Міністерство освіти і науки України
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни

"СИСТЕМИ ОБРОБКИ РОЗПОДІЛЕНИХ БАЗ ДАНИХ"

для студентів спеціальності "Комп'ютерна інженерія"

Тернопіль 2021

Методичні вказівки до лабораторних робіт з дисципліни «Системи обробки розподілених баз даних” / Н.Я. Савка, Ю.М. Батько / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2021. 45 с.

Укладачі: Н.Я. Савка, к.т.н., старший викладач кафедри комп’ютерної інженерії, Західноукраїнський національний університет
Ю.М. Батько, к.т.н., старший викладач кафедри комп’ютерної інженерії, Західноукраїнський національний університет

Відповідальний за випуск: Березький О.М., д.т.н., професор, завідувач кафедри комп’ютерної інженерії, Західноукраїнський національний університет

Рецензенти: Луцик І.Б., к.т.н., доцент кафедри комп’ютерних технологій, Тернопільський національний педагогічний університет

Порплиця Н.П., к.т.н., доцент кафедри комп’ютерних наук, Західноукраїнський національний університет

Методичні вказівки розглянуто та рекомендовано до друку на засіданні кафедри комп’ютерної інженерії протокол № 6 від 28 січня 2021 р.

Методичні вказівки затверджено на засіданні групи забезпечення спеціальності "Комп’ютерна інженерія" протокол № 4 від 28 січня 2021 р.

Методичні вказівки затверджено на засіданні вченої ради факультету комп’ютерних інформаційних технологій протокол № 4 від 8 лютого 2021 р.

ЗМІСТ

ВСТУП

Методичні вказівки до виконання лабораторних робіт призначені для підготовки та виконання лабораторних робіт з дисципліни “Системи обробки розподілених баз даних” для студентів спеціальності “Комп’ютерна інженерія”.

Сьогодні інформація перетворилася на один із найбільш важливих ресурсів, а інформаційні системи стали необхідним інструментом практично у всіх сферах діяльності. Розвиток автоматизованих систем обробки даних характеризується зміною акценту з процедурної обробки даних на структуру і зберігання даних, що призводить до необхідності використання банків даних, що забезпечують зберігання та підтримку у системі інтегрованої бази даних (БД), яка є динамічною інформаційною моделлю предметної області, тобто деякої частини реального світу.

База даних – це сукупність взаємозв'язаних даних, що зберігаються разом, за наявності такої мінімальної надмірності, яка допускає їх використання оптимальним чином для одного або декількох додатків; дані запам'ятовуються так, щоб вони були незалежні від програм, що використовують ці дані. Для додавання нових або модифікації існуючих даних, а також для пошуку даних у базі даних застосовується загальний керований спосіб.

Дані структуруються таким чином, щоб забезпечити можливість подальшого нарощування додатків. Керування і підтримку баз даних здійснює система керування (управління) базами даних (СК(У)БД). Це складні програмні системи, що працюють на різних операційних платформах. Саме СКБД надає засоби визначення й маніпулювання даними, зробивши дані незалежними від прикладних програм, що їх використовують.

У вищезазначеному контексті навчальна дисципліна "Системи обробки розподілених баз даних" є однією з найважливіших. Вона відноситься до системних дисциплін і становить той фундамент, на якому базується проектування та безпосередньо створення інформаційних систем у бізнесі. Для розробника ІС істотним моментом при використанні концепції баз даних є те, що дані стають певним чином організовані, здобувають якусь упорядкованість і внутрішню структуру, а також те, що є деякий набір уніфікованих операцій обробки даних і декларативних засобів подання даних.

У результаті вивчення дисципліни у студентів формуються теоретичні знання та практичні навички застосування існуючих систем управління базами даних; вживання ефективних моделей забезпечення даних на основі вивчення предметної області, методів аналізу, пошуку та використання існуючих систем управління базами даних; знайомство з існуючими системами управління базами даних реляційного типу та нереляційного типу; забезпечення теоретичної та інженерної підготовки фахівців у галузі проектування та використання систем управління базами даних.

У методичних вказівках описано 6 лабораторних робіт, які входять до складу робочої програми із вказаної дисципліни. У ньому наведено теоретичні відомості про моделювання предметної області, основні принципи організації та проектування баз даних, структури та моделі даних, системи керування базами даних. Наявність теоретичного матеріалу обумовлено тим, що практично складно забезпечити фронтальний метод виконання лабораторних робіт, крім того, не завжди можливо синхронізувати в часі лекційні й лабораторні заняття. Наведено послідовність досліджень, які проводяться під час лабораторної роботи, вимоги щодо опрацювання результатів, а також контрольні запитання, які орієнтують студента на конкретний напрям досліджень.

ЛАБОРАТОРНА РОБОТА 1
ПРОЕКТУВАННЯ БАЗИ ДАНИХ

Мета: навчитися використовувати основні прийоми проектування БД.

Завдання

1. Ознайомитися із теоретичними відомостями щодо рівнів проектування баз даних.
2. Проаналізувати предметну область для проектування бази даних.
3. Розроити модель сутність-зв’язок для конкретної бази даних.

Теоретичні відомості

На концептуальному рівні здійснюється інтегрований опис предметної області, для якої розробляється БД, незалежно від ЇЇ сприйняття окремими користувачами та способів реалізації в комп'ютерній системі. Означимо основні поняття, що використовуються на концептуальному рівні.

Предметна область (ПО) — частина реального світу, для якої здійснюється концептуальне моделювання.

Концептуальна модель ПО — формальне зображення сукупності думок, які характеризують можливі стани ПО, а також переходи з одного стану в інший (включно з класифікацією наявних у ПО сутностей, чинних правил, законів, обмежень тощо).

Концептуальне моделювання ПО — процес побудови концептуальної моделі ПО, яка б відображувала ПО з урахуванням вимог, висунутих до цього процесу.

Концептуальна схема — фіксація концептуальної моделі ПО засобами конкретних мов моделей даних. У СКБД концептуальна модель подається у вигляді концептуальної схеми.

До розробки бази даних залучається великий колектив: експерти, системні аналітики, проектувальники, розробники, ті, хто займається впровадженням і супроводом. Усі вони повинні однозначно розуміти, чим є ПО, в чому зміст використаних понять, як вони взаємопов'язані між собою, які обмеження висуваються до моделі ПО тощо. Спільність понять має забезпечувати концептуальна модель.

1. Концептуальна схема відображує лише концептуально важливі аспекти ПО, виключаючи будь-які аспекти зовнішнього або внутрішнього відображення даних. Ця модель не повинна відображувати конкретні потреби окремих користувачів або застосувань. Вона має фіксувати, чим є ПО в цілому, а не з точки зору інтересів або потреб користувачів. Для отримання цілісного уявлення про ПО її модель має інтегрувати думки, погляди та інтереси окремих користувачів, але саме інтегрувати, а не виражати їхні конкретні побажання.

2. Визначення допустимих меж еволюції бази даних. У процесі експлуатації база даних може розвиватися, проте цей розвиток може відбуватися тільки в межах, допустимих для концептуальної схеми.

3. Відображення зовнішніх схем на внутрішню. Саме через концептуальну схему зовнішні дані відображуються на внутрішні, й навпаки. У такий спосіб створюється єдина основа для опису даних і підтримки цих відображень.

4. Забезпечення незалежності даних. Наявність відображень концептуальний-зовнішній і концептуальний-внутрішній дає змогу вирішувати проблему логічної та фізичної незалежності даних. Будь-які зміни в тій чи іншій зовнішній моделі не повинні спричиняти зміни в концептуальній або внутрішній моделях. У цьому випадку має змінитися тільки відповідне відображення «концептуальний-зовнішній». Аналогічно, будь-які зміни у внутрішній моделі не зачіпають концептуальну модель і моделі зовнішнього рівня, а тільки приводять до змін відображення «концептуальний-внутрішній».

5. Централізоване адміністрування. Саме через концептуальну схему здійснюється адміністрування баз даних.

6. Стійкість. Концептуальна схема не має підладжуватися до вимог тих чи інших користувачів (зовнішній рівень) або до вимог зберігання даних (внутрішній рівень). Будучи моделлю ПО, вона має змінюватися тільки тоді, коли входить у суперечність із нею.

Існує багато мов, які претендують на роль мов концептуального моделювання ПО. Найпопулярнішими і широковживаними є мови, що належать до класу так званих графічних мов, які оперують поняттями «сутність-атрибут-зв'язок» (Entity-Relationship language).

Хід роботи

1. Обрати предметну область, в якій виникає необхідність створення бази даних.
2. Ознайомитись з концептуальним рівнем моделювання предметної області.
3. Визначити основні сутності предметної області.
4. Визначити атрибути сутностей.
5. Структурувати інформацію про сутності та відповідні їм атрибути.
6. Зробити висновки щодо отриманих результатів, визначити основні етапи і принципи отримання інформації, охарактеризувати предметну область в термінах сутностей та атрибутів.

4. Дати назви сутностям та їхнім атрибутам (ідентифікатори іменників).

3. Встановити зв’язки, якими зв’язані сутності (дієслова).

5. Розробити діаграми ER-екземплярів для першого встановленого зв’язку.

6. Розробити відповідні ER-діаграми для всіх зв’язків (названих дієсловами).

7. Встановити клас приналежності одної сутності бінарного зв’язку (тобто, встановити значення «1», або «n»).

8. Встановити клас приналежності другої сутності бінарного зв’язку (тобто, встановити значення «1», або «n»).

Питання для самоконтролю

1. Що таке БД?
2. Що таке СУБД?
3. Що таке проектування БД?
4. Рівні проектування баз даних?
5. Модель сутність-зв’язок?
6. Поняття сутності?
7. Типи зв’язків між сутностями?

ЛАБОРАТОРНА РОБОТА 2
СТВОРЕННЯ ПРОСТИХ ЗАПИТІВ, ЗАПИТІВ ДЛЯ ЗАПИСУ, ТА РЕДАГУВАННЯ ДАНИХ

Мета: набути практичних навиків створення запитів маніпуляції даними на мові SQL.

Завдання

1 Онайомитися із основними характеристиками мови запитів дл БО SQL.

2. Дослідити основні оператори маніпулювання даними групи DML.

3. Сформувати прості запити на вибірку, додавання, оновлення, видалення даних у БД.

Теоретичні відомості

Мова SQL (з англ. Structured query language — мова структурованих запитів) – декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних і її модифікації, системи контролю за доступом до бази даних. Сама по собі SQL не є ні системою керування базами даних, ні окремим програмним продуктом. SQL може формувати інтерактивні запити або, будучи вбудованою в прикладні програми, виступати в якості інструкцій для керування даними. Стандарт SQL вміщує функції для визначення зміни, перевірки і захисту даних.

Ядро SQL формує командна мова, яка дозволяє здійснювати пошук, вставку, оновлення, і вилучення даних, використовуючи систему управління і адміністративні функції. SQL також включає CLI (Call Level Interface) для доступу і управління базами даних дистанційно.

Незалежність від конкретної СУБД. Незважаючи на наявність діалектів і відмінностей в синтаксисі, в більшості тексти SQL-запитів, що містять DDL і DML, можуть бути перенесені з однієї СУБД в іншу. **Наявність стандартів і набору тестів** для виявлення сумісності і відповідності конкретній реалізації SQL загальноприйнятому стандарту тільки сприяє «стабілізації» мови. **Декларативність.** За допомогою SQL програміст описує тільки те, які дані потрібно витягнути або модифікувати. Те, яким чином це зробити, вирішує СУБД безпосередньо при обробці SQL-запиту. Проте не варто думати, що це універсальний принцип. Програміст описує набір даних для вибірки або модифікації, проте йому потрібно розуміти, як СУБД розбиратиме текст його запиту. Особливо критичними такі моменти стають при роботі з великими базами даних і зі складними запитами — чим складніше сконструйований запит, тим більше він допускає варіантів написання, різних за швидкістю виконання, але тих самих за набором даних.

Розробники реляційної моделі даних вказують на те, що SQL не є істинно реляційною мовою, зважаючи на такі проблеми:

- рядки, що повторюються;
- невизначені значення (null);
- порядк стовпчиків зліва направо;
- стовпці без імені та імена стовпців, що дублюються;
- відсутність підтримки властивості «=»;
- використання покажчиків;
- висока надлишковість;

1 Синтаксис оператора SELECT

Оператор SELECT має таку структуру:

```
SELECT [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS] [HIGH_PRIORITY]
    [DISTINCT | DISTINCTROW | ALL]
  select_expression, ...
  [INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
  [FROM table_references
    [WHERE where_definition]
    [GROUP BY {unsigned_integer | col_name | formula} [ASC | DESC], ...]
    [HAVING where_definition]
    [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC], ...]
    [LIMIT [offset,] rows]
    [PROCEDURE procedure_name]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT застосовується для вилучення рядків, вибраних з однієї або декількох таблиць. Вираз select_expression задає стовпці, в яких необхідно проводити вибірку. Крім того, оператор SELECT можна використовувати для вибірки рядків, що визначені без посилання на якусь таблицю.

Наприклад:

```
mysql> SELECT 1 + 1;
    -> 2
```

При вказівці ключових слів слід точно дотримуватися порядоку, зазначеного вище.

Наприклад, оператор HAVING повинен розташовуватися після всіх виразів оператора GROUP BY і перед всіма виразами оператора ORDER BY .

Використовуючи ключове слово AS можна визначити назву. Псевдонім використовується в якості імені стовпця в даному виразі і може застосовуватися в ORDER BY або HAVING.

Наприклад:

```
mysql> SELECT CONCAT (last_name, ',', first_name) AS full_name
    FROM mytable ORDER BY full_name;
```

Посилання на стовпці можуть задаватися у вигляді col_name , tbl_name.col_name або db_name.tbl_name.col_name . У виразах tbl_name або db_name.tbl_name немає необхідності вказувати префікс для посилань на стовпці в команді SELECT , якщо ці посилання не можна витлумачити неоднозначно. See section 6.1.2 Імена баз даних, таблиць, стовпців, індекси псевдоніми , де наведені приклади неоднозначних випадків, для яких потрібні більш чіткі визначення посилань на стовпці.

У операторах ORDER BY і GROUP BY для виведення інформації, можна використовувати або імена стовпців, або їх псевдоніми, або їх позиції (місця розташування). Нумерація позицій стовпців починається з 1 :

```
mysql> SELECT college, region, seed FROM tournament
    ORDER BY region, seed;
mysql> SELECT college, region AS r, seed AS s FROM tournament
    ORDER BY r, s;
mysql> SELECT college, region, seed FROM tournament
    ORDER BY 2, 3;
```

Для того щоб сортування проводилася в зворотному порядку, в операторі ORDER BY до імені заданого стовпця, в якому відбувається сортування, слід додати ключове слово DESC (спадаючий). За замовчуванням прийняте сортування в зростаючому порядку, яке можна задати явно за допомогою ключового слова ASC .

У виразі WHERE можна використовувати будь-яку з функцій, яка підтримується в MySQL. Вираз HAVING може посилатися на будь-який стовпець або псевдонім, згаданий у вираженні select_expression. Оператор HAVING відпрацьовується останнім. Не використовуйте цей вираз для визначення того, що повинно бути визначено в WHERE . Наприклад, не можна задати наступний оператор:

Параметри (опції) DISTINCT , DISTINCTROW і ALL вказують, чи повинні повертатися записи, що дублюються.. За умовчанням встановлений параметр (ALL), тобто повертаються всі рядки. DISTINCT і DISTINCTROW є синонімами і вказують, що повторювані рядки в результуючому наборі даних мають бути видалені.

При вказівці параметра SQL_BUFFER_RESULT MySQL буде заносити результат в тимчасову таблицю. Таким чином MySQL отримує можливість раніше зняти блокування таблиці; це корисно також для випадків, коли для посилки результату клієнтові потрібен значний час.

2. Синтаксис оператора INSERT

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name [(col_name, ...)]
    VALUES (expression, ...), (...), ...
або INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name [(col_name, ...)]
    SELECT ...
або INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name
    SET col_name = expression, col_name = expression, ...
```

Оператор INSERT вставляє нові рядки в існуючу таблицю. Форма даної команди INSERT ... VALUES INSERT ... VALUES вставляє рядки відповідно до точно зазначених в команді значень. Форма INSERT ... SELECT INSERT ... SELECT вставляє рядки, обрані з іншої таблиці або таблиць. tbl_name задає назву таблиці, в яку повинні бути внесені рядки. Стовпці, для яких задані величини в команді, вказуються в списку імен стовпців або в частині SET.

Якщо не зазначений список стовпців для INSERT ... VALUES INSERT ... VALUES або INSERT ... SELECT INSERT ... SELECT , то величини для всіх стовпців повинні бути визначені в списку VALUES() або в результаті роботи SELECT. Якщо порядок стовпців у таблиці невідомий, для його отримання можна використовувати DESCRIBE tbl_name.

Duplicates показує кількість рядків, які не могли бути внесені, оскільки вони дублювали б значення деяких існуючих унікальних індексів. Покажчик Warnings показує кількість спроб внести величину в стовпець, який з якоїсь причини опинився проблематичним. Попередження виникають при виконанні будь-якого з таких умов:

3 Синтаксис оператора UPDATE

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
    SET col_name1 = expr1 [, col_name2 = expr2, ...]
    [WHERE where_definition]
    [LIMIT #]
```

Оператор UPDATE оновлює стовпці відповідно до їх нових значень в рядках існуючої таблиці. У виразі SET вказується, які саме стовпці слід модифікувати і які величини повинні бути в них встановлені. У виразі WHERE, якщо воно присутнє, задається, які рядки підлягають оновленню. В

інших випадках оновлюються всі рядки. Якщо задано вираз ORDER BY, то рядки будуть оновлюватися у зазначеному в ньому порядку.

Оператор UPDATE значення присвоює зліва направо

4 Синтаксис оператора DELETE

```
DELETE [LOW_PRIORITY | QUICK] FROM table_name
    [WHERE where_definition]
    [ORDER BY ...]
    [LIMIT rows]
```

або

```
DELETE [LOW_PRIORITY | QUICK] table_name [. *] [, Table_name [. *] ...]
    FROM table-references
    [WHERE where_definition]
```

або

```
DELETE [LOW_PRIORITY | QUICK]
    FROM table_name [. *], [Table_name [. *] ...]
    USING table-references
    [WHERE where_definition]
```

Оператор DELETE видаляє з таблиці table_name рядки, що задовольняють заданим в where_definition умовам, і повертає кількість видалених записів.

Якщо оператор DELETE запускається без умови WHERE, то видаляються всі рядки.

Хід роботи

1. Виконати простий запит на вибірку даних із БД.
2. Виконати запит на створення запису.
3. Виконати запит на редагування запису.
4. Виконати запит на видалення запису із БД.
5. Виконати запит на вибірку із сортуванням результатів в порядку спадання.

Питання для самоконтролю

1. Що таке SQL?
2. Оператори групи DML?
3. Оператори групи DDL?
4. Що таке процес маніпулювання даними?
5. Будова простого запиту до БД?
6. Оператор впорядкування даних по спаданню і зростанню?
7. Оператор групування даних?

ЛАБОРАТОРНА РОБОТА 3
СКЛАДНІ ЗАПИТИ НА ВИБІРКУ І ГРУПУВАННЯ. ПОБУДОВА ЗАПИТІВ НА ОСНОВІ КІЛЬКОХ ТАБЛИЦЬ

Мета: навчитися створювати складні запити до БД на вибірку та групування, набути практичних навичок використання різних типів об'єднань та підзапитів.

Завдання

1. Ознайомитися із теоретичними відомостями щодо створення під запитів та запитів на об'єднання таблиць.
2. Сформувати суладні запити із використанням операторів об'єднання та перевірки виконання умови запиту.

Теоретичні відомості

3.1 Підзапити. Види та застосування вкладених підзапитів.

Стандартом SQL передбачена можливість вкладати запити один в одний, що має велике практичне застосування. В результаті такого вкладення, одні запити можуть управляти іншими. При цьому вводиться поняття підзапиту. Підзапит – це запит, який міститься в іншому запиті SQL. Він являє собою повноцінний SELECT-вираз, результат виконання якого використовується в іншому запиті. Розміщуватись вони можуть майже в довільному місці SQL виразу, наприклад, замість одного з імен в списку SELECT, в операторі FROM, при вказанні умови в операторах WHERE або HAVING. Найчастіше зустрічається використання підзапитів при вказанні умови. Слід відмітити, що самі по собі підзапити не додають жодних функціональних можливостей, але іноді з ними запити стають більш читабельнішими, ніж при складній вибірці.

При використанні підзапитів часто використовують поняття зовнішнього та внутрішнього запиту. Спочатку виконується підзапит, тобто внутрішній запит, який розміщується, наприклад, в інструкції WHERE, а потім основний, тобто зовнішній запит, який може бути інструкцією SELECT, INSERT, DELETE або UPDATE. При цьому вкладений підзапит завжди обмежується дужками.

Щоб краще зрозуміти як користуватись підзапитами і як вони

```
SELECT Product.*
FROM Product, Producer
WHERE Product.IdProducer  = Producer.IdProducer AND Producer.Name='ЗАТ "Рівне-Хліб"';
```

Результат:

Запрос7 : запрос на выборку

Код	Name	IdCategory	Price	Quantity	IdProducer	IdMeasurement	IdMarkup
25	Хліб чорний	Хлібо-булочні	2,00	25	ЗАТ "Рівне-Хліб"	шт.	Базова
29	Батон свіжий	Хлібо-булочні	2,00	20	ЗАТ "Рівне-Хліб"	шт.	Базова
30	Ватрушка	Хлібо-булочні	5,00	20	ЗАТ "Рівне-Хліб"	шт.	Базова

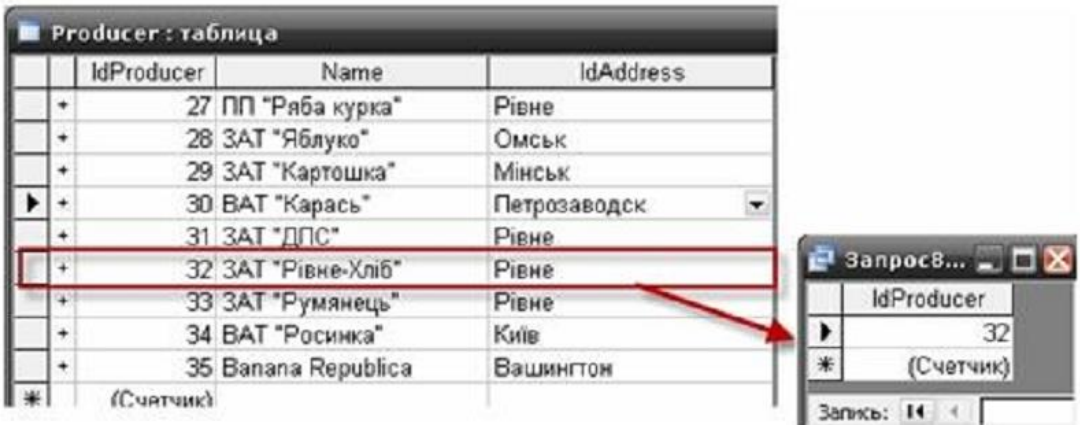
Запись: 1 из 3

працюють, розглянемо все на прикладі. Для початку, напишемо запит без використання підзапитів, який виводить всю інформацію про товари лише одного постачальника. За звичайних умов слід написати:

З використанням підзапитів запит набуде вигляду:

```
SELECT  *
FROM Product
WHERE IdProducer  =
                  (SELECT IdProducer
                  FROM Producer
                   WHERE Name='ЗАТ "Рівне-Хліб"');
```

Результат буде аналогічний. Що ж відбувається при використанні підзапиту? Спочатку цілісно виконується підзапит, який розміщений в інструкції WHERE. Результатом його виконання буде ідентифікатор – значення поля Name рівне ЗАТ “Рівне-Хліб”.



Отриманий результат повертається в основний запит і використовується при його виконанні. Тобто зовнішні ключі, які використовуються для зв’язку з таблицею Producer співставляються з первинним ключем, який є результатом виконання внутрішнього підзапиту. В результаті будуть обрані лише ті товари, які були виготовлені на ЗАТ «Рівне-Хліб».

Обмеження при використанні під запитів

1) Результатом підзапиту повинно бути лише одне значення, причому його тип даних повинен відповідати типу даних основного запиту.

Наприклад, потрібно вивести всю інформацію про товари двох виробників: ЗАТ «Рівне-Хліб» та ВАТ «Росинка».

```
SELECT  *
FROM Product
WHERE IdProducer  =
                 (SELECT IdProducer
                 FROM Producer
                  WHERE Name='ЗАТ "Рівне-Хліб"' OR    Name='ВАТ "Росинка"');
```

Даний запит призведе до помилки, оскільки результатом даного підзапиту є кілька значень, тобто два ідентифікатори, значення полів Name яких рівне ЗАТ “Рівне-Хліб” та ВАТ “Росинка”.

Виходом з цієї ситуації є використання оператора IN, який застосовується для перебору значень результату роботи внутрішнього підзапиту:

```
SELECT  *
FROM Product
WHERE IdProducer IN
                (SELECT IdProducer
                FROM Producer
                 WHERE Name='ЗАТ "Рівне-Хліб"' OR    Name='ВАТ "Росинка"');
```

2) Результатом підзапиту може бути NULL-запис. В такому випадку результат роботи підзапиту буде оцінений як UNKNOWN, що рівносильне FALSE. В результаті попередній запит є вірним навіть без використання оператора IN, але за умови, якщо товарів одного з вказаних виробників або обох в базі даних не існує. В такому випадку підзапит на виході поверне один або жодного записів.

В деяких випадках, для підстраховки, існує також можливість використання оператора DISTINCT для гарантованого отримання одного запису на виході підзапиту.

3) Згідно стандарту підзапити є непереміщувані, тобто наступний запит являється вірним:

SELECT * FROM Product WHERE IdProducer = (SELECT IdProducer FROM Producer WHERE Name='ЗАТ "Рівне-Хліб"' OR Name='ВАТ "Росинка"');

Але, якщо поміняти тіло підзапиту з порівнюваним значенням місцями, повинна згенеруватись помилка:

SELECT * FROM Product WHERE (SELECT IdProducer FROM Producer WHERE Name='ЗАТ "Рівне-Хліб"' OR Name='ВАТ "Росинка"') = IdProducer;

ПРИМІТКА! Дане правило не стосується середовища СУБД MS Access, оскільки вона розглядає і перший і другий варіант як однакові.

4) Вкладений запит не можна використовувати в інструкції ORDER BY.

5) При використанні підзапитів для перевірки результату не можна використовувати оператори BETWEEN, LIKE, IS NULL.

6) Якщо підзапит використовується з немодифікованим оператором порівняння, тобто звичаним знаком рівності (=), без використання ключових слів SOME, ANY або ALL, то він не може містити оператори групування GROUP BY та HAVING.

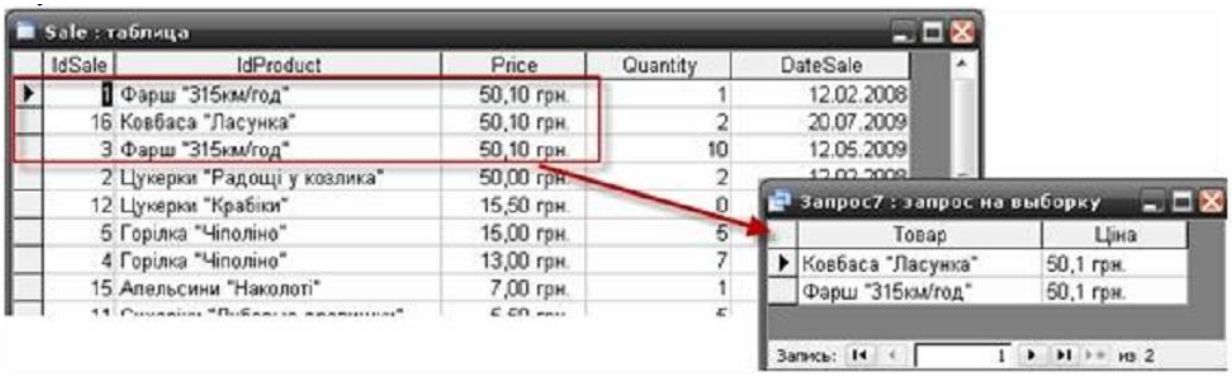
В підзапитах допускається використовувати функції агрегування, оскільки їх результатом є єдине значення.

Приклади

1. Запит на отримання інформації про найдорожчий товар, тобто товар з найбільшою ціною продажу:

SELECT DISTINCT Product.Name as [Товар], Sale.Price &' грн.' as [Ціна] FROM Product, Sale WHERE Product.IdProduct = Sale.IdProduct AND Sale.Price = (SELECT max(Sale.Price) FROM Sale);

Результат:



2.Запит, який виводить на екран імена та прізвища всіх постачальників, які поставляли товар в проміжку між 01/06/2009 та поточною датою:

SELECT Name as [Постачальник] FROM Supplier WHERE IdSupplier IN (SELECT IdSupplier FROM Delivery WHERE DateDelivery BETWEEN #01/06/2009# AND Date());

Результат:

3. Отримати інформацію про всіх постачальників, які поставляли товар більше, ніж 2 рази. Вибірку відсортувати по назві виробника:

SELECT * FROM Supplier s WHERE 2 > (SELECT COUNT(Delivery.IdSupplier) FROM Delivery WHERE s.IdSupplier = Delivery.IdSupplier) ORDER BY 2;

4. Визначити, які товари були вироблені в м. Київ:

SELECT Name as [Товар], Format(Price, '## ###.00 грн.') as [Ціна] FROM Product WHERE IdProduct IN (SELECT DISTINCT IdProduct FROM Delivery WHERE IdSupplier IN (SELECT s.IdSupplier FROM Supplier s, Address addr WHERE s.IdAddress=addr.IdAddress AND addr.Town = 'Київ'));

Як вже було сказано, підзапит можна використовувати у виразі FROM зовнішнього запиту. Це дозволяє створити тимчасову таблицю і додати її в запит. Наприклад:

SELECT IdProduct, Name, IdCategory FROM Product WHERE Price BETWEEN 20 AND 50;

Даний запит виведе на екран список товарів та ідентифікатори їх категорій, ціна яких знаходиться в межах 20-50 грн. Цей запит можна використати в рамках іншого, щоб отримати додаткову інформацію. Наприклад, використаємо його, щоб вивести список товарів, категорій «М'ясні» та «Ковбасні», ціни яких знаходяться у вищевказаному діапазоні.

SELECT pr.Name as [Товар] FROM (SELECT IdProduct, Name, IdCategory

FROM Product WHERE Price BETWEEN 20 AND 50) as pr, Category as c WHERE pr.IdCategory = c.IdCategory AND c.Name IN ('Мясні', 'Ковбасні');

Отже, використовуємо підзапит для створення таблиці, яка містить лише три поля: IdProduct, Name та IdCategory. Цій таблиці присвоюємо псевдонім pr. Після цього до створеної таблиці можна звернутись із запитом, як до будь-якої іншої таблиці.

3.2 Оператори EXIST, ANY, SOME, ALL

Для того, щоб обробити кілька записів, які поверне підзапит (багаторядковий підзапит), використовуємо оператор IN. Крім оператора IN, існує ще 4 логічних оператора: EXISTS, ALL, ANY та SOME.

Оператор EXISTS використовується, коли необхідно визначити наявність значень, які відповідають умові в підзапиті. Якщо дані на виході підзапиту існують, тоді даний оператор поверне значення true, інакше - false. При використанні оператора EXISTS ми насправді використовуємо в підзапиті дані зовнішнього запиту. Такий запит іноді називають зв'язаним або корельованим підзапитом.

Наприклад: вивести інформацію про всіх постачальників, які коли-небудь поставляли товари в магазин:

SELECT * FROM Supplier WHERE EXISTS (SELECT * FROM Delivery WHERE Supplier.IdSupplier = Delivery.IdSupplier) ORDER BY 3;

Проаналізуємо, що вийшло в результаті. В підзапиті ми шукаємо записи таблиці Delivery, в яких значення IdSupplier співпадає з значенням Supplier.IdSupplier. Кожен запис таблиці Supplier співставляється з результатом підзапиту і У ВИПАДКУ ІСНУВАННЯ (WHERE EXISTS) інформація про постачальника додається в результуючу таблицю.

Доречі, те, що повертається підзапитом в підзапит EXISTS або NOT EXISTS абсолютно немає значення. В зв'язку з цим, інколи повертають довільне константне значення. Все, що необхідно знати, - це сам факт наявності або відсутності записів, які відповідають критерію підзапита.

Оператори ALL, ANY та SOME використовуються для порівняння одного значення з множиною даних, які повертаються підзапитом. Їх можна комбінувати з усіма операторами порівняння і можуть включати інструкції GROUP BY та HAVING. Варто зазначити, що оператори ANY та SOME у загальному є ідентичними.

Значення

= ANY рівне довільному значенню, яке повертається під запитом. Прирівнюється до дії оператора IN і може бути ним замінений;

< ANY менше найбільшого значення, яке повертається підзапитом. Це можна трактувати як «менше будь-якого значення»;

> ANY більше найменшого значення, яке повертається підзапитом. Це можна трактувати як «більше будь-якого значення».

Для демонстрації роботи даного оператора перепишемо запит на отримання інформації про постачальників, які поставляли товари в магазин з використанням оператора ANY:

SELECT IdSupplier, Name FROM Supplier WHERE IdSupplier = ANY (

SELECT IdSupplier FROM Delivery) ORDER BY 2;

В даному випадку оператор ANY співставляє всі значення поля IdSupplier з таблиці Delivery та повертає результат true, якщо БУДЬ_ЯКЕ (ANY) значення співпаде з значенням поля IdSupplier.

Щодо оператора SOME (який-небудь), то він дасть аналогічний результат.

SELECT IdSupplier, Name FROM Supplier WHERE IdSupplier = SOME (SELECT IdSupplier FROM Delivery) ORDER BY 2;

Різниця між операторами ANY та SOME полягає лише в термінології та заключається в тому, щоб дозволити людям використовувати той термін, який найбільш підходить в даній ситуації.

Використання оператора ALL також нескладне. Даний оператор повертає значення true, якщо кожне значення, яке буде отримане в результаті роботи підзапиту, відповідає умові зовнішнього запиту.

Оператор значення

> ALL більше найбільшого значення, яке повертається підзапитом. Це можна трактувати як «більше всіх значень»;

< ALL меньше найменшого значення, яке повертається підзапитом. Це можна трактувати як «менше всіх значень».

В якості прикладу виконаємо запит, в якому поставимо ціллю довести, що товари категорії “Фрукти” мають найбільший попит

```
SELECT Product.Name FROM Product, Sale
WHERE Product.IdProduct = Sale.IdProduct
AND Sale.Quantity > ALL ( SELECT Sale.Quantity
FROM Sale, Product, Category
WHERE Sale.IdProduct=Product.IdProduct
AND Product.IdCategory=Category.IdCategory AND
Category.Name='Фрукти');
```

Підзапит поверне список значень кількості продаж (Sale.Quantity) товарів категорії “Фрукти”. Потім зовнішній запит шукає кількість продаж товарів, які були більшими, ніж дані, використовуючи для цього вираз Sale.Qauntity>ALL(кількість продаж). Якщо даний запит не поверне жодного запису – це буде підтвердженням того, що товари вказаної категорії дійсно мають найбільший попит, інакше виведеться перелік товарів, які продаються більше, ніж товар, вказаний в підзапиті.

3.3 Об’єднання запитів. Оператори UNION та UNION ALL.

3.3.1 Об’єднання – це зв’язування даних, що містяться в двох таблицях або запитах в один результуючий набір. Оскільки об’єднання запитів та таблиць відрізняється, розглянемо окремо один і другий спосіб об’єднання.

Об’єднання запитів здійснюється за допомогою оператора SQL UNION. З його допомогою можна об’єднати результати від 2 до 255 результатів запитів в один результуючий набір. В результаті такого об’єднання однакові записи по замовчуванню знищуються, але при аявності ключового слова ALL (тобто при використанні оператора UNION ALL) повертаються всі записи, в тому числі і однакові.

Синтаксис оператора UNION такий:

SELECT <список_полів>
[FROM <список_таблиць>] [WHERE <умова>]
[GROUP BY <список_полів_для_групування>] [HAVING <умова_на_групу>]
UNION [ALL]
SELECT <список_полів>
[FROM <список_таблиць>] [WHERE <умова>]
[GROUP BY <список_полів_для_групування>] [HAVING <умова_на_групу>];
[ORDER BY <умова_сортування>];

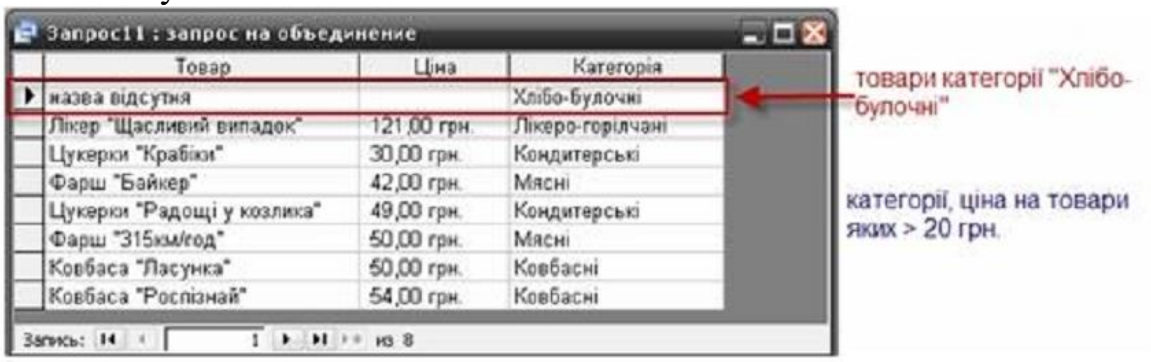
При використанні оператора UNION притримуються таких правил:

- кількість, послідовність і типи даних полів в обох списках SELECT повинні співпадати;
- оператори GROUP BY та HAVING використовуються лише в одному запиті;
- жодна з таблиць не може бути відсортована окремо; можна сортувати лише результуючий запит, тому оператор ORDER BY можна використовувати лише в кінці оператора UNION;
- імена полів результуючої вибірки визначаються списком полів першого оператора SELECT.

Для початку виберемо всі товари, ціна яких більше 20 грн., АБО код категорії товару рівний 1.

SELECT Name FROM Product WHERE Price > 20 UNION SELECT Name FROM Product WHERE IdCategory=1;

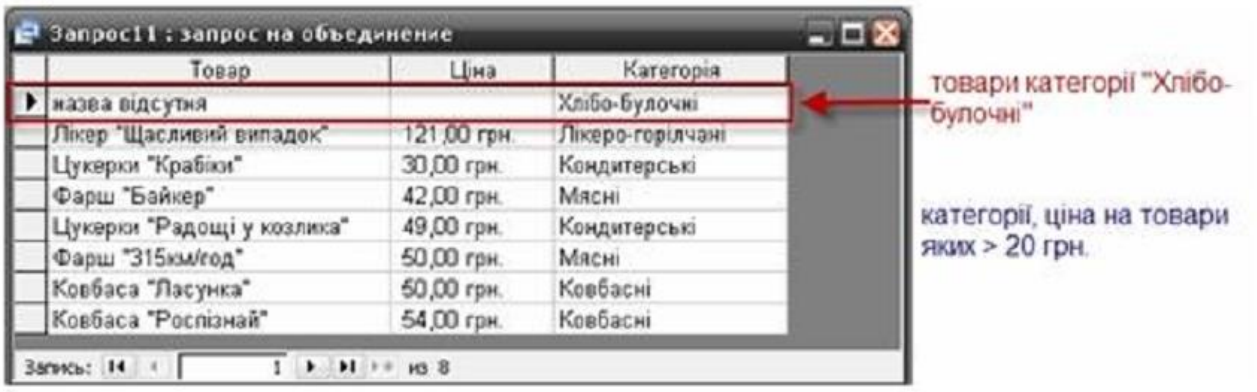
Результат:



А тепер виберемо всі товари, ціна яких більша 20 грн. АБО, які відносяться до категорії “Хлібо-булочні”.

SELECT pr.Name as [Товар], Format(pr.Price, '## ###.00 грн.') as [Ціна], c.Name as [Категорія] FROM Product pr, Category c
WHERE pr.IdCategory = c.IdCategory AND pr.Price > 20 UNION
SELECT 'назва відсутня', NULL, Name
FRO M Category WHERE Name ='Хлібо-булочні';

Результат:



3.3.2 Об'єднання таблиць.

Однією з найсильніших сторін SQL є можливість зв'язувати дані, що розміщуються в окремих таблицях. Це зв'язування здійснюється за рахунок об'єднання таблиць, які вказуються в списку FROM. Разом з цим задається спосіб або тип об'єднання.

Умова об'єднання, яка вказується після оператора ON являє собою вираз, аналогічний умові відбору, який використовувався в старому стандарті у виразі WHERE. Вона задає як будуть відноситись між собою записи в двох таблицях. Більшість операцій зв'язування виконуються на основі виразів еквівалентності, таких як ПолеА = ПолеВ. Однак умова об'єднання може бути і більша, при цьому всі вирази, які входять в умову, об'єднуються за допомогою логічних операторів AND або OR.

3.3.3 Внутрішні об'єднання. Оператор INNER JOIN

Являє собою звичайне об'єднання двох або більше таблиць. В результаті такого об'єднання отримується нова таблиця, записи якої задовольняють відповідним умовам. Внутрішні об'єднання повертають дані, якщо знаходять спільну інформацію в обох таблицях.

Для прикладу, відобразимо інформацію про товари та їх категорії.

```
SELECT pr.Name as [Товар], c.Name as [Категорія] FROM Product pr, Category c WHERE pr.IdCategory = c.IdCategory;
```

або

```
SELECT pr.Name as [Товар], c.Name as [Категорія] FROM Product pr INNER JOIN Category c ON pr.IdCategory = c.IdCategory;
```

Результат буде однаковий. Фактично, відмінність полягає лише в тому, що зв'язки між таблицями вказуються за допомогою оператора INNER JOIN, а зв'язки по ключових полях описуються після оператора ON. Всі інші оператори діють так само.

Умова повинна розміщуватись в інструкції WHERE. Отже, накладемо умову на виведення товарів лише категорії "Бакалія":

```
SELECT Product.Name as [Товар], Category.Name as [Категорія], Producer.Name as [Виробник] FROM Category INNER JOIN (Product INNER JOIN Producer ON Product.IdProducer=Producer.IdProducer) ON Product.IdCategory=Category.IdCategory WHERE Category.Name='Бакалія';
```

3.3.4 Зовнішнє об'єднання та його типи.

Зовнішні об'єднання здійснюються за допомогою оператора OUTER

JOIN та використовуються в випадку, коли потрібно, щоб запит повертав всі записи з однієї або більше таблиць, незалежно від того, чи мають вони відповідні записи в іншій таблиці. Фактично, вони дозволяють обмежити кількість результуючих полів однієї таблиці, не обмежуючи при цьому їх для іншої таблиці.

Типи зовнішніх об'єднань:

1 LEFT OUTER JOIN – лівостороннє об'єднання, при якому записи першої таблиці (зліва) включаються в результуючу таблицю повністю, а з другої таблиці (справа) в результат включаються лише ті, що мають пару в першій таблиці. В якості пари для записів першої таблиці, які не мають пари в іншій використовують пусті (NULL) поля;

2 RIGHT OUTER JOIN – правостороннє об'єднання – навпаки;

3. FULL OUTER JOIN – повне об'єднання – включає всі записи з обох таблиць.

Варто зазначити, що СУБД MS Access підтримує лише два перших типи зовнішніх об'єднань: ліве і праве.

Наприклад, вивести список постачальників, інформація про яких присутня в базі даних, не залежно від того, постачав він вже якийсь товар в магазин чи ні:

```
SELECT Supplier.Name, Product.Name FROM Supplier LEFT OUTER JOIN
(Delivery LEFT OUTER JOIN Product ON Product.IdProduct = Delivery.IdProduct)
ON Supplier.IdSupplier  = Delivery.IdSupplier;
```

Результат:

Запрос9 : запрос на выборку

Supplier.Name	Product.Name
ПП Вася	
ЗАТ "Хвилинка"	Цукерки "Крабіки"
ТзОВ "Хрещатик"	Картопля "Зелене Чудо"
ТзОВ "Хрещатик"	Цукерки "Крабіки"
ПП Кулаков В.В.	
ТзОВ "Вмерти, але доставити"	Фарш "315км/год"
ПП "Швидкий вітер"	Фарш "315км/год"

В результаті запиту, якщо постачальник не постачав ще товар в магазин, йому відповідає NULL-значення в полі назви товару,що постачається

І навпаки, якщо необхідно відобразити повний список товарів, з інформацією про їх постачальників, незалежно від того відома про них інформація чи ні:

```
SELECT Supplier.Name AS Постачальник, Product.Name AS Товар FROM
Supplier RIGHT OUTER JOIN  (Product RIGHT OUTER JOIN Delivery ON
Product.IdProduct = Delivery.IdProduct) ON Supplier.IdSupplier =
Delivery.IdSupplier;
```

Такий запит дозволить одразу виявити товари, інформацію про постачальників яких не заповнили.

3.4 Самооб’єднання таблиць.

Аналогічно об’єднанню кількох таблиць, таблицю можна об’єднати саму з собою. Це може знадобитись, коли потрібні зв’язки між рядками однієї і тієї ж таблиці.

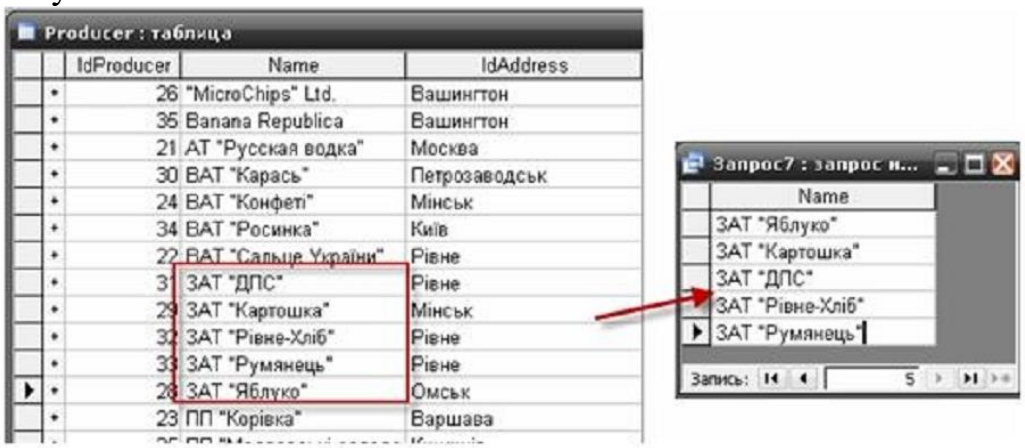
Наприклад, запит на виведення інформації про виробників, назви яких починаються з «ЗАТ», тобто форма відповідальності яких Закрите Акціонерне Товариство:

```
SELECT p1.Name FROM Producer p1, Producer p2 WHERE p1.IdProducer=p2.IdProducer AND p1.Name LIKE 'ЗАТ*';
```

або

```
SELECT p1.Name FROM Producer p1 INNER JOIN Producer p2 ON p1.IdProducer =p2.IdProducer WHERE p1.Name LIKE 'ЗАТ*';
```

Результат:



В такому запиті для таблиці Producer ми визначили два різних псевдоніми, тобто ми повідомляємо саму СУБД, що ми хочемо мати дві різні таблиці, які повинні містити однакові дані. Після цього ми їх об’єднуємо так само, як і будь-які інші таблиці. А далі отримуємо записи, що задовольняють умову.

Хід роботи

1. Побудувати складні запити до БД обраної предметної області із використанням всіх типів обєднань.
2. Побудувати складні запити до БД обраної предметної області із використанням операторів EXIST, ANY, SOME, ALL.
3. Сформувати запити на об’єднання із використанням операторів UNION та UNION ALL, обґрунтувати результат виконання запитів.
4. Сформувати складні запити із використанням підзапитів.

Питання для самоконтролю

1. Що таке під запит?
2. Що таке вкладений запит?
3. Оператори об’єднання таблиць?
4. Прератори об’єднання запитів?
5. Що є результатом застосування предиката EXIST у запиті?

6. Яка відмінність між результатом виконання запиту із використанням операторів ANY та SOME?
7. Для чого необхідне самооб’єднання таблиць?

ЛАБОРАТОРНА РОБОТА 4
СТВОРЕННЯ ЗБЕРЕЖЕНИХ ПРОЦЕДУР ТА ФУНКЦІЙ

Мета: навчитися створювати збережені процедури та функції

Завдання

1. Ознайомтися із понятями збережена процедура та функція.
2. Створити збережену процедуру.
3. Створти бережену функцію.

Теоретичні відомості

Часто при виконанні рутинних операцій потрібно здійснювати послідовність однакових запитів. Збережені процедури дозволяють об'єднати послідовність таких запитів і зберегти їх на сервері. Після цього клієнтам не доведеться посилати серверу послідовність запитів, достатньо надіслати один запит на виконання збереженої процедури.

Збережені процедури мають переваги.

- Повторне використання коду – після того, як процедура, створена, її можна викликати з будь-яких додатків і SQL-запитів, більше не потрібно щоразу програмувати одні й ті самі дії, завдяки чому зменшується ризик проникнення у додатки помилок і зменшується час розробки програми.
- Скорочення мережевого трафіку – при використанні збережених процедур замість того, щоб відсилати по мережі кожен запит і отримати на кожен із запитів відповідь, набагато економніше послати серверу запит на виконання збереженої процедури і відразу отримати відповідь.
- Безпека – збережені процедури використовуються для всіх стандартних банківських операцій. Процедура гарантує, що послідовність дій буде узгодженою і не призведе до порушення цілісності даних через те, що один оператор не буде виконаний. Крім того, для виконання збереженої процедури користувач повинен мати відповідний привілей. При цьому прав доступу до таблиць мати зовсім не обов'язково. Таким чином, адміністратор бази даних отримує більш широкі можливості у плані захисту даних і управління доступом користувачів до об'єктів бази даних.
- Простота доступу – основним імперативом розробки програмного забезпечення проголошено зменшення складності коду. Збережені процедури дозволяють інкапсулювати складний код і оформити його у вигляді простого виклику з осмисленим ім'ям. При створенні нового каталогу набагато простіше оперувати процедурою create_new_catalog (), ніж кількома операторами незрозумілого призначення. При реєстрації угоди, що

вимагає пошуку в таблицях catalogs products, users, і редагуванні таблиць orders і users набагато простіше оформити послідовність SQL-операторів для оформлення угоди у збережену процедуру ordering (). Навіть якщо послідовність операторів добре знайома програмісту, застосування процедур дозволяє перейти на більш високий рівень абстракції і оперувати не таблицями, а сутностями реального світу, такими як угода, каталог, продукція та ін.

- Виконання ділової логіки – збережені процедури можуть виконувати перевірку умов виконання замовлення, наприклад, підраховуючи число товарних позицій на складі, і відхиляти замовлення у випадку, якщо його недостатньо. Це дозволяє перенести код збереження цілісності бази даних з прикладної програми на сервер бази даних. Так, як прикладна програма може виконуватися на сотні машин мережі, а сервер, як правило, один, стійкість системи при переміщенні логіки на сервер різко зростає. Крім того, в цьому випадку легше змінити мову розробки зовнішнього застосування, тому що велика частина логіки оформлена у вигляді збережених процедур і не залежить від мови розробки програмного додатку.

Створення збереженої процедури

CREATE PROCEDURE sp_name ([parameter [,...]]) [characteristic ...] routine_body

CREATE FUNCTION sp_name ([parameter[,...]]) RETURNS type [characteristic ...] routine_body

Тут sp_name ім'я збереженої процедури, в дужках, наступних за ім'ям, передається необов'язковий список параметрів, перерахованих через кому. Кожен параметр Parameter дозволяє передати в процедуру або з неї вхідні дані або результат роботи функції і має наступний синтаксис:

[IN | OUT | INOUT] param_name type

Параметру param_name передує одне з ключових слів in, out, inout, які дозволяють задати напрямок передачі даних:

- in – дані передаються строго всередину збереженої процедури, але якщо параметру з даним модифікатором всередині функції присвоюється нове значення, після виходу з неї він не зберігається і параметр приймає значення, яке він мав до виклику процедури;
- out – дані передаються строго зі збереженої процедури, навіть якщо параметр має якесь початкове значення, всередині збереженої процедури це значення не береться до уваги. З іншого боку, якщо параметр змінюється всередині процедури, після виклику процедури параметр має значення, присвоєне йому всередині процедури;
- inout – значення цього параметра приймається до уваги всередині процедури, так і зберігає своє значення після виходу з неї.

Список аргументів, вкладених у круглі дужки, повинен бути присутнім завжди. Якщо аргументи відсутні, слід використовувати порожній список аргументів ().

Після імені параметра param_name вказується його тип. За замовчуванням, якщо жоден з модифікаторів не вказаний, вважається, що параметр оголошений з ключовим словом in.

Оператор CREATE FUNCTION дозволяє задати функцію користувача, тобто такий вид процедури, який повертає єдине значення. Тип цього значення дозволяє задати оператор RETURN.

Тіло процедури routine_body складається з складеного оператора begin ... end, всередині якого можуть розташовуватися інші оператори, у тому числі й інші складові оператори begin ... end. Оператор має наступний синтаксис:

[label:] BEGIN
statements
END [label]

Якщо оператор починається з необов'язкової мітки label, в якості якої може виступати будь унікальне ім'я, то він може закінчуватися виразом end label

Оператор begin ... end може виглядати так, як це представлено у прикладі.

Приклад:

```
begin
update tbll set coll = '1234.56';
update tbl2 set col2 = '1234.56';
END
```

В якості одного з операторів всередині складеного оператора BEGIN ... END може виступати інший складений оператор (приклад).

Приклад:

```
BEGIN
UPDATE tbll SET coll = '1234.56';
inner: BEGIN
UPDATE tbl2 SET col2 = '1234.56';
UPDATE tbl3 SET col3 = '1234.56';
END inner;
END
```

Як видно з прикладу, у внутрішньому складеному операторі BEGIN ... END перед оператором BEGIN використовується мітка inner, яка обов'язково згадується і у операторі END. Слід зазначити, що якщо процедура, що зберігається містить тільки один запит, то можна не використовувати складений оператор BEGIN ... END.

Після оператора END можна як поміщати, так і не поміщати крапку з комою тут і далі крапка з комою після END поміщатися не буде.

Основні труднощі, що виникають при роботі з збереженими процедурами, полягають в тому, що обов'язковий символ крапки з комою ";" наприкінці кожного запиту сприймається консольним клієнтом як сигнал до відправлення запиту на сервер. Для того щоб уникнути цього, при роботі з збереженими процедурами слід перевизначити роздільник запитів за допомогою параметра – delimiter = // консольного клієнта mysql. У цьому

випадку для позначення закінчення введення замість крапки з комою необхідно буде використовувати послідовність "//".

Приклад:

mysql – u root – delimiter=//

Крім того, можна змінювати роздільник у будь-який момент в консольному клієнті mysql. Для цього необхідно скористатися командою delimiter, після якої вказати роздільник.

Ім'я процедури не може перевищувати 64 символи та не залежить від регістра.

Приклад:

```
CREATE PROCEDURE ray_version ()
BEGIN
SELECT VERSION();
END //
```

Якщо тіло процедури містить єдиний запит, можна не використовувати складений оператор BEGIN ... END.

Приклад:

```
CREATE PROCEDURE my_version ()
SELECT VERSION(); //
```

Функція my_version () нічого не робить, крім того, що виводить версію сервера MySQL. Для того щоб викликати збережену процедуру, необхідно застосувати оператор **CALL**, після якого міститься ім'я процедури і її параметри в круглих дужках.

Приклад:

```
CALL my_version ();
```

При виклику процедури, на відміну від вбудованих функцій, між ім'ям функції і круглими дужками допускається пропуск.

При іменуванні функцій слід уникати назв, що збігаються з іменами внутрішніх функцій MySQL.

Приклад:

```
CREATE PROCEDURE pi ()
BEGIN
SELECT VERSION();
END //
```

У прикладі оголошується процедура, pi (), яка виводить поточну версію сервера MySQL. Слід звернути увагу, що між ім'ям процедури і круглими дужками поміщений пробіл – в іншому випадку СКБД MySQL інтерпретує послідовність pi () як виклик вбудованої функції, що повертає число пі. При виклику процедури використання пропусків в цьому випадку є обов'язковим. Перевантаження функцій в MySQL не допустима.

Щоб уникнути подібних накладок рекомендується уникати використання назв збережених процедур, що збігаються з іменами вбудованих функцій MySQL. Якщо все ж таки це необхідно, то і у визначенні функції, і при її виклику слід використовувати пробіл між ім'ям процедури і круглими дужками.

Параметри процедури

У попередніх прикладах збережені процедури не містили параметрів. Як згадувалося раніше, кожен параметр може бути оголошений одним з модифікаторів IN, OUT або INOUT. У наступному прикладі наводиться приклад функції, яка присвоює змінній користувача @ x нове значення.

Приклад:

```
CREATE PROCEDURE set_K (IN value INT)
BEGIN
SET @x = value;
END //
CALL set_x(123456)//
SELECT @x//
@x
123456
```

Як видно з прикладу, через параметр value функції передається числове значення 123456, яке вона присвоює змінній користувача @ x. Модифікатор in повідомляє СКБД MySQL, що за допомогою параметра value користувачі передають дані всередину функції.

На відміну від змінної @ x, яка є глобальною і доступна як всередині збереженої процедури set_x (), так і поза нею, параметри функції є локальними і доступні для використання тільки усередині функції.

Слід зазначити, що імена параметрів при оголошенні збереженої процедури і при виклику не обов'язково повинні збігатися. Усередині процедури всі локальні змінні використовуються без символу @ в той час як для глобальних змінних символ перед ім'ям обов'язковий4.

Якщо локальна або користувацька змінні не ініціюються за допомогою оператора SET чи ключового слова DEFAULT, вони отримують значення NULL.

Для того, щоб через параметр можна було і передати значення в середину процедури і отримати значення, яке потрапляє в параметр в результаті обчислень всередині процедури, його слід оголосити з модифікатором INOUT.

Приклад:

```
CREATE PROCEDURE set_y (INOUT value INT)
BEGIN
SET @x = value;
SET value = 7;
END //
SET @val = 123456//
CALL set_y(@val)//
SELECT @x, @val//
@x          @val
123456     7
```

Тепер через параметр value можна як передавати значення всередину процедури, так і витягувати значення, які отримує параметр всередині

процедури. Проте рекомендується використовувати тільки IN і OUT-параметри, не вдаючись до комбінованих INOUT-параметрів, оскільки це призводить до нечитабельного і непослідовного коду.

Робота з таблицями бази даних. Розглянемо процедури, які здійснюють запити до таблиць бази даних. Створимо функцію numcatalogs (), яка підраховує кількість записів у таблиці catalogs навчальної бази даних shop.

Приклад:

```
CREATE PROCEDURE numcatalogs (OUT total INT)
BEGIN
SELECT COUNT(*) INTO total FROM catalogs;
END //
```

Процедура numcatalogs () має один цілочисельний (int) параметр total, в який зберігається число записів в таблиці catalogs. Здійснюється це за допомогою оператора SELECT ... INTO ... FROM, який дозволяє зберігати результати безпосередньо у вихідному параметрі total функції numcatalogs.

Процедура успадковує базу даних за замовчуванням від викликаючого оператора, тому при зверненні до таблиць інших баз даних необхідно використовувати розширені імена. Використання операторaUSE в збережених процедурах заборонено.

Для використання будь-якої змінної у функції потрібно її оголошення при допомозі оператора DECLARE, який має синтаксис:

DECLARE var_name[,...] type [DEFAULT value].

Один оператор declare дозволяє оголосити відразу кілька змінних одного типу причому необов'язкове слово default дозволяє призначити значення.

Приклад:

```
CREATE PROCEDURE declare_var ()
BEGIN
DECLARE id, num INT(11) DEFAULT 0;
DECLARE name, hello, temp TINYTEXT;
END //
```

У прикладі оголошуються дві змінні типу int (11) - id і num, ініційовані значенням 0, і три текстові змінні name, hello і temp, оголошені без додаткової ініціалізації. Ініціювати локальні змінні можна і пізніше при допомозі оператора set.

Оператор DECLARE може з'являтися тільки всередині блоку BEGIN ... END, область видимості оголошеної змінної також обмежена цим блоком. Це означає, що в різних блоках BEGIN ... END можуть бути оголошені змінні з однаковими іменами, і діяти вони будуть тільки в рамках даного блоку, не перетинаючись з змінними інших блоків.

Слід зазначити, що не допускається повторне оголошення змінної в рамках одного блоку BEGIN ... END. Це призводить до виникнення помилки 1331: "Повторне оголошення змінної".

Збережені функції. Крім форми CREATE PROCEDURE, що створює процедуру, допускається використання CREATE FUNCTION, яка створює функцію. Функція на відміну від процедури може викликатися безпосередньо, без використання оператора CALL і повертати одне значення, яке підставляється на місце виклику функції.

Створимо найпростішу функцію say_hello (), яка буде приймати єдиний вхідний параметр з ім'ям name і повертати фразу "Hello, name!", Де замість підрядка name буде підставлено значення параметра name.

Приклад:

```
CREATE FUNCTION say_hello (name CHAR(20)) RETURNS CHAR(50)
BEGIN
RETURN CONCAT(' Hello, ',name,'!');
END //
SELECT say_hello('world'), say_hello('softtime')//
say_hello('world')          say_hello('softtime')
Hello, world!               Hello, softtime!
```

Після оголошення параметрів функції слідує оператор returns, який задає тип значення, що повертається функцією. Повернути значення з функції можна за допомогою оператора return, який може бути викликаний у будь-якій точці функції. Виклик оператора return означає, що функція повинна негайно завершити виконання і повернути значення, передане в якості аргументу оператора return.

При оголошенні параметрів функції використання ключових слів IN, INOUT і OUT неприпустимо Всі параметри, що передані функції, є вхідними.

Функція обов'язково повинна містити оператор returns, що встановлює тип значення функції, і хоча б один оператор return в тілі функції, який повертає це значення.

Приклад:

```
CREATE FUNCTION func_catalog (id INT)
RETURNS TINYTEXT
BEGIN
DECLARE catalog TINYTEXT;
SELECT name INTO catalog FROM catalogs
WHERE id_catalog = id LIMIT 1;
RETURN catalog;"
SELECT name INTO catalog FROM catalogs
WHERE id_catalog = id + 1 LIMIT 1;
RETURN catalog;
END //
SELECT func_catalog (1)//
func_catalog(1)
Процессори
```

Збережена функція func_catalog () приймає єдиний параметр id-первинний ключ таблиці catalogs. Прийнявши як параметр id значення 1, функція повертає результат ("процесори"), досягнувши першого оператора

return. При цьому другий оператор select і return не досягаються ніколи (інакше поверталося б значення "Оперативна пам'ять"). Це не означає, що двох операторів return у тілі функції не повинно зустрічатися.

Група характеристик збережених процедур.

Синтаксис збережених процедур допускає використання характеристик characteristic У визначенні CREATE PROCEDURE та CREATE FUNCTION:

LANGUAGE SQL
| [NOT] DETERMINISTIC
| SQL SECURITY {DEFINER | INVOKER}
|COMMENT 'string'

Ці ключові слова описують характеристики збережених процедур і функцій. Вони розміщуються після списку параметрів, але до початку тіла збереженої процедури. Характеристика LANGUAGE SQL не має особливого сенсу і повідомляє, що процедура написана на мові SQL .

Ключове слово DETERMINISTIC дозволяє повідомити оптимізатору, що процедура завжди повертає один і той же результат для одних і тих самих вхідних параметрів, в іншому випадку слід використовувати ключове слово NOT DETERMINISTIC.

Ключове слово SQL SECURITY може бути записано в двох формах: SQL SECURITY DEFINER і SQL SECURITY INVOKER. Якщо використовується форма SQL SECURITY DEFINER, то процедура викликається з привілеями користувача, що створив її, при використанні SQL SECURITY INVOKER процедура викликається з привілеями користувача, що викликає процедуру оператором CALL.

Якщо ключове слово SQL SECURITY не вказано, за замовчуванням встановлюється режим SQL SECURITY DEFINER. Процедура виконується з привілеями користувача, що її створив.

Ключове слово COMMENT дозволяє позначити процедуру коротким описом, який відобразить оператор SHOW CREATE PROCEDURE і SHOW CREATE FUNCTION. Ключове слово COMMENT є розширенням MySQL і може не підтримуватися іншими СУБД.

Оператори керування потоком даних

Збережені процедури це не просто зручні контейнери для групи запитів, вони дозволяють реалізувати досить складну логіку, використовуючи оператори розгалуження та цикли.

Оператор IF...THEN...ELSE. Оператор IF дозволяє реалізувати розгалуження програми за умовою і має такий синтаксис:

IF search_condition THEN statement_list
[ELSEIF search_condition THEN statement_list] ...
[ELSE statement_list] END IF

Оператор IF може бути із додатковим блоком else, після якого виконуються оператори, якщо умова виявилася хибною. Після ключового слова ELSE ставити крапку з комою не потрібно, оскільки цим самим єдиний оператор IF розбивається на частини – крапку з комою ставлять після ключового слова END IF.

Оператор CASE дозволяє здійснити множинний вибір і має дві форми. Синтаксис першої форми оператора виглядає таким чином:

CASE case_value

WHEN when_value THEN statement_list [WHEN when_value THEN statement_list] ... [ELSE statement_list] END CASE

Синтаксис другої форми:

CASE

WHEN search_condition THEN statement_list

[WHEN search_condition THEN statement_list] ...

[ELSE statement_list] END CASE

Синтаксис оператора case всередині збереженої процедури трохи відрізняється від синтаксису SQL-виразу CASE. Оператор CASE не може містити конструкціію ELSE NULL, і його виконання завершується за допомогою виразу END CASE, а не END.

Оператор WHILE

Оператор WHILE виконує цикл і має таий синтаксис:

[label:] while search_condition DO

statement_list

END while [label]

Цикл WHILE виконує оператори statement_list до тих пір, поки умова search_condition істинна. При кожній ітерації умова search_condition перевіряється і якщо при черговій перевірці воно буде хибним (0), то цикл завершить своє виконання. Це означає, що якщо умова search_condition хибна з самого початку, цикл не виконає жодної ітерації.

Для дострокового виходу з циклу призначений оператор LEAVE, який має синтаксис: LEAVE label. Оператор LEAVE припиняє виконання блоку, позначеного міткою label.

Ще одним оператором, виконуючим дострокове припинення циклу, є оператор ITERATE, який синтаксис: **ITERATE label.** На відміну від оператора LEAVE, оператор ITERATE не припиняє виконання циклу, він лише виконує дострокове припинення поточної ітерації.

Оператор **REPEAT**, так само як і оператор WHILE, реалізує цикл:

[label:] REPEAT

statement_list UNTIL search_condition

END REPEAT [label]

Відмінною особливістю такого циклу є той факт, що умова циклу search_condition перевіряється не на початку, як у циклі WHILE, а в кінці оператора (ключове слово UNTIL). Таким чином, цикл виконує, принаймі, одну ітерацію незалежно від умови. Слід зазначити, що цикл REPEAT виконується, поки умова search_concition помилкова.

Оператор REPEAT може бути використаний з необов'язковою міткою label, по якій можна здійснювати достроковий вихід з циклу за допомогою операторів LEAVE і ITERATE.

Оператор **LOOP** призначений для реалізації циклів і має таий синтаксис:

[label:] LOOP
statement_list END LOOP [label]

Цикл LOOP, на відміну від операторів WHILE і REPEAT, не має умов виходу, тому даний вид циклу повинен обов'язково мати у своєму складі оператор LEAVE.

Оператор **GOTO** дозволяє здійснювати безумовний перехід і має наступний синтаксис: **GOTO label.**

Оператор goto здійснює перехід до оператора, позначеного міткою label. Це може бути як оператор begin, так і будь-який з циклів: while, repeat та loop. Крім того, мітка може бути не прив'язана ні до одного з операторів процедури, а оголошена за допомогою оператора label, синтаксис якого **LABEL label.**

Метадані. Існує три способи переглянути дані, пов'язані із збереженими процедурами або функціями:

1. оператори show procedure status / show function status.
2. оператори show create procedure / show create function.
3. запит SELECT from mysq.proc.

Оператор SHOW PROCEDURE STATUS – переглянути список вже створених збережених процедур **SHOW PROCEDURE STATUS [LIKE 'pattern'].**

Оператор повертає список збережених процедур, який не включає збережених функцій. При використанні ключового слова LIKE можна вивести інформацію тільки про ті процедури, імена яких задовольняють шаблону pattern.

Приклад:

SHOW PROCEDURE STATUS LIKE 'bin%';
Db: shop
Name: binrand
Type: PROCEDURE
Definer: root@localhost
Modified: 2005-07-18 23:43:23
Created: 2005-07-18 23:43:23
Security_type: DEFINER
Comment:
Db: shop
Name: binstring
Type: PROCEDURE
Definer: root@localhost
Modified: 2005-07-18 14:09:55
Created: 2005-07-18 14:09:55
Security _type: DEFINER
Comment:

Для перегляду списку збережених функцій призначений оператор SHOW FUNCTION STATUS: **SHOW FUNCTION STATUS [LIKE 'pattern'].**

Оператор виводить список збережених функцій, що не включає до свого складу збережених процедур. При використанні ключового слова LIKE можна вивести інформацію тільки про ті функції, імена яких задовольняють шаблону pattern.

Ще одним оператором, який дозволяє отримати інформацію про збережені процедури, є оператор SHOW CREATE PROCEDURE, який має синтаксис: SHOW CREATE PROCEDURE procname. Оператор виводить синтаксис оператора CREATE PROCEDURE, за допомогою якого була створена процедура procname.

Оператор SHOW CREATE PROCEDURE виводить інформацію тільки для збережених процедур, для збережених функцій необхідно скористатися оператором **SHOW CREATE FUNCTION funcname.** Оператор виводить синтаксис оператора CREATE FUNCTION, за допомогою якого була створена процедура funcname.

Видалення збережених процедур. Для видалення збережених процедур використовується оператор DROP PROCEDURE, який має синтаксис:**DROP PROCEDURE [IF EXISTS] nameproc.**

Оператор DROP ROCEDURE дозволяє видалити збережену процедуру nameproc. Якщо процедури з таким ім'ям не існує, синтаксис оператора повертає помилку. Для видалення збережених функцій потрібно икористати опратор DROP FUNCTION.

Редагування збережених процедур. Для зміни характеристик процедури призначений оператор ALTER PROCEDURE. Редагування збереженої функції виконується за допомогою оператора ALTER FUNCTION. Оператори мають такі синтаксиси:

ALTER PROCEDURE sp_name [characteristic ...]

ALTER FUNCTION sp_name [characteristic ...]

Характеристика characteristic може приймати такі значення:

1. SQL SECURITY (DEFINER | INVOKER) - даний парамеир визначає режим виконання: процедура виконується або з правами створившого користувача, (DEFINER), або з правами користувача, що викликав її (INVOKER).
2. COMMENT 'string' - даний параметр дозволяє призначити коментар для процедури.

Зауваження:

- Для виконання операторів ALTER PROCEDURE та ALTER FUNCTION необхідно володіти привілеєм ALTER ROUTINE. Даний привілей автоматично передається користувачеві, що створив збережену процедуру (DEFINER). Користувач, що викликав процедуру, має права (INVOKER).

Хід роботи

1. Написати збережувану процедуру для обраної предметної області із використанням 3 таблиць.
2. Створити функцію для БД обраної предметної області.

3.Створити тригер для роботи із даними у БД.

4. Сформувати висновки щодо переваг збережуваних процедур, особливостей роботи тригера та результатів виконання функції.

Питання для самоконтролю

1. Що таке збережена процедура?
2. Види збережених процедур?
3. Що є результатом виконання функції?
4. Оператори для роботи із процедурами?
5. Що таке тіло процедури?
6. Оператор виклику процедури?

ЛАБОАТОРНА РОБОТА 5
НЕРЕЛЯЦІЙНІ БАЗИ ДАНИХ NOSQL. СУБД MONGODB

Мета: ознайомитися із перевагами та недоліками нереляційних баз даних, навчитися перетворювати моделі даних реляційних баз даних у документо-орієнтований вигляд.

Завдання

1. Ознайомитися із основними перевагами та недоліками не реляційних баз даних NOSQL.
2. Ознайомитися із документо-орієнтованою СУБД MongoDB.
3. Ознайомитися із форматами обміну даними у базі даних MongoDB.
4. Провести моделювання даних реляційної бази даних обраної предметної області на основі документо-орієнтованого підходу.

Теоретичні відомості

5.1 Нереляційні бази даних. Останнім часом для задач оперативної обробки погано структурованих даних все більшої популярності набувають NoSQL (від англ Not Only SQL –
Не тільки SQL) БД. Вони використовуються не тільки як елемент сховища даних, але й як саме сховище. NoSQL - це багато технологій, підходів, проектів спрямованих на реалізацію моделей баз даних, що мають суттєві відмінності від традиційних СУБД, що працюють з мовою SQL.

Концепція NoSQL не заперечує SQL, вона лише прагне вирішити проблеми і питання, з якими не досить добре справляється реляційна СУБД. Найчастіше дані в NoSQL представляються у вигляді хеш-таблиць, дерев, документів та інших структур. Концепція NoSQL надає високу доступність і стійкість даних до поділу, але при цьому в NoSQL не все гаразд із узгодженістю даних.

На сьогодні існує велика кількість NoSQL баз даних. NoSQL БД, як правило, є гнучкими рішеннями, що дозволяють масштабувати їх на безліч серверів з мінімальними витратами часу і коштів. Подібного роду рішення підходять, в тому числі, і для зберігання слабоструктурованої інформації. NoSQL БД забезпечують високу швидкість виконання, низькі витрати на масштабування, зберігання і обробку великих обсягів даних. Відмінною рисою NoSQL БД є відсутність необхідності використовувати реляційні моделі даних. Подібного роду бази даних в більшості випадків не мають графічного інтерфейсу або він мінімальний.

Взаємодія з БД відбувається через інтерфейс програмування додатків або через мережу. Для роботи з NoSQL БД потрібно написати додаток, що взаємодіє з інтерфейсом програмування додатків БД і виконує необхідні функції. По суті, NoSQL БД виступає в ролі сховища даних додатку, а всю логіку роботи з БД реалізовано в додатку і покладено на плечі програміста.

Сховище виконує основні функції по оперуванню даними: зберігання, вилучення, пошук. Наприклад, така операція як з'єднання (JOIN) в NoSQL реалізується самим програмістом.

Зважаючи на відсутність необхідності використовувати реляційні моделі даних в NoSQL, дані можуть зберігатися в зовсім не структурованому вигляді. Так, в будь-який документ можна додати довільне поле. Документ може містити вкладений документ, утворюючи ієрархію. Поля в БД визначаються на рівні документа. Це означає, що будь-який документ може мати унікальний набір полів, відмінний від інших документів.

На рисунку 5.1 представлено приклад документа, що включає, крім полів, вкладені документи. Так, як база даних не зберігає модель даних, зміна моделі даних вимагає змінити програмний код додатка, не вносячи жодних змін в БД.

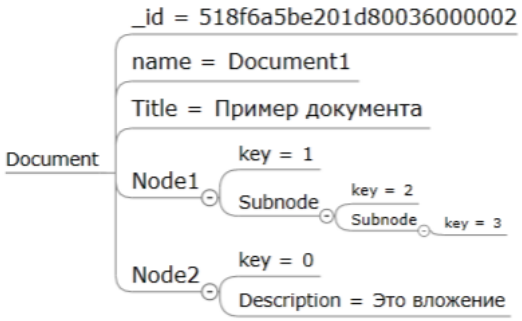


Рисунок 5.1 – Приклад документа в NoSQL БД

Характеристики, властиві для NoSQL баз даних:

- в NoSQL СУБД не використовується SQL (мається на увазі оператори DML);
- відсутність необхідності зберігати дані з певною структурою;
- висока продуктивність рішень, висока доступність;
- розподілене зберігання даних;
- здатність до горизонтального масштабування на вимогу для деякого набору операцій на багатьох серверах;

- ефективне використання розподілених індексів і пам'яті для запитів;
- відсутність транзакцій;
- прості протоколи доступу до збережених даних;
- відсутність підтримки транзакционной цілісності ACID (atomicity, consistency, isolation, durability - атомарність, узгодженість, ізольованість, довговічність).

Для того щоб зрозуміти схему подання даних в NoSQL БД, необхідно згадати, як дані представлені в реляційних БД (РБД). РБД є набором доменів (відносин, таблиць), імена яких збігаються з іменами схем відносин в схемі БД. Відносини складаються з кортежів, що відповідають одній схемі відносини (рядків). Кортеж (набір іменованих значень заданого типу) складається з атрибутів (стовпців). Між сутностями існують зв'язку.

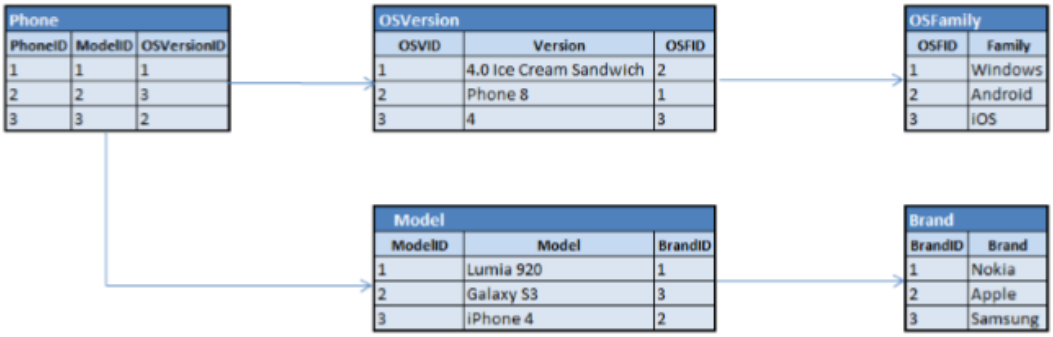
Якби призначенням бази даних було тільки збереження окремих, не пов'язаних між собою даних, то її структура могла б бути дуже проста. Проте однією із основних вимог до організації бази даних є забезпечення можливості відшукання одних сутностей за значеннями інших, для чого необхідно встановити між ними певні зв’язки. Для структурування даних існує процес нормалізації. Нормалізація – це процес структурування моделі даних, що забезпечує зв'язність і відсутність надмірності в даних.

У NoSQL БД дані зберігаються у вигляді пар ключ-значення. Будь-якому запису в БД відповідає ключ. БД складається з колекцій. Колекція є еквівалентом таблиці. База даних може мати нуль або більше колекцій. Колекція складається з документів, отже, еквівалент кортежу - документ.

Документи представлені, як об'єкти з полями і значеннями, що становлять пари ключ-значення. Документи складаються з полів, які подібні до атрибутів. Як зазначалося раніше, колекція не містить інформації про структуру даних, що вній розміщені, подібного роду інформацію містить кожен окремий документ.

База даних має індекси. Індекси в NoSQL БД майже еквівалентні індексам в РБД. NoSQL БД, як правило, замість даних повертають курсор, з яким ми можемо оперувати (підраховувати записи, пропускати їх), не завантажуючи самі дані. Курсор – одержуваний при виконанні запиту результуючий набір і пов'язаний з ним покажчик поточного запису.

Для прикладу, існує база даних, що містить інформацію про мобільні телефони в реляційному і нереляційному вигляді. Припустимо, що опис телефону має такі параметри: марка телефону, модель, сімейство ОС і її версія. У реляційному вигляді після проведення операції нормалізації, модель даних буде виглядати так, як представлено на рисунку 5.2.



Phone

PhoneID	ModelID	OSVersionID
1	1	1
2	2	3
3	3	2

OSVersion

OSVID	Version	OSFID
1	4.0 Ice Cream Sandwich	2
2	Phone 8	1
3	4	3

OSFamily

OSFID	Family
1	Windows
2	Android
3	iOS

Model

ModelID	Model	BrandID
1	Lumia 920	1
2	Galaxy S3	3
3	iPhone 4	2

Brand

BrandID	Brand
1	Nokia
2	Apple
3	Samsung

Рисунок 5.2 – Схема бази даних «Телефон»

В нереляційному вигляді база даних буде зберігати три документа, що відповідають кожному із збережених записів, причому кожен документ може містити властиві тільки йому поля. Модель БД в НЕреляційному вигляді представлено на рисунку 5.3.

Phone

ID	Attributes
1	Brand:Nokia Model:Lumia 920 OSFamily:Windows OSVersion:8
2	Brand:Apple Model:iPhone 4 OSFamily:iOS OSVersion:4
3	Brand:Samsung Model:Galaxy S3 OSFamily:Android OSVersion:4.0 Ice Cream Sandwich Display:4.8 HD Super AMOLED

Рисунок 5.3 – База даних «Телефон у не реляційному вигляді

5.2. Документо-орієнтована БД MongoDB. Однією з найпопулярніших NoSQL СУБД в даний час є MongoDB – документно-орієнтована база даних з відкритим вихідним кодом.

Особливості MongoDB:

- документно-орієнтоване сховище;
- повна підтримка індексів;
- реплікація даних;
- висока доступність даних;
- здатність до горизонтального масштабування;
- авто-шардінг (можливість рознесення даних по декількох серверах);
- підтримка запитів;
- підтримка Map / Reduce;
- підтримка GridFS.

Так, як MongoDB відноситься до NoSQL БД і є документно-орієнтованою, то кожен запис в ній є документом без жорстко заданої схеми. Кожен документ може містити вкладені документи. MongoDB володіє хорошою швидкістю роботи з даними (читання / запису), хорошою масштабністю, володіє системою розподілених обчислень з високим ступенем відмовостійкості.

Реплікація – це тиражування змін даних з головного сервера БД на одному або декількох залежних серверах. Шардінг – поділ даних на рівні ресурсів, розбиття даних за будь-якою ознакою. Концепція шардінга полягає в логічному поділі даних по різних ресурсах.

Для управління документами використовується нотація JSON, для їх зберігання – BSON. MongoDB не працює на транзакційній цілісності ACID. Це означає, що в MongoDB відсутнє поняття «транзакція». Наприклад, дані, що змінюються одним клієнтом, одночасно можуть читатися іншим. Атомарність присутня, але тільки на рівні цілого документа.

Використання MongoDB додатку відбувається через прикладний інтерфейс програмування. Компанія – розробник MongoDB крім самої БД також розробляє і підтримує драйвери, необхідні для розробки додатків з використанням MongoDB. Взаємодія з БД через API (інтерфейс програмування додатків) є основним способом взаємодії з базою даних. Крім використання драйверів з сервером баз даних Mongo можна взаємодіяти через графічні утиліти, що розробляються сторонніми фірмами. Різні графічні утиліти, призначені для взаємодії з базами даних mongo, позиціонуються як засоби адміністрування.

MongoDB не містить графічного інтерфейсу адміністрування. Більшість адміністративних функцій виконується через командний рядок. З MongoDB також можна взаємодіяти за допомогою http.

«Mongo.exe» – це інтерактивна оболонка JavaScript інтерфейсу для MongoDB, яка надає потужний інтерфейс для системних адміністраторів, а також дозволяє розробникам тестувати запити і операції безпосередньо з базою даних. «Mongo.exe» надає також повнофункціональне JavaScript середовище для використання з MongoDB. «Mongo.exe» є консольним додатком.

В консолі Mongo є кілька глобальних команд, наприклад «help», «use». Команда «help» дозволяє отримати коротку довідку щодо команд консоля Mongo. Команда «use» дозволяє вибрати базу даних. Синтаксис команди: «**use <ім'я бази даних>**».

Примітка: обрана база даних може і не існувати на момент введення команди. У такому випадку MongoDB автоматично створить БД при створенні першої колекції.

Не глобальні команди застосовуються до колекцій або бази даних. Команди, які використовуються стосовно поточної бази даних, починаються із вказівки об'єкта бази даних: «db», наприклад «db.help ()» або «db.stats ()». Команди, які використовуються стосовно конкретної колекції, крім об'єкта бази даних використовують також назву колекції: «db. <Ім'я колекції>», наприклад, «db.test.help ()» або «db.test.count ()».

Для отримання імені БД необхідно ввести команду «getName» стосовно поточного об'єкту БД: db.getName ().

Додавання документа в БД проводиться за допомогою команди: db.test.save ({name: "test parameter"}).

Синтаксис команд, які можна застосувати до колекції, такий: «<об'єкт бд>. <Колекція>. <Функція> (<список параметрів>)». Список параметрів задається в форматі JSON. Для додавання запису необхідно вказати ім'я запису і її значення через двокрапку. Зверніть увагу, що кожна команда «save» створює новий документ. У разі помилки в команді клієнт видасть відповідне попередження. У разі успішності операції повідомлень виведено не буде. Для отримання всіх документів в колекції необхідно ввести в консоль команду: db.test.find ().

MongoExplorer – це інструмент управління MongoDB. Особливості програми:

- MongoExplorer легкий у використанні;
- відображає всі колекції і документи бази даних;
- використовує зручне дерево для відображення документів;
- повністю підтримує drag'n'drop;
- in-place редагування документів.

MongoExplorer являє собою додаток, який надає основні можливості по взаємодії з БД Mongo: додавання, редагування та видалення колекцій / документів / полів. Інтерфейс програми не перевантажений та зручний для знайомства з БД. На відміну від командного рядка, взаємодія з БД відбувається з використанням графічного інтерфейсу.Екранну форму графічного інтерфейсу MongoExplorer наведено на рисунку 5.4.

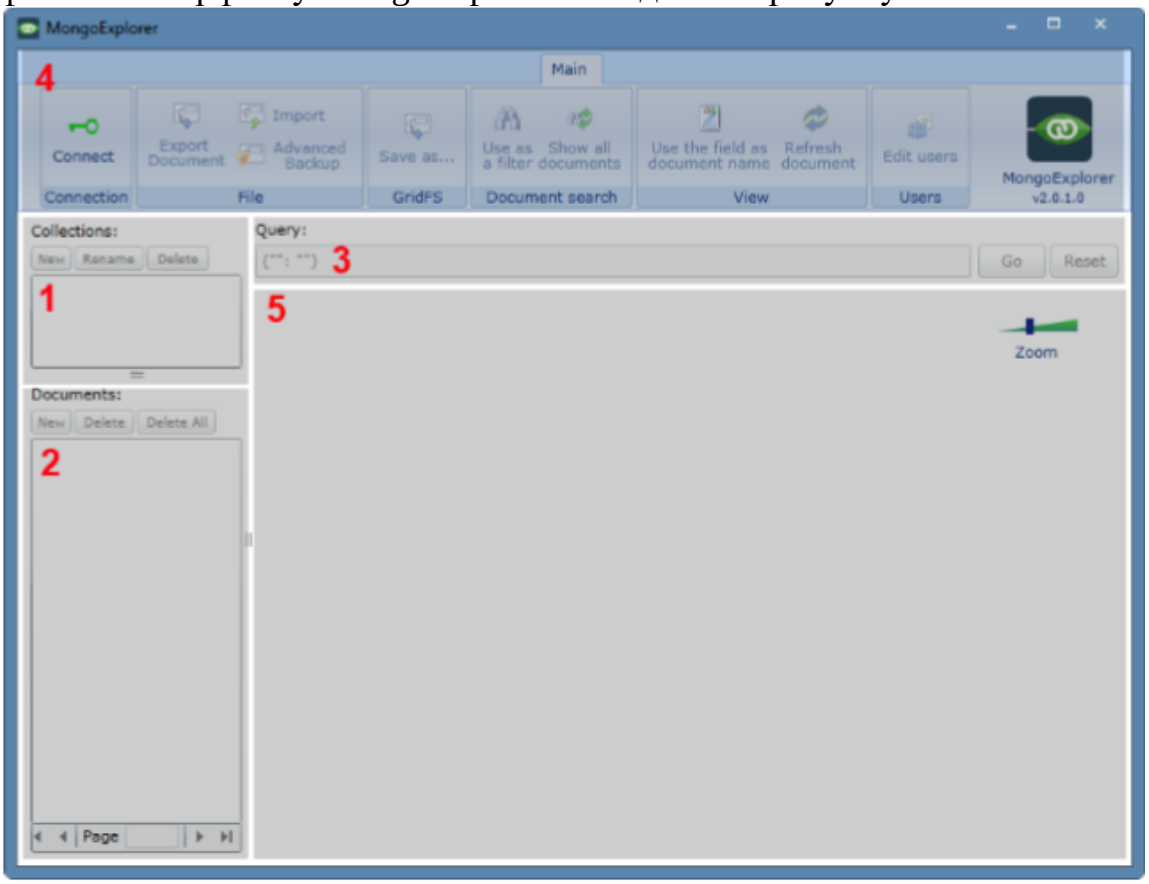


Рисунок 5.4 – Екранна форма додатку MongoExplorer

Панель роботи з колекціями містить перелік колекцій, присутніх в базі даних, а також надає можливості для створення, перейменування і видалення колекцій.

Панель роботи з документами також містить перелік усіх документів бази даних і кнопки управління документами: створення документів і їх видалення.

Панель вводу запитів являє собою поле, призначене для введення запиту. Результат виконання запиту відображається в панелях 2 і 5.

Панель команд містить команди: підключення до бази даних (і відключення від неї), експорт документа в JSON форматі, імпорт документа в JSON форматі, створення резервної копії БД, збереження файла на диск (при використанні GridFS), пошук по документу і використання фільтра, відображення всіх документів, використовувати значення поля в якості імені документа, оновити документи, робота зі списком користувачів.

Область роботи з документом представляє документ в графічному вигляді.

Для взаємодії з базою даних Mongo через Mongo Explorer необхідно клікнути на кнопку «Connect», що знаходиться в панелі команд. У вікні необхідно вказати ім'я бази даних, до якої потрібне підключення. Далі необхідно клікнути на кнопку «ОК». Якщо ніяких повідомлень про помилки не надходило, то підключення пройшло успішно.

Після підключення до бази даних в області колекцій відкрити список існуючих в базі даних колекцій. Для перегляду документів, що належать колекції, необхідно клікнути на імені колекції.

Для додавання поля в документ необхідно:

- вибрати документ в списку документів;
- активувати контекстне меню на імені документа в області документа;
- вибрати пункт меню «Insert child»;
- у вікні ввести ім'я і вибрати тип запису.
- для введення значення запису необхідно двічі клацнути мишею на значенні запису і змінити його.

Видалення запису відбувається за допомогою пункту «Remove» контекстного меню або із застосуванням кнопки «Delete».

5.3 Формати обміну та представлення даних у MongoDB.

5.3.1 Формат обміну даними JSON

JSON (від англ. «JavaScript Object Notation») – це текстовий формат обміну даними. Він заснований на підмножині мови програмування JavaScript. JSON незалежний від мови програмування і будується на двох структурах: набір пар ключ / значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативний масив. Ключем може бути тільки рядок, значенням – будь-яка форма; пронумерований набір значень.

Так як JSON використовується для обміну даними між різними мовами програмування, то варто будувати його на вказаних структурах. У JSON використовуються такі їх форми:

Об'єкт – це неврегульована кількість пар ім'я / значення, у фігурних дужках {}. Між ім'ям і значенням стоїть символ ':' (двокрапка), а пари ім'я / значення розділяються комами.

Масив (одновимірний) – це множина значень, що мають порядкові номери (індекси). Масив у квадратних дужках []. Значення відокремлюються комами. Значення може бути рядком в подвійних лапках, числом, значенням true або false, об'єктом, масивом, або значенням null. Ці структури можуть бути вкладені одна в одну.

Рядок – це впорядкована множина з нуля або більше символів юникоду, вкладену в подвійні лапки, з використанням escape-послідовностей, починаються з зворотної косої межі (backslash). Символи представляються простим рядком.

Ім'я – це рядок.. Число дуже схоже на C або Java-число, за винятком того, що використовується тільки десятковий формат. Пробіли можуть бути вставлені між будь-якими двома сімволамі.

Ілюстрація JSON-представлення деяких об'єктів:

```
{
    "hello": "world"
}
{
    "BSON": ["awesome", 5.05, 1986]
}
```

Для того, щоб занести інформацію про телефони в MongoDB, дані необхідно представити в JSON вигляді. Інформація про телефон «Nokia Lumia 920» в JSON вигляді буде виглядати таким чином:

```
{
    "Name": "NL920",
    "Brand": "Nokia",
    "Model": "Lumia 920",
    "OSFamily": "Windows",
    "OSVersion": "8"
}
```

Phone	
ID	Attributes
1	Brand:Nokia Model:Lumia 920 OSFamily:Windows OSVersion:8
2	Brand:Apple Model:iPhone 4 OSFamily:iOS OSVersion:4
3	Brand:Samsung Model:Galaxy S3 OSFamily:Android OSVersion:4.0 Ice Cream Sandwich Display:4.8 HD Super AMOLED

Рисунок 5.5 – База даних «Телефон» у не реляційному вигляді

Поле «Name» містить ім'я документа. Інформацію про всі телефони буде представлено як сукупність документів, що описують телефони:

```
{
    "Brand": "Nokia",
    "Model": "Lumia 920",
    "OSFamily": "Windows",
    "OSVersion": "8"
}
{
    "Brand": "Apple",
    "Model": "iPhone 4",
    "OSFamily": "iOS",
    "OSVersion": "4"
}
{
    "Brand": "Samsung",
    "Model": "Galaxy S3",
    "OSFamily": "Android",
    "OSVersion": "4.0 Ice Cream Sandwich",
    "Display": "4.8 HD Super AMOLED"
}
```

5.3.2 Формат обміну даними BSON. BSON (від англ. «Binary JavaScript Object Notation») – бінарна версія JSON. Також як і JSON BSON підтримує вбудовування документів і масиви в інші документи і масиви. BSON також містить розширення, які дозволяють оперувати з даними, які не є частиною специфікації JSON. BSON не має схеми даних, що дає йому деякі переваги в гнучкості і деякі недоліки в ефективності використання дискового простору. У BSON може зберігатися нуль або більше пар ключ / значення. Дані зберігаються як єдине ціле, як один документ. Всі операції по модифікації даних зачіпають зміну всього документу.

Представленя JSON документів в BSON буде виглядати таким чином:

```
{"hello": "world"}                    →    "\x16\x00\x00\x00\x02hello\x00
                                            \x06\x00\x00\x00world\x00\x00"

                                          "\x31\x00\x00\x00\x04BSON\x00\x26\x
00
                                           \x00\x00\x020\x00\x08\x00\x00
{"BSON": ["awesome", 5.05, 1986]}     →   \x00awesome\x00\x011\x00\x33\x33\x
33
                                           \x33\x33\x33
                                           \x14\x40\x102\x00\xc2\x07\x00\x00
                                           \x00\x00"
```

5.4 Моделювання даних. Нереляційні СУБД дозволяють проектувати модель предметної області у вигляді набору об'єктів. При цьому, інформація про одну сутність розкидана по різних таблицях РБД в нереляційній БД буде зібрана в одному об'єкті. Основною відмінністю MongoDB від РБД є відсутність аналога операції з'єднання (JOIN). Якщо виникає необхідність з'єднання у базі даних, то вони реалізуються в програмному коді програми.

Для того, щоб знайти дані, пов'язані з будь-яким документом, як правило, необхідно виконати другий запит. Для зв'язування документів можна зберігати їх разом з «_id» зв'язаних документів.

Як приклад проілюструємо збереження інформації про виробника телефонів у вигляді зв'язаного запису:

```
{
    _id: ObjectId ("1"),
    "Name": "Nokia",
    "BrandName": "Nokia",
    "BrandCountry": "Finland"
}
```

На документ «Nokia» будуть посилатися інші документи, яким необхідно в якості виробника вказати фірму, описану в ньому. Для створення зв'язаного документа необхідно знати поле «_id» документа «Nokia».

Запис із зазначенням фірми-виробника буде виглядати таким чином:

```
{
    _id: ObjectId ("2"),
    "Name": "L920",
    "Model": "Lumia 920",
    "OSFamily": "Windows",
    "OSVersion": "8",
    "Brand": ObjectId ("1")
}
```

Зверніть увагу, що значення поля «Brand» документа «L920» і поля «_id» документа «Nokia» збігаються. Поле «_id» може бути будь-яким унікальним значенням. Щоб знайти всі телефони, вироблені під брендом «Nokia», необхідно виконати запит із зазначенням значення його поля «_id»:

```
db.phones.find ({Brand: ObjectId ("1")})
```

Якщо необхідно вказати більше, ніж один зв'язаний документ, то можна використовувати масиви:

```
"Brand": [ObjectId ("1"), ObjectId ("3")]
```

Одним із способів позбутися зв'язків між документами є використання вкладених документів. Наприклад, наведений вище приклад можна переписати, використовуючи відомості про фірму Nokia у вигляді вкладеного документа:

```
{    "Name": "L920",
    "Model": "Lumia 920",
    "OSFamily": "Windows",
    "OSVersion": "8",
    "Brand":    {
                "BrandName": "Nokia",
                "BrandCountry": "Finland"
                }
}
```

Вкладені документи можна використовувати для моделювання зв’язків «один-до-багатьох». Для цього необхідно використовувати масив вкладених документів.

Хід роботи

1. Встановити MongoDB.
2. Підключитися до тестової бази даних.
3. Додайти дані в БД, що стосуються обраної предметної області, з використанням командного рядка.
4. Вибрати додані на попередньому кроці дані дані за допомогою командного рядка.
5. Додайте дані в базу даних з використанням Mongo Explorer.
6. Представити спроектовану БД в нереляційних вигляді записану в форматі JSON.

Питання для самоконтролю

1. Що означає термін NoSQL?
2. Які переваги надають NoSQL бази даних в порівнянні з реляційними базами даних?
3. Якими особливостями володіє MongoDB?
5. На які групи діляться додатки, що входять до складу MongoDB?
6. Чи створює MongoDB за замовчуванням будь-яку базу даних? Якщо створює, назвіть її ім'я.
7. Які існують способи взаємодії з БД Mongo?
8. Якими особливостями володіє Mongo Explorer?
9. Який формат команд командного рядка?
10. Особливості форматів JSON і BSON?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Диго С.М. Проектирование и использование баз данных. М.: ЕАОИ, 2008. 171 с.
2. Пасічник В.В., Берко А.Ю., Верес О.М. Системи баз даних та знань. Книга 1. Організація баз даних та знань: навч.посібник. Львів: “Магнолія 2006”, 2008. 456 с.
3. Берко А.Ю., Верес О.М**,** Пасічник В.В. Системи баз даних та знань. Книга 2. Системи управління базами даних та знань: навч. посібник. Львів: «Магнолія-2006». – 584 с.
4. Пасічник В.В., Резніченко В.А. Організація баз даних та знань. . К.: Видавнича група BHV, 2006. 384 с.
5. Бэнкер К. MongoDB в действии / пер. А. Слинкин. М.: ДМК Пресс, 2014. 394 с.
6. Обзор основных sql запросов: веб-сайт − URL: https://itvdn.com/ru/blog/article/m-sql (дата звернення: 25.01.2020).
7. Основы SQL на примере задачи : веб-сайт − URL: https://habr.com/ru/post/123636 (дата звернення: 24.01.2020).
8. Официальный портал MongoDB: веб-сайт − URL: http://www.mongodb.org (дата звернення: 5.02.2020).
9. Что такое NoSQL и MongoDb?: веб-сайт − URL: http://www.andrey- vasiliev.com/no-sql/chto-takoe-nosql-i-mongodb (дата звернення: 6.02.2020).
10. Install MongoDB on Windows?: веб-сайт − URL: http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows (дата звернення: 30.01.2020).
11. JSON: веб-сайт – URL: http://json.org (дата звернення: 06.02.2020)
12. BSON – Binary JSON: веб-сайт – URL: http://bsonspec.org (дата звернення: 06.02.2020).

Підписано до друку 09.02.2021 р.
Формат 60х84/16. Папір офсетний.
Друк на дублікаторі.
Умов.- друк арк. 1.4 Обл.-вид. арк 1.5.
Тираж 25 прим.

Віддруковано ФОП Шпак В.Д.
Свідоцтво про державну реєстрацію
серія В02 №924434 від 11.12.2006 р.
Свідоцтво платника податку: Е № 897220