Міністерство освіти і науки України
Український державний університет науки і технологій

Комп'ютерних технологій і систем
(назва факультету)
Комп'ютерні інформаційні технології
(повна назва кафедри)

Пояснювальна записка
до кваліфікаційної роботи
Магістр
(ступінь вищої освіти)

на тему: Аналіз механізмів відновлення даних у середовищі СКБД SQLite
за освітньою програмою 12 Інформаційні технології
зі спеціальності: 121 Інженерія програмного забезпечення

Виконав студент групи:
(підпис студента) Євген БАЮК
(Ім'я ПРІЗВИЩЕ)

Керівник:
(підпис) доц. Олександр ІВАНОВ
(посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер:
(підпис) доц. Світлана ВОЛКОВА
(посада, Ім'я ПРІЗВИЩЕ)

Консультанти:
Кошторис на розробку ПЗ
(назва розділу)
(підпис) доц. Микола ГНЕНИЙ
(посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень праць інших авторів без відповідних посилань.
Студент
(підпис)

Дніпро – 2022 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Дніпровський національний університет залізничного транспорту

імені академіка В. Лазаряна

Кафедра Комп'ютерні інформаційні технології

«ДО ЗАХИСТУ»

Завідувач кафедри

_____ /Віктор ШИНКАРЕНКО/

«_____» _____ 20____р.

Дипломна робота

на здобуття освітнього ступеня «магістр»

Галузь знань **12 Інформаційні технології**

Спеціальність **121 Інженерія програмного забезпечення**

Тема **Аналіз механізмів відновлення даних у середовищі СКБД SQLite**

Theme **Analysis of data recovery mechanisms in SQLite DBMS environment**

Керівник дипломної роботи доц. _____ Олександр ІВАНОВ

Нормоконтролер доц. _____ Світлана ВОЛКОВА

Студент групи ПЗ1921 _____ Євген Баюк

Student Baiuk Yevhen

Дніпро – 2022

Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна

Факультет Комп'ютерних технологій і систем кафедра Комп'ютерні інформаційні технології

Спеціальність Інженерія програмного забезпечення

«ЗАТВЕРДЖУЮ»

Завідувач кафедри

_____ проф. Віктор Шинкаренко

(підпис)

«___»_____2021 р.

ЗАВДАННЯ

до дипломної роботи на здобуття ОС_____Магістр_____

(освітній ступень)

студента групи (ПЗ1821) 961-М Баюк Євген Вадимович

(номер групи) (ПІБ)

1 Тема дипломної роботи: Дослідження Аналіз механізмів відновлення данних у середовищі СКБД SQLite

затверджена наказом по університету від «12» листопада 2019 р. № 833ст.

2 Термін подання студентом закінченого проекту «16» грудня 2020 р.

3 Вихідні дані до дипломного проекту_____

_____

4 Зміст пояснювальної записки (перелік питань до розробки) проведення аналізу сучасного стану досліджень часової ефективності гібридизації біонічних алгоритмів, визначення актуальних проблем, розробка власної гібридизації та проведення експериментів.

5 Перелік демонстраційного матеріалу презентація обраного для створення гібридизацій алгоритму, функції оптимізації та їх особливості, розроблена гібридизація, результати експериментів, висновки; відео демонстрація роботи розробленого програмного інструментарію для проведення досліджень.

6. Консультанти (з назвами розділів):

Розділ	Консультант	Підпис, дата	
		завдання видав	завдання прийняв
Техніко-економічні розрахунки	**доц. Гненний М.В.**		

КАЛЕНДАРНИЙ ПЛАН

п/п	Назва розділів дипломного проекту	Термін виконання розділів проекту (роботи)	Примітка
	Вступ	1.09.2021 – 10.09.2021	
	Огляд літератури	10.09.2021 – 15.09.2021	
	Постановка задачі, технічне завдання	15.09.2021 – 30.09.2021	
	Створення тестової програми	01.10.2021 – 15.10.2021	
	Перші тестування баз даних із різним наповненням	15.10.2021 – 22.10.2021	30%
	Застосування скрипту для автоматичного заповнення бази даних	22.10.2021 – 30.10.2021	
	Аналіз результатів	30.10.2021 – 11.11.2021	
	Розрахунок показників відновлення даних	11.11.2021 – 14.11.2021	
	Охорона праці	14.11.2021 – 18.11.2021	60%
0	Оформлення пояснювальної записки	18.11.2021 – 01.12.2021	
1	Демонстраційні матеріали	5.11.2021 – 16.12.2021	100%

Дата видачі завдання «12» листопада 2018 р.

Керівник дипломного проекту _____ **Іванов О.П.**

(підпис) (ПІБ)

Завдання прийняв до виконання _____ **Баюк Є.В.**

(підпис) **(ПІБ)**

РЕФЕРАТ

Об’єктом даного дослідження є процес пошуку методів зберігання інформації та її відновлення в разі видалення або пошкодження файлів баз даних типу SQLite.

Предметом дослідження є створення алгоритмів відновлення видаленої інформації з баз даних за показниками часової ефективності.

Метою даної роботи є визначення доцільності застосування алгоритмів відновлення інформації, а саме – можливість пришвидшити процес пошуку даних.

Методи дослідження: емпіричний науковий метод, аналіз та порівняння результатів роботи аналогічних програмних систем.

Пояснювальна записка складається зі вступу, 5 розділів, висновків, бібліографічного списку та 2 додатків.

Вступ – описується сутність роботи та її актуальність (3 сторінки).

Першому розділ – опис аналізу сучасних видів баз даних за науковими літературними джерелами (1**8** сторінок).

Другий розділ – опис і аналіз формату файлів баз даних типу SQLite та структур даних, що використовуються для збереження інформації (34 сторінок).

Третій розділ – описано процес проектування і розробки програмного забезпечення для дослідження (3 сторінок).

Четвертому розділ – визначення досліджуваних програмних аналогів та результати проведених експерементів(**6**).

П‘ятий розділ – розкриті питання охорони та безпеки праці в надзвичайних ситуаціях (**9** сторінок).

Додатки – технічне завдання і робочий проект. Таблиць – 28, рисунків – 18, бібліографія – 53 джерел.

Ключові слова: SQLite, відновлення, збереження.

ЗМІСТ

ВСТУП

Аналіз механізмів відновлення данних у середовищі СКБД SQLite – це робота, метою якої є визначення можливих способів відновлення видаленої інформації та з'ясування доцільності їх застосування. Під доцільністю розуміється здатність покращити цілісність відновлених даних.

Актуальність роботи. На сьогоднішній існує тенденція розвитку технологій пов'язаних із збереженням даних у СКБД SQLite. Простота і зручність вбудовування SQLite привели до того, що бібліотека використовується в браузерах, музичних плеєрах і багатьох інших програмах.

Зокрема, SQLite використовується в:

- Adobe Integrated Runtime - середовище для запуску додатків;
- Gears - відкрите програмне забезпечення компанії Google, що дозволяє використовувати веб-додатки за допомогою браузерів;
- Фреймворк Qt;
- Платформа XUL і всі додатки, засновані на цій платформі, в тому числі:
    - Mozilla Firefox (починаючи з версії 3.0);
    - Mozilla Thunderbird (починаючи з версії 3.0);
    - Songbird;
    - SQLite Manager;
- Skype;
- Viber;
- WhatsApp;
- Деякі моделі GPS-навігаторів Garmin;

Багато програм підтримують SQLite як формат зберігання даних (особливо в Mac OS і iOS, Android), в тому числі:

- 1C: Підприємство 7.7 (за допомогою зовнішнього компонента);
- 1C: Підприємство 8.3 (для зберігання записів журналу реєстрації);
- Adobe Photoshop Lightroom;
- FlylinkDC ++;

- AIMP ;
- Banshee;
- Calibre;
- Eserv;
- F-Spot;
- Nextcloud;
- FAR Manager (починаючи з версії 3.0);
- Gajim;
- Google Chrome;
- Miranda IM (за допомогою плагіна драйвера бази даних [14]);
- MyChat;
- Opera (починаючи з версії 10.50);
- qutIM;
- QGIS;
- Safari;
- XnView;
- Garena.

Об’єкт дослідження. Об’єктом даного дослідження є процес пошуку методів зберігання інформації та її відновлення в разі видалення або пошкодження файлів баз даних типу SQLite.

Предмет дослідження. Предметом дослідження є створення алгоритмів відновлення видаленої інформації з баз даних за показниками часової ефективності.

Мета і завдання дослідження. Метою даної роботи є визначення доцільності застосування алгоритмів відновлення інформації, а саме – можливість пришвидшити процес пошуку даних. Поставлена мета зумовлює необхідність вирішення наступного ряду завдань:

- розробка програмного середовища для проведення порівняльних експериментів;

- провести пошук аналогічних програмних продуктів та провести порівняльний аналіз.

Методи дослідження. Емпіричний науковий метод – проведення експериментів, аналіз та порівняння результатів роботи аналогічних програмних систем.

Практичне значення. Практичне значення роботи викладене наступним чином.

Результати проведених досліджень дозволяють встановити доцільність використання алгоритмів відновлення видаленої інформації з СКБД SQLite.

Отримані результати, а також інструментарій може бути використаний в у вищих навчальних закладах та науково-дослідницьких центрах задля подальших досліджень і пошуку покращених способів вирішення задач відновлення даних із СКБД.

1 ОПИС АНАЛІЗУ СУЧАСНИХ ВИДІВ БАЗ ДАНИХ ЗА НАУКОВИМИ ЛІТЕРАТУРНИМИ ДЖЕРЕЛАМИ

Одним з найважливіших компонентів будь-якої складної програмно-апаратної системи є зв’язка база даних (БД) – система управління базами даних (СУБД). Парадигма БД сформувалася в середині 60-років XX сторіччя як результат багатьох спроб позбавитися від недоліків файлових систем організації інформації. Основними з яких є [1]:

-ізольованість даних (програмні додатки, що працюють паралельно, не можуть одночасно змінювати записи в одному і тому ж файлі, а сумісна обробка файлів досить важка);

-залежність програм від даних (наявність опису структури даних в прикладній програмі потребує при зміні структури файлу внесення відповідних змін в програму);

-дублювання даних (різні програмні додатки, які використовують однакову інформацію про один і той же об’єкт, що зберігається в різних файлах, можуть потенційно призвести до порушення цілісності даних);

-відсутність опису даних (в файлах даних, які оброблюються прикладними програмами, дані зберігаються без відокремленого їхнього опису, що ускладнює документування інформаційної системи, а це потенційно призводить до появи помилок).

1.1 Постановка задачі

Нашим завданням є аналіз сучасного стану справ в області БД.

БД можна умовно розділити на типи в залежності від моделі даних, на основі якої вони побудовані. Тому аналіз існуючих БД будемо робити, відштовхуючись саме від моделі даних.

1.2 Історичний огляд розвитку БД

Основний розвиток СУБД почався в 60-роки XX сторіччя, коли корпорація IBM разом з фірмою NAA (North American Aviation) розробили першу СУБД – ієрархічну систему IMS (Information Management System) (див., наприклад, [2]). В середині 60-років поява системи IDS (Integrated Data Store) надала розвитку СУБД новий імпульс. Розвиток цієї СУБД призвів до появи нового типу СУБД – мережевого. В 1970 р. Е.Ф. Кодд в статті [3], визначив реляційну модель даних, яка усувала недоліки попередніх моделей. В запропонованій моделі Кодда можна було виділити три аспекти: цілісний, структурний і маніпуляційний. В основі реляційної моделі лежать структури даних на "плоских" нормалізованих відношеннях, обмеження цілісності визначаються засобами логіки першого порядку, а маніпуляції над даними відбуваються засобами реляційної алгебри (реляційного числення на доменах чи кортежах). Саме завдяки розвиненому в математиці апарату теорії множин, логіки першого порядку та теорії відношень реляційна модель набула широкого розповсюдження. В цій моделі Кодд запропонував 8 операцій реляційної алгебри: традиційні операції над таблицями, які розуміються як множини рядків (об'єднання, перетин, різниця) та спеціальні операції над таблицями (проекція, декартове з'єднання, з'єднання (theta-, equi-), ділення, селекція), що використовують специфіку таблиць і рядків [3]. Набір операцій реляційної алгебри, запропонований Коддом, з часом був розширений відповідно до потреб мов запитів. Крім вказаних вище операцій до сигнатури реляційної алгебри зараз також відносять операції перейменування та активного доповнення [4, 5].

Наступні типи СУБД, які з'явилися після реляційних, можна віднести до так званих "постреляційних". Так, в [6] застосовується поняття "постреляційні" БД для розширених реляційних, багатомірних та об'єктноорієнтованих СУБД.

Основи об'єктно-орієнтованого програмування (ООП), зокрема, об'єктна модель даних (ОМД), були закладені наприкінці 60-х років минулого сторіччя [7] і з цього моменту набувала вдосконалення. Поява ОМД пов'язана з поняттям абстрактних типів даних (АТД) [8, 9]. Саме ООП дало поштовх до створення в

середині 80-років минулого сторіччя нового типу СУБД – об’єктних. У 1996 р. з виходом СУБД Informix Universal Server корпорації Informix почалася епоха об’єктно-реляційних БД. Наприкінці 90-х років з’явився новий напрямок в розвитку СУБД – так звані NoSQL (Not only SQL) СУБД, націлені на обробку великих об’ємів слабоструктурованої інформації.

Зауважимо, що існують багато визначень терміну БД, які взаємно доповнюють одне одного. Так в [10] БД – це по суті, не що інше, як комп’ютеризована система збереження записів. В [11] БД визначається, в широкому розумінні, як набір логічно зв’язаних даних (або опис цих даних), призначений для задоволення інформаційних потреб організації. На більш конкретному рівні під БД розуміється єдине велике сховище даних, яке однократно визначається, а потім використовується одночасно багатьма користувачами. В [12] БД – це самодокументоване зібрання інтегрованих записів. БД є самодокументованою (selfdescribing), якщо вона містить з даними користувача опис власної структури. Цей опис називається словником даних (data dictionary), каталогом даних (data directory) або метаданими (metadata). Слід зауважити, що згідно [12] БД є моделлю моделі. БД не моделює реальність або якусь її частину, а є моделлю моделі користувача (user model). В [13] під БД розуміється сукупність призначених для машинної обробки даних, яка слугує для задоволення потреб багатьох користувачів в рамках однієї або декількох організацій. В [14] під БД розуміється набір даних, що знаходиться під контролем СУБД.

Історично першим типом БД по суті була файлова система, яка, як було сказано вище, мала ряд принципових недоліків, усунення яких спричинило подальший бурхливий розвиток БД.

1.3 Реляційні БД

В основі функціонування реляційних БД лежить реляційна модель даних. Наведемо одне із багатьох означень реляційної БД.

Реляційна БД – це скінченний набір відношень. Відношення використовуються для представлення сутностей та зв'язків між сутностями [15]. В основі реляційної моделі лежать такі складові [11]:

- відношення, фізичним представленням якого є "плоска" таблиця, що складається із стовпців та рядків; рядки відповідають окремим записам, а стовпці – атрибутам;
- атрибут – іменований стовпець відношення; атрибути можуть розташовуватися в таблиці в довільному порядку;
- домен – множина допустимих значень одного або декількох атрибутів; кожен атрибут визначається на деякому домені; домен може бути системним або створеним користувачем на основі системних доменів;
- кортеж (синоніми – рядок, запис) – рядок відношення; кортежі в таблиці можуть бути розташовані в довільному порядку, при цьому відношення буде залишатися таким же самим, а отже мати той же зміст.

Зробимо декілька зауважень згідно [4, 17] щодо основного поняття реляційних БД – поняття реляції (синоніми – таблиці, відношення). Як всяке інтуїтивне поняття реляція потенційно припускає багато уточнень. Спочатку як одне з таких уточнень Коддом було вибране класичне логіко-математичне поняття скінченномісного відношення як підмножини декартова добутку множин, взятих у деякому порядку. На початковому етапі таке уточнення обумовило успішність та популярність реляційного підходу в БД. Дійсно, змістовні поняття уточнювались і досліджувались за допомогою добре розвиненого формального апарату (теорія відношень, фрагменти мов логіки предикатів першого порядку та ін.) [3, 16].

Разом з тим, традиційне уточнення реляції має ряд недоліків. Наприклад, інформаційний зміст реляції, не залежить від порядку слідування компонент, а для відношення цей порядок суттєвий.

Використання стандартних імен 1,2,.. для доступу до компонент елементів реляції (кортежам у даному випадку) обтяжливе, що, зокрема, призводить до появи в

метамові, а не в мові-об’єкті довільних (“мнемонічних”) імен як позначень стандартних.

Результатом традиційного декартова добутку відношень є бінарне відношення, яке складають пари кортежів відношень-аргументів, а результатом декартова добутку (краще казати, з’єднання) відношень в реляційній моделі – відношення арності n+m, де n,m – арності відношень-аргументів, яке складають кортежі, що отримуються конкатенацією вихідних кортежів. Ця, на перший погляд, невелика різниця в означеннях призводить до суттєвих розбіжностей властивостей цих двох операцій: так декартове з’єднання асоціативне (що випливає з асоціативності конкатенації), а вихідний класичний декартів добуток – ні.

Аналогічно, стандартна операція проектування відношення за компонентою дає множину, а операція проектування відношення в реляційній моделі – відношення меншої арності, ніж відношення-аргумент [4].

На сьогоднішній час існує багато формальних моделей реляційних БД. Особливу увагу слід звернути на роботи [18, 19], які виконані під керівництвом професора Д.Б. Буя. В основі цих робіт лежить композиційний підхід до програмування, який був запропонований академіком НАН України В.Н. Редьком [20].

Наведемо основні результати цих робіт. В [18] задаються теоретико-множинні операції на таблицях, які є аналогами звичайних теоретикомножинних операцій об’єднання, перетину та різниці.

Згідно [4, 18] таблиця визначається наступним чином. Зафіксуємо наступні дві множини: A, елементи якої назвемо атрибутами, і D – універсальний домен. Довільну скінченну множину атрибутів $\subseteq$ назвемо схемою.

Рядком схеми R називається іменна множина на парі R, D (тобто множина пар вигляду <атрибут, значення>), проекція якої за першою компонентою рівна R.

Таблицею схеми R називається пара $(\hat{t}, R)$, де $\hat{t}$ – скінченна множина рядків схеми R, яка називається станом таблиці, друга компонента таблиці схеми називається, природньо, схемою. Відмітимо, що для непорожніх станів схема таблиці може бути відновлена по множині рядків.

Але для порожньої множини рядків це неможливе. Множину всіх рядків (таблиць) схеми R позначимо S(R) (відповідно T(R)), а множину всіх рядків (таблиць) – S (відповідно T). Таким чином, $\stackrel{def}{=} \cup\subseteq$, $\stackrel{def}{=} \cup\subseteq$. Схема може бути порожньою; при цьому існує єдиний рядок порожньої схеми, який позначається ε. Рядки будемо позначати **S**, **S1, S2,** ..., таблиці – t, t1, t2,... .

Далі буде розглянута таблична алгебра $\langle T, \boldsymbol{\Omega}\rangle$, де T − носій, $\boldsymbol{\Omega}$ − сигнатура.Під носієм табличної алгебри будемо розуміти множину всіх таблиць, під сигнатурою – множину операцій на таблицях, що складається з обмежень стандартних теоретико-множинних операцій об'єднання, перетину та різниці на множину односхемних таблиць, а також певних спеціальних операцій.

Розглянемо формальне означення деяких операцій над таблицями.

Об'єднання двох таблиць (схеми R):

$$\cup_R: T(R) \times T(R) \to T(R);$$
$$(\hat{t}_1, R) \cup_R (\hat{t}_2, R) = (\hat{t}_1 \cup \hat{t}_2, R).$$

Перетин двох таблиць (схеми R):

$$\cap_R: T(R) \times T(R) \to T(R);$$
$$(\hat{t}_1, R) \cap_R (\hat{t}_2, R) = (\hat{t}_1 \cap \hat{t}_2, R).$$

Різниця двох таблиць (схеми R):

$$\setminus_R: T(R) \times T(R) \to T(R);$$
$$(\hat{t}_1, R) \setminus_R (\hat{t}_2, R) = (\hat{t}_1 \setminus \hat{t}_2, R).$$

Також визначаються операції внутрішнього з'єднання, а саме декартове з'єднання (cross join, Cartesian join), внутрішнє природнє з'єднання (inner natural join),

внутрішнє з'єднання за атрибутами … 0 (inner join on … 0) та внутрішнє з'єднання за предикатом p (inner join on p). Наприклад, під декартовим з'єднанням

розуміється часткова операція вигляду

$$C_j: T \times T \nrightarrow T,$$
$$domC_j \stackrel{\text{def}}{=} \{(\hat{t}_1, R_1), (\hat{t}_2, R_2) | R_1 \cap R_2 = \emptyset\}.$$

Стан результуючої таблиці на таблицях аргументах $(\hat{t}_1, R_1), (\hat{t}_2, R_2)$ визначається формулою

$$\{s | \exists s_1 \exists s_2 (s_1 \in \hat{t}_1 \wedge s_2 \in \hat{t}_2 \wedge s = s_1 \cup s_2)\}.$$

Схема результуючої таблиці дорівнює $R_1 \cup R_2$.

Серед операцій зовнішнього з'єднання в роботі задаються зовнішнє ліве з'єднання (outer left join), зовнішнє праве з'єднання (outer right join), зовнішнє повне з'єднання (outer full join) та зовнішнє з'єднання об'єднанням (outer union join).

В [19] вперше введено до розгляду табличну алгебру нескінченних таблиць, яка є узагальненням табличної алгебри. Узагальнено класичні результати Кодда-Лакруа-Піротта щодо еквівалентності реляційних алгебр та числень на кортежах (доменах). При цьому числення поповнені предикатними та функціональними сигнатурами, у той час у формулюваннях згаданих класичних результатів розглядають лише бінарні предикати, а функціональна сигнатура взагалі порожня. Побудовано мультимножинну табличну алгебру, сигнатуру якої поповнено новими операціями: операціями внутрішніх і зовнішніх з'єднань, операцією напівз'єднання, агрегатними операціями.

1.4 Мова SQL для реляційних БД.

Реляційна модель тісно пов'язана з мовою SQL (Structured Query Language – мова структурованих запитів).

Фактично коли говориться про мову програмування для реляційних БД, то мається на увазі саме SQL [18]. Дамо одне із означень мови

SQL. SQL – формальна, непроцедурна мова програмування, що застосовується для створення, модифікації та керування даними в реляційній БД, яка керується відповідною СУБД.

Мову SQL можна поділити на дві частини: мову DDL (Data Definition Language) та мову DML (Data Manipulation Language). Засобами мови DDL можна створювати (оператор Create), модифіковувати (оператор Alter або Modify), знищувати (оператор Delete) елементи БД, а також можна керувати повноваженнями користувачів БД (оператори Grant, Revoke). Засобами мови DML можна здійснювати додавання (оператор Insert), знищення (оператор Delete), редагування (оператор Update) даних БД, а також їх вибірку (оператор запиту Select).

Для здійснення фільтрування рядків таблиці в мові SQL після службового слова Where використовуються предикати на рядках. Предикати можуть повертати третє логічне значення unknown разом з стандартними значеннями істини та хиби (true та false, відповідно). Це означає, що в SQL використовується трьохзначна логіка замість класичної двозначної (булевої), а саме це так звана сильна тризначна логіка Кліні.

Атомарні предикати з’єднуються в складні вирази булевими операціями кон’юнкції, диз’юнкції та заперечення, розширеними на трьохзначну логіку.

1.5 Оптимізація обробки запитів.

Одною з основних задач, які стоять перед СУБД, є “оптимізація обробки” запитів. Під “оптимізацією обробки” запитів СУБД розуміють стратегії, які підвищують ефективність процедур обробки запитів. Такі стратегії представляють собою евристичні методи, доцільність використання яких обґрунтовується статистичними методами. Поняття “запит”, по своїй суті, є виразом деякої мови і розглядається в наступних трьох контекстах [1].

Як стандартна вимога доступу користувача до даних БД (в цьому випадку “оптимізація” здійснюється за рахунок програмування відповідних процедур пошуку даних і перетворення входу користувача в результат необхідного формату).

Як транзакція, що здійснює зміну даних БД на підставі їх поточного значення.

Як вираз, який СУБД використовує для авторизації користувача, забезпечення цілісності даних і синхронізації багатокористувацького доступу до БД.

Однією з характеристик оптимізації запитів – є “вартість” запиту. Вартість запиту має декілька складових [1].

Комунікаційна вартість, тобто вартість передачі даних з їх місцеположення у вторинну пам'ять (пам’ять, з якої завантажуються дані для власне обчислення), а також вартість переміщення результату обчислення в його місце розташування.

Вартість доступу до вторинної пам’яті, тобто вартість завантаження порцій даних з вторинної пам'яті в основну пам’ять, яка використовується при обчисленні.

Вартість запам'ятовування, тобто вартість часу використання вторинної пам’яті і пам’яті буферів.

Вартість обчислення, тобто вартість часу, який використовується безпосередньо для обчислення.

1.6 Аномалії БД та їх усунення.

БД може зберігати досить великий об’єм інформації, яка розподілена між таблицями цієї БД. Через це виникає ситуація, коли одні і ті ж дані можуть знаходитися (дублюватися) в декількох таблицях. Тоді можна говорити про надлишковість такої інформації в БД, наявність якої в БД вказує на можливі проблеми підтримки цілісності. При роботі з таблицями БД, які мають надлишкові дані, виникають проблеми, що називаються

“аномаліями”. Можна умовно розділити аномалії на три види:

-аномалія включення – проблема, пов’язана з додаванням інформації в БД;

-аномалія модифікування – проблема, пов’язана зі змінами інформації в БД;

-аномалії знищення – проблема, пов’язана з видаленням інформації з БД.

Питання усунення аномалій БД постає на етапі проектування БД. Саме під час проектування БД за рахунок нормалізації повинні вирішуватися задачі мінімізації дублювання даних і спрощення методів їх оновлення та обробки. Нормалізацією називається формальна процедура, в ході якої атрибути даних групуються в таблиці, а таблиці групуються в БД. Задачами нормалізації є [21]:

- виключення в таблицях інформації, що повторюється;

-створення структури, в якій передбачена можливість її майбутніх змін;

-створення структури, в якій вплив структурних змін на додатки, що використовують дані цієї БД, зведено до мінімуму.

Коротко розглянемо формальні основи нормалізації реляційних БД. Вперше термін “нормалізація” був застосований у 1970 р. Е.Ф. Коддом для назви процедури усунення непростих доменів [3]. Історично поняття першої нормальної форми (1НФ) трансформувалося та уточнювалося. Так, спочатку це поняття уточнювалося наступним чином: “Відношення знаходиться в 1НФ, якщо кожне значення у відношенні є атомарним елементом даних” [10]. Потім цьому поняттю надавався більш уточнений зміст: “Відношення знаходиться в 1НФ, якщо домен кожного атрибута містить тільки атомарні значення і значення кожного атрибуту набувають тільки повних значень з цього домену”. З появою складних типів даних в СУБД, поняття 1НФ уточнювалося наступним чином: “Зміннавідношення знаходиться у 1НФ тоді і тільки тоді, коли в кожному її допустимому значенні кожний кортеж містить тільки одне значення для кожного атрибуту” [10]. У 1971 р. Е.Ф. Кодд вказує на надлишковість даних та аномалії, які виникають при здійсненні операцій над відношеннями, вперше представляє концепцію функціональної залежності (ФЗ), демонструє можливість її використання для розв’язання проблем проектування БД, наводить означення другої нормальної форми (2НФ), транзитивної ФЗ та третьої нормальної форми (3НФ).

Означення транзитивної ФЗ (див. наприклад, фундаментальну монографію Е.Ф. Кодда [22]) сформульоване Коддом у вигляді: “Нехай , ? і 7 різні атрибути у відношенні і нехай вони задовольняють наступним умовам: виконуються

ФЗ A → B, B → C та не виконується ФЗ B → A.

Тоді ФЗ A→ C називається транзитивною залежністю”, на даний час має два уточнення [21]:

1) атрибут C € A ∪B, де A і B – множини атрибутів [4]; дана умова виключає з розгляду тривіальні ФЗ (ФЗ називається тривіальною, якщо її права частина є підмножиною лівої); 2) B ⊈ A, де і ? – множини атрибутів [23]; цим також виключаються тривіальні ФЗ в посилці A → B.

Для приведення відношення до 3НФ користуються багатьма алгоритмами [23]. Так, Хез (Heath) сформулював теорему, яка обґрунтовувала приведення відношення до 3НФ за допомогою декомпозиції без втрат відношення

R(A, B, C) на проекції $R_1 = \pi_R(A, B)$ та $R_2 = \pi_R(A, C)$ за умови виконання ФЗ A → B. Також на сьогодні відомими класичними алгоритмами зведення схеми відношення до 3НФ є алгоритми Ульмана, Делобеля-Гейсі (DelobelGasey), результатами яких не завжди є схема у 3НФ, Берштейна (Bernstein), Іслура (Isloor), Неклюдової-Цаленка, який дає кількісно оптимальну схему БД; оригінальний алгоритм зведення схеми відношення до 3НФ через побудову так званих кільцевих покриттів запропонував Мейер [5]. Переваги та недоліки більшості з вказаних алгоритмів, а також їх відповідність різним визначенням еквівалентності реляційних схем розглянуті у монографії В.П. Дрібаса [23]. Пошук ефективних алгоритмів розв’язання задачі синтезу оптимальної схеми БД у 3НФ продовжується і сьогодні (див. огляд [21]).

1.7 Об’єктні БД.

Складна будова об’єктивної дійсності спонукає людину використовувати класифікаційні схеми, які дозволяють цілісно представляти цю реальність у вигляді

об’єктів. Тому абсолютно природнім стало впровадження підтримки об’єктного підходу в мови програмування та в БД, зокрема. В основі об’єктного підходу лежить ОМД. В [11] ця модель визначається як модель, що враховує семантику об’єктів – головного поняття ООП. Основна ідея об’єктного підходу полягає в об’єднанні даних і операцій, які виконуються над цими даними, в одне концептуально замкнуте поняття – клас. Дані класу не повинні змінюватися ззовні, а доступ до даних слід здійснювати тільки через функціїчлени (методи класу). Програма, яка написана на об’єктній мові, взаємодіє з сукупностю об’єктів, кожен з яких належить до певного АТД (класу) і має інтерфейс у вигляді сукупності методів для взаємодії з іншими об’єктами за допомогою повідомлень.

Розробка систем об’єктних БД (ОБД) розпочалася в 80-х роках минулого століття в зв’язку з впровадженням складних програмних додатків, пов’язаних з автоматизованим проектуванням (Computer Aided Design – CAD), автоматизованим виробництвом (Computer Aided Manufacturing – CAM), системами, які основані на знаннях, тощо. Використання реляційного підходу в таких застосуваннях показало його неефективність. Іншими словами, поява ОБД була зумовлена більш адекватним представленням та моделюванням сутностей реального світу в ОБД в порівнянні з реляційними БД.

Об’єктна теорія побудована з використанням базових понять об’єктного підходу Г. Буча та трикутника Г. Фреге, виходячи з наступних принципів [24, 25]:

-загальності об'єктного визначення: всі сутності – об'єкти;

-унікальності: кожен об’єкт – унікальний елемент;

-об’єктної впорядкованості: всі об’єкти впорядковані у відповідності з певними відношеннями;

Цілісності об’єктної моделі: об’єкти і відношення між ними однозначно визначаються в моделі на певному рівні абстракції;

• інтероперабельності об’єктів: об’єкти пов’язуються операціями викликів на множинах вхідних і вихідних інтерфейсів.

Треба зазначити, що об’єктному підходу в програмуванні та в БД присвячено багато праць. Та, відповідно, існує багато визначень ООП та ООБД. Ми будемо визначати ООБД, як БД, в яких об’єктний підхід поєднується з функціональністю БД. Цей підхід надає можливість більш точно моделювати предметну область. В свою чергу, функціональність об’єктних СУБД потрібна для забезпечення цілісності даних, а також паралельного доступу до них. Використання ООП надає суттєві переваги ООБД. Спільним для різних ООБД є використання об’єктноорієнтованих мов програмування, основними властивостями яких є інкапсуляція, поліморфізм, успадкування. Отже, формалізація успадкування класів є кроком у створенні загального підходу до побудови ООБД та їхньої роботи в цілому.

Обов’язковою складовою ОБД є, по-перше, можливість маніпулювання об’єктами, що представляються у вигляді сутностей в адресному просторі обчислювальної системи, яка з’являється при створенні екземпляра класу, та, по-друге, використання концепцій ООП. Разом з цим, кожна одиниця (об’єкт) інформації в таких БД володіє двома характеристиками: станом та поведінкою. Стан визначається сукупністю поточних значень атрибутів та зв’язків, які має об’єкт. Значеннями атрибутів можуть виступати як елементи примітивних (системних) типів даних, так і інші об’єкти. Останній факт дає можливість визначати об’єкт рекурсивно через інші об’єкти. Стан визначає реакцію об’єкта на подію, яка в нього поступає. Об’єкт зберігає свій стан на протязі часу між двома послідовними подіями, які він приймає.

Поведінка визначається можливістю об’єкта реагувати на дії ззовні. Реагування на дії ззовні відбувається через методи об’єкта, які в нього інтегровані. Кожен об’єкт в БД володіє унікальним ідентифікатором OID (Object Identifier). Об’єкти, які мають однакові схеми та поведінку, групуються в класи, створюючи при цьому інстанс

(instance) класу. Об’єкт може належати як інстансу одного класу, так і інстансу декількох класів у випадку звичайного та множинного успадкування [26].

Існує наступна відмінність у використанні типів даних атрибутів. Так, в реляційній БД, як правило, атрибути мають прості (примітивні) типи даних, а в ООБД значеннями атрибутів можуть бути складні типи даних. Причому складні типи даних можуть бути вбудовані в СУБД, а можуть бути створені користувачем

(user-defined types).

1.7.1 Запити в об’єктних БД.

Для опису та маніпулювання інформацією в об’єктних БД використовують відповідні мови. За аналогією з реляційними СУБД в об’єктних існує мова опису об’єктів ODL (Object Definition Language), що дозволяє описати структуру даних та схему БД для об’єктної моделі, розробленої консорціумом ODMG (Object Database Management Group), та мова OQL (Object Query Language), що слугує для пошука об’єктів у БД. З іншого боку, існують такі об’єктні СУБД (NeoDatis, db4o, objectdb, тощо), в яких створення схеми класу та екземплярів класу покладається на мову програмування (Java, C++, тощо). Такі СУБД надають користувачу бібліотеки (класи) для взаємодії з БД.

В кожній розвиненій об’єктній СУБД присутні наступні операції над об’єктами, які можна позначити абревіатурою CRUD: створення об’єкту та поміщення його в БД (Create, Storing), зчитування об’єкту з БД (Retriving), зміна об’єкту з подальшим його записом в БД (Update), знищення об’єкту з БД (Deleting).

На відміну від реляційних СУБД, в яких існує єдиний підхід до реалізації вибірки даних (мова SQL), в об’єктних СУБД існує декілька шляхів реалізації цієї можливості: Java Persistence Query

- Language (JPQL), Query by Example (QBE), Native ;
- Queries (NQ), Simple Object Database Access ;

- (S.O.D.A) та інші. Реалізацію вибірки даних обирає безпосередньо користувач ООБД в залежності від поставленої задачі.

1.7.2 Об’єктно-реляційні БД (ОРБД)

В основі функціонування об’єктно-реляційних БД лежить інтеграція елементів ОМД в реляційну модель. Фактично ОРБД представляє собою розширення функціональних можливостей реляційних БД. Прикладами такого розширення може виступати:

1) збереження в атрибуті значень складних типів даних;
2) реалізація успадкування таблиць;
3) надання рядку таблиці об’єктного ідентифікатора, який відіграє роль первинного ключа (primary key) таблиці.

Зауважимо, що реалізація ідей об’єктного підходу для реляційних БД дещо відрізняється від реалізації безпосередньо в об’єктному підході. Покажемо це на прикладі. Розглянемо успадкування таблиць в об’єктно-реляційній СУБД postgresql v. 9.2. Нехай маємо дві таблиці cities (міста) та capitals (столиці). Створимо ці таблиці (тобто задамо їх імена та схеми):

```
CREATE TABLE cities (
name       text,
population real,
altitude   int );
```

```
CREATE TABLE capitals (
  state      char(2)
) INHERITS (cities);
```

Рисунок 1

Таблиця capitals успадковує всі атрибути батьківської таблиці cities. Тоді, коли додається запис в таблицю capitals, цей же запис додається в таблицю cities (без атрибута state). Як відомо, в ООП при створенні об’єкта похідного класу цей об’єкт не створюється в батьківському класі.

1.7.3 Object Relation Mapping (ORM)

Слід відмітити, що існує принципова можливість перетворити об’єктну модель в реляційну. При такому підході, в загальному випадку, кожному класу буде відповідати певне відношення (таблиця). Хоча програмні системи, які реалізовують ORM, можуть одному класу співставити більше ніж одну реляційну таблицю та декільком класам співставити одну реляційну таблицю. Атрибути класу стануть атрибутами відповідного відношення, а екземпляру класу буде відповідати певний рядок у відношенні. Цю можливість можна реалізувати за допомогою так званого відображення клас-відношення (ORM – Object Relation Mapping). Така задача виникає у випадку, коли прикладна програма, що виступає клієнтом по відношенню до БД, реалізована на об’єктно-орієнтованій мові програмування, а дані зберігаються в реляційній БД.

Щодо практичної реалізації, то існує багато програмних продуктів, які надають можливість користування ORM. Серед таких програм слід виділити Hibernate. Крім того, можна здійснювати зворотне перетворення реляційної моделі в об’єктну. Така задача може виникнути, коли ми хочемо в об’єктному стилі переписати програмуклієнт, створену для роботи з реляційною БД [27].

1.8 NoSQL БД

З кожним роком кількість інформації, яка повинна зберігатися та оброблюватися в БД, стрімко зростає. Також сучасні СУБД стикаються з інформацією, яка не є структурованою, зростають вимоги до надзвичайно швидкого виконання запитів та оновлення фізично розподілених даних. Ця тенденція вимагає постійного підвищення функціональних можливостей як серверів (hardware: підвищення тактової частоти процесорів, пам’яті, зменшення часу записузчитування інформації з накопичувачів), так і реалізації механізмів підвищення швидкодії роботи з інформацією в самої БД на логічному рівні (software). Якщо розглянемо реляційні БД, то для підтримки цілісності та прискорення виконання запитів на великому об’ємі інформації були реалізовані механізми: нормалізація, денормалізація та шардінг

(sharding, так зване вертикальне та горизонтальне масштабування, яке є архітектурним рішенням). Суть шардінгу полягає в розділенні БД на окремі частини з подальшим перенесенням цих частин на окремі сервери. Шардінг можно розділити на два типи: вертикальний та горизонтальний. Вертикальний шардінг – це спосіб виділення таблиці або декількох таблиць з метою подальшого перенесення на інший сервер БД. Горизонтальний шардінг – це спосіб розділення однієї таблиці на частини з метою перенесення кожної частини на окремий сервер. І ці механізми давали певний час змогу пришвидшити роботу реляційних БД, в якій зберігається досить велика кількість інформації. Але з часом навіть ці механізми не дозволяли виконувати покладені на них завдання по обробці великих об'ємів інформації. В результаті перед розробниками програмного забезпечення постало завдання знайти нові механізми пришвидшення обробки інформації в БД. Це і дало поштовх для розвитку нового типу БД – NoSQL (Not Only SQL – не тільки SQL). Зауважимо, що термін NoSQL визначає скоріше напрямок розвитку БД, а ніж технологію. За моделлю даних БД типу NoSQL можна умовно поділити на 4 типи:

1) документо-орієнтовані (OrientDB, MongoDB);
2) БД типу "ключ-значення" (Redis, Memcached);
3) стовпцеві БД (Cassandra, HBase);
4) графові БД (OrientDB, HyperGraphDB, Infinite Graph).

Треба зауважити, що існують такі NoSQL СУБД, які не можна однозначно віднести до якогось одного з перелічених вище типів. Це пов'язано з тим, що границі між вказаними вище моделями даних дуже часто розмиті і такі БД не можна однозначно віднести до одного типу. Наприклад, СУБД OrientDB можна одночасно віднести як до типу документо-орієнтованих, так і графових.

Незважаючи на те, що NoSQL СУБД мають переваги, в першу чергу для обробки і збереження великої кількості інформації, вони все ж таки мають недолік, а саме можливість агрегувати структури даних. В цьому контексті реляційні БД мають перевагу завдяки відсутності агрегатної структури, яка дозволяє здійснювати

різноплановий доступ до даних. Окрім цього, вони забезпечують зручний механізм, який дозволяє здійснювати пошук даних незалежно від того, як ці дані зберігаються [28]. Тому при виборі відповідної моделі даних (СУБД) потрібно підходити виважено.

Поширене використання NoSQL БД призвело до створення формальних моделей таких БД. Так, в роботі [29] побудовано дві моделі даних, що уточнюють структури даних документоорієнтованих БД. В основі побудови першої моделі лежить теорія множин, в основі другої – теорія мультимножин, яка дає можливість використовувати дублікати записів в межах одного документа. Введено відношення піддокументу та підзапису, які уточнюють входження одного документа в інший.

Висновки

В статті зроблений аналіз типів БД, які поширено використовуються в інформаційних системах. Розповсюджене використання, стабільність роботи реляційних БД забезпечується наявністю формальних моделей та мови запитів SQL. Розглянуті характеристики мови SQL, а також мов запитів для інших типів БД. Для реляційних СУБД існує єдиний підхід до реалізації вибірки даних (мова SQL), в об’єктних СУБД єдиного підходу не існує. Для об’єктних (постреляційних), NoSQL БД також побудовані формальні моделі, які описують роботу таких БД.

2 ОПИС І АНАЛІЗ ФОРМАТУ ФАЙЛІВ БАЗ ДАНИХ ТИПУ SQLite ТА СТРУКТУР ДАНИХ, ЩО ВИКОРИСТОВУЮТЬСЯ ДЛЯ ЗБЕРЕЖЕННЯ ІНФОРМАЦІЇ

2.1. Файл бази даних

Повний стан бази даних SQLite зазвичай міститься в одному файлі на диску, який називається "основний файл бази даних".

Під час транзакції SQLite зберігає додаткову інформацію у другому файлі, який називається "журнал відкату", або якщо SQLite знаходиться в режимі WAL , - файл журналу попереднього запису.

2.1.1. Гарячі журнали

Якщо програма або хост-комп'ютер виходять з ладу до завершення транзакції, журнал відкату або журнал попереднього запису містить інформацію, необхідну для відновлення основного файлу бази даних до стійкого стану. Коли журнал відкату або журнал попереднього запису містить інформацію, необхідну для відновлення стану бази даних, вони називаються "гарячим журналом" або "гарячим WAL-файлом". Гарячі журнали та файли WAL є лише фактором під час сценаріїв відновлення помилок, тому вони нечасті, але вони є частиною стану бази даних SQLite і тому їх не можна ігнорувати. Цей документ визначає формат журналу відката та журналу журналу попереднього запису, але акцент робиться на головному файлі бази даних.

2.1.2. Сторінки

Основний файл бази даних складається з однієї або декількох сторінок. Розмір сторінки - потужність двох між 512 і 65536 включно. Всі сторінки в одній базі даних однакового розміру. Розмір сторінки для файлу бази даних визначається 2-байтовим цілим числом, розташованим у зміщенні 16 байт від початку файлу бази даних.

Сторінки нумеруються, починаючи з 1. Максимальний номер сторінки - 2147483646 (2 31 - 2). База даних SQLite мінімального розміру - це одна 512-байтна сторінка. База даних максимального розміру складе 2147483646 сторінок при 65536 байт на сторінку або 140,737,488,224,256 байт (близько 140 терабайт). Зазвичай

SQLite буде досягти максимального обмеження розміру файлу базової файлової системи або дискового обладнання задовго до того, як він досягне власного внутрішнього обмеження розміру.

Зазвичай, бази даних SQLite мають тенденцію коливатися в розмірі від кількох кілобайт до кількох гігабайт, хоча відомо, що у виробництві існують бази даних SQLite, що мають терабайт.

У будь-який момент часу кожна сторінка основної бази даних використовує одноразове використання, яке є одним із наступних:

- Сторінка блокування;
- Сторінка фрілістів;
- Сторінка вільного перекладу;
- Вільна сторінка листів;
- Сторінка b-дерева;
- Внутрішня сторінка таблиці дерева b;
- Сторінка листівки b-дерева;
- Внутрішня сторінка b-дерева із покажчиком;
- Сторінка аркуша b-дерева;
- Сторінка переповнення корисного навантаження;
- Сторінка карти вказівникаю.

Усі записи з основного файлу бази даних починаються з межі сторінки, а всі записи - це ціла кількість розмірів сторінок. Читання також зазвичай є цілим числом сторінок за розміром, за винятком того, що коли база даних вперше відкривається, перші 100 байт файлу бази даних (заголовок файлу бази даних) читаються як одиниця розміру підсторінки.

Перш ніж будь-яка інформаційна сторінка бази даних буде змінена, оригінальний немодифікований вміст цієї сторінки записується в журнал відкату. Якщо транзакція перервана і її потрібно повернути назад, журнал відката може бути використаний для відновлення бази даних до початкового стану. Сторінки аркушів фрілістів не містять жодної інформації, яку потрібно було б відновити під

час відкату, і тому вони не записуються в журнал до внесення змін, щоб зменшити введення / виведення диска.

2.1.3. Заголовок бази даних

Перші 100 байт файлу бази даних містять заголовок файлу бази даних. Заголовок файлу бази даних розділений на поля, як показано в таблиці нижче. Усі багатобайтові поля в заголовку файлу бази даних зберігаються першим найзначнішим байтом (big-endian).

Таблиця 2.1. Формат заголовка бази даних

Зсув	Розмір	Опис
0	16	Рядок заголовка: "Формат SQLite 3 \ 000"
16	2	Розмір сторінки бази даних у байтах. Повинна бути потужність двох між 512 і 32768 включно, або значення 1, що становить розмір сторінки 65536.
18	1	Версія для запису у форматі файлу. 1 за спадщиною; 2 для WAL .
19	1	Формат файлу для читання версії. 1 за спадщиною; 2 для WAL .
20	1	Байти невикористаного "зарезервованого" місця в кінці кожної сторінки. Зазвичай 0.
21	1	Максимальна частка вбудованої корисної навантаження. Повинно бути 64.
22	1	Мінімальна частка вбудованої корисної навантаження. Повинно бути 32.
23	1	Фракція корисного навантаження листя. Повинно бути 32.
24	4	Лічильник зміни файлів.

28	4	Розмір файлу бази даних у сторінках. "Розмір бази даних в заголовку".
32	4	Номер сторінки першої сторінки вільного списку.
36	4	Загальна кількість сторінок вільного списку.
40	4	Печиво схеми.
44	4	Номер формату схеми. Підтримувані формати схем складають 1, 2, 3 і 4.
48	4	Стандартний розмір кешу сторінки
52	4	Номер сторінки найбільшої кореневої сторінки b-дерева в режимах автоматичного вакуумування або поступового вакууму, або нуль інакше.
56	4	Кодування тексту бази даних. Значення 1 означає UTF-8. Значення 2 означає UTF-16le. Значення 3 означає UTF-16be.
60	4	"Користувацька версія" як прочитана та встановлена прагмою user_version .
64	4	Справжнє (не нульове) для інкрементально-вакуумного режиму. Неправдиво (нуль) інакше.
68	4	"Ідентифікатор програми", встановлений PRAGMA application_id .
72	20	Зарезервовано для розширення. Повинно бути нульовим.
92	4	Версій закінчення дії для номера .
96	4	SQLITE_VERSION_NUMBER

2.1.3.1. Чарівний рядок заголовка

Кожен дійсний файл бази даних SQLite починається з наступних 16 байт (у шістнадцятковій формі): 53 51 4c 69 74 65 20 66 6f 72 6d 61 74 20 33 00. Ця послідовність байтів відповідає рядку UTF-8 "**SQLite format 3**", включаючи символ нульового термінатора в кінці.

2.1.3.2. Розмір сторінки

Двобайтне значення, що починається зі зміщення 16, визначає розмір сторінки бази даних. Для версій SQLite 3.7.0.1 (2010-08-04) та новіших версій це значення інтерпретується як ціле число великого ендіану і має бути потужністю двох між 512 та 32768 включно. Починаючи з версії 3.7.1 SQLite (2010-08-23), підтримується розмір сторінки в 65536 байт. Значення 65536 не вміститься в двобайтове ціле число, тому для визначення розміру сторінки 65536 байт значення при зміщенні 16 становить 0x00 0x01. Це значення можна інтерпретувати як великий ендіаніст 1 і розглядати його як магічне число для відображення розміру сторінки 65536. Або можна переглянути двобайтове поле як невелике ендіатичне число і сказати, що воно являє собою розмір сторінки, розділений на 256. Ці дві інтерпретації поля розміру сторінки еквівалентні.

2.1.3.3. Номери версій у форматі файлу

Версія для запису формату файлу та версія для зчитування формату файлу при зміщеннях 18 та 19 призначені для покращення формату файлів у майбутніх версіях SQLite.

У поточних версіях SQLite обидва ці значення мають 1 для режимів зворотного прокату і 2 для режиму WAL журналу. Якщо версія SQLite, закодована до поточної специфікації формату файлу, зустрічається з файлом бази даних, де версія для читання дорівнює 1 або 2, але версія запису перевищує 2, то файл бази даних повинен розглядатися як лише для читання. Якщо зустрічається файл бази даних з прочитаною версією, що перевищує 2, то цю базу даних не можна читати чи записувати.

2.1.3.4. Зарезервовані байти на сторінку

SQLite має можливість виділяти невелику кількість зайвих байтів у кінці кожної сторінки для використання розширеннями. Ці додаткові байти використовуються, наприклад, розширенням шифрування SQLite для зберігання нон-цензури та/або криптографічної контрольної суми, пов'язаної з кожною сторінкою. Розмір «зарезервованого простору» в 1-байтовому цілому цілому при зміщенні 20 - це кількість байтів пробілу в кінці кожної сторінки, яку потрібно резервувати для розширень. Зазвичай це значення 0. Значення може бути непарним.

"Використовуваний розмір" сторінки бази даних - це розмір сторінки, визначений 2-байтовим цілим числом при зміщенні 16 у заголовку за вирахуванням "зарезервованого" розміру простору, записаного в 1-байтовому цілому цілому при зміщенні 20 у заголовку. Корисний розмір сторінки може бути непарним номером. Однак допустимий розмір не може бути меншим за 480. Іншими словами, якщо розмір сторінки становить 512, розмір зарезервованого простору не може перевищувати 32.

2.1.3.5. Фракції корисного навантаження

Максимальні та мінімальні вбудовані фракції корисного навантаження та значення фракції корисного навантаження листів повинні бути 64, 32 та 32. Спочатку ці значення були призначені для налаштування параметрів, які можна було б використовувати для зміни формату зберігання алгоритму b-дерева. Однак ця функціональність не підтримується, і немає поточних планів додавати підтримку в майбутньому. Отже, ці три байти фіксуються за вказаними значеннями.

2.1.3.6. Лічильник зміни файлів

Лічильник зміни файлів - це 4-байтне ціле велике ціле число зі зміщенням 24, яке збільшується щоразу, коли файл баз даних розблокується після зміни. Коли два чи більше процесів читають один і той же файл бази даних, кожен процес може виявити зміни бази даних від інших процесів, контролюючи лічильник змін. Процес, як правило, хоче стерти кеш сторінки своєї бази даних, коли інший процес змінив базу даних, оскільки кеш став розмитим. Лічильник зміни файлів полегшує це.

У режимі WAL зміни бази даних виявляються за допомогою wal-index, і тому лічильник змін не потрібен. Отже, лічильник змін може не збільшуватися для кожної транзакції в режимі WAL.

2.1.3.7. Розмір бази даних у заголовку

4-байтне ціле велике ендіанське значення зі зміщенням 28 у заголовку зберігає розмір файлу бази даних на сторінках. Якщо цей розмір розміру даних в заголовку недійсний (див. Наступний параграф), то розмір бази даних обчислюється, переглядаючи фактичний розмір файлу бази даних. Старіші версії SQLite ігнорували розмір бази даних у заголовку та використовували виключно фактичний розмір файлу. Новіші версії SQLite використовують розмір бази даних у заголовку, якщо він доступний, але повертається до фактичного розміру файлу, якщо розмір бази даних у заголовку недійсний.

Розмір бази даних в заголовку вважається дійсним лише в тому випадку, якщо він не дорівнює нулю і якщо 4-байтовий лічильник зміни при зміщенні 24 точно відповідає 4-байтовій версії-дійсно для номерапри зміщенні 92. Розмір бази даних у заголовку завжди дійсний, коли база даних змінюється лише за допомогою останніх версій SQLite, версій 3.7.0 (2010-07-21) та новіших версій. Якщо застаріла версія SQLite пише в базу даних, вона не буде знати оновлення розміру бази даних у заголовку, і тому розмір бази даних у заголовку може бути невірним. Але застарілі версії SQLite також залишать змінну версії для номера при зміщенні 92 незмінною, щоб вона не відповідала лічильнику змін. Отже, недійсні розміри баз даних у заголовку можна виявити (ігнорувати), спостерігаючи, коли лічильник змін не відповідає номеру версії-дійсності.

2.1.3.8. Безкоштовний список сторінок

Невикористані сторінки у файлі бази даних зберігаються у вільному списку. 4-байтне ціле велике ендіанське значення зі зміщенням 32 зберігає номер сторінки першої сторінки фрілісте або нуль, якщо фріліст порожній. 4-байтне ціле велике ендіанське значення в зсуві 36 магазинів зберігає загальну кількість сторінок у вільному списку.

2.1.3.9. Cookie схеми

Файл cookie схеми - це 4-байтне ціле велике ендіанське значення зі зміщенням 40, яке збільшується кожного разу, коли змінюється схема бази даних. Підготовлений оператор складається відповідно до конкретної версії схеми бази даних. Коли схема бази даних змінюється, оператор повинен бути підготовлений. Коли підготовлений оператор запускається, він спочатку перевіряє файл cookie схеми, щоб переконатися, що значення є таким самим, як коли було підготовлено оператор, і якщо файл cookie схеми змінився, виписка або автоматично повторно перетворюється та перезавантажується, або переривається з помилкою SQLITE_SCHEMA .

2.1.3.10. Номер формату схеми

Номер формату схеми - це 4-байтне ціле велике ендіанське ціле число при зміщенні 44. Номер формату схеми схожий на формат файлу для читання і запису номерів версій при зміщеннях 18 і 19, за винятком того, що номер формату схеми відноситься до високого рівня SQL форматування, а не низькорівневого форматування b-дерева. Наразі визначено чотири номери формату схеми:

Формат 1 розуміється всіма версіями SQLite до версії 3.0.0 (2004-06-18).

Формат 2 додає можливість рядків у одній таблиці мати різну кількість стовпців, щоб підтримувати функцію ALTER TABLE ... ADD COLUMN . Підтримка формату читання та запису 2 була додана у версії 3.1.3 SQLite 2005-02-20.

Формат 3 додає можливість додаткових стовпців, доданих ALTER TABLE ... ADD COLUMN, щоб мати значення, які не є NULL за замовчуванням. Ця можливість була додана у версії 3.1.4 SQLite 2005-03-11.

Формат 4 змушує SQLite поважати ключове слово DESC у деклараціях індексу. (Ключове слово DESC ігнорується в індексах для форматів 1, 2 і 3.) Формат 4 також додає два нових булевих значення типу запису (серійні типи 8 і 9). Підтримка формату 4 була додана в SQLite 3.3.0 2006-01-10.

Нові файли бази даних, створені SQLite, використовують формат 4 за замовчуванням. Legacy_file_format прагма може бути використаний , щоб викликати SQLite для створення нових файлів бази даних з використанням формату 1. Формат

номера версії можна за замовчуванням 1 замість 4, встановивши SQLITE_DEFAULT_FILE_FORMAT = 1 під час компіляції.

2.1.3.11. Запропонований розмір кешу

4-байтне ціле число з великим ендіанським підписом при зміщенні 48 є запропонованим розміром кешу на сторінках для файлу бази даних. Значення є лише пропозицією, і SQLite не зобов'язаний його шанувати. В якості запропонованого розміру використовується абсолютне значення цілого числа. Запропонований розмір кешу можна встановити, використовуючи прагму default_cache_size .

2.1.3.12. Поступові параметри вакууму

Два цілих 4-байтові цілі великі ендіани при зміщеннях 52 та 64 використовуються для керування режимами автовакууму та інкрементального вакууму . Якщо ціле число зі зміщенням 52 дорівнює нулю, сторінки файла вказівника (ptrmap) опускаються з файлу бази даних, і не підтримується ні auto_vacuum, ні incremental_vacuum. Якщо ціле число зі зміщенням 52 не дорівнює нулю, то це номер сторінки найбільшої кореневої сторінки у файлі бази даних, файл бази даних містить сторінки ptrmap, а режим повинен бути або auto_vacuum, або incremental_vacuum. В останньому випадку ціле число при зміщенні 64 є істинним для інкрементального_вакууму і хибним для авто_вакууму. Якщо ціле число при зміщенні 52 дорівнює нулю, то ціле число при зміщенні 64 також має бути рівним нулю.

2.1.3.13. Кодування тексту

4-байтове ціле число великого ендіана при зміщенні 56 визначає кодування, яке використовується для всіх текстових рядків, що зберігаються в базі даних. Значення 1 означає UTF-8. Значення 2 означає UTF-16le. Значення 3 означає UTF-16be. Інші значення не дозволяються. Файл заголовка sqlite3.h визначає макроси C-препроцесора SQLITE_UTF8 як 1, SQLITE_UTF16LE як 2, а SQLITE_UTF16BE як 3, щоб використовувати замість числових кодів для кодування тексту.

2.1.3.14. Номер версії користувача

4-байтове ціле число великого ендіану при зміщенні 60 - це версія користувача, яка встановлюється і запитується прагмою user_version . Користувацьку версію SQLite не використовує.

2.1.3.15. Ідентифікатор програми

4-байтне ціле велике ендіанське значення зі зміщенням 68 - це "ідентифікатор програми", який може бути встановлений командою PRAGMA application_id з метою ідентифікації бази даних як належної або асоційованої з певною програмою. Ідентифікатор програми призначений для файлів баз даних, що використовуються як формат файлу програми . Ідентифікатор програми може використовуватися утилітами, такими як **file**, щоб визначити конкретний тип файлу, а не просто звітувати про "SQLite3 База даних". Список призначених ідентифікаторів додатків можна переглянути, звернувшись до файла magic.txt у сховище джерела SQLite.

2.1.3.16. Номер версії бібліотеки SQLite

4- байтне ціле велике ендіанське значення зі зміщенням 96 зберігає значення SQLITE_VERSION_NUMBER для бібліотеки SQLite, яка нещодавно змінила файл бази даних. 4-байтове ціле число великого ендіана при зміщенні 92 - це значення лічильника змін, коли зберігався номер версії. Ціле число зі зміщенням 92 вказує, для якої транзакції дійсний номер версії, а іноді його називають "номером версії-дійсності".

Простір заголовка відведено для розширення**.** Усі інші байти заголовка файлів бази даних зарезервовані для подальшого розширення і повинні бути встановлені в нуль.

2.1.4. Сторінка блокування

Сторінка блокування байтів - це єдина сторінка файлу бази даних, яка містить байти із зрушеннями між 1073741824 та 1073742335, включно. Файл бази даних розміром менше 1073741824 байт не містить жодної сторінки блокування. Файл бази даних більше 1073741824 містить рівно одну сторінку блокування байтів.

Сторінка блокування байта забороняється для використання спеціальною реалізацією VFS для операційної системи при реалізації примітивів блокування файлів бази даних. SQLite не використовує сторінку блокування байтів. Ядро SQLite ніколи не читатиме і не записує сторінку блокування байтів, хоча реалізація VFS, визначена операційною системою, може вибрати читання або запис байтів на сторінці блокування байтів відповідно до потреб і розбірливості базової системи. Реалізації VFS unix і win32 , вбудовані в SQLite, не записуються на сторінку блокування байтів, однак сторонні реалізації VFS для інших операційних систем можуть бути.

Сторінка блокування байта виникла з необхідності підтримувати Win95, яка була переважаючою операційною системою при розробці цього формату файлів і яка підтримувала лише обов'язкове блокування файлів. Усі сучасні операційні системи, які нам відомі, підтримують дорадче блокування файлів, і тому сторінка блокування байтів вже насправді не потрібна, але зберігається для зворотної сумісності.

2.1.5. Фріліст

Файл бази даних може містити одну або кілька сторінок, які не використовуються. Невикористані сторінки можуть виникати, наприклад, при видаленні інформації з бази даних. Невикористані сторінки зберігаються у вільному списку та повторно використовуються, коли потрібні додаткові сторінки.

Фріліст організований у вигляді зв'язаного списку сторінок магістральних списків із кожною сторінкою магістралі, що містить номери сторінок для нуля або більше листів археологічних листів.

Сторінка стовбура фріліста складається з масиву 4-байтових цілих чисел великих величин. Розмір масиву становить стільки ж цілих чисел, скільки вміститься у просторі сторінки. Мінімальний простір - 480 байт, тому масив завжди матиме щонайменше 120 записів. Перше ціле число на сторінці магістрального перекладу - це номер сторінки наступної сторінки магістральних списків у списку або нульовий, якщо це остання сторінка магістральних фрілістів. Друге ціле число на сторінці вільної магістралі - це кількість покажчиків аркушів сторінки. Викличте друге ціле

число на сторінці стовбура вільного списку L. Якщо L більше нуля, то цілі числа з індексами масиву між 2 та L + 1 включно містять номери сторінок для фріліст-листів.

На сторінках фріліст-листів немає жодної інформації. SQLite уникає читання або запису фріліст-листів, щоб зменшити введення / виведення диска.

Помилка у версіях SQLite до 3.6.0 (2008-07-16) спричинила повідомлення про базу даних про базу даних, якщо будь-яка з останніх 6 записів у масиві сторінок фрілістних магістралей містила ненульові значення. Новіші версії SQLite не мають цієї проблеми. Однак новіші версії SQLite все ще уникають використання останніх шести записів у масиві сторінок фрілістних магістралей для того, щоб файли баз даних, створені новішими версіями SQLite, могли читати старіші версії SQLite.

Кількість сторінок вільного списку зберігається у вигляді 4-байтового цілого числа великих ендіанів у заголовку бази даних зі зміщенням 36 від початку файлу. Заголовок бази даних також зберігає номер сторінки першої сторінки стовбура вільної списки у вигляді 4-байтового цілого цілого числа з великим ендіанієм при зміщенні 32 від початку файлу.

2.1.6. Сторінки B-дерева

Алгоритм b-tree забезпечує зберігання ключів / даних унікальними та впорядкованими ключами на сторінках, орієнтованих на зберігання. Довідкову інформацію про b-дерева див. У Knuth, The Art of Computer Programming , Volume 3 "Сортування та пошук", стор. 471-479. SQLite використовує два види b-дерев. Алгоритм, який Кнут називає "B * -Tree", зберігає всі дані у листках дерева. SQLite називає цю різновид b-дерева "таблицею b-дерево". Алгоритм, який Кнут називає просто "B-Tree", зберігає разом і ключ, і дані, як у листках, так і на внутрішніх сторінках. У реалізації SQLite оригінальний алгоритм B-Tree зберігає лише ключі, повністю опускаючи дані, і називається "деревом індексу".

Сторінка b-дерева - це або внутрішня сторінка, або аркуш сторінки. На аркуші аркуша містяться ключі, а у випадку дерева b таблиці кожного ключа є пов’язані дані. Внутрішня сторінка містить K клавіші разом із вказівниками K + 1 на дочірні

сторінки b-дерева. "Вказівник" на внутрішній b-tree сторінці - це лише 31-бітний цілочисельний номер дочірньої сторінки.

Визначте глибину листя b-дерева, яка повинна бути 1, а глибина будь-якого внутрішнього b-дерева бути на одну більшу, ніж максимальна глибина будь-якого з його дітей. У добре сформованій базі даних всі діти внутрішнього b-дерева мають однакову глибину.

На внутрішній b-tree сторінці вказівники та клавіші логічно чергуються з вказівником на обох кінцях. (Попереднє речення слід розуміти концептуально - власне розташування клавіш і покажчиків на сторінці складніше і буде описано в подальшому.) Усі клавіші на одній сторінці унікальні і логічно організовані у порядку зростання зліва праворуч. (Знову ж таки, це впорядкування логічне, а не фізичне. Фактичне розташування клавіш на сторінці довільне.) Для будь-якого ключа X вказівники зліва від X посилаються на сторінки b-дерева, на яких усі клавіші менші або рівні на X. Покажчики праворуч від X посилаються на сторінки, де всі клавіші перевищують X.

На внутрішній сторінці b-дерева кожен ключ та вказівник зліва наліво об'єднуються у структуру, що називається "комірка". Найбільш правий покажчик тримається окремо. Сторінка аркуша b-дерева не має вказівників, але вона все ще використовує структуру комірок, щоб утримувати клавіші для індексованих b-дерев або ключі та вміст для b-дерев таблиці. Дані також містяться в комірці.

Кожна сторінка b-дерева має максимум одну батьківську сторінку b-дерева. Сторінка b-дерева без батьківського називається кореневою. Сторінка кореневого дерева b разом із закриттям його дітей утворює повне b-дерево. Можна (і насправді досить поширене) мати повне b-дерево, яке складається з однієї сторінки, яка є і листям, і коренем. Оскільки є вказівники від батьків до дітей, кожна сторінка повного b-дерева може бути розміщена, якщо відома лише коренева сторінка. Отже, b-дерева ідентифікуються за номером кореневої сторінки.

Сторінка b-дерева - це або сторінка b-дерева таблиці, або сторінка b-дерева індексу. Всі сторінки в кожному повному b-дереві мають один і той же тип: або

таблиця, або індекс. У файлі бази даних є одна таблиця b-дерев для кожної рядкової таблиці в схемі бази даних, включаючи системні таблиці, такі як sqlite_master. У файлі бази даних для одного індексу схеми є одне b-дерево індексу, включаючи неявні індекси, створені обмеженнями унікальності. Немає b-дерев, пов'язаних з віртуальними таблицями . Конкретні реалізації віртуальних таблиць можуть використовувати тіньові таблиці для зберігання, але ці тіньові таблиці матимуть окремі записи в схемі бази даних. WITHOUT ROWID таблиці використовують b-дерева індексу, а не таблиці b-дерева, тому у файлі бази даних є одне b-дерево індексу для кожної WITHOUT ROWID таблиці. B-дерево, відповідне таблиці sqlite_master, завжди є b-деревом таблиці і завжди має кореневу сторінку 1. Таблиця sqlite_master містить номер кореневої сторінки для кожної іншої таблиці та індекс у файлі бази даних.

Кожен запис у b-дереві таблиці складається з 64-розрядного цілочисельного підписаного ключа та до 2147483647 байт довільних даних. (Ключ дерева b таблиці відповідає рядковій таблиці SQL, яку реалізує b-дерево.) Внутрішня таблиця b-дерев містить лише ключі та покажчики для дітей. Усі дані містяться в таблиці листя дерев.

Кожен запис у b-дереві індексу складається з довільного ключа довжиною до 2147483647 байт та відсутніх даних.

Визначте "корисне навантаження" комірки, яка буде секцією довільної довжини комірки. У дереві індексу b ключ завжди є довільним за довжиною, а отже, корисне навантаження є ключовим. Немає елементів довільної довжини в клітинках внутрішніх сторінок b-дерева таблиці, тому ці комірки не мають корисного навантаження. Сторінки аркушів b-дерев містять довільний вміст довжини, тому для комірок на цих сторінках корисним навантаженням є вміст.

Коли розмір корисної навантаження для комірки перевищує певний поріг (буде визначено пізніше), то на першій сторінці дерева b зберігаються лише перші кілька байт корисного навантаження, а залишок зберігається у пов'язаному списку переповнених сторінок вмісту.

Сторінка b-дерева поділена на регіони в такому порядку:

1. 100-байтний заголовок файлу бази даних (знайдено лише на сторінці 1);
2. 8 або 12 байт заголовка сторінки b-дерева;
3. Масив вказівника комірок;
4. Нерозподілений простір4
5. Область вмісту комірок;
6. Зарезервований регіон.

Заголовок файлів бази даних на 100 байтів можна знайти лише на сторінці 1, яка завжди є b-деревом таблиці таблиці. Всі інші сторінки b-дерева у файлі бази даних опускають цей 100-байтний заголовок.

Зарезервована область - це область невикористаного простору в кінці кожної сторінки (крім сторінки блокування), яку розширення можуть використовувати для зберігання інформації про кожну сторінку. Розмір зарезервованої області визначається однобайтове неподписане ціле число, знайдене при зміщенні 20 у заголовок файлу бази даних. Розмір зарезервованої області зазвичай дорівнює нулю.

Заголовок сторінки b-дерева має розмір 8 байт для листкових сторінок і 12 байт для внутрішніх сторінок. Усі багатобайтові значення в заголовку сторінки є великими. Заголовок сторінки b-дерева складається з таких полів:

Таблиця 2.2. Формат заголовка сторінки на B-дереві

Зсув	Розмір	Опис
0	1	Однобайтовий прапор зі зміщенням 0, що вказує на тип сторінки b-tree. Значення 2 (0x02) означає, що сторінка - це внутрішня сторінка b-дерева. Значення 5 (0x05) означає, що сторінка - це внутрішня сторінка b-дерева. Значення 10 (0x0a) означає, що сторінка - це сторінка з деревним індексом. Значення 13 (0x0d) означає, що сторінка - це сторінка b-дерева аркушів.

		Будь-яке інше значення для типу сторінки b-tree - це помилка.
1	2	Двобайтове ціле число при зміщенні 1 дає початок першого вільного блоку на сторінці, або дорівнює нулю, якщо немає вільних блоків.
3	2	Двобайтове ціле число при зміщенні 3 дає кількість комірок на сторінці.
5	2	Двобайтне ціле число при зміщенні 5 позначає початок області вмісту комірки. Нульове значення для цього цілого числа інтерпретується як 65536.
7	1	Однобайтове ціле число при зміщенні 7 дає кількість фрагментованих вільних байтів у межах вмісту комірки.
8	4	Чотирибайтовий номер сторінки при зміщенні 8 є найбільш правим вказівником. Це значення відображається лише у заголовку внутрішніх сторінок b-дерева та пропускається з усіх інших сторінок.

Масив вказівника комірок сторінки b-дерева негайно слідує за заголовком сторінки b-дерева. Нехай K - кількість комірок на btree. Масив покажчика комірки складається з 2-байтових цілих чисел, зміщених до вмісту комірки. Покажчики комірок розташовані в ключовому порядку з найбільшою лівою клітиною (комірка з найменшою клавішею) першою та найбільш правою клітиною (комірка з найбільшим ключем) останньою.

Вміст комірок зберігається в області вмісту комірки сторінки b-дерева. SQLite прагне розмістити комірки якомога ближче до кінця сторінки b-дерева, щоб залишити місце для майбутнього зростання масиву покажчиків комірок. Область між останнім входом масиву вказівника комірки та початком першої комірки є нерозподіленою областю.

Якщо на сторінці немає клітинок (що можливе лише для кореневої сторінки таблиці, що не містить рядків), зміщення до області вмісту комірки буде дорівнює розміру сторінки за вирахуванням байтів зарезервованого простору. Якщо база даних використовує розмір сторінки 65536 байт, а зарезервований простір дорівнює нулю (звичайне значення для зарезервованого простору), тоді зміщення вмісту комірки порожньої сторінки хоче бути 65536. Однак це ціле число занадто велике, щоб зберігати в 2-байтове ціле число без знака, тому на його місці використовується значення 0.

Фріблок - це структура, яка використовується для ідентифікації нерозподіленого простору на сторінці b-дерева. Фріблоки організовані як ланцюжок. Перші 2 байти вільного блоку - це велике ціле число, яке є зміщенням на b-дереві сторінки наступного вільного блоку в ланцюзі, або нуль, якщо фріблок є останнім у ланцюжку. Третій та четвертий байти кожного вільного блоку утворюють ціле число великого ендіану, яке є розміром фріблока в байтах, включаючи 4-байтний заголовок. Безкоштовні блоки завжди підключаються в порядку збільшення зсуву. Друге поле заголовка сторінки b-дерева - це зміщення першого фріблока, або нуль, якщо на сторінці немає вільних блоків. На добре сформованій сторінці b-дерева завжди буде принаймні одна клітинка перед першим вільним блоком.

Для вільного блокування потрібно не менше 4-х байт простору. Якщо в зоні вмісту комірки є ізольована група з 1, 2 або 3 невикористаних байтів, ці байти містять фрагмент. Загальна кількість байтів у всіх фрагментах зберігається у п’ятому полі заголовка сторінки b-дерева. На добре сформованій сторінці b-дерева загальна кількість байтів у фрагментах не може перевищувати 60.

Загальна кількість вільного місця на b-дереві складається з розміру нерозподіленої області плюс загального розміру всіх вільних блоків плюс кількості фрагментованих вільних байтів. SQLite час від часу може реорганізувати сторінку b-дерева, щоб не було вільних блоків або байтів фрагментів, усі невикористані байти містяться в нерозподіленій області простору, а всі комірки упаковані щільно в кінці сторінки. Це називається "дефрагментація" сторінки b-дерева.

Ціле число або "varint" змінної довжини - це статичне кодування Хаффмана з 64-бітовими цілими числами з двома доповненнями, що використовує менше місця для малих позитивних значень. Варіант має довжину від 1 до 9 байт. Стрічка складається з нуля або більше байтів, у яких встановлено біт високого порядку, а потім - один байт з чітким бітом високого порядку, або дев'ять байтів, залежно від того, що коротше. Нижні сім біт кожного з перших восьми байтів і всі 8 бітів дев’ятого байта використовуються для реконструкції 64-бітного цілого числа двокомпонентів. Варіанти є великими-ендіанськими: біти, взяті з попереднього байта varint, є більш значущими, ніж біти, взяті з пізніших байтів.

Формат комірки залежить від того, на якій сторінці b-дерева відображається комірка. Наступна таблиця показує елементи комірки, у порядку появи, для різних типів сторінок b-дерева.

Таблиця клітини B-дерева (заголовок 0x0d):

- varint - загальна кількість байтів корисного навантаження, включаючи будь-яке переповнення;
- varint, який є цілим ключем, він називається "rowid".

Початкова частина корисного навантаження, яка не переливається на переповнені сторінки.

4-байтовий цілочисельний номер з великим ендіанським номером для першої сторінки списку переповнених сторінок - пропущено, якщо все корисне навантаження входить на сторінку b-дерева.

Внутрішня клітинка таблиці B-Tree (заголовок 0x05):

4-байтовий номер сторінки з великим ендіанієм, який є вказівником зліва.

varint, який є цілим числом

Індекс B-Tree Leaf Cell (заголовок 0x0a):

varint, який є загальною кількістю байтів ключового корисного навантаження, включаючи будь-яке переповнення.

Початкова частина корисного навантаження, яка не переливається на переповнені сторінки.

4-байтовий цілочисельний номер з великим ендіанським номером для першої сторінки списку переповнених сторінок - пропущено, якщо все корисне навантаження входить на сторінку b-дерева.

Індекс B-Tree Internal Cell (заголовок 0x02):

4-байтовий номер сторінки з великим ендіанієм, який є вказівником зліва.

varint, який є загальною кількістю байтів ключового корисного навантаження, включаючи будь-яке переповнення.

Початкова частина корисного навантаження, яка не переливається на переповнені сторінки.

4-байтовий цілочисельний номер з великим ендіанським номером для першої сторінки списку переповнених сторінок - пропущено, якщо все корисне навантаження входить на сторінку b-дерева.

Інформацію вище можна переробити у формат таблиці таким чином:

Таблиця 2.3. Формат клітини B-дерева

Тип даних	З'являється у ...				Опис
	Лист таблиці (0x0d)	Покажчик таблиці (0x05)	Індексний лист (0x0a)	Покажчик індексу (0x02)	
4-байтове ціле число		✔		✔	Номер сторінки лівої дитини
варінт	✔		✔	✔	Кількість байтів корисного навантаження
варінт	✔	✔			Rowid
байтовий масив	✔		✔	✔	Корисне навантаження
4-байтове ціле число	✔		✔	✔	Номер першої сторінки, що переповнюється

Кількість корисного навантаження, що розподіляється на сторінки переповнення, також залежить від типу сторінки. Для наступних обчислень, нехай U - корисний розмір сторінки бази даних, загальний розмір сторінки за винятком відведеного місця в кінці кожної сторінки. І нехай P - розмір корисної навантаження. Далі символ X являє собою максимальну кількість корисного навантаження, яка може зберігатися безпосередньо на b-дереві сторінки, не розливаючись на сторінку переповнення, а символ M являє собою мінімальну кількість корисного навантаження, яка повинна зберігатися на btree сторінці до того, як розсипання буде дозволено .

Листова клітина B-дерева:

Нехай X - це U-35. Якщо розмір корисного навантаження P менший або дорівнює X, то весь корисний вантаж зберігається на аркуші b-дерева. Нехай M буде ((U-12) * 32/255) -23 і нехай K буде M + ((PM)% (U-4)). Якщо P більший за X, то кількість байтів, збережених на сторінці аркуша b-дерева таблиці, дорівнює K, якщо K менше або дорівнює X або M в іншому випадку. Кількість байтів, що зберігаються на листковій сторінці, ніколи не менша за M.

Внутрішня клітина B-дерева:

Внутрішні сторінки столів b-дерев не мають корисного навантаження, тому ніколи не буде корисного навантаження.

Індекс листа B-дерева або внутрішньої клітини:

Нехай X буде ((U-12) * 64/255) -23. Якщо розмір корисного навантаження P менший або дорівнює X, то весь корисний вантаж зберігається на b-дереві сторінки. Нехай M буде ((U-12) * 32/255) -23 і нехай K буде M + ((PM)% (U-4)). Якщо P більше X, то кількість байтів, збережених на сторінці дерева b індексу, дорівнює K, якщо K менше або дорівнює X або M в іншому випадку. Кількість байтів, що зберігаються на індексній сторінці, ніколи не менша за M.

Ось альтернативний опис того ж обчислення:

X - U-35 для сторінок білого аркуша таблиці або ((U-12) * 64/255) -23 для індексних сторінок.

M завжди ((U-12) * 32/255) -23.

Нехай K - M + ((PM)% (U-4)).

Якщо P <= X, то всі P байти корисного навантаження зберігаються безпосередньо на btree сторінці без переповнення.

Якщо P> X і K <= X, то перші K байти P зберігаються на btree сторінці, а решта байти PK зберігаються на переливних сторінках.

Якщо P> X і K> X, то перші M байти P зберігаються на btree-сторінці, а решта байт PM зберігаються на переливних сторінках.

Пороги переповнення розроблені таким чином, щоб забезпечити мінімальний показник 4 для дерев індексу b і забезпечити, щоб на сторінці b-дерева було достатньо корисного навантаження, до якого заголовок запису зазвичай можна отримати, не звертаючись до сторінки переповнення. Зрозумівши, дизайнер логіки bQL-дерева SQLite зрозумів, що ці пороги можна було зробити набагато простішими. Однак обчислення неможливо змінити, якщо це не призведе до несумісного формату файлів. І поточні обчислення працюють добре, навіть якщо вони трохи складні.

2.1.7. Сторінки переповнення комірок корисної навантаження

Коли корисна навантаження клітини b-дерева занадто велика для сторінки b-дерева, надлишок розливається на сторінки переповнення. Переповнені сторінки утворюють зв'язаний список. Перші чотири байти кожної сторінки, що переповнюється, - це велике ціле число, яке є номером наступної сторінки ланцюга, або нулем для кінцевої сторінки ланцюга. П'ятий байт через останній корисний байт використовується для утримання вмісту переповнення.

2.1.8. Карта вказівників або сторінки Ptrmap

Сторінки карти вказівників або сторінки ptrmap - це додаткові сторінки, вставлені в базу даних, щоб зробити роботу режимів auto_vacuum та incremental_vacuum більш ефективними. Інші типи сторінок у базі даних зазвичай мають вказівки від батьків до дитини. Наприклад, внутрішня сторінка b-дерева містить вказівники на її дочірні сторінки b-дерева, а ланцюг переповнення має вказівник від попередніх до пізніших посилань ланцюга. Сторінка ptrmap містить інформацію про зв'язки, що йде в зворотному напрямку, від дитини до батьків.

Сторінки Ptrmap повинні існувати у будь-якому файлі бази даних, який має значення нульового найбільшого кореневого b-дерева на сторінці зі зміщенням 52 у заголовку бази даних. Якщо найбільше значення корінкової сторінки b-дерева дорівнює нулю, то база даних не повинна містити сторінок ptrmap.

У базі даних зі сторінками ptrmap перша сторінка ptrmap - це сторінка 2. Сторінка ptrmap складається з масиву 5-байтних записів. Нехай J - кількість 5-

байтних записів, які вмістяться у просторі сторінки. (Іншими словами, J = U / 5.) Перша сторінка ptrmap міститиме інформацію про вказівники на задній сторінці для сторінок 3 по J + 2, включно. Друга сторінка карти вказівника знаходитиметься на сторінці J + 3, і ця сторінка ptrmap надаватиме інформацію щодо вказівника для сторінок J + 4 по 2 * J + 3 включно. І так далі для всього файлу бази даних.

У базі даних, яка використовує сторінки ptrmap, усі сторінки в місцях, визначених обчисленням у попередньому абзаці, повинні бути сторінками ptrmap, а жодна інша сторінка не може бути сторінкою ptrmap. За винятком випадків, якщо сторінка блокування байтів потрапляє на той самий номер сторінки, що і сторінка ptrmap, тоді ptrmap переміщується на наступну сторінку для цього одного випадку.

Кожен 5-байтний запис на сторінці ptrmap надає інформацію про зворотне посилання про одну зі сторінок, що негайно слідують за картою вказівника. Якщо сторінка B - сторінка ptrmap, то інформація про зворотне посилання на сторінку B + 1 надається першим записом на карті вказівника. Інформацію про сторінку B + 2 надає другий запис. І так далі.

Кожен запис 5-байтного ptrmap складається з одного байта інформації про "тип сторінки" з наступним 4-байтним номером сторінки. Розпізнається п'ять типів сторінок:

- Сторінка кореневого дерева b. Номер сторінки повинен бути нульовим;
- Сторінка фрілістів. Номер сторінки повинен бути нульовим;
- Перша сторінка ланцюга переповнення корисних навантажень комірок. Номер сторінки - це сторінка b-дерева, яка містить комірку, вміст якої переповнюється;
- Сторінка в ланцюзі переповнення, відмінна від першої. Номер сторінки - це попередня сторінка ланцюга переповнення;
- Сторінка не-кореневого дерева b. Номер сторінки є батьківською сторінкою b-дерева.

У будь-якому файлі бази даних, що містить сторінки ptrmap, всі кореневі сторінки b-дерева повинні бути перед будь-якою некореневою сторінкою b-дерева, сторінкою переповнення завантаження комірок або сторінкою вільних списків. Це

обмеження забезпечує те, що коренева сторінка ніколи не буде переміщена під час автоматичного вакуумування або збільшення вакууму. Логіка автоматичного вакуумування не знає, як оновити поле root_page таблиці sqlite_master, тому необхідно запобігти переміщенню кореневих сторінок під час автоматичного вакуумування, щоб зберегти цілісність таблиці sqlite_master. Корінні сторінки переміщуються на початок файлу бази даних операціями CREATE TABLE, CREATE INDEX, DROP TABLE та DROP INDEX.

2.2. Schema рівень

Вищенаведений текст описує аспекти формату файлів SQLite на низькому рівні. Механізм b-tree забезпечує потужний та ефективний засіб доступу до великого набору даних. У цьому розділі буде описано, як низькорівневий шар b-дерева використовується для реалізації можливостей SQL вищого рівня.

2.2.1. Формат запису

Дані для сторінки аркуша b-дерева таблиці та ключа сторінки індексу b-дерева характеризувались вище як довільна послідовність байтів. Попереднє обговорення зазначало, що один ключ є меншим за інший, але не визначав, що означає "менше". У цьому розділі буде розглянуто ці недоліки.

Корисне навантаження, або дані b-дерева таблиці, або ключі b-дерева індексу, завжди знаходиться у "форматі запису". Формат запису визначає послідовність значень, що відповідає стовпцям таблиці або індексу. Формат запису визначає кількість стовпців, тип даних кожного стовпця та вміст кожного стовпця.

Формат запису широко використовує ціле чи змінне представлення змінної довжини 64-бітових цілих чисел, визначених вище.

Запис містить заголовок і тіло в такому порядку. Заголовок починається з однієї varint, яка визначає загальну кількість байтів у заголовку. Значення varint - це розмір заголовка в байтах, включаючи сам розмір. Слідом за розмірною фарбою є один або кілька додаткових лаків, по одному на стовпчик. Ці додаткові лаки називаються

номерами "серійного типу" і визначають тип даних кожного стовпця відповідно до наступної діаграми:

Таблиця 2.4. Коди послідовного типу формату запису

Тип серії	Розмір вмісту	Значення
0	0	Значення - NULL.
1	1	Значення - це 8-бітове ціле число подвійних доповнень.
2	2	Значення - це 16-бітове ціле ціле число з двома бітами.
3	3	Значення - це 24-бітове ціле ціле число з двома екземплярами.
4	4	Значення - це велике ціле 32-бітове ціле число.
5	6	Значення - це ціле 48-бітове ціле ціле число.
6	8	Значення - це велике ціле 64-бітове ціле число.
7	8	Значення - 64-бітове число з плаваючою комою IEEE 754-2008 IEEE.
8	0	Значення - ціле число 0. (Доступно лише для формату схеми 4 і вище.)
9	0	Значення - ціле число 1. (Доступно лише для формату схеми 4 і вище.)
10,11	змінна	Зарезервовано для внутрішнього використання. Ці коди послідовного типу ніколи не з'являться у добре сформованому файлі бази даних, але вони можуть використовуватися у тимчасових та тимчасових

		файлах баз даних, які SQLite іноді генерує для власного використання. Значення цих кодів можуть переходити від одного випуску SQLite до іншого.
N≥12 і навіть	(N-12) / 2	Значення - BLOB, що становить (N-12) / 2 байти в довжину.
N≥13 і непарне	(N-13) / 2	Значення - це рядок у кодуванні тексту та довжиною (N-13) / 2 байти. Нульовий термінатор не зберігається.

Фарба розміру заголовка та лаки серійного типу зазвичай складаються з одного байту. Фарби серійного типу для великих рядків і BLOB можуть поширюватися на два-три байтові, але це швидше виняток, ніж правило. Формат varint дуже ефективний при кодуванні заголовка запису.

Значення для кожного стовпця в записі одразу слідують за заголовком. Для серійних типів 0, 8, 9, 12 і 13 значення в довжину становить нульові байти. Якщо всі стовпці таких типів, то розділ тіла запису порожній.

Запис може мати менше значень, ніж кількість стовпців у відповідній таблиці. Це може статися, наприклад, після того, як оператор ALTER TABLE ... ADD COLUMN SQL збільшив кількість стовпців у схемі таблиці без зміни попередніх рядків таблиці. Пропущені значення в кінці запису заповнюються, використовуючи значення за замовчуванням для відповідних стовпців, визначених у схемі таблиці.

2.2.2. Запис порядку сортування

Порядок ключів у дереві індексу b визначається порядком сортування записів, які представляють ключі. Запишіть порівняння прогресує колонку за стовпцем. Стовпці запису оглядаються зліва направо. Перша пара стовпців, які не є

рівними, визначає відносний порядок двох записів. Порядок сортування окремих стовпців такий:

Значення NULL (серійний тип 0) спочатку сортуйте.

Числові значення (серійні типи з 1 по 9) сортують за NULL і в числовому порядку.

Текстові значення (непарні серійні типи 13 і більше) сортують за числовими значеннями в порядку, визначеному функцією згортання стовпців .

Значення BLOB (навіть серійні типи 12 і більше) сортуються останніми і в порядку, визначеному memcmp ().

Функція складання для кожного стовпця необхідна для обчислення порядку текстових полів. SQLite визначає три вбудовані функції зіставлення:

Таблиця 2.5. Функції зіставлення SQLite

BINARY	Вбудоване порівняння BINARY порівнює рядки байт за байтом, використовуючи функцію memcmp () зі стандартної бібліотеки C.
NOCASE	Порівняння NOCASE подібне до BINARY, за винятком того, що великі символи ASCII («A» - «Z») складаються у їхні малі еквіваленти перед запуском порівняння. Лише ASCII символи складаються з регістру. NOCASE не реалізує універсальне порівняння unicode без випадків.
RTRIM	RTRIM схожий на BINARY, за винятком того, що додаткові пробіли в кінці будь-якого рядка не змінюють результат. Іншими словами, рядки будуть порівнювати однакові між собою, якщо вони відрізняються лише кількістю пробілів на кінці.

До SQLite можна додати додаткові функції, що стосуються додатків, за допомогою інтерфейсу sqlite3_create_collation () .

За замовчуванням функція згортання для всіх рядків - BINARY. Альтернативні функції згортання стовпців таблиць можуть бути визначені в операторі CREATE TABLE за допомогою пункту COLLATE у визначенні стовпця . Коли стовпчик індексується, однакова функція згортання, визначена в операторі CREATE TABLE ,

використовується за стовпцем в індексі за замовчуванням, хоча це може бути замінено за допомогою пункту COLLATE в операторі CREATE INDEX .

2.2.3. Представлення таблиць SQL

Кожна звичайна таблиця SQL у схемі бази даних представлена на диску на дереві таблиці. Кожен запис в b-дереві таблиці відповідає рядку таблиці SQL. ROWID таблиці SQL є 64-розрядним цілим числом ключа для кожного запису в таблиці B-дереві.

Вміст кожного рядка таблиці SQL зберігається у файлі бази даних, спочатку об'єднуючи значення в різних стовпцях у байтовий масив у форматі запису, а потім зберігаючи цей байтовий масив як корисне навантаження у записі в b-дереві таблиці. Порядок значень у записі такий самий, як порядок стовпців у визначенні таблиці SQL. Якщо таблиця SQL містить стовпець INTEGER PRIMARY KEY (який псевдонімом називається rowid), то цей стовпець відображається в записі як значення NULL. При посиланні на стовпець INTEGER PRIMARY KEY SQLite завжди використовуватиме ключ дерева b, а не значення NULL .

Якщо спорідненість стовпця РЕАЛЬНА і цей стовпець містить значення, яке можна перетворити на ціле число без втрати інформації (якщо значення не містить дробової частини і не надто велике, щоб бути представлене як ціле число), то стовпець може бути зберігається в записі як ціле число. SQLite перетворить значення назад у плаваючу точку під час вилучення його із запису.

2.2.4. Представлення БЕЗ РОЗМІРНИХ таблиць

Якщо таблиця SQL створена за допомогою пункту "БЕЗ ROWID" в кінці її оператора CREATE TABLE, то ця таблиця є БЕЗ ROWID таблиці та використовує інше представлення на диску. БЕЗ ROWID таблиці використовує індексне b-дерево, а не табличне b-дерево для зберігання. Ключ для кожного запису в b-дереві БЕЗ ROWID - це запис, що складається з стовпців PRIMARY KEY, за якими слідують усі решта стовпців таблиці. Стовпці первинного ключа відображаються в тому порядку, в якому вони були оголошені в пункті PRIMARY KEY, а решта стовпці

відображаються в тому порядку, в якому вони виникають у операторі CREATE TABLE.

Отже, кодування вмісту для таблиці WITHOUT ROWID є такою ж, як кодування вмісту для звичайної рядної таблиці, за винятком того, що порядок стовпців переставляють таким чином, щоб перші стовпці PRIMARY KEY були першими, а вміст використовувався як ключ у індекс b-дерева, а не як дані таблиці b-дерева. Спеціальні правила кодування стовпців з реальною спорідненістю застосовуються до WITHOUT ROWID таблиць так само, як і до рядкових таблиць.

2.2.5. Представлення індексів SQL

Кожен індекс SQL, явно декларований через оператор CREATE INDEX або мається на увазі обмеження UNIQUE або PRIMARY KEY, відповідає дереву індексу b у файлі бази даних. Кожен запис у дереві індексу b відповідає одному рядку у пов'язаній таблиці SQL. Ключ до b-дерева індексу - це запис, що складається з стовпців, які індексуються, а потім ключ відповідного рядка таблиці. Для звичайних таблиць ключ рядка - це рядковий , а для БЕЗ РОЗІДНИХ таблиць ключ рядка - ПЕРШИЙ КЛЮЧ. Оскільки кожен рядок у таблиці має унікальний ключ рядка, всі ключі в індексі є унікальними.

У звичайному індексі відображається індивідуальне відображення між рядками в таблиці та записами в кожному індексі, пов'язаному з цією таблицею. Однак у частковому індексі дерево індексу b містить лише записи, відповідні рядкам таблиці, для яких вираз WHERE в операторі CREATE INDEX є істинним. Відповідні рядки в b-деревах індексу та таблиці мають однакові значення рядка чи первинного ключа та містять однакове значення для всіх індексованих стовпців.

2.2.6. Зберігання схеми баз даних SQL

Сторінка 1 файлу бази даних - це коренева сторінка дерева b таблиці, яка містить спеціальну таблицю під назвою "sqlite_master" (або "sqlite_temp_master" у випадку бази даних TEMP), на якій зберігається повна схема бази даних. Структура таблиці sqlite_master така, ніби вона була створена за допомогою наступного SQL:

```
CREATE TABLE sqlite_schema(
  type text,
  name text,
  tbl_name text,
  rootpage integer,
  sql text
);
```

Таблиця sqlite_master містить один рядок для кожної таблиці, індекс, перегляд та тригер (спільно "об'єкти") в схемі бази даних, за винятком записів для самої таблиці sqlite_master. Таблиця sqlite_master містить записи для внутрішніх об'єктів схеми, крім об'єктів, визначених програмою та програмістом.

Стовпець sqlite_master.type буде одним із наступних текстових рядків: "таблиця", "індекс", "перегляд" або "тригер" відповідно до визначеного типу об'єкта. Рядок 'table' використовується як для звичайних, так і для віртуальних таблиць .

Стовпець sqlite_master.name буде містити ім'я об'єкта. Унікальний і PRIMARY KEY обмеження на столах викликають SQLite для створення внутрішніх індексів з іменами виду «sqlite_autoindex_TABLE_N» , де Таблиця замінюється ім'ям таблиці, яка містить обмеження і N є цілим числом , починаючи з 1 , і збільшується на одиницю з кожним обмеженням видно в таблиці визначення. У БЕЗ ROWID таблиці немає запису sqlite_master для PRIMARY KEY, але ім'я "sqlite_autoindex_TABLE_N" відведено для PRIMARY KEY так, як ніби запис sqlite_master існує. Це вплине на нумерацію наступних UNIQUE обмежень. "Sqlite_autoindex_TABLE_N" , або в рядкових таблицях, або БЕЗ РОЗМІРНИХ таблиць.

Стовпець sqlite_master.tbl_name містить ім'я таблиці або представлення, з яким об'єкт пов'язаний. Для таблиці або подання стовпець tbl_name - це копія стовпця імен. Для індексу tbl_name - це ім'я таблиці, яка індексується. Для тригера стовпець tbl_name зберігає назву таблиці або перегляду, що викликає активацію тригера.

Стовпець sqlite_master.rootpage зберігає номер сторінки кореневої сторінки b-дерева для таблиць та індексів. Для рядків, що визначають представлення, тригери та віртуальні таблиці, стовпець кореневої сторінки дорівнює 0 або NULL.

Стовпець sqlite_master.sql зберігає текст SQL, який описує об'єкт. Цей текст SQL є CREATE TABLE , CREATE VIRTUAL TABLE , CREATE INDEX , CREATE VIEW або CREATE TRIGGER, який, якщо його оцінювати по відношенню до файлу бази даних, коли він є основною базою даних з'єднання з базою даних , відтворив би об'єкт. Текст, як правило, є копією оригінального висловлювання, що використовується для створення об'єкта, але із застосованими нормалізаціями, щоб текст відповідав таким правилам:

Ключові слова CREATE, TABLE, VIEW, TRIGGER та INDEX на початку оператора перетворюються на всі великі літери.

Ключове слово TEMP або TEMPORARY видаляється, якщо воно виникає після початкового ключового слова CREATE.

Будь-який класифікатор імені бази даних, який виникає до імені створеного об'єкта, видаляється.

Провідні пробіли видаляються.

Усі пробіли, що слідують за першими двома ключовими словами, перетворюються в єдиний пробіл.

Текст у стовпці sqlite_master.sql - це копія оригінального тексту оператора CREATE, який створив об'єкт, за винятком нормалізованого, як описано вище та зміненого наступними операторами ALTER TABLE . Sqlite_master.sql - NULL для внутрішніх індексів , які автоматично створюються за допомогою UNIQUE або PRIMARY KEY .

2.2.6.1. Об'єкти внутрішньої схеми

На додаток до таблиць, індексів, представлень та тригерів, створених програмою та / або розробником з використанням CREATE висловлювань SQL, таблиця sqlite_master може містити нуль або більше записів для внутрішніх об'єктів схеми , які створюються SQLite для власного внутрішнього використання. Назви

об'єктів внутрішньої схеми завжди починаються з "sqlite_", а будь-яка таблиця, індекс, перегляд або тригер, ім'я яких починається з "sqlite_", є об'єктом внутрішньої схеми. SQLite забороняє програмам створювати об'єкти, імена яких починаються з "sqlite_".

Об'єкти внутрішньої схеми, використовувані SQLite, можуть включати в себе наступне:

Індекси з іменами форми "sqlite_autoindex_TABLE_N", які використовуються для реалізації UNIQUE та PRIMARY KEY обмежень у звичайних таблицях.

Таблиця з назвою "sqlite_sequence", яка використовується для відстеження максимального історичного ПЕРШОГО КЛЮЧА INTEGER для таблиці за допомогою AUTOINCREMENT .

Таблиці з іменами форми "sqlite_statN", де N - ціле число. Такі таблиці зберігають статистику баз даних, зібрану командою ANALYZE і використовувану планувальником запитів, щоб допомогти визначити найкращий алгоритм, який слід використовувати для кожного запиту.

Нові імена об’єктів внутрішньої схеми, які завжди починаються з "sqlite_", можуть бути додані до файлового формату SQLite у майбутніх випусках.

2.2.6.2. Таблиця sqlite_sequence

Таблиця sqlite_sequence - це внутрішня таблиця, яка використовується для сприяння реалізації AUTOINCREMENT . Таблиця sqlite_sequence створюється автоматично, коли створюється будь-яка звичайна таблиця з первинним ключем цілого числа AUTOINCREMENT. Після створення таблиці sqlite_sequence таблиця існує в таблиці sqlite_master назавжди; її не можна скинути. Схема таблиці sqlite_sequence така:

CREATE TABLE sqlite_sequence (ім'я, послідовність);

У таблиці sqlite_sequence є один рядок для кожної звичайної таблиці, яка використовує AUTOINCREMENT. Назва таблиці (як вона відображається в sqlite_master.name) знаходиться в полі sqlite_sequence.main, а найбільший ІНТЕГРОВИЙ ПЕРШИЙ КЛЮЧ, який коли-небудь вставлений у цю таблицю,

знаходиться у полі sqlite_sequence.seq. Нові автоматично згенеровані цілі цілі первинних ключів для таблиць AUTOINCREMENT гарантовано перевищують поле sqlite_sequence.seq для цієї таблиці. Якщо поле sqlite_sequence.seq таблиці AUTOINCREMENT вже є найбільшим цілим значенням (9223372036854775807), тоді спроба додати нові рядки до цієї таблиці з автоматично сформованим цілим числом первинних не вдасться з SQLITE_FULLпомилка. Поле sqlite_sequence.seq автоматично оновлюється, якщо потрібно, коли нові записи вставляються в таблицю AUTOINCREMENT. Рядок sqlite_sequence для таблиці AUTOINCREMENT автоматично видаляється при падінні таблиці. Якщо рядок sqlite_sequence для таблиці AUTOINCREMENT не існує, коли оновлена таблиця AUTOINCREMENT, то створюється новий рядок sqlite_sequence. Якщо значення sqlite_sequence.seq для таблиці AUTOINCREMENT вручну встановлюється на щось інше, ніж ціле число, і є наступна спроба вставити або оновити таблицю AUTOINCREMENT, тоді поведінка не визначена.

Коду програми дозволяється змінювати таблицю sqlite_sequence, додавати нові рядки, видаляти рядки або змінювати існуючі рядки. Однак код програми не може створити таблицю sqlite_sequence, якщо вона ще не існує. Код програми може видалити всі записи з таблиці sqlite_sequence, але код програми не може скинути таблицю sqlite_sequence.

2.2.6.3. Таблиця sqlite_stat1

Sqlite_stat1 - це внутрішня таблиця, створена командою ANALYZE і використовується для зберігання додаткової інформації про таблиці та індекси, які планувальник запитів може використовувати, щоб допомогти їй знайти кращі способи виконання запитів. Програми можуть оновлювати, видаляти з, вставляти в таблицю sqlite_stat1 або видаляти її, але не можуть створювати або змінювати таблицю sqlite_stat1. Схема таблиці sqlite_stat1 така:

CREATE TABLE sqlite_stat1 (tbl, idx, stat);

Зазвичай існує один рядок на індекс, при цьому індекс ідентифікується ім'ям у стовпці sqlite_stat1.idx. Стовпець sqlite_stat1.tbl - це назва таблиці, до якої належить

індекс. У кожному такому рядку стовпець sqlite_stat.stat буде рядком, що складається зі списку цілих чисел, що супроводжується нулем або більше аргументів. Перше ціле число цього списку - приблизна кількість рядків в індексі. (Кількість рядків в індексі така ж, як кількість рядків у таблиці, за винятком часткових індексів.) Друге ціле число - це приблизна кількість рядків в індексі, які мають однакове значення в першому стовпчику індексу. Третє ціле число - це число число рядків в індексі, які мають однакове значення для перших двох стовпців. N -те ціле число (для N> 1) - це орієнтовна середня кількість рядків в індексі, які мають однакове значення для перших стовпців N-1. Для індексу K-стовпців, у стовпчику stat будуть цілі числа K + 1. Якщо індекс унікальний, то останнім цілим числом буде 1.

Перелік цілих чисел у стовпці stat може необов'язково супроводжуватися аргументами, кожне з яких - це послідовність символів, що не знаходяться з пробілом. Всім аргументам передує єдиний пробіл. Нерозпізнані аргументи мовчки ігноруються.

Якщо аргумент "не упорядкований" присутній, тоді планувальник запитів припускає, що індекс не упорядкований і не буде використовувати індекс для запиту діапазону або для сортування.

Аргумент "sz = NNN" (де NNN являє собою послідовність з 1 або більше цифр) означає, що середній розмір рядка для всіх записів таблиці або індексу становить NNN байт на рядок. Планувальник запитів SQLite може використовувати інформацію про передбачуваний розмір рядків, надану маркером "sz = NNN", щоб допомогти йому вибрати менші таблиці та індекси, для яких потрібно менше вводу / виводу диска.

Наявність маркера "noskipscan" у полі sqlite_stat1.stat індексу не дозволяє використовувати цей індекс при оптимізації пропуску сканування .

Нові текстові лексеми можуть бути додані в кінець стовпця stat у майбутніх удосконаленнях до SQLite. Для сумісності нерозпізнані лексеми в кінці стовпця stat мовчки ігноруються.

Якщо стовпець sqlite_stat1.idx дорівнює NULL, то стовпець sqlite_stat1.stat містить єдине ціле число, яке є приблизною кількістю рядків у таблиці, ідентифікованих sqlite_stat1.tbl. Якщо стовпець sqlite_stat1.idx такий же, як стовпець sqlite_stat1.tbl, то таблиця - БЕЗ ROWID таблиці, а поле sqlite_stat1.stat містить інформацію про btree індексу, що реалізує таблицю БЕЗ ROWID.

2.2.6.4. Таблиця sqlite_stat2

Sqlite_stat2 створюється та використовується лише у тому випадку, якщо SQLite компілюється із SQLITE_ENABLE_STAT2 та якщо номер версії SQLite знаходиться між 3.6.18 (2009-09-11) та 3.7.8 (2011-09-19). Таблиця sqlite_stat2 не читається і не записується жодною версією SQLite до 3.6.18, ні після 3.7.8. Таблиця sqlite_stat2 містить додаткову інформацію про розподіл ключів в індексі. Схема таблиці sqlite_stat2 така:

CREATE TABLE sqlite_stat2 (tbl, idx, sampleno, sample);

Стовпець sqlite_stat2.idx та стовпець sqlite_stat2.tbl у кожному рядку таблиці sqlite_stat2 ідентифікують індекс, описаний цим рядком. Зазвичай у таблиці sqlite_stat2 є 10 рядків для кожного індексу.

Записи sqlite_stat2 для індексу, які мають sqlite_stat2.sampleno між 0 і 9 включно, є зразками самого лівого ключового значення індексу, взятого в рівномірно розташованих точках уздовж індексу. Нехай C - кількість рядків в індексі. Потім вибіркові рядки задаються по

рядовий номер = (i * C * 2 + C) / 20

Змінна i в попередньому виразі коливається між 0 і 9. Концептуально простір індексу поділяється на 10 рівномірних відрізків, а вибірки є середнім рядом від кожного відра.

Формат для sqlite_stat2 записується тут для попереднього посилання. Останні версії SQLite більше не підтримують sqlite_stat2, а таблиця sqlite_stat2, якщо вона існує, просто ігнорується.

2.2.6.5. Таблиця sqlite_stat3

Sqlite_stat3 використовується лише в тому випадку, якщо SQLite компілюється з SQLITE_ENABLE_STAT3 або SQLITE_ENABLE_STAT4 і якщо номер версії SQLite становить 3.7.9 (2011-11-01) або вище. Таблиця sqlite_stat3 не читається і не записується жодною версією SQLite до 3.7.9. Якщо використовується параметр часу компіляції SQLITE_ENABLE_STAT4 і номер версії SQLite становить 3.8.1 (2013-10-17) або вище, то sqlite_stat3 може бути прочитаний, але не записаний. Таблиця sqlite_stat3 містить додаткову інформацію про розподіл ключів в індексі, інформацію, яку планувальник запитів може використовувати для створення кращих та швидших алгоритмів запитів. Схема таблиці sqlite_stat3 така:

CREATE TABLE sqlite_stat3 (tbl, idx, nEq, nLt, nDLt, зразок);

Зазвичай у таблиці sqlite_stat3 є кілька записів для кожного індексу. Колонка sqlite_stat3.sample містить вміст самого лівого поля індексу, ідентифікованого sqlite_stat3.idx та sqlite_stat3.tbl. Стовпець sqlite_stat3.nEq містить приблизну кількість записів в індексі, найменший лівий стовпець точно відповідає вибірці. Sqlite_stat3.nLt містить приблизну кількість записів в індексі, найменший лівий стовпець якого менший за вибірку. Стовпець sqlite_stat3.nDLt містить приблизну кількість різних лівих записів в індексі, менших, ніж вибірки.

На індекс може бути довільна кількість записів sqlite_stat3. Проаналізуйте команду зазвичай генерує sqlite_stat3 таблиць , які містять від 10 до 40 зразків, які розподілені по всьому ключовому простору і з великими значеннями Neq.

У добре сформованій таблиці sqlite_stat3 зразки для будь-якого одного індексу повинні відображатися в тому ж порядку, що і в індексі. Іншими словами, якщо запис з найбільш лівим стовпцем S1 знаходиться раніше у дереві індексу b, ніж запис із найменшим стовпцем S2, то у таблиці sqlite_stat3 зразок S1 повинен мати менший рядковий від порівняння зразок S2.

Таблиця 2.6. Таблиця sqlite_stat4

tbl:	Стовпець sqlite_stat4.tbl містить ім'я таблиці, що володіє індексом, який описує рядок
idx:	У стовпці sqlite_stat4.idx міститься назва індексу, який описує рядок, або у випадку запису sqlite_stat4 для таблиці БЕЗ РОЗУМУ назва самої таблиці.
зразок:	Стовпець sqlite_stat4.sample містить BLOB у форматі запису, який кодує індексовані стовпці, за якими послідовно виконано rowid для таблиці rowid, або стовпчики первинного ключа для БЕЗ РОЗУМНОЇ таблиці. BLUE_stat4.sample BLOB для таблиці БЕЗ ROWID містить лише стовпці первинного ключа. Нехай кількість стовпців, закодованих блоком sqlite_stat4.sample, дорівнює N. Для індексів у звичайній рядкової таблиці N буде на один більше, ніж кількість індексованих стовпців. Для індексів БЕЗ ROWID таблиць N буде числом стовпців, що індексуються плюс кількість стовпців у первинному ключі. Для таблиці БЕЗ РОУДІВ N буде числом стовпців у первинному ключі.
nEq:	Стовпець sqlite_stat4.nEq містить список N цілих чисел, де K-число ціле число - це приблизна кількість записів в індексі, чий лівий K-стовпець K точно відповідає K-лівому стовпчику вибірки.
nLt:	Стовпець sqlite_stat4.nLt містить перелік N цілих чисел, де K-е ціле число - це приблизна кількість записів в індексі, K-більшість стовпців ліворуч у сукупності менше, ніж K-ліві стовпці вибірки.
nDLt:	Стовпець sqlite_stat4.nDLt містить список N цілих чисел, де K-число ціле число - це приблизна кількість записів в індексі, які є чіткими в перших стовпцях K і де найбільше лівих K стовпців збігається менше, ніж лівий-найбільше K стовпці вибірки.

Sqlite_stat4 створюється лише тоді, коли SQLite компілюється з SQLITE_ENABLE_STAT4 та якщо номер версії SQLite становить 3.8.1 (2013-10-17)

або вище. Таблиця sqlite_stat4 не читається і не записується жодною версією SQLite до 3.8.1. Таблиця sqlite_stat4 містить додаткову інформацію про розподіл ключів в індексі або розподіл ключів у первинному ключі БЕЗ РОЗВИДАНОЇ таблиці. Планувальник запитів іноді може використовувати додаткову інформацію в таблиці sqlite_stat4 для створення кращих та швидших алгоритмів запитів. Схема таблиці sqlite_stat4 така:

CREATE TABLE sqlite_stat4 (tbl, idx, nEq, nLt, nDLt, зразок);

Як правило, від 10 до 40 записів у таблиці sqlite_stat4 для кожного індексу, для якого доступна статистика, однак ці обмеження не є складними межами. Значення стовпців таблиці sqlite_stat4 такі:

Sqlite_stat4 - це узагальнення таблиці sqlite_stat3. Таблиця sqlite_stat3 надає інформацію про лівий стовпець індексу, тоді як таблиця sqlite_stat4 містить інформацію про всі стовпці індексу.

На індекс може бути довільна кількість записів sqlite_stat4. Проаналізуйте команду зазвичай генерує sqlite_stat4 таблиць , які містять від 10 до 40 зразків, які розподілені по всьому ключовому простору і з великими значеннями Neq.

У добре сформованій таблиці sqlite_stat4 зразки для будь-якого одного індексу повинні відображатися в тому ж порядку, що і в індексі. Іншими словами, якщо запис S1 знаходиться раніше у дереві індексу b, ніж запис S2, то в таблиці sqlite_stat4 зразок S1 повинен мати менший рядковий від порівняння зразок S2.

2.3. Журнал відкату

Журнал відкату - це файл, пов'язаний із кожним файлом бази даних SQLite, який містить інформацію, яка використовується для відновлення файлу бази даних до його початкового стану під час транзакції. Файл журналу відкату завжди знаходиться в тому ж каталозі, що і файл бази даних, і має те саме ім'я, що і файл бази даних, але додається рядок " -journal ". Тут може бути лише один журнал відката, пов’язаний із базою даних, і, отже, одночасно може бути відкрита лише одна транзакція запису проти однієї бази даних.

Якщо транзакція перервана через збій програми, збої в операційній системі або збої або збої в роботі апаратного забезпечення, то база даних може залишитися в непослідовному стані. Наступного разу, коли SQLite намагатиметься відкрити файл бази даних, буде виявлено наявність журналу відкату, і журнал буде автоматично відтворюватися для відновлення бази даних до її стану на початку неповної транзакції.

Журнал відката вважається дійсним лише тоді, коли він існує та містить дійсне заголовка. Отже, транзакцію можна здійснити одним із трьох способів:

- Файл журналу відкату можна видалити;
- Файл журналу відкату можна скоротити до нульової довжини;
- Заголовок журналу відкату можна перезаписати невірним текстом заголовка (наприклад, усі нулі).

Ці три способи здійснення транзакції відповідають налаштуванням DELETE, TRUNCATE та PERSIST відповідно до прагми журналу_mode відповідно .

Дійсний журнал відката починається із заголовка у такому форматі:

Таблиця 2.7. Формат заголовка журналу відкат

Зсув	Розмір	Опис
0	8	Рядок заголовка: 0xd9, 0xd5, 0x05, 0xf9, 0x20, 0xa1, 0x63, 0xd7
8	4	«Кількість сторінок» – кількість сторінок у наступному сегменті журналу, або -1, щоб означати весь вміст до кінця файлу
12	4	Випадкове значення для контрольної суми
16	4	Початковий розмір бази даних у сторінках
20	4	Розмір дискового сектора передбачається процесом, який писав цей журнал.
24	4	Розмір сторінок у цьому журналі.

Заголовок журналу відкату закреслений нулями до розміру одного сектора (як визначено цілим числом сектора при зміщенні 20). Заголовок знаходиться в секторі сам по собі, так що якщо втрата електроенергії відбудеться під час запису сектора, інформація, що слідує за заголовком, буде (сподіваємось) непошкодженою.

Після заголовка та нульової прокладки – це нуль або більше записів сторінок. Кожна сторінка записує копію вмісту сторінки з файлу бази даних до її зміни. Одна і та сама сторінка може не з’являтися більше одного разу в одному журналі про відкат. Для відкату неповної транзакції процес повинен просто прочитати журнал відкату від початку до кінця та записувати сторінки, знайдені в журналі, назад у файл бази даних у відповідному місці.

Нехай розмір сторінки бази даних (значення цілого числа при зміщенні 24 у заголовку журналу) дорівнює H. Тоді формат запису сторінки такий:

Таблиця 2.8. Формат запису сторінки відкатного журналу

Зсув	Розмір	Опис
0	4	Номер сторінки у файлі бази даних
4	N	Оригінальний вміст сторінки до початку транзакції
N + 4	4	Контрольна сума

Контрольна сума - це непідписане 32-бітове ціле число, обчислене таким чином:

Ініціалізуйте контрольну суму до значення нульової контрольної суми, знайденого в заголовку журналу, при зміщенні 12.

Ініціалізуйте індекс X як N-200 (де N - розмір сторінки бази даних у байтах.

Інтерпретуйте байт при зміщенні X на сторінку як 8-бітне ціле число без підпису та додайте значення цього цілого числа до контрольної суми.

Віднімiть 200 від X.

Якщо X більше або дорівнює нулю, поверніться до кроку 3.

Значення контрольної суми використовується для захисту від неповних записів записів сторінки журналу після відключення живлення. Кожен раз, коли починається транзакція, використовується інша випадкова записка, щоб мінімізувати ризик того,

що неписані сектори випадково можуть містити дані з тієї ж сторінки, що була частиною попередніх журналів. Змінюючи поняття для кожної транзакції, несвіжі дані на диску все одно генеруватимуть неправильну контрольну суму та будуть виявлені з високою ймовірністю. Контрольна сума використовує лише рідкісний зразок 32-розрядних слів із запису даних з міркувань продуктивності - проектні дослідження під час фаз планування SQLite 3.0.0 показали значне враження продуктивності при контрольній сумі всій сторінці.

Нехай значення підрахунку сторінок при зміщенні 8 у заголовку журналу буде M. Якщо M більше нуля, то після запису на M сторінку журнальний файл може бути нульовим, доповненим наступним кратним розміром сектора, і може бути вставлений інший заголовок журналу. Усі заголовки журналу в одному журналі повинні містити однаковий розмір сторінки бази даних та розмір сектора.

Якщо в початковому заголовку журналу M дорівнює -1, то кількість записів на сторінці, що випливає, обчислюється шляхом обчислення кількості записів сторінок, що вміщуватимуться у доступному просторі, що залишився у файлі журналу.

2.4. Журнал запису наперед

Починаючи з версії 3.7.0 (2010-07-21), SQLite підтримує новий механізм управління транзакціями, який називається "журнал запису" або "WAL". Коли база даних знаходиться в режимі WAL, всі з'єднання з цією базою даних повинні використовувати WAL. Конкретна база даних буде використовувати журнал відкату або WAL, але не обидва одночасно. WAL завжди знаходиться в тому ж каталозі, що і файл бази даних, і має те саме ім'я, що і файл бази даних, але додається рядок " -wal ".

2.4.1. Формат файлу WAL

WAL файл складається з заголовка , за яким слід нуль або більше «кадри». Кожен кадр записує переглянутий вміст однієї сторінки з файлу бази даних. Усі зміни в базі даних реєструються, записавши кадри в WAL. Операції здійснюються, коли записаний кадр, який містить маркер фіксації. Одна WAL може

і, як правило, записує кілька транзакцій. Періодично вміст WAL передається назад у файл бази даних в рамках операції, званої "контрольна точка".

Один WAL-файл можна використовувати повторно. Іншими словами, WAL може заповнити кадри, а потім перевірити, а потім нові кадри можуть замінити старі. WAL завжди зростає від початку до кінця. Контрольні суми та лічильники, приєднані до кожного кадру, використовуються для визначення того, які кадри в межах WAL є дійсними та які є залишками попередніх контрольних точок.

Заголовок WAL має розмір 32 байти і складається з наступних восьми великих 32-бітових цілочисельних знаків без ензиму:

Таблиця 2.9. Формат заголовка WAL

Зсув	Розмір	Опис
0	4	Магічне число. 0x377f0682 або 0x377f0683
4	4	Версія формату файлу. В даний час 3007000.
8	4	Розмір сторінки бази даних. Приклад: 1024
12	4	Послідовний номер контрольної точки
16	4	Сіль-1: випадкове ціле число, що збільшується з кожною контрольною точкою
20	4	Сіль-2: різне випадкове число для кожної контрольної точки
24	4	Checksum-1: перша частина контрольної суми на перших 24 байтах заголовка
28	4	Checksum-2: Друга частина контрольної суми на перших 24 байтах заголовка

Одразу слідом за заголовком wal є нульовий або більше кадрів. Кожен кадр складається з 24-байтового заголовка кадру з наступним байтом даних про розмір сторінки. Заголовок кадру - це шість 32-бітових цілих цілочисельних знаків без великого ендіану:

Таблиця 2.10. Формат заголовка кадру WAL

Зсув	Розмір	Опис
0	4	Номер сторінки
4	4	Для записів фіксації розмір файлу бази даних на сторінках після фіксації. Для всіх інших записів нуль.
8	4	Сіль-1 скопійовано із заголовка WAL
12	4	Сіль-2 скопійовано із заголовка WAL
16	4	Контрольна сума-1: Сукупна контрольна сума через та включення цієї сторінки
20	4	Контрольна сума-2: друга половина сукупної контрольної суми

Кадр вважається дійсним, якщо і лише за умови виконання наступних умов:

Значення солі-1 та солі-2 у кадрі-заголовку відповідають значенням солі у wal-заголовку

Значення контрольної суми у підсумкових 8 байтах заголовка кадру точно відповідають контрольній сумі, обчисленій послідовно на перших 24 байтах заголовка WAL та перших 8 байтах та вмісті всіх кадрів до поточного кадру та включаючи його.

2.4.2. Алгоритм контрольної суми

Контрольна сума обчислюється, інтерпретуючи вхід як парне число непідписаних 32-бітових цілих чисел: x (0) - x (N). 32-розрядні цілі числа є великими, якщо магічне число в перших 4-х байтах заголовка WAL дорівнює 0x377f0683, а цілі числа є мало-ендіанськими, якщо магічне число 0x377f0682. Значення контрольної суми завжди зберігаються у заголовку кадру у форматі big-endian незалежно від того, який порядок байт використовується для обчислення контрольної суми.

Алгоритм контрольної суми працює лише для вмісту, який є кратним 8 байт у довжину. Іншими словами, якщо входи від x (0) до x (N), то N має бути непарним. Алгоритм контрольної суми такий:

s0 = s1 = 0

```
для і від 0 до n-1 крок 2:
  s0 + = x (i) + s1;
  s1 + = x (i + 1) + s0;
endfor
# результат у s0 та s1
```

Виходи s0 та s1 є зваженими контрольними сумами, використовуючи ваги Фібоначчі у зворотному порядку. (Найбільша вага Фібоначчі виникає на першому елементі послідовності, що підсумовується.) Значення s1 охоплює всі 32-бітні цілочисельні члени послідовності, тоді як s0 - кінцевий член.

2.4.3. Алгоритм контрольної точки

На контрольній точці WAL спочатку очищається до постійного зберігання за допомогою методу xSync VFS . Потім дійсний вміст WAL передається у файл бази даних. Нарешті, база даних очищається до постійного сховища за допомогою іншого виклику методу xSync. Операції xSync служать перешкодою для запису - всі записи, запущені до xSync, повинні завершитися перед тим, як записувати, що запускається після початку xSync.

Контрольно-пропускний пункт не повинен працювати до завершення. Можливо, деякі читачі все ще використовують старі транзакції з даними, які містяться у файлі бази даних. У такому випадку перенесення вмісту для нових транзакцій з файлу WAL в базу даних видалить вміст з-під читачів, які все ще використовують старі транзакції. Щоб уникнути цього, контрольно-пропускні пункти запускаються до завершення лише у тому випадку, коли всі читачі використовують останню транзакцію в WAL.

2.4.4. Скидання WAL

Після повної контрольної точки, якщо ніяких інших з'єднань немає в транзакціях, що використовують WAL, то наступні транзакції запису можуть замінити файл WAL з початку. Це називається "скидання WAL". На початку першої нової операції запису значення WAL заголовка сіль-1 збільшується, а значення солі-

2 рандомізоване. Ці зміни в солях недійсні для старих кадрів у WAL, які вже були перевірені, але ще не переписані, і не дозволяють повторно перевірити їх.

Файл WAL за бажанням може бути усічений, але цього не потрібно. Продуктивність зазвичай трохи краща, якщо WAL не обрізаний, оскільки файлові системи, як правило, замінять наявний файл швидше, ніж вони виростуть файл.

2.4.5. Алгоритм читання

Щоб прочитати сторінку з бази даних (називайте її номером сторінки P), читач спочатку перевіряє WAL, щоб побачити, чи містить вона сторінку P. Якщо так, то останній дійсний екземпляр сторінки P, за яким слідує кадр фіксації або є Кадр фіксації сам стає значенням прочитаного. Якщо WAL не містить копій сторінки P, які є дійсними, і які є фреймом фіксації, або слідом за ними кадр фіксації, то сторінка P читається з файлу бази даних.

Щоб розпочати транзакцію зчитування, зчитувач записує кількість кадрів значень у WAL як "mxFrame". (Більш детально) Читач використовує це записане значення mxFrame для всіх наступних операцій зчитування. Нові транзакції можуть бути додані до WAL, але поки читач використовує своє оригінальне значення mxFrame і ігнорує згодом доданий контент, читач побачить послідовний знімок бази даних з однієї точки часу. Цей прийом дозволяє одночасно читачам переглядати різні версії вмісту бази даних.

Алгоритм зчитування в попередніх параграфах працює правильно, але оскільки кадри для сторінки P можуть з’являтися в будь-якому місці WAL, читачеві необхідно сканувати весь WAL, шукаючи кадри сторінки P. Якщо WAL великий (типовий кілька мегабайт), сканування може бути повільним, а продуктивність читання страждає. Для подолання цієї проблеми підтримується окрема структура даних, яка називається wal-index, щоб прискорити пошук кадрів певної сторінки.

2.4.6. Формат індексу WAL

Концептуально wal-index є спільною пам'яттю, хоча в поточних реалізаціях VFS використовується файл, відображений у пам'яті для перенесення операційної системи. Файл, відображений на пам'ять, знаходиться в тому ж каталозі, що і база даних, і має те саме ім’я, що і база даних із доданим суфіксом " -shm ". Оскільки wal-index є спільною пам'яттю, SQLite не підтримує journal_mode = WAL в мережевій файловій системі, коли клієнти знаходяться на різних машинах, оскільки всі клієнти бази даних повинні мати можливість спільної пам’яті.

Мета wal-index - швидко відповісти на це питання:

З огляду на номер сторінки P та максимальний індекс кадру WAL M, поверніть найбільший індекс кадру WAL для сторінки P, що не перевищує M, або поверніть NULL, якщо для сторінки P немає кадрів, які не перевищують M.

Значення M у попередньому параграфі - це значення "mxFrame", визначене в розділі 4.4, яке зчитується на початку транзакції і яке визначає максимальний кадр з WAL, який читач використовуватиме.

Індекс wal є тимчасовим. Після аварії Wal-індекс реконструюється з вихідного файлу WAL. VFS потрібно або скорочувати, або нулювати заголовок wal-індексу, коли завершується останнє з'єднання з ним. Оскільки wal-index є тимчасовим, він може використовувати специфічний для архітектури формат; це не повинно бути кросплатформенним. Отже, на відміну від форматів баз даних та файлів WAL, які зберігають усі значення як великі ендіани, wal-індекс зберігає багатобайтові значення в натурному порядку байтів хост-комп'ютера.

3 ОПИСАНО ПРОЦЕС ПРОЕКТУВАННЯ І РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДОСЛІДЖЕННЯ

3.1 Зовнішнє проектування

3.1.1 Вхідні дані

Вхідними даним даної системи є:

- файл бази даних (опція -f) для проведення аналізу на вміст видалених даних – треба передати повний шлях до бази даних, або ім’я файлу, якщо він знаходиться поруч із програмою;
- тип виводу (опція -o) результатів пошуку даних – у якому форматі виводити знайдені дані: hex-значення, ascii-літери, або нічого, якщо вас інтересує тільки інформація по аналізу файлу бази даних (розмір сторінки даних, кількість сторінок даних, тип кодування і т.і.);
- опція розширеного виводу (опція -v) – вмикає вивід додаткової інформації по аналізу файлу бази даних (розмір кешу, розмір зарезервованого місця і т.і.).

Вхідні параметри мають пройти валідацію.

При вказаних невірно параметрах їх список виводяться на екран:

```
C:\Users\Vlad\Documents\Projects\jeka\SqliteJekaRecover\recoversqlite-master>C:\python27-x64\python.exe sqliterec.py -f
JioCloud_DeletedRowsForTestingJekaProgra
error: file JioCloud_DeletedRowsForTestingJekaProgra not found!
Usage: sqliterec.py [options] -f <file>

Options:
    -h          this help
    -v          verbose
    -f <file>   sqlite file
    -o [h|a|n]  output as hex (default), ascii, or nothing
```

3.1.2 Вихідні дані

Вихідними даними програмного застосунку є інформація про базу даних та знайдені данні у форматі, який був запрошений через вхідні опції.

Вміст інформації по аналізу файлу бази даних еквівалентний переданній базі данних, яку користувач передає при роботі з програмою через інтерфейс терміналу. Дані містять наступну інформацію:

- заголовок бази даних;
- розмір сторінки.

3.1.3 Формалізація задачі.

Формалізація задачі зовнішнього проектування представлена у вигляді діаграми варіантів використання (рис. 3.1).

Користувач в даному випадку – дослідник, який взаємодіє із програмним забезпеченням представлений у вигляді актора. Прецеденти відображають можливі способи використання даного ПО.

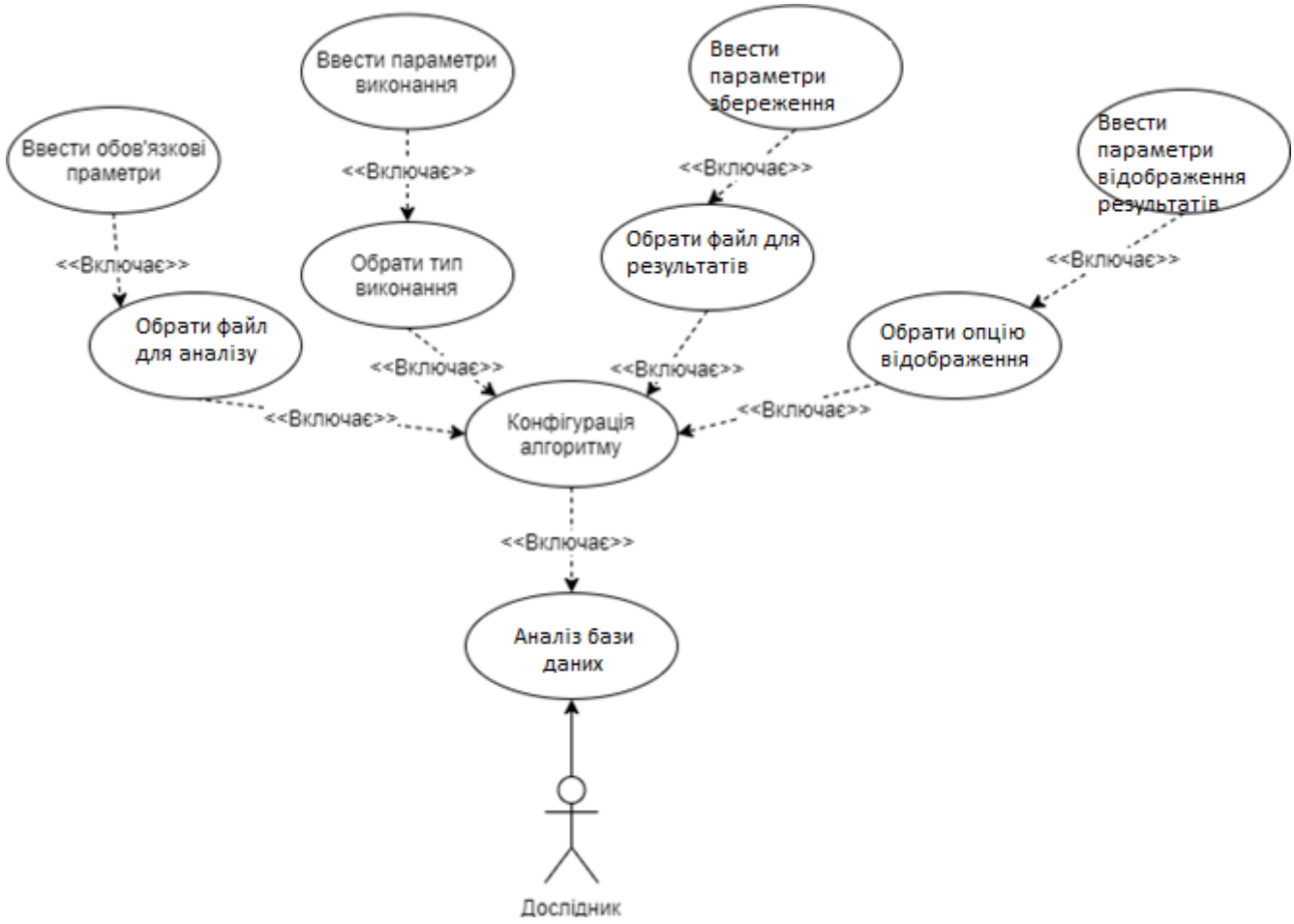

Рисунок 3.1 – Діаграма прецедентів

3.2 Базова архітектура системи

Так як розроблювальний інструментарій має за головну мету – дослідження можливості відновлення видалених даних у файлах баз даних SQLite – структура та користувацький інтерфейс доволі прості. Також слід відмітити, що основна увага була приділена можливим способам відображення знайдених даних, для спрощення аналізу. Проте під час проектування було враховано подальше змінення користувацького інтерфейсу, а також можливе використання в вигляді окремої бібліотеки. Дана можливість була досягнута за рахунок використання 2 окремих модулів програми:

- для аналізу файлу бази даних;
- для керування модулем аналізу файлів баз даних і відображення результатів.

3.3 Внутрішнє проектування

3.3.1 Огляд обраної мови програмування

Вибір мови програмування був серед популярних об’єктно-орієнтованих.

Для створення даного програмного інструментарію було використано мову C# на платформі .NET.

Платформа .NET – це безкоштовна платформа з відкритим кодом для створення різноманітних типів програмного забезпечення. Саме завдяки даним характеристикам даної платформи, в подальшому можна буде провести тестування та експерименти на різник платформах та операційних системах.

Також слід зазначити, що для даної платформи існує велика спільнота, великий ресурс із документацією та можливостями проходити навчання з певних тем. Інструментарій, який необхідно мати для розробки під дану платформу – Visual Studio Community – безкоштовний та надає можливість оновлення і розширення під певні потреби.

Мова C# надає наступні переваги при розробці програмного забезпечення:

- підтримка строгої типізації даних, що, в свою чергу, надає можливість виявляти несумісності та помилки при операціях над різними типами даних на етапах розробки та компіляції;
- завдяки об’єктно-орієнтовному підходу є можливість проектувати системи, які можуть бути швидко розширюваними;
- немає необхідності витрачати час на вирішення проблем пов’язаних із витоком пам’яті, цим займається «збірник сміття»;
- велика кількість вбудованих бібліотек для вирішення різноманітних проблем;
- підтримка асинхронного виконання методів – немає потреби блокувати певний потік, поки виконується довготривала операція;

- наявність LINQ – технології, яка пришвидшує розробку певних елементів коду, які працюють із колекціями чи масивами даних;
- можливість створення багато поточних програм.

3.4 Проектування користувацького інтерфейсу

Під час проектування інтерфейсу необхідно виділити головну ціль створення додатку, оцінивши при цьому мінімальний функціонал необхідний для отримання результатів дослідження.

Користувацький інтерфейс було спроектовано у вигляді вікна із можливостями взаємодії шляхом послідовного вибору відповідних опцій або заповненню необхідних параметрів, шляхом натискання на відповідні графічні елементи інтерфейсу.

Так, як система проектувалась із мінімальними залежностями між елементами – це забезпечую програмний продукт можливістю розширення та змінення моделі представлення користувацького інтерфейсу іншою реалізацією

Висновки до розділу 3

В розділі описано процес зовнішнього та внутрішнього проектування інструментального забезпечення необхідного для проведення відновлення відиланих даних із СКБД SQLite.

Розроблена система, завдяки використанню шаблонів проектування, має можливість швидкого розширення функціоналу за потребою, за рахунок розподілення окремих обов’язків на окремі блоки.

4 ВИЗНАЧЕННЯ ДОСЛІДЖУВАНИХ ПРОГРАМНИХ АНАЛОГІВ ТА РЕЗУЛЬТАТИ ПРОВЕДЕНИХ ЕКСПЕРЕМЕНТІВ

4.1. Підготовка тестового середовища

В ході цього дослідження для перегляду баз даних використовувалася програма DB Browser for SQLite.

Посилання: https://sqlitebrowser.org.

Ця програма дозволяє як переглядати, так і редагувати вміст бази даних. Видаленні дані вона не відображає:

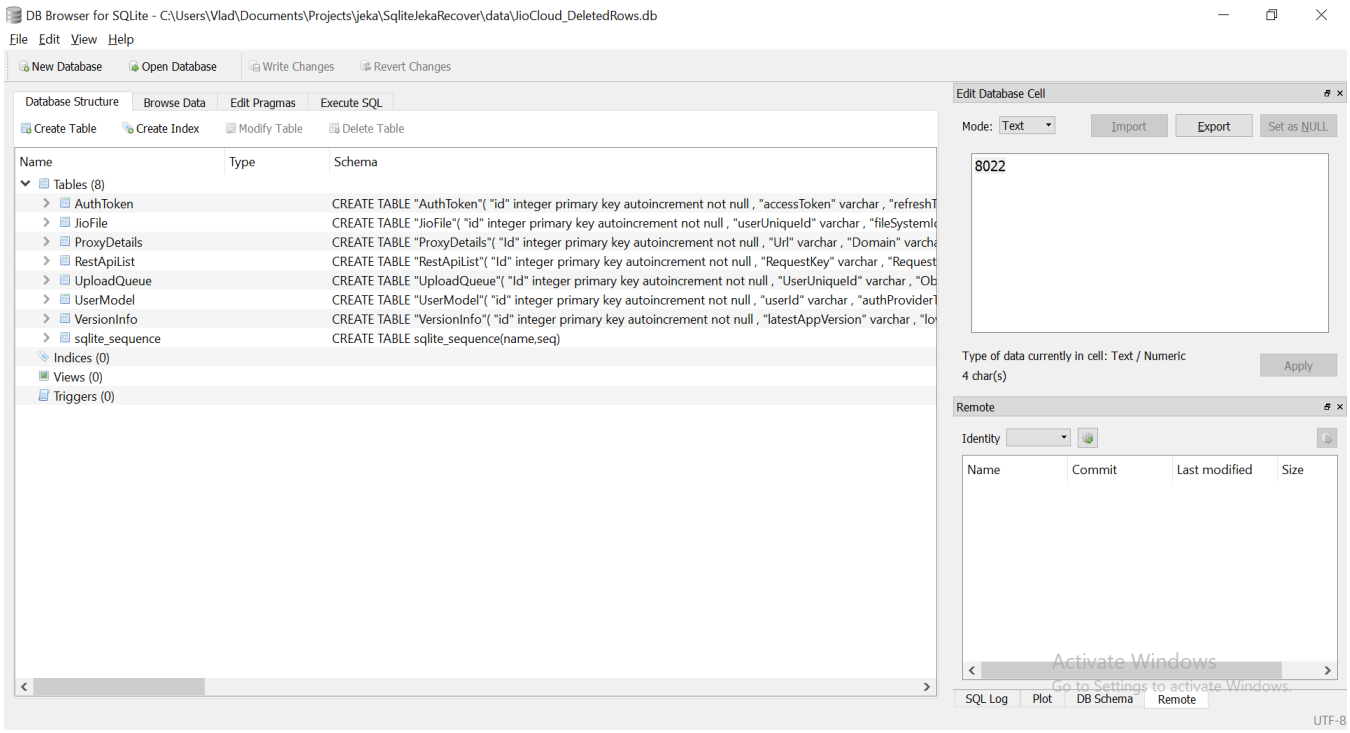

Рисунок 4.1. Користувацький інтерфейс додатку «DB Browser for SQLite»

Експерименти

Для проведення експериментів та перевірки роботи було розроблено програмний продукт що дозволяє:

- виконати аналіз заголовку та вивести інформацію про структуру бази даних;
- виконати операцію відновлення видалених даних;
- зберегти результати у обраний файл.

Використовуючи мову програмування Python було написано скрипт для автоматичного заповнювання бази даних данними та полегшення тестування. Скрипт використовує модуль sqlite3 для проведення операцій з базою даних.

Для запуску треба виконати наступну команда із терміналу Windows, за умови що Python-інтерпретатор вже встановлений у системі та прописаний у змінній середовища PATH:

```
python add_data.py
```

Python-скрипт для автоматичного заповнювання бази даних данними та полегшення тестування представлений нижче:

```
import sqlite3
from sqlite3 import Error

# значение ниже - это сколько надо вставить строк в таблицу с файлами
FILES_COUNT = 1000
# значение ниже - это имя базы в которую надо вставить файлы
DATABASE_PATH = "JioCloud.db"

def create_connection(db_file):
    conn = None
    try:
        conn = sqlite3.connect(db_file)
    except Error as e:
        print(e)
    return conn
```

```
def insert(conn, files):
    """
    Вставляем массив файлов в таблицу JioFile
    """
    sql = 'INSERT INTO JioFile (userUniqueId,sourceName) VALUES (?,?)'
    cur = conn.cursor()
    cur.executemany(sql, files)
    conn.commit()
    cur.close()


def main():
    print('opening db...')
    # подключаемся к базе данных
    conn = create_connection(DATABASE_PATH)


    with conn:
      # создаем массив из 1000 файлов.
      # файл - это пара их 2ух строк: имя пользователя, имя файла
      files = [('jeka','file_' + str(i)) for i in range(FILES_COUNT)]
      # вставляем в базу массив файлов
      print('inserting {} rows...'.format(FILES_COUNT))
      insert(conn, files)
      print('End')



if __name__ == '__main__':
    main()
```

В результаті виконання скрипту було створено тестову базу даних з таблицею JioFile на 1000 рядків (це значення регулюється відповідною змінною на початку скрипта, тобто його можна за потреби збільшити) на якій стало можливим проведення експерементів з видалення та наступного відновлення даних:

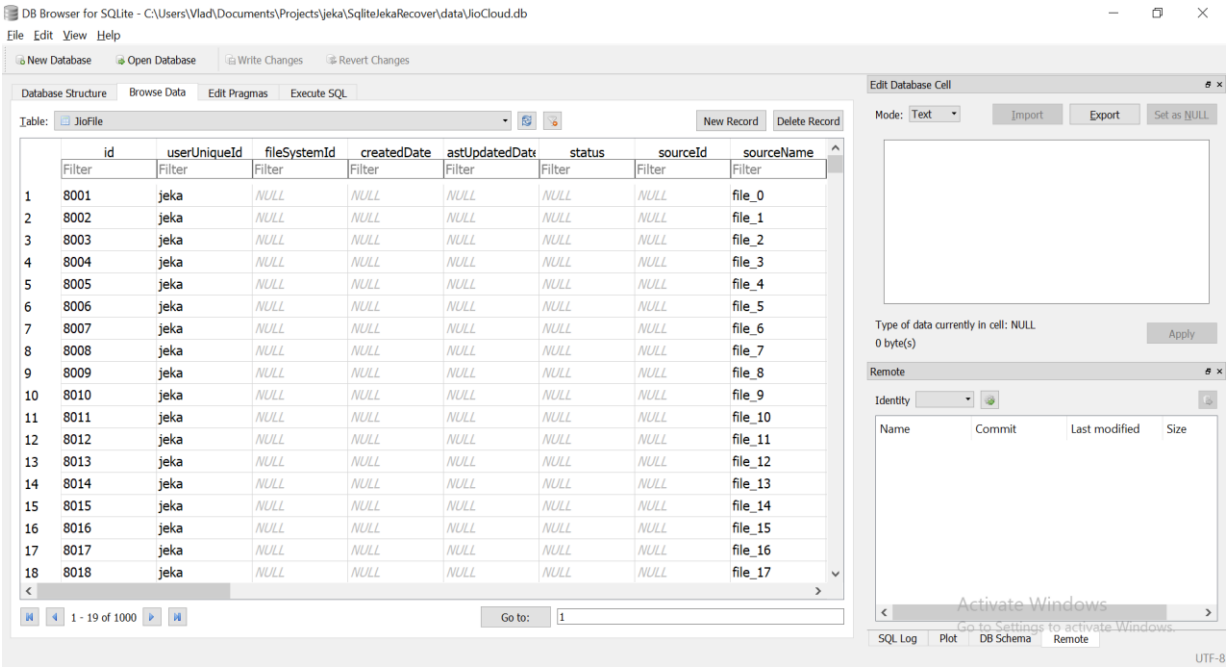

Рисунок 4.2. Наповнення тестової таблиці даними

Видалили 21 перший рядок з таблиці JioFile:

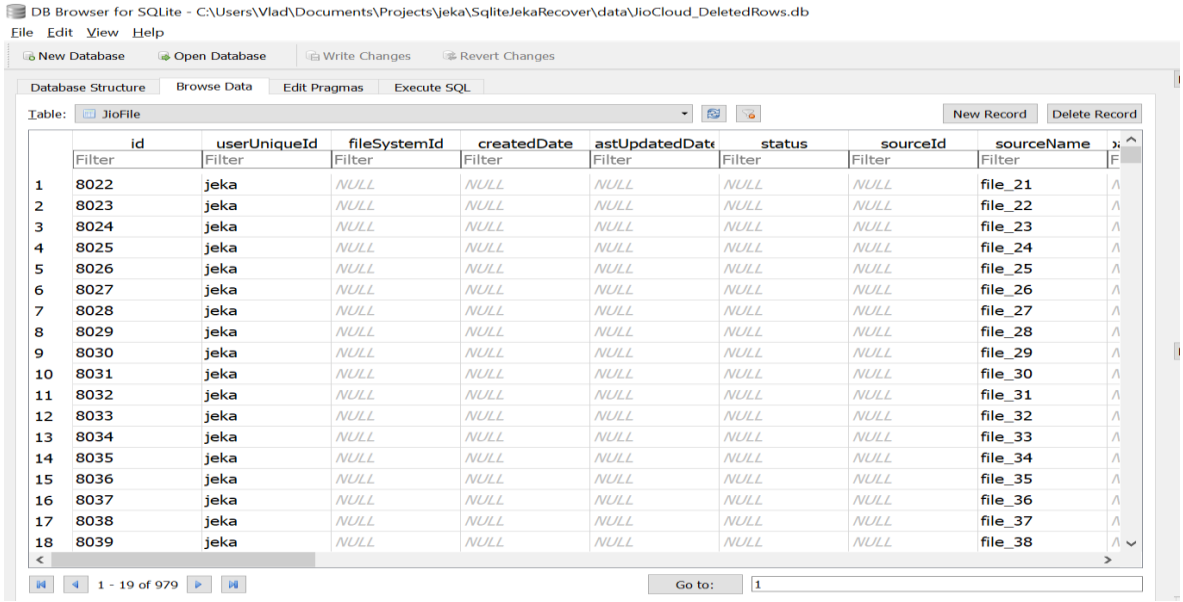

Рисунок 4.3. Видалення частини даних із тестової таблиці

Наступним кроком у проведеному дослідженні було проведення порівняння з 2-ма аналогами.

4.2. SQLite-Deleted-Records-Parser

Посилання: https://github.com/mdegrazia/SQLite-Deleted-Records-Parser

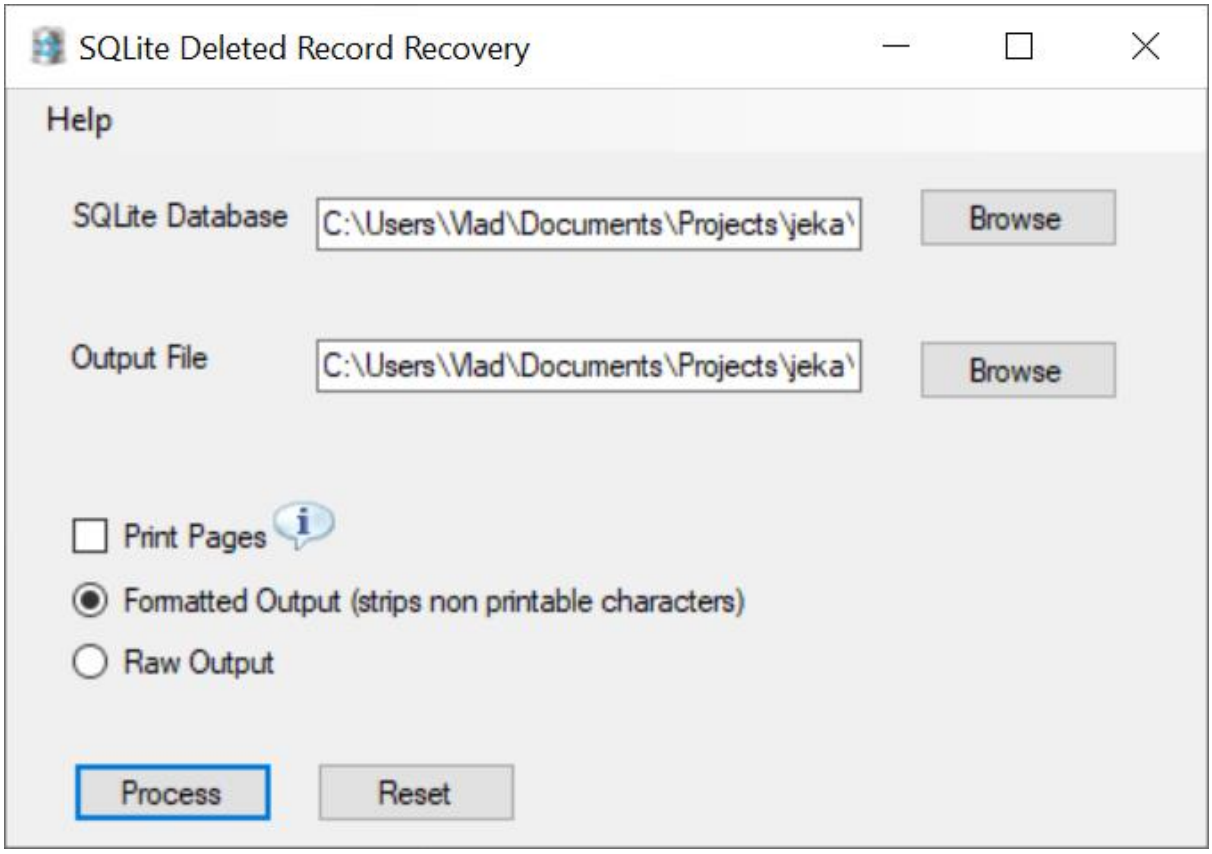

Рисунок 4.4. Користувацький інтерфейс додатку «SQLite Deleted Records Parser»

Після натискання “Process” програма створює файл report.tsv.

Цей файл можна переглянути в «Онлайн-засіб перегляду Excel».

Посилання: https://sheet.zoho.com/sheet/excelviewer.

Звіт до видалення

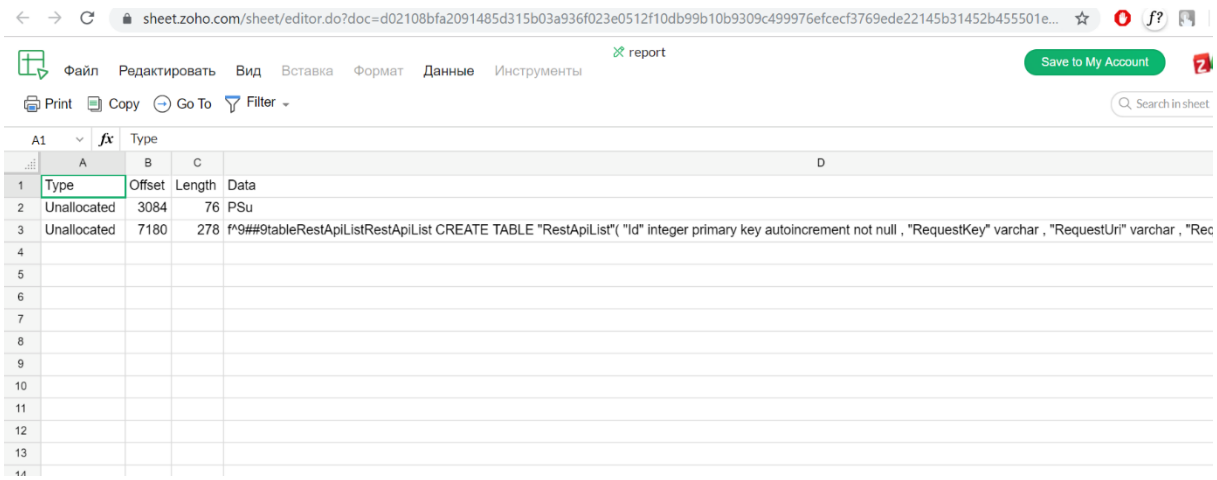

Рисунок 4.5. Звіт перед видаленням даних

Звіт після видалення

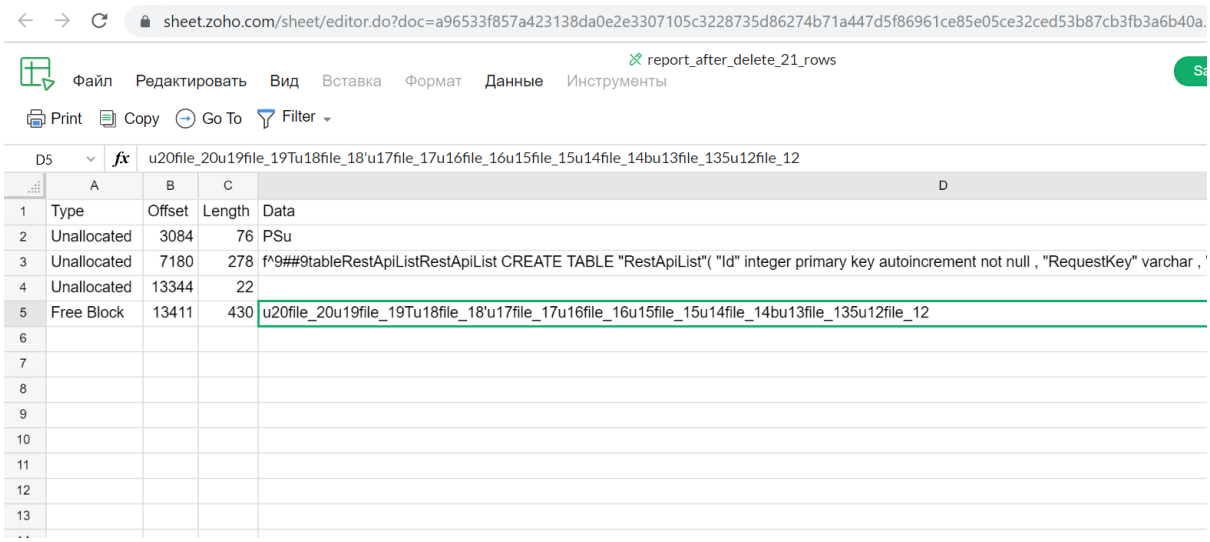

Рисунок 4.6. Звіт після видалення даних

Free Block - дані помічені як звільнені, тобто ті дані які видалили, але ще не записали на їх місце нові. Ці дані і є відновлені.

Загалом складності після відновлення було відновлено 40% даних.

4.3. Undark

Посилання: https://github.com/witwall/undark.

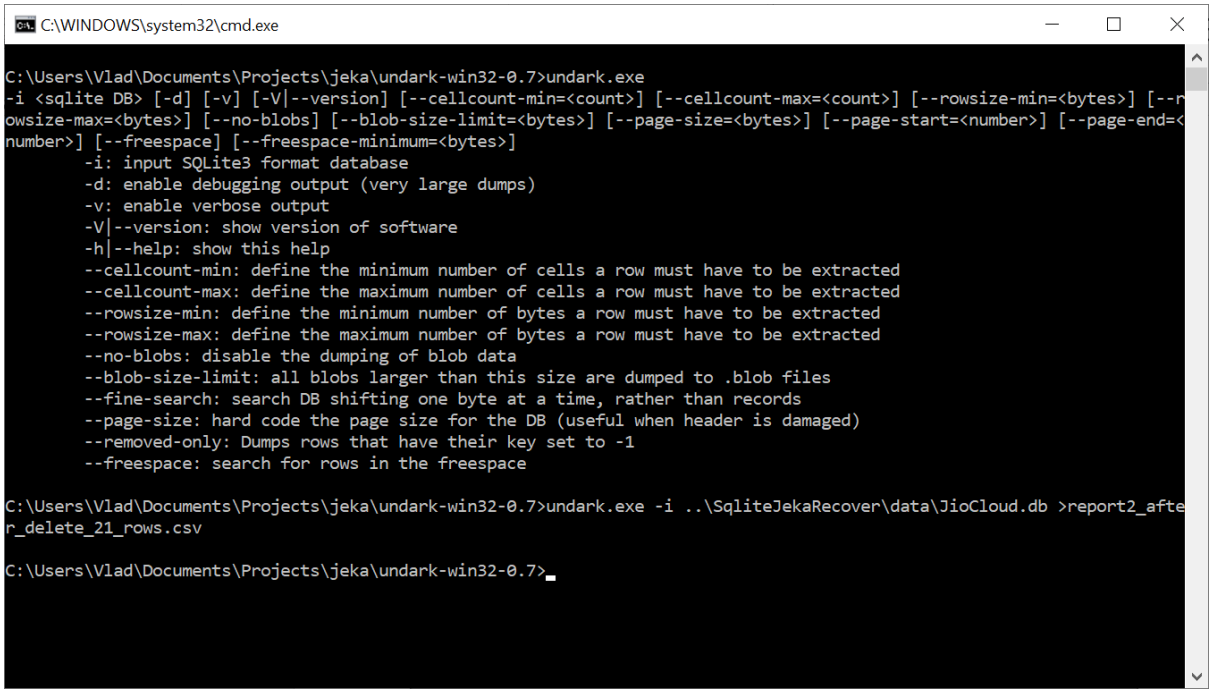

Рисунок 4.7. Консольний інтерфейс додатку «undark»

Після виконання команди вище на зображенні створюється файл report2_after_delete_21_rows.csv.

Згенерований файл можна переглянути в «Online CSV Viewer».

Посилання: https://www.becsv.com/csv-viewer.php.

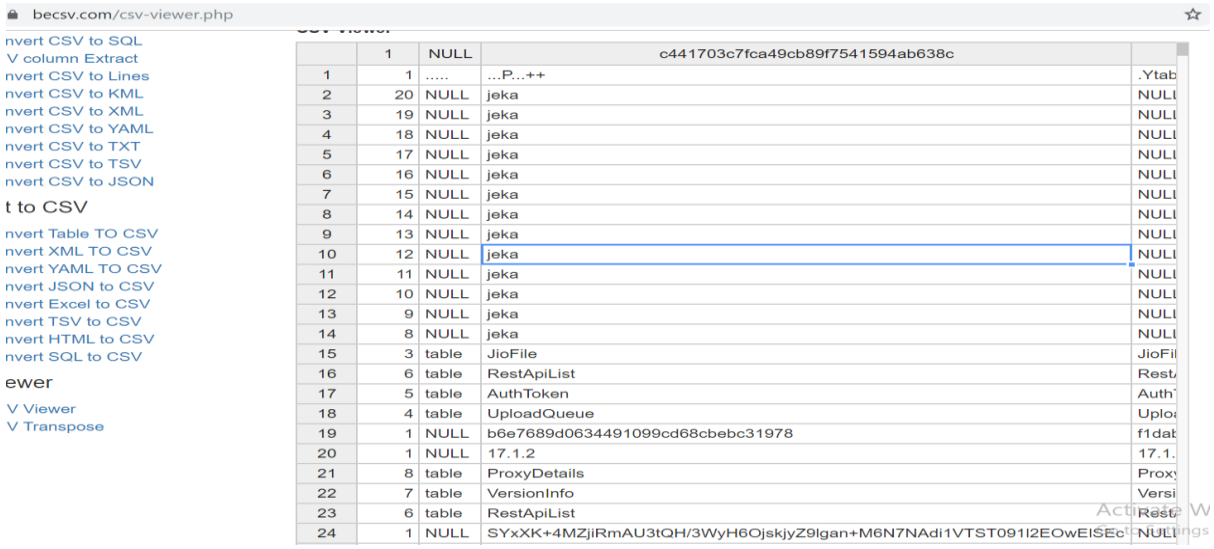

Рисунок 4.8. Створений файл з результатами аналізу додатку «undark»

Як бачимо дана програма виводить в csv файл весь вміст бази, разом з вилученими даними: рядка з ід (перший стовпець) від 8 до 21.

Загалом складності після відновлення було відновлено 60% даних.

4.4. Розроблена програма

Програма має наступні опції обробки файлу бази даних:

- аналіз файлу і вивід інформації про структуру бази даних, але без самого відновлення видалених даних (рис. 4.9);
- запуск операції пошуку і відновлення видалених даних у файлі бази даних, що вже включає в себе операцію аналізу файлу (рис. 4.10).

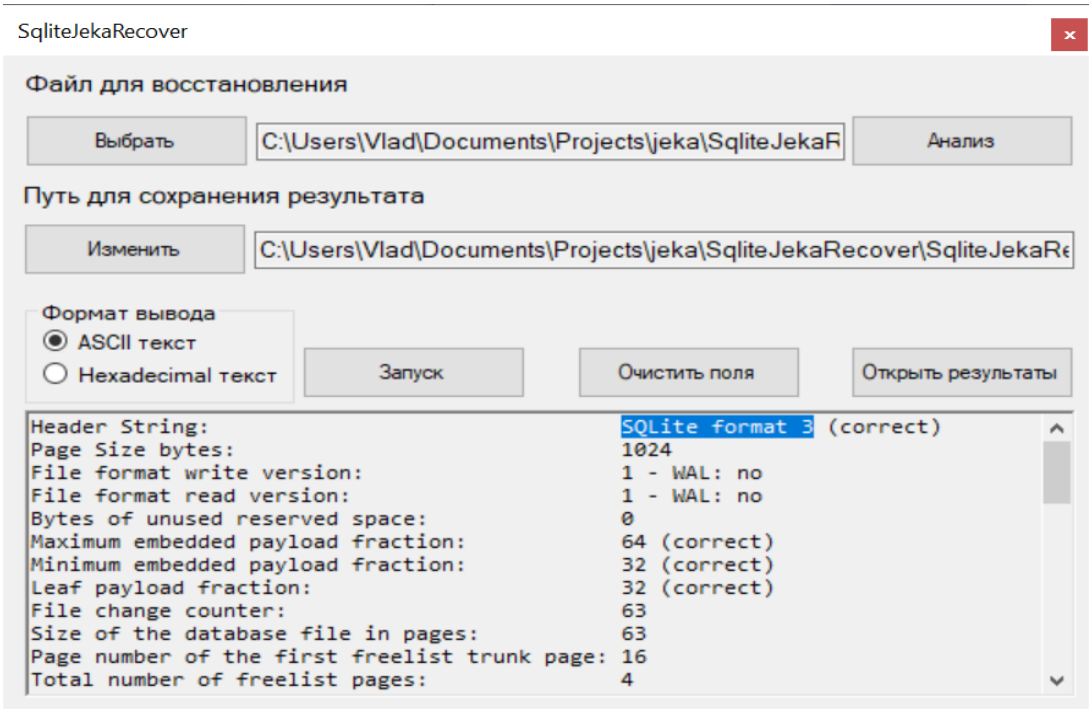

Рисунок 4.9. Аналіз файлу бази даних

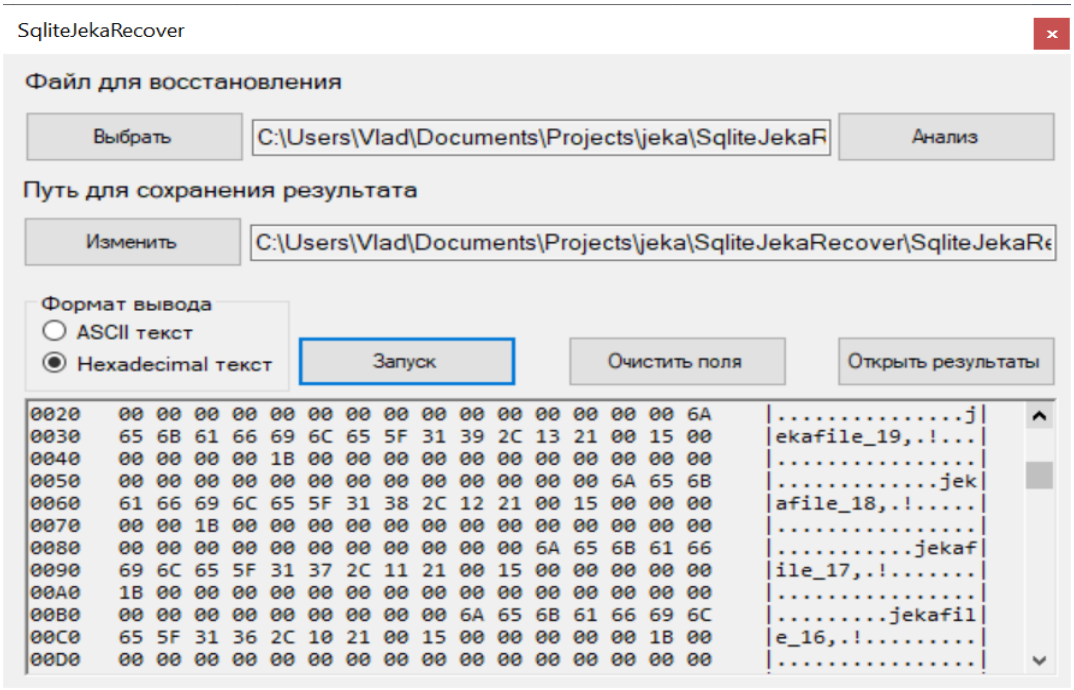

Рисунок 4.10. Запуск відновлення даних, hex-формат

У свою чергу для другої опції є можливість обрати режим відображення відновлених даних:

- ASCII текст - зручно, якщо вас цікавить насамперед текстова інформація у базі даних (рис. 4.11);
- Hexadecimal текст - якщо вас цікавить побайтовий вивід інформації (рис. 4.10).

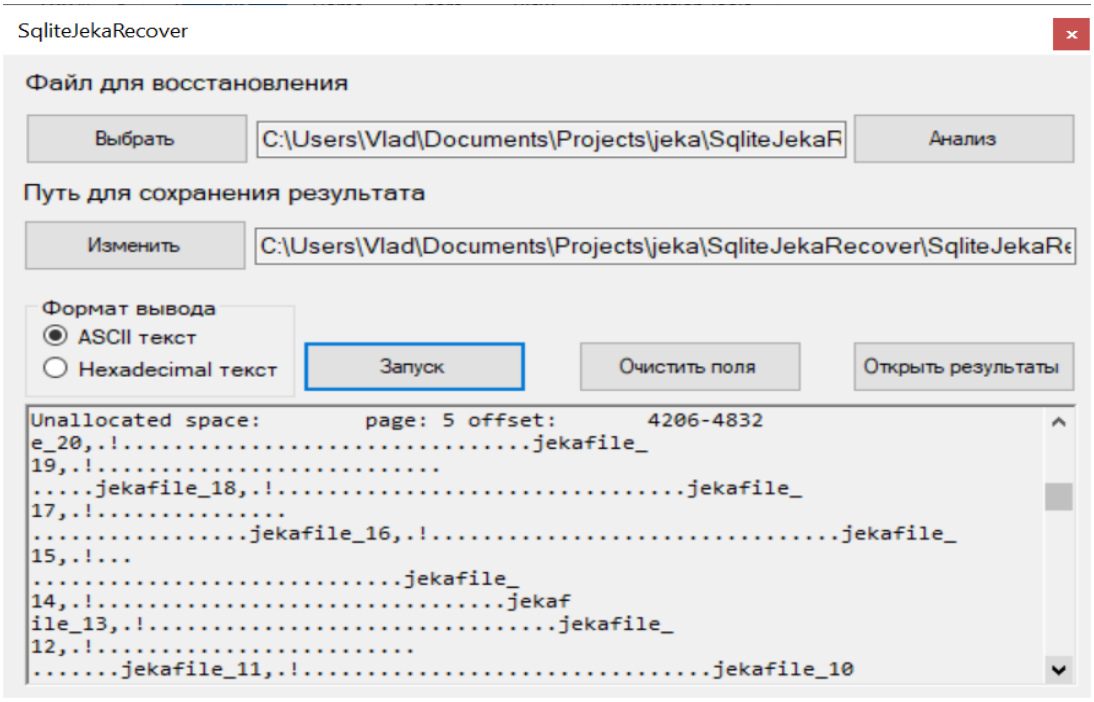

Рисунок 4.11. Запуск відновлення даних, ascii-формат

Висновки

Розроблений програмний інструмент дозволяє вирішити задачі відновлення даних, їх перегляду та збереження результатів. Також у цій роботі приводиться список програмних рішень, у яких використовується СКБД SQLite та які можна використовувати для тестування та аналізу методів відновлення даних на тих видах файлів баз даних, які використовують приведені у цій роботі програмні рішення для вирішення різноманітних повсякденних задач, як, наприклад, програми бухгалтерського обліку, браузерні рішення, чи програми засновані на їх движку (ядрі), як-от Firefox та Thunderbird.

Експериментальні результати показують, що запропонований алгоритм відновлення даних має високу надійність і дозволяє відновити приблизно 80% даних при умові, що вони ще не були перезаписані поверх іншими даними.

ВИСНОВКИ ДИПЛОМНОЇ РОБОТИ

В роботі було досліджено та проаналізовано сучасний стан реляційних СКБД, механізми зберігання даних та можливість відновлення видаленої інформаіїї.

Встановлено, що існує можливість відновлення видалених даних при умові, що вони ще не були перезаписані поверх іншими даними.

В роботі представлені інструменти для комплексної задачі відновлення даних, перегляду та збереження результатів, що зменшують час на пошук. Також у цьому документі приводиться список програмних рішень, у яких використовується СКБД SQLite та які можна використовувати для тестування та аналізу методів відновлення даних на тих видах файлів баз даних, які використовують приведені у статті програмні рішення для вирішення різноманітних повсякденних задач, як, наприклад, програми бухгалтерського обліку (1С), браузерні рішення, чи програми засновані на їх движку (ядрі), як-от Firefox та Thunderbird, різноманітні Android-додатки та iOS-додатки, які використовують SQLite як засіб постійного збереження даних, наприклад WhatsApp, Viber та інші месенджери.

Експериментальні результати показують, що запропонований алгоритм відновлення даних має високу надійність і дозволяє відновити приблизно 80% даних при умові, що вони ще не були перезаписані поверх іншими даними. Для проведення відповідних досліджень було створено програмний інструмент, використовуючи парадигму об’єктно-орієнтованого програмування із застосуванням шаблонів проектування. Програмний додаток легко модифікувати та вносити нові функції для експериментів чи розширення виконуваних задач.

В подальшому рекомендується розширення варіантів перегляду даних, пошуку інформації та оптимізації швидкості роботи програми для аналізу великих масивів даних.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. *Буй Д.Б.* Сложность операций в базах данных (Обзор) / Д.Б. Буй, В.Г. Скобелев / Радиоэлектронные и компьютерные системы.Харків. “ХАІ”. – 2014. – №2 (66). – С. 53-59.
2. *Klein, B.* An Introduction to IMS. Second Edition /B. Klein, R. Long, K. Ray // IBM Press, 2012, 622 p.
3. *Codd, E.F.* A Relational Model of Data for Large Shared Data Banks / E.F. Codd // Communications of the ACM. – 1970. – Vol. 13, num. 6. – P. 377-387.4.*Мейер Д.*Теория реляционных баз данных: [пер. сангл.] / Д. Мейер. – Москва: Мир, 1987. – 608 с.
4. *Редько В.Н.* Реляційні бази даних: табличні алгебри та SQL-подібні мови / В.Н. Редько, Ю.Й. Брона, Д.Б. Буй, С.А. Поляков. – К.: Видавничий дім “Академперіодика”, 2001. – 196 с.
5. BUI, D., SKOBELEV, V. (2014) *Complexity of operations in database systems (a survey).*Radioelectronic and computersystems 66 (2), pp. 5359.
6. KLEIN, B.,LONG, R. RAY, K. (2012) *AnIntroduction to IMS. Second Edition*. IBM Press.
7. CODD, E. (1970) A Relational Model of Data for Large Shared Data Banks. *Communication softhe ACM*. Vol. 13 (6), pp. 377-387.
8. MAIER, D. (1983) *The theory of relational databases*. ComputerSciencePress.
9. REDKO, V., BRONA, Y, BUY, D, POLYAKOV, S (2001) *Relyacionniebazydannych: tablichnyalgebry ta SQL-podibnimovy*. K.:Academ periodika.
10. *Кирстен В.*Постреляционная СУБДCache. Объектно-ориентированная разработка приложений / Вольфанг Кирстен, Михаэль Ирингер. – М.: Бином, 2008. – 401 с.
11. *Фридман А.Л.* Основы объектноориентированной разработки программных систем / А.Л. Фридман – М.: “Финансы и статистика”, 2000. – 190с.
12. *Cook, W.* Object-Oriented Programming Versus Abstract Data Types [Online] – Hewlett-Packard Laboratories. Availablefrom:

    www.cs.utexas.edu/users/wcook/papers/OOPvsADT/CookOOPvsADT90.pdf
13. *Bubel R.* A Formalisation of Java Strings for Program Specification and Verification.

14. *Дейт К.Дж*. Введение в системы баз данных, 8-е издание.: Пер. С англ. / К.Дж. Дейт. – М., СПб, Киев: “Вильямс”, 2005. − 1328с.

15. *Конолли Т.* Базы данных. Проектирование, реализация и сопровождение. Теория и практика. Третьеиздание / Т. Конолли, К. Бегг. – Москва Санкт-Петербург-Киев, 2003. – 1427 с.

16. *Гарсиа-Молина Г.* Системы баз данных.

Полныйкурс.:Пер. сангл. / ГекторГарсиаМолина, ДжеффриД. Ульман. – М.: Издательскийдом “Вильямс”, 2003. – 1088 с.

17. *Бурлаков П.В.*Введениевсистемыбазданных. Учебное пособие / П.В. Бурлаков, В.Ю. Петров. – Санкт-Петербург, 2010. – 129с.

18. *Грей П.*Логика, алгебраибазыданных. – М.: Машиностроение, 1989. – 360 с.

19. *Редько В.Н.* К основаниямтеорииреляционныхмоделейбазданных / В.Н. Редько, Д.Б. Буй // Кибернетикаисистемныйанализ . – 1996. – №4. – С. 3-12.

20. *Поляков С.А.*Композиційнасемантикаядра SQL-подібнихмов. Дисертаціяназдобуттявченогоступенякандидатафізико-математичнихнаук: спец.01.05.03“Математичнетапрограмнезабезпеченняобчислювальнихмашиніс истем” – К.: 2011.– 118 с.

21. KIRSTEN, V. et al. (2003) *A post-relational database Cache. Object-oriented application development.* Springer.

22. FRIDMAN, A. (2000) *Fundamentals of objectoriented software development.*Moskov:Financy i statistika.

23. COOK, W. (1990) *Object-Oriented Programming Versus Abstract Data Types.* [Online] HewlettPackardLaboratories. Availablefrom: www.cs.utexas.edu/users/wcook/papers/OOPvsADT/ CookOOPvsADT90.pdf

24. BUBEL, R. (2011) *A Formalisation of Java Strings for Program Specification and Verification* [Online] . – Available from:

25. DATE, C.J. (2003) *Introduction to Database Systems, 8th Edition.*Addison-Wesley.

26. CONOLLY, T., BEGG C. (2002) *Database systems. A practical approach to Design, Implementation, and Management. Thirdedition*. Addison-Wesley.

27. KROENKE, D. (2002) *Database processing. Fundamentals, Design, and Implementation.* Prentice Hall.

28. TEOREY, T. (1999) *Database Modeling and Design.*MorganKaufman.

29. GARSIA-MOLINA, H. at al. (2001) *Database Systems: The Complete Book.* Prentice Hall.

30. BURLAKOV, P. (2010) *Vvedenie v systemiibazdannych.Uchebnoeposobie*. S.-Peterburg.

31. GREY, P. (1989) *Logika, algebra ibazydannych*. M.:Mashinostroenie

32. REDKO, V., BUY, D. (1996) *On the foundations of the theory of relational database models*. CyberneticsandSystemsAnalysis.No 4. – pp. 3-12.

33. POLYAKOV, S. (1999)Compositional semantics of SQL-like language: tabular data structure, composition, examples.*Bulletin of Kiev University. August. Sci. science*. Vol. 1, 2. – pp. 130140, pp. 183-194.

34. *Глушко І.М.* Числення та розширеннясигнатур табличних алгебр. Дисертація наздобуттявченогоступенякандидатафізикоматематичнихнаук: спец. 01.05.01 – К.: 2013. – 142 с.

35. *Редько В.Н.* Экзистенциальные основаниякомпозиционнойпарадигмы / В.Н. Редько // Кибернетикаисистемныйанализ. – 2008. – № 2. – С. 3-12.

36. *Буй Д.Б.* Обзор современной теориинормализациивреляционныхбазахданных / Д.Б. Буй, А.В. Пузікова // Материалы XVII Международной конференции “Проблемытеоретическойкибернетики” (Казань, 16 – 20 июня 2014 г.). ПодредакциейЮ.И. Журавлева. – Казань: Отечество, 2014. – С. 39-43.

37. *Codd E.F.* A Relational Model of Database Management : version 2. / E. Codd. – Addison-Wesley, 1990. – 567 p.

38. *Дрибас В.П.*Реляционныемоделибазданнях / *В.П.*Дрибас.– Минск: Белорус. ун-т,1982. – 192 с.

39. *Лаврищева Е.М.* Software engineeringкомп'ютернихсистем. Парадигми, технологии,

Case-инструменты программирования /Е.М. Лаврищева. – К.: "Науковадумка", 2013. –282 с.

BaguiSikha. Achievements and Weaknesses of Object-Oriented Databases / SikhaBagui // Journal of object technology, vol. 2, № 4, July-August 2003. pp. 29-41.

40. *Buy Dmitriy*. Formal specification of the NoSQL data model / Dmitriy Buy, Sergey Polyakov, Iuliia Hryshko // Informatics'2013, November 5th-7th, Spiska Nova Ves, 2013, pp. 284-288.

41. REDKO, V. (2008) *Existential foundation compositional paradigm.* Journal"Cybernetics and systems analysis", no. 2., pp. 3-12.

42. BUI, D., PUZIKOVA, A. (2014) Obzor sovremennoy teorii normalisatsii v relyacionnych baz dannych. *Proceedings of the XVII International Conference "Problems of Theoretical Cybernetics"*, 16 – 20 June, Kazan, pp. 39-43.

43. CODD, E. (1990) *A Relational Model of Database Management: version 2.* / E. Codd. – Addison-Wesley, 1990. – 567 p.

44. DRIBAS, V. (1982) *Relyacionnie modeli baz dannich*. Minsk: Belorus. Universitet.

45. LAVRISHCHEVA, E (2013) *Software engineering komputernych system. Paradigmy, tehnologii, Case-instrumentyprogrammi-rovaniya*. K.:"Naukovadumka".

46. ABADI, M, CARDELLI, L. (1996) *A theory of objets,* Springer [Online] . – Available from: http://books.google.om.ua/books?id=4xT3LgCPP5U

47. BAGUI, SIKHA (2003) *Achievements and Weaknesses of Object-Oriented Databases* / SikhaBagui // Journal of object technology, vol. 2(4).pp. 29-41.

48. KUNGURTSEV, A.B., NEIZVESTNY,A.S.

(2007) *Mathematical model of objective representation of a relational database*. ProceedingsoftheOdessaPolytechnicUniversity. 27(1). pp. 130-134.

49. SADALAGE, J. PRAMOD, FOWLER, MARTIN (2012) NoSQL Distilled: *A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-WesleyProfessional. BUY, D., POLYAKOV, S., HRYSHKO, Y. *Formal specification of the NoSQL data model*. Informatics'2013, November 5th-7th, SpiskaNovaVes, 2013, pp. 284-288.