накопичену знижку.
- Порахувати кількість продуктів, які було продано з кожної категорії.
- Визначити загальну суму продажів за певний період часу.
- Знайти всі замовлення, в яких було застосовано певну знижку.
- Вивести список продуктів, відсортованих за їхньою ціною у спадному порядку.

Наприклад, для показу списку всіх магазинів у базі даних, можна використати запит SELECT. Ось приклад запиту:

SELECT * FROM Магазин;

Цей запит вибирає всі рядки з таблиці "Магазин" і повертає усі дані про магазини, включаючи всі атрибути цієї таблиці.

Для виведення всіх виробників сиру та вина з їхніми контактними даними, можна використати наступний запит:

SELECT * FROM Виробник;

Цей запит вибирає всі рядки з таблиці "Виробник" і повертає усі дані про виробників, включаючи всі атрибути цієї таблиці.

Для знаходження всіх замовлень, зроблених певним клієнтом, треба використати наступний запит

SELECT * FROM Замовлення WHERE customer_id = <customer_id>;

У цьому запиті <customer_id> потрібно замінити на фактичний ідентифікатор клієнта, для якого ви хочете знайти замовлення. Цей запит вибирає всі рядки з таблиці "Замовлення", де значення стовпця customer_id співпадає з вказаним значенням <customer_id>.

Для знаходження всіх замовлень, зроблених на певну суму, слід використати наступний запит:

SELECT * FROM Замовлення WHERE загальна_сума = <total_amount>;

У цьому запиті <total_amount> потрібно замінити на фактичну суму, для якої ви хочете знайти замовлення. Цей запит вибирає всі рядки з таблиці "Замовлення", де значення стовпця загальна_сума співпадає з вказаним значенням <total_amount>.

Для знаходження трьох клієнтів, які витратили найбільше коштів на покупку в мережі магазинів, можна використати наступний запит:

```
SELECT Клієнт.customer_id, Клієнт.ім'я, Клієнт.прізвище, SUM(Замовлення.загальна_сума) AS сума_витрат
FROM Клієнт
JOIN Замовлення ON Клієнт.customer_id = Замовлення.customer_id
GROUP BY Клієнт.customer_id, Клієнт.ім'я, Клієнт.прізвище
ORDER BY сума_витрат DESC
LIMIT 3;
```

Цей запит використовує з'єднання (JOIN) між таблицями "Клієнт" і "Замовлення" на основі стовпця customer_id. Він групує рядки за ідентифікатором клієнта, обчислює суму витрат кожного клієнта за допомогою функції SUM() для стовпця загальна_сума в таблиці "Замовлення". Результати сортуються за зменшенням суми витрат (від найбільшої до найменшої) і обмежуються першими трьома рядками за допомогою ключового слова LIMIT.

Цей запит поверне три клієнти, які витратили найбільше коштів на покупку в мережі магазинів, разом із сумою їх витрат.

Запит до бази даних мережі магазинів з використанням GROUP BY та HAVING може мати наступний вигляд:

SELECT Клієнт.ім'я, Клієнт.прізвище, COUNT(Замовлення.order_id) AS кількість_замовлень

FROM Клієнт

JOIN Замовлення ON Клієнт.customer_id = Замовлення.customer_id

GROUP BY Клієнт.customer_id, Клієнт.ім'я, Клієнт.прізвище

HAVING COUNT(Замовлення.order_id) > 5;

У цьому запиті використовуються дві таблиці "Клієнт" і "Замовлення". Запит об'єднує ці таблиці за допомогою оператора JOIN за спільним стовпцем customer_id. Згрупування (GROUP BY) проводиться за стовпцями customer_id, ім'я та прізвище клієнта. Функція COUNT використовується для підрахунку кількості замовлень для кожного клієнта. HAVING використовується для фільтрації результатів на основі агрегатної функції COUNT. У даному прикладі фільтруються тільки ті рядки, де кількість замовлень більше 5.

Цей запит поверне список клієнтів разом з кількістю замовлень для кожного клієнта, які зробили більше 5 замовлень в мережі магазинів.

Ось три різних приклади запитів до бази даних мережі магазинів з використанням агрегатних функцій:

Запит на підрахунок загальної кількості замовлень:

SELECT COUNT(*) AS загальна_кількість_замовлень

FROM Замовлення;

Цей запит поверне загальну кількість замовлень у базі даних.

Запит на визначення середньої ціни товару:

SELECT AVG(Товар.ціна) AS середня_ціна_товару

FROM Товар;

Цей запит обчислить середню ціну товару в базі даних.

Запит на визначення найвищої суми замовлення:

SELECT MAX(Замовлення.загальна_сума) AS найвища_сума_замовлення

FROM Замовлення;

Цей запит поверне найвищу суму замовлення в базі даних.

Ці запити демонструють використання різних агрегатних функцій, таких як COUNT(), AVG() та MAX(). Їх можна змінити або додати інші функції, щоб виконати різні обчислення або аналізи в конкретній базі даних мережі магазинів.

Ось три різних приклади запитів до бази даних мережі магазинів з використанням оператора UPDATE:

Запит на зміну ціни певного товару:

UPDATE Товар

SET ціна = 455.99

WHERE назва = 'Сир Чеддер';

Цей запит змінить ціну товару з назвою "Сир Чеддер" на нову ціну 455.99.

Запит на оновлення інформації про клієнта:

UPDATE Клієнт

SET телефон = '+380987654321'

WHERE customer_id = 123;

Цей запит оновить номер телефону клієнта з ідентифікатором 123 на новий номер '+380987654321'.

Запит на збільшення знижки для всіх клієнтів:

```
UPDATE Клієнт
SET знижка = знижка + 5;
```

Цей запит збільшить поточний розмір знижки для всіх клієнтів на 5 одиниць.

Ці запити використовують оператор UPDATE для зміни даних у вказаній базі даних

Ось приклади запитів до бази даних мережі магазинів з використанням підзапитів (subqueries):

Запит на вибірку клієнтів, які зробили замовлення з сумою більше середньої суми замовлень:

```
SELECT Ім'я, Прізвище
FROM Клієнт
WHERE customer_id IN (
    SELECT customer_id
    FROM Замовлення
    GROUP BY customer_id
    HAVING SUM(загальна_сума) > (
        SELECT AVG(загальна_сума)
        FROM Замовлення
    )
```

```
);
```

Цей запит вибирає ім'я та прізвище клієнтів, які зробили замовлення з сумою більше середньої суми замовлень усіх клієнтів.

Запит на вибірку товарів, які є унікальними (не повторюються) у замовленнях:

```
SELECT назва
FROM Товар
WHERE товар_id IN (
    SELECT товар_id
    FROM Замовлення_Товари
    GROUP BY товар_id
    HAVING COUNT(*) = 1
);
```

Цей запит вибирає назви товарів, які є унікальними у замовленнях, тобто вони замовлялися лише один раз.

У цих прикладах підзапити використовуються для отримання додаткової інформації, фільтрації або порівняння даних у базі даних.

Ось приклади запитів до бази даних мережі магазинів з використанням оператора SELECT DISTINCT:

Запит на вибірку унікальних типів вина, доступних у магазинах:

```
SELECT DISTINCT тип_вина
FROM Товар
WHERE категорія = 'вино';
```

Цей запит поверне унікальні типи вина, які є доступними у магазинах, з таблиці "Товар", за умови, що категорія товару є "вино".

Запит на вибірку унікальних дат замовлень:

SELECT DISTINCT дата_замовлення

FROM Замовлення;

Цей запит поверне унікальні дати замовлень з таблиці "Замовлення".

У цих прикладах оператор SELECT DISTINCT використовується для отримання унікальних значень з вибраних стовпців.

Ось приклади запитів до бази даних мережі магазинів з використанням оператора ORDER BY:

Запит на вибірку клієнтів, відсортованих за алфавітом прізвища в зростаючому порядку:

SELECT Ім'я, Прізвище

FROM Клієнт

ORDER BY Прізвище ASC;

Цей запит вибирає ім'я та прізвище клієнтів з таблиці "Клієнт" і сортує їх за прізвищем в зростаючому порядку.

Запит на вибірку товарів, відсортованих за спаданням ціни:

SELECT назва, ціна

FROM Товар

ORDER BY ціна DESC;

Цей запит вибирає назву та ціну товарів з таблиці "Товар" і сортує їх за ціною в спадаючому порядку.

Запит на вибірку замовлень, відсортованих за датою в зростаючому порядку:

```
SELECT *
FROM Замовлення
ORDER BY дата_замовлення ASC;
```

Цей запит вибирає всі стовпці з таблиці "Замовлення" і сортує їх за датою замовлення в зростаючому порядку.

У цих прикладах оператор ORDER BY використовується для сортування результатів запиту за вказаним стовпцем у заданому порядку (зростаючому або спадаючому).

Ось приклади запитів до бази даних мережі магазинів з використанням оператора BETWEEN:

Запит на вибірку замовлень, зроблених протягом певного періоду:

```
SELECT *
FROM Замовлення
WHERE дата_замовлення BETWEEN '2023-01-01' AND '2023-03-31';
```

Цей запит вибирає всі замовлення з таблиці "Замовлення", які були зроблені між 1 січня 2023 року і 31 березня 2023 року.

Запит на вибірку товарів за діапазоном цін:

```
SELECT *
FROM Товар
WHERE ціна BETWEEN 500 AND 700;
```

Цей запит вибирає всі товари з таблиці "Товар", ціна яких знаходиться у діапазоні від 500 до 700.

У цих прикладах оператор BETWEEN використовується для вибірки даних, які знаходяться у вказаному діапазоні.

Перелік запитів, які можна створити до заданої бази даних достатньо великий, тому були наведені тільки загальні формулювання найбільш актуальних запитів. Відповідно до конкретних потреб та реалізації ці запити можуть модифікуватися.

ВИСНОВКИ

При виконанні кваліфікаційної роботи на тему: «Проектування бази даних для системи управління складом та продажем товарів для роздрібної торгівлі» було розглянуто та спроектовано модель бази даних роздрібнених магазинів «Сирний сомельє». Мережа магазинів містить дані про наявність товару, надає послуги з продажу голландських сирів та імпортного алкоголю. База даних магазину містить інформацію постійних клієнтів, накопичення знижок, прихід товару, яка потребує сортування та обробки.

В даній роботі було проведено розгорнуте інфологічне дослідження мережі магазинів, створена основа для фізичної моделі бази даних, була побудована ER-діаграма моделі бази даних. Для спрощення роботи в базі даних роздрібнених магазинів, щоб швидко та зручно отримувати потрібну інформацію, були створені типові прості та складні запити. Також розроблені тригери, що дозволяють працювати з накопичувальною системою знижок, підраховувати загальну суму покупки, автоматично оновлювати комірки потрібних таблиць.

Робота виконувалась, використовуючи систему управління базами даних DB Browser for SQL lite. Також, було встановлено, що мова SQL надає всі можливості для спрощення роботи в базі даних, щоб швидко та зручно отримувати потрібну інформацію.

При виконанні роботи були використані реальні найменування товарів для магазину «Сирний сомельє».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до оформлення кваліфікаційних робіт здобувачів вищої освіти першого бакалаврського та другого магістерського рівнів / укладач Ющенко О.В. − Суми: Сумський державний університет, 2023. – 53 с.
2. https://www.solarwinds.com/resources/it-glossary/sql-database
3. https://www.simplilearn.com/tutorials/sql-tutorial/what-is-normalization-in-sql
4. https://www.cioinsight.com/big-data/what-is-a-dbms/
5. The Definitive Guide to SQLite by Grant Allen, Mike Owens
6. SQLite Database System Design and Implementation (2015) by Sibsankar Haldar
7. Введення в базу даних Sqlite - Yoip. Yoip. URL: http://yoip.com.ua/vvedennya-v-bazu-danih-sqlite/
8. Учасники проектів Вікімедіа. Реляційна база даних – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Реляційна_база_даних
9. Учасники проектів Вікімедіа. SQLite – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/SQLite
10. Що таке sqlite? - визначення з техопедії - Бази даних - 2022. Icy Science. URL: https://uk.theastrologypage.com/sqlite
11. ER-модель. URL: https://ua.wikipedia.org/wiki/ER-модель
12. SQLITE команди URL: http://old.code.mu/sql/union.html
13. SQLite Home Page. URL: https://www.sqlite.org/index.html
14. SQLite Tutorial - An Easy Way to Master SQLite Fast. SQLite Tutorial. URL: https://www.sqlitetutorial.net/
15. SQLite. URL: https://lecturesdb.readthedocs.io/databases/sqlite.html

ДОДАТОК А

Створення та заповнення таблиць даними в DB Browser for SQLite для магазину з продажу імпортних товарів молочної та алкогольної продукції «Сирний Сомельє».

Щоб створити таблицю в "DB Browser for SQLite", необхідно спочатку запустити програму і створити базу даних у форматі .db.

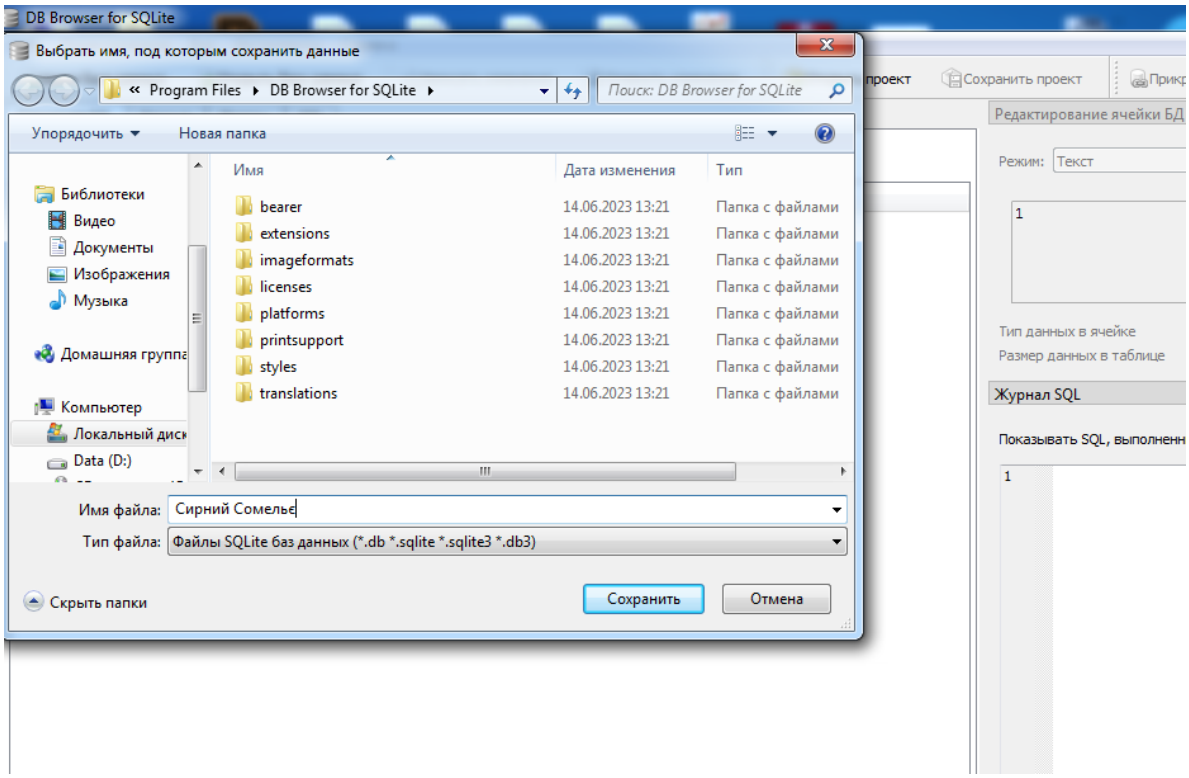

Рис. А.1. – Створення бази даних «Сирний Сомельє».

Після створення бази даних у форматі .db і відкриття цієї бази даних треба натиснути на поле «Створити таблицю» і ввести його в поле "Створити таблицю". Щоб створити правильну базу даних, потрібно використовувати вже створену ER-діаграм: відтворити раніше створені поля з таблиці та їх властивості.

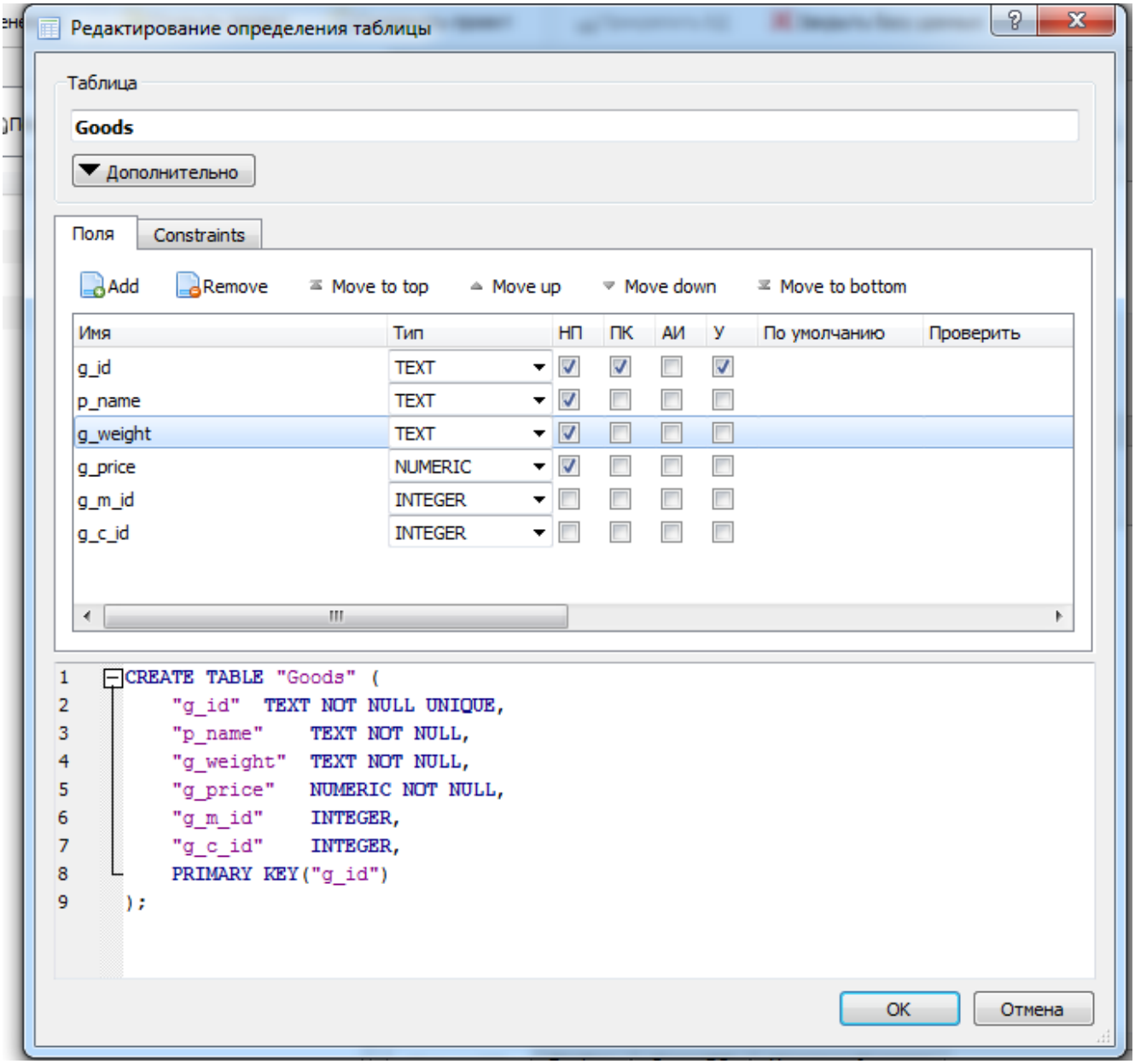

Рис. А.2. – Створення таблиці «Goods».

Кожне значення, що зберігається в базі даних SQLite, має один з декількох класів зберігання. Існує клас зберігання TEXT, де стовпець зберігає всі дані. Клас зберігання TEXT або BLOB використовується для зберігання всіх даних, NUMERIC - це стовпець, який може містити значення, використовуючи всі п'ять класів зберігання, і зазвичай вибирається при введенні числових даних у цьому стовпці. INTEGER вибирається, коли до цього стовпця вводяться числові дані; Стовпчик INTEGER має ті самі властивості, що й стовпчик NUMERIC, за винятком виразу CAST, який використовується для перетворення значень з одного типу даних в інший. REAL має ті ж властивості, що і стовпець NUMERIC, за винятком

виразу CAST, який використовується для перетворення значень з одного типу даних в інший. Стовпчик REAL має ті самі властивості, що й стовпчик NUMERIC, за винятком того, що він представляє цілі значення як числа з плаваючою комою.

За такою самою послідовністю створюємо всі інші таблиці.

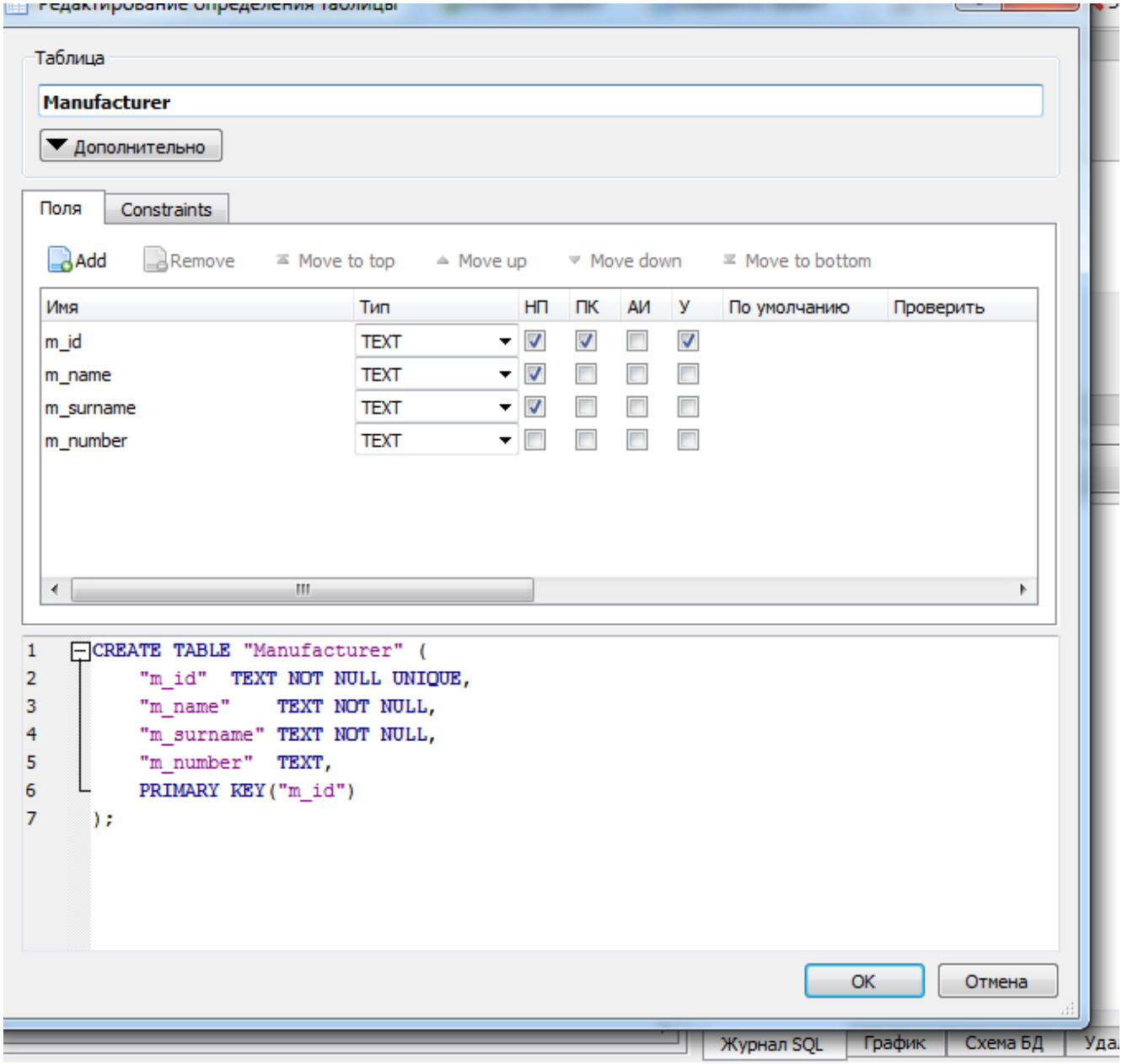

Рис. А.3. – Створення таблиці «Manufacturer».

PRIMARY KEY (в нашому випадку "m_id") - це первинний ключ. Цей параметр встановлюється для унікальної ідентифікації конкретного або іншого запису таблиці, або встановлюється для унікальної ідентифікації інших записів таблиці. Він може бути тільки один і не може бути NULL. Крім того, первинний ключ

зв'язується з зовнішнім ключем, таким чином встановлюється зв'язок між таблицями для обробки інформації Між таблицями встановлюється зв'язок для обробки інформації.

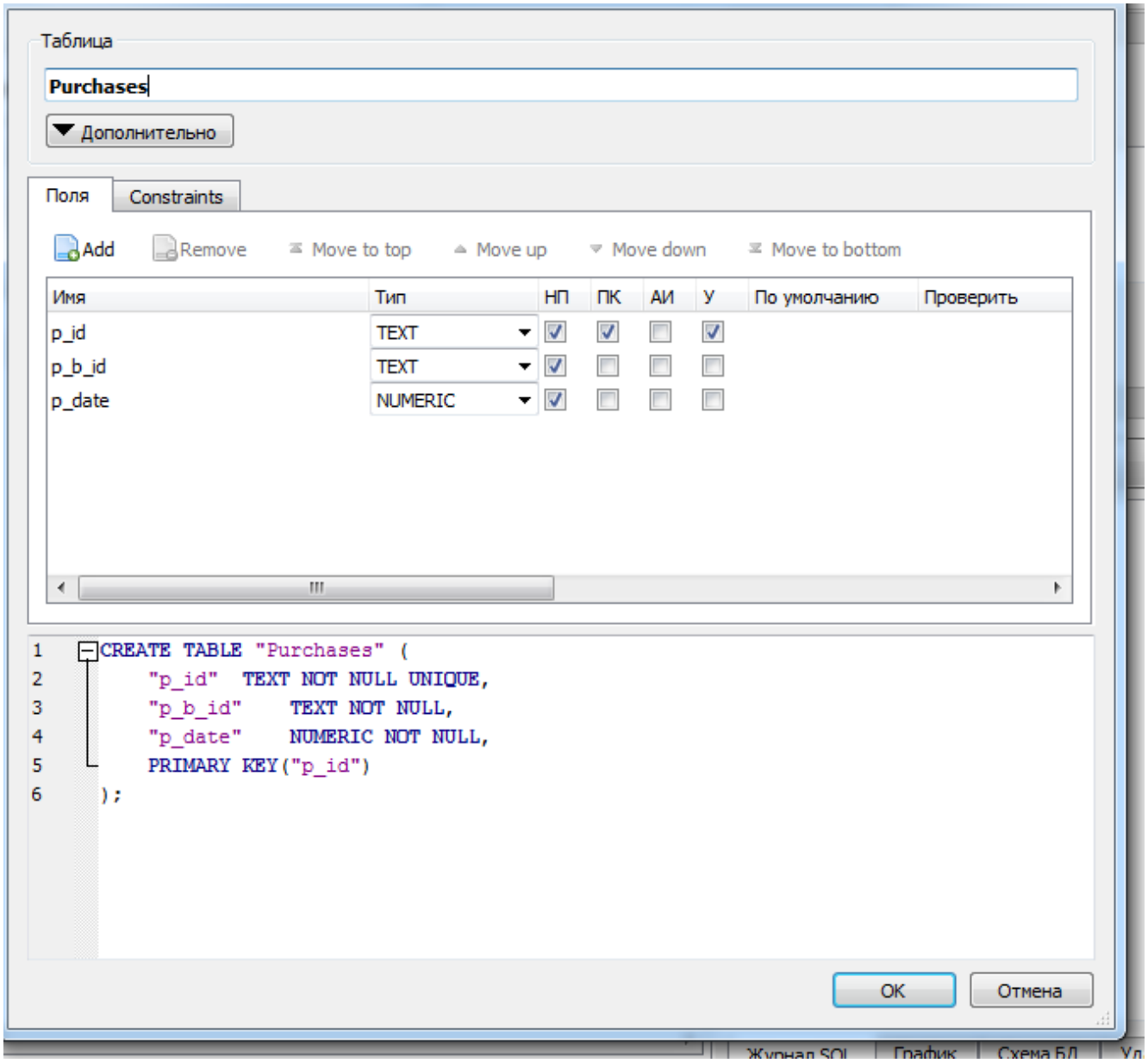

Рис. А.4. – Створення таблиці «Purchases».

Властивості в таблиці «p_id» та «p_b_id» мають TEXT NOT NULL та UNIQUE, в той час як «p_date» має NUMERICAL NOT NULL.

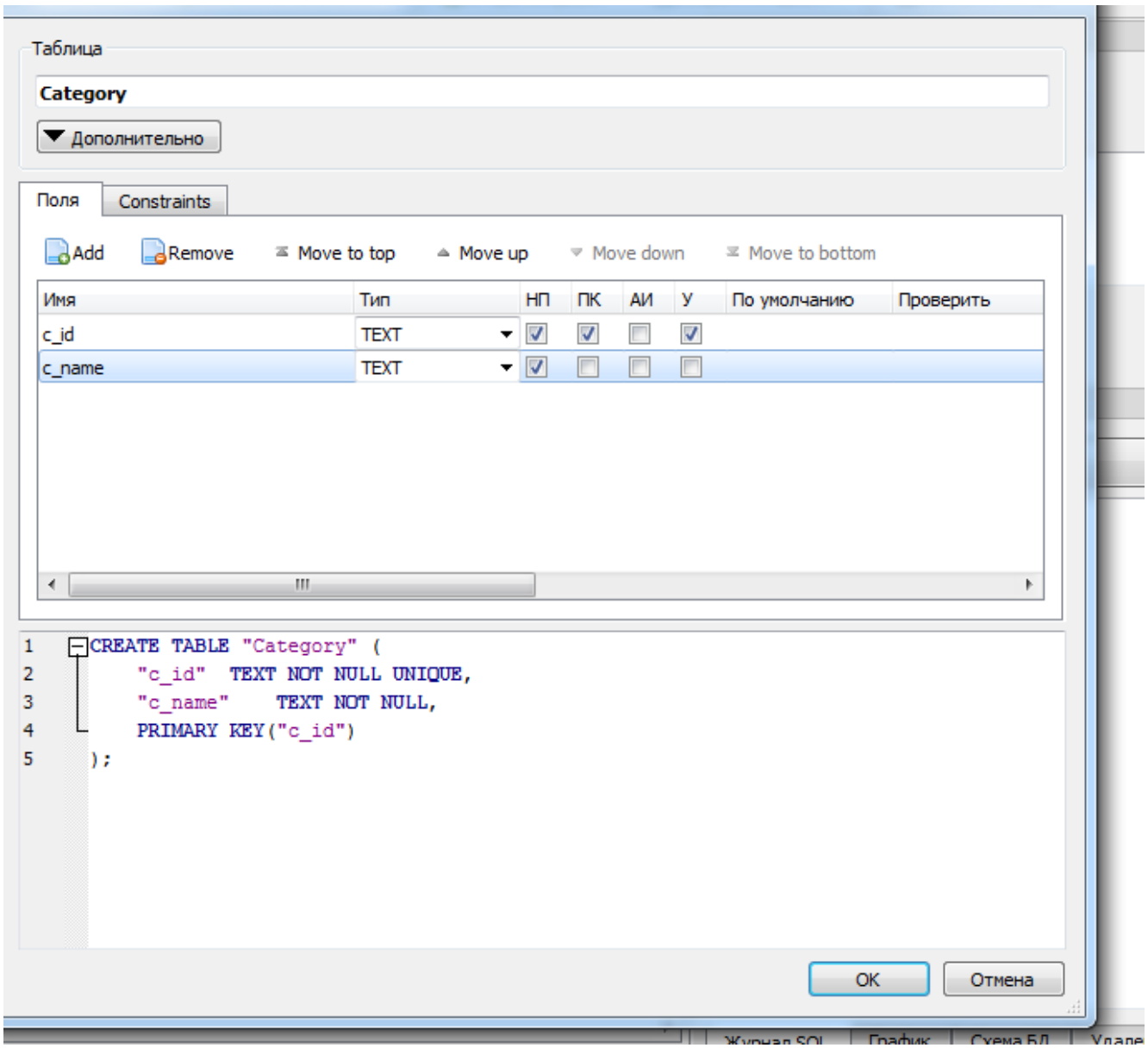

Рис. А.5. – Створення таблиці «Category».

Таблиця «Category» має властивості «c_id» та «c_name» TEXT NOT NULL та UNIQUE.

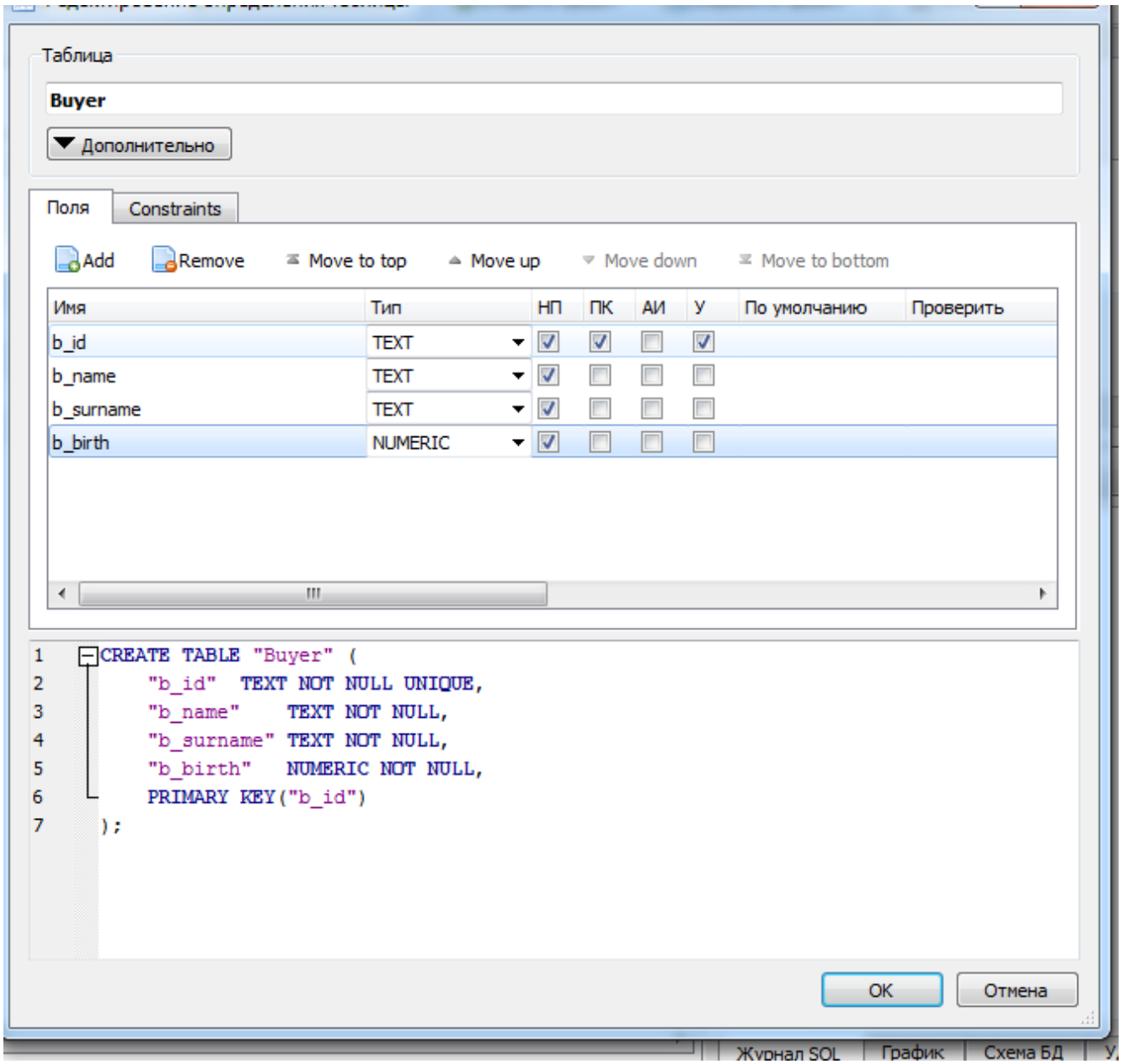

Рис. А.6. – Створення таблиці «Buyer»

Таблиця «b_id» типу INTEGER і властивість «b_id» має значення NOT NULL. Властивість NOT NULL означає, що вона ніколи не буває порожньою. UNIQUE вказує на те, що вони є унікальними. Далі йде поле «b_name», яке повинно бути записано як ім'я, NOT NULL в TEXT повинно бути обране, тому що покупець не може існувати без ім’я. NOT NULL в TEXT, тому що «b_surname» має однаковий тип з «b_name», а «b_birth» має тип NUMERIC і NOT NULL. В таблиці «Manufacturer» властивість «m_number» має тип TEXT, бо NUMERIC і NOT NULL тут не є обов'язковими. Це пов'язано з тим, що номер телефону може бути недоступним для виробника.

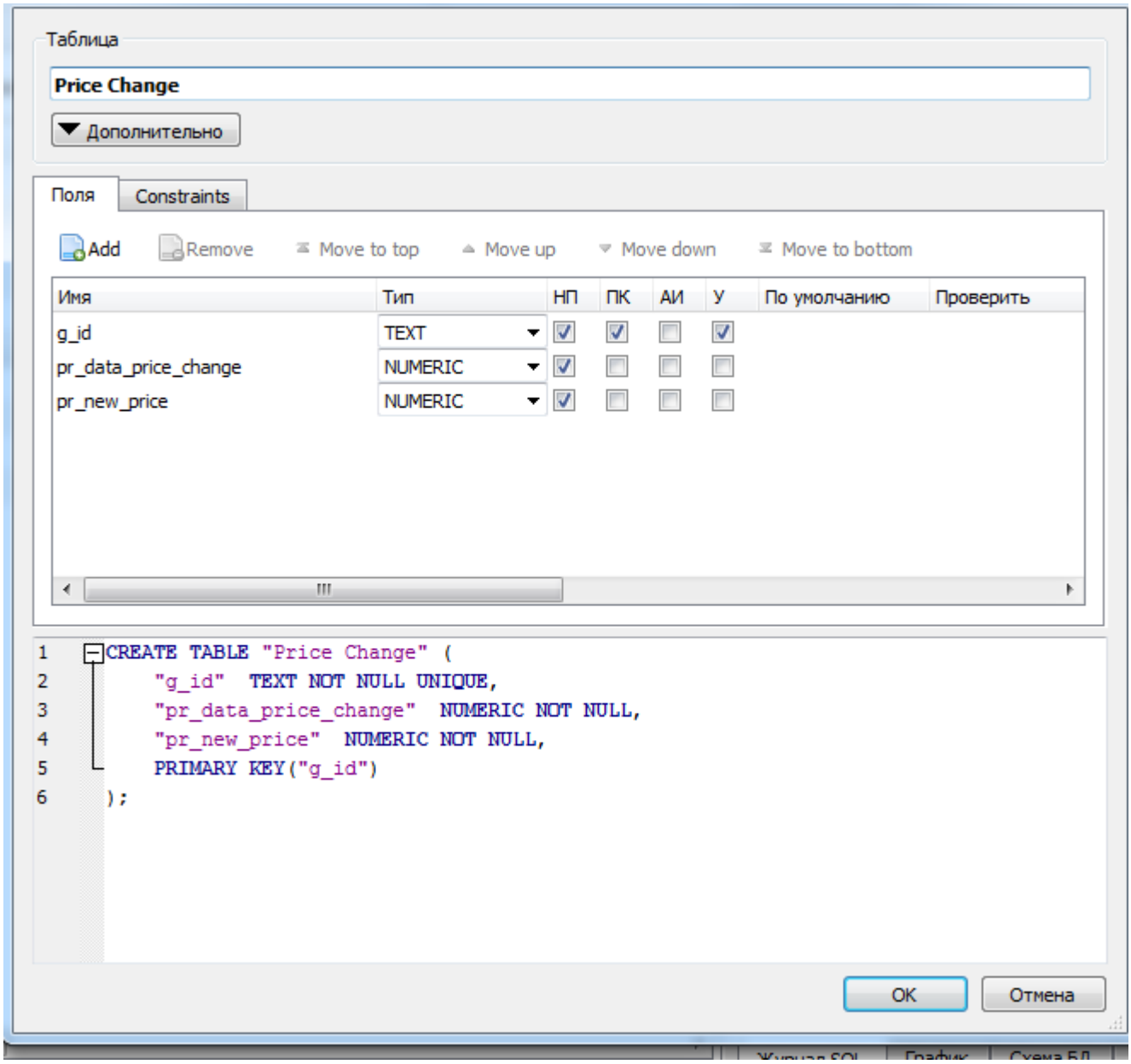

Рис. А.7. – Створення додаткової таблиці «Price_Change»

Останні дві таблиці «Price_Change» та «Purchases_item» додаткові таблиці. «Price_Change» таблиця стежить за цінами, та автоматично заповнює і змінює поле, якщо ціни знизились або навпаки зросли. Таблиця «Purchases_item» дає змогу швидко отримати інформацію про кількість, рахунок та ціну товару.

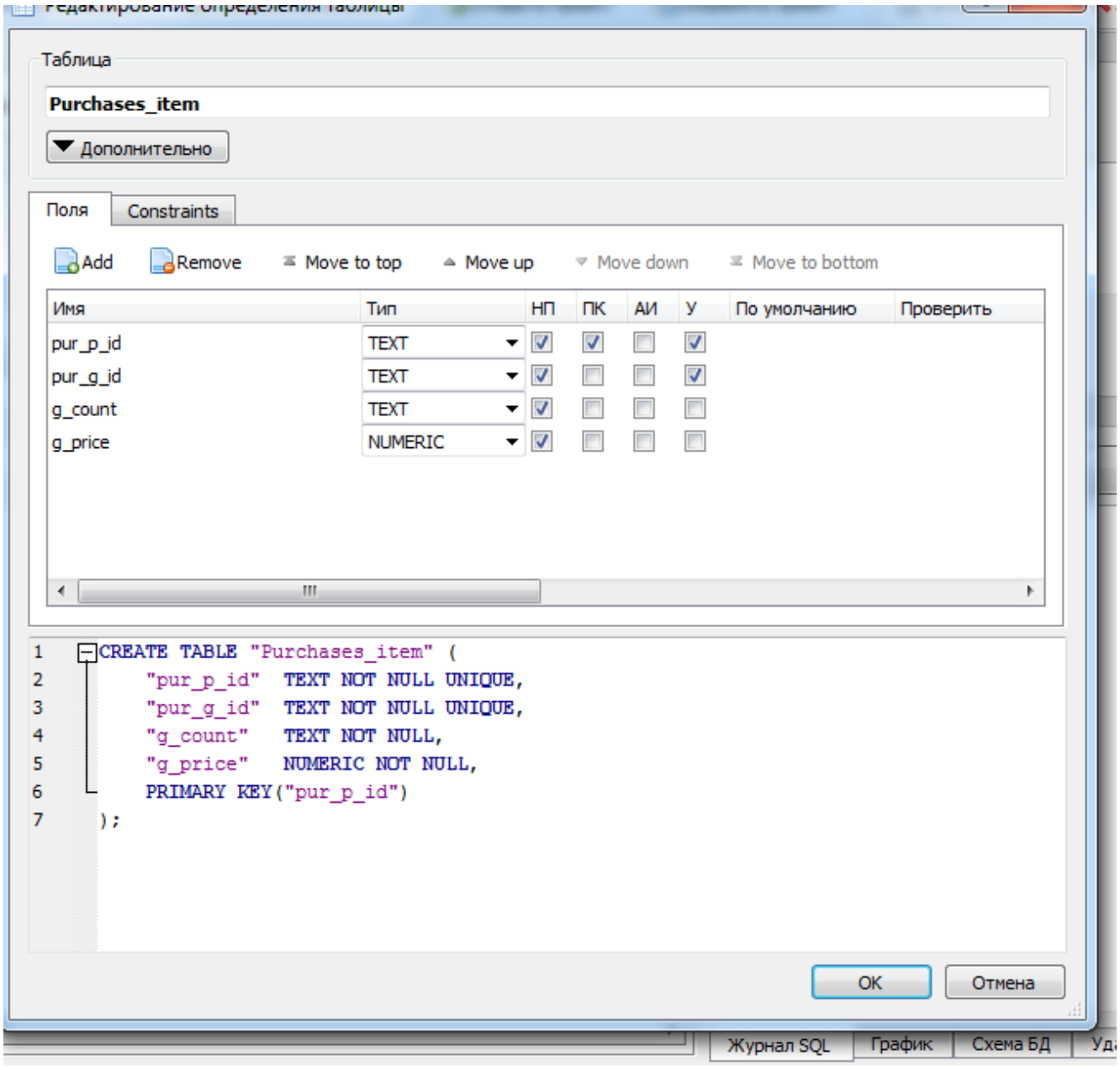

Рис. А.8 – Створення додаткової таблиці «Purchases_item».

Створивши всі таблиці, можна переходити до їх заповнення, створення тригерів та запитів за прикладами.