РОБОЧА ВЕРСІЯ

В. В. Жировецький, В. Я. Костів, С. В. Літинський, Р. Б. Малець

ЛАБОРАТОРНИЙ ПРАКТИКУМ ДО КУРСУ

"Бази даних та інформаційні системи"

для студентів факультету прикладної математики та інформатики

ЛНУ імені Івана Франка

Львів 2023

ЗМІСТ

ВСТУП

Це видання присвячено практичним аспектам вивчення та опанування навичками та вміннями з курсів «Бази даних та інформаційні системи», «Бази даних», «Бази даних та основи SQL», розроблених авторами. Перелічені курси викладалися та викладаються на факультеті прикладної математики та інформатики для студентів спеціальностей 122 – «Комп'ютерні науки», 014 – «Середня освіта (Інформатика)», 124 – «Системний аналіз», 125 – «Кібербезпека» а також 112 - «Статистика» механіко-математичного факультету Львівського національного університету імені Івана Франка. Пропонований лабораторний практикум також можна рекомендувати і викладачам, і студентам, які прагнуть розвивати практичні навички з проєктування та використання даних і спеціалізуються в галузі інформаційних систем та технологій.

Він складається з 16 лабораторних, кожна з яких охоплює певну тему та має чітко окреслену мету.

Навчально-методичні матеріали поряд з покроковим описом ходу виконання лабораторних робіт, детально висвітлюють необхідні теоретичні відомості та основні аспекти роботи в середовищі PostgreSQL. Кожна лабораторна робота містить перелік корисних посилань на електронні ресурси, що сприятиме результативності виконання кожного завдання та опануванню навичками роботи з базами даних. Посібник проілюстрований прикладами оформлення виконаних завдань, написаний доступною, живою мовою, та має на меті сформувати базові навички і вдосконалити професійну компетенцію. Бази даних є обов'язковою складовою значної кількості сучасних інформаційних систем. Частка систем, які використовують реляційні бази даних у 2019 році оцінювалася в 60.48% [2], що свідчить про відчутні переваги реляційної моделі даних при використанні її в інформаційних системах.

Додатковою можливістю, яка спрощує роботу з реляційними базами даних, є наявність стандартизованої мови запитів SQL, що забезпечує обробку даних в однаковий спосіб в усіх реляційних системах керування базами даних. Відмінності діалектів SQL переважно стосуються додаткових можливостей кожної СКБД, у той час як типові класи запитів до даних реалізуються майже без відмінностей у різних СКБД.

У цьому лабораторному практикумі запропоновано підтримку для практичного вивчення реляційних баз даних та їх можливостей. Послідовність лабораторних робіт передбачає поступову розробку реляційної бази даних від проєктування її концептуальної моделі аж до реалізації функціональності за допомогою технології збережених процедур та дослідження принципів функціонування ACID-транзакцій в СКБД PostgreSQL. Лабораторні роботи

дозволяють на практиці освоїти основні етапи побудови реляційної бази даних, ознайомитися із засобами підтримки цілісності даних, інструментами для покращення продуктивності системи та зручності обробки даних і розширення функціональності бази даних для подальшого використання у складі інформаційної системи. Передбачено, що протягом усієї роботи над навчальною базою даних студент активно використовуватиме мову запитів SQL, що дозволить глибше засвоїти основні елементи цієї мови запитів.

Окрім засвоєння реляційного підходу до даних, останню частину практикуму присвячено нереляційному способу представлення даних, а саме: частково-структурованому формату даних на прикладі розширюваної мови розмітки XML. Лабораторні роботи з XML передбачають ознайомлення з нереляційним представленням даних, як частково-структурованих документів (що є характерним для NOSQL баз даних, які використовують документну модель), а також засобами обробки даних у такому форматі. Роботи з DTD та Schema дають можливість ознайомитися зі способами задання правил перевірки структури і вмісту XML документів. XPath, XQuery та XSLT дозволять засвоїти основні інструменти обробки даних в XML та перетворення частково-структурованих даних у нові типи відображення.

ЛАБОРАТОРНА РОБОТА № 1
«Побудова концептуальної моделі бази даних у вигляді ER-діаграми»

Тема: *«Побудова концептуальної моделі бази даних у вигляді ER-діаграми».*

Мета роботи: Ознайомлення з процесом проєктування бази даних на прикладі побудови концептуальної моделі бази даних у вигляді ER-діаграми («сутність–зв'язок»).

Теоретичний матеріал

Перелік розділів та понять, з якими необхідно ознайомитись для виконання завдання лабораторної роботи:

1. Поняття бази даних.
2. Поняття системи керування базою даних (СКБД).
3. Поняття моделі даних (ієрархічна, мережна, реляційна).
4. Рівні архітектури системи бази даних (зовнішній, концептуальний, внутрішній).
5. Модель даних «сутність–зв'язок» (ER-діаграма).
   - 5.1. Множина сутностей.
   - 5.2. Набори атрибутів. Ключові атрибути (потенційні ключі).
   - 5.3. Зв'язки.
     - 5.3.1. Види бінарних зв'язків.
     - 5.3.2. Багатосторонні зв'язки. Їх перетворення в бінарні.
   - 5.4. Підкласи в ER-моделі.

Літературу та перелік електронних ресурсів, необхідних для ознайомлення з теоретичним матеріалом теми, наведено у списку рекомендованих джерел до теми.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Відповідно до свого завдання намалювати ER-діаграму в нотації П. Чена з обов'язковим позначенням ключових атрибутів.
3. Оформити звіт про виконання лабораторної роботи, який має містити:
   - титульну сторінку (див. додаток А);
   - тему, мету та завдання лабораторної роботи;
   - розроблену ER-діаграму або фото діаграми, якщо малювали на папері.

4. Завантажити звіт у канал «БД та ІС. Лабораторна робота» своєї команди в Microsoft Teams.

Контрольні питання

1. Дайте визначення поняття бази даних.
2. Які функції системи керування базою даних (СКБД)?
3. Дайте визначення моделі даних.
4. Що таке ієрархічна модель даних?
5. Що таке мережева модель даних?
6. Які три аспекти виділяють у реляційній моделі?
7. Які існують рівні архітектури системи бази даних?
8. Назвіть три основні типи елементів моделі даних «сутність–зв'язок» (ER-діаграма).
9. Поясніть поняття ключовий атрибут.
10. Які є види бінарних зв'язків?
11. Поясніть застосування підкласів в ER-моделі.

Список рекомендованих джерел до теми

1. Garcia-Molina H. Database Systems: 2nd Edition / H. Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. – Pearson Education Inc. – 2009.– 1203 p.
2. Date C. J. An Introduction to Database Systems (8th Edition) / C. J. Date. – 8th ed. – Pearson Education Inc. – 2004. – 1040 p.
3. Пасічник В. В. Організація баз даних та знань / В. В. Пасічник, В. А. Резніченко.– Київ: Видавнича група BHV. – 2006. — 384 с.

Варіанти завдань

1. Розробити базу даних для сайту обслуговування рекламних агенцій. Рекламна агенція має назву, юридичну адресу та список контактних осіб про кожну з яких відомо ім'я, прізвище, електронну адресу та телефон. Агенції проводять рекламні кампанії, що тривають у певний період, мають назву, слоган і коротку характеристику. В рамках кампанії випускається реклама з датами показу, частотою показу, пов'язаними зображенням/відео/аудіо тощо. Кожна кампанія проводиться для певного клієнта, стосовно якого виставляються рахунки за проведення кампаній і фіксуються платежі. Користувач сайту матиме змогу обробляти наявні агенції, клієнтів, рекламні кампанії, а також виконувати пошук

для визначення оплачених рахунків, клієнтів із заборгованостями, загальної вартості рекламних повідомлень тощо.

2. Розробити базу даних для сайту аеропорту. Він обслуговує рейси і вхідні/вихідні, і транзитні. Кожен рейс має визначені дні тижня, період навігації, авіакомпанію та аеропорти між якими він відбувається. Кожен аеропорт перебуває в певній країні, має офіційну назву, унікальний код IATA. Авіакомпанії мають назву, офіційну адресу, список контактних даних, код IATA та флот літаків певних типів, місткості, з серійними номерами, які можуть бути у власності авіакомпанії, або орендованими. Користувач сайту матиме можливість обробляти інформацію про аеропорти, авіакомпанії та рейси, а також використовувати пошук рейсів у певні дні з/до певних аеропортів за критеріями ціни перельоту, авіакомпанії, типу літаків.

3. Розробити базу даних для сайту продажу автомобілів. Сайт підтримує інформацію про клієнтів. Для кожного з них зберігається ім’я, контактні дані(адреса, телефони, email), вподобання (якими автомобілями клієнт цікавився раніше). Крім того, база даних сайту містить інформацію про автомобілі, які продаються (марка, модель, колір, особливості комплектації), виставлені та оплачені рахунки за продані автомобілі. Користувач сайту захоче мати можливість обробляти інформацію про клієнтів, автомобілі та оплати, а також використовувати пошук по клієнтах по критеріях вподобань при наявних автомобілях, оплати рахунків тощо

4. Розробити базу даних для системи автоматизації банку. Система підтримує інформацію про філії та відділення банку, клієнтів банку – як фізичних так і юридичних осіб, клієнтські рахунки та операції. Філія банку має назву, контактну інформацію та множину коррахунків (рахунки банку асоційовані з філією). Контактна інформація містить адресу та множину телефонів. Відділення банку відкривається філією за певною адресою і володіє інформацією про клієнтів, яких відділення обслуговує. Кожен клієнт має податковий код, назву (або ім’я), контактну інформацію та перелік осіб з правом підпису у випадку клієнта юридичної особи. Операції здійснюються між рахунками клієнтів і впливають на їхні залишки. Користувач системи матиме можливість обробляти інформацію про філії, відділення, клієнтів та їхні операції, а також використовувати пошук серед клієнтів щодо операцій, відділень, де вони обслуговуються тощо.

5. Розробити базу даних для веб представлення системи управління документами. Система підтримує інформацію про документи, які проходять через неї. Крім власне документа важливими є інформація про його створення, історія змін,

користувачі, які вносили зміни, та інформація про офлайн представлення документа (паперовий, CD, USB-флеш-накопичувач). Користувачі можуть створювати і змінювати документи і асоціювати їх з офлайн представленням, а також шукати їх за користувачами, які змінювали їх, станом чи представленням.

6. Розробити базу даних для екзаменаційного сайту. Він повинен відображати інформацію про предмети, іспити, студентів та викладачів. Сайт містить інформацію про сесії, які відбувалися у ВУЗі. Сесія для кожного семестру містить інформацію про період, іспити, які відбувалися протягом сесії, викладачів, які приймали іспити, результати іспитів, враховуючи отримані талони і перездачі. Користувач сайту може створювати сесії, іспити і вносити інформацію про отримані результати, а також шукати результати складених іспитів, студентів, які склали на відмінно або були відраховані тощо.

7. Розробити базу даних для сайту планування подій. Сайт відображає інформацію про події у місті, які відбуваються у певні періоди, за певними адресами. Події проводяться організаторами, які мають назву та контактну інформацію. Для їх проведення можуть залучатися спонсори, які мають назву та контактні дані, а також певний рівень спонсорства (головний, преміальний, звичайний тощо). Для подій важливою є інформація про кількість людей, котрі відвідали подію або придбали квитки на неї. Користувач має можливість створювати нові події, додавати їх організаторів, спонсорів до подій, а також шукати події за адресами, організаторами тощо.

8. Розробити базу даних для сайту розкладів ВНЗу. Сайт повинен відображати інформацію про розклад занять на факультетах ВНЗу. Зокрема, має бути доступною інформація про предмети, викладачів, аудиторії, типи пар (лекція, практична, семінар, консультація тощо). Пари повинні описуватися часом, протягом якого вони відбуваються. Користувач може створювати нові пари з предметів в аудиторіях для певних академічних груп, а також шукати зведений розклад для викладача, академічної групи, дня тижня тощо.

9. Розробити базу даних для сайту зберігання зображень. Він повинен забезпечувати розподіл доступу користувачам на основі ролей. Користувачі в ролі адміністратора повинні мати змогу виконувати операції із будь-якими зображеннями. Користувачі в ролі автора мають можливість завантажити зображення, додати до нього опис та інші важливі деталі, створити папки для зображень, поділитися ними з іншими користувачами та надати доступ до перегляду зображень іншим. Користувачі в ролі глядача можуть переглядати зображення дозволені для перегляду.

10. Розробити базу даних для системи управління кадрами організації. Система має забезпечувати збір та обробку інформації про структуру організації (відділи, департаменти, управління), працівників, їх переміщення на посадах та між відділами, інформацію про відпустки, лікарняні, відгули, понаднормову роботу тощо. Користувач такої системи може додавати нових працівників, переміщувати їх між підрозділами і/або посадами, а також шукати деталі про них за їхніми посадами, прізвищами та ін.

11. Розробити базу даних для сайту відстежування помилок у програмному забезпеченні. Сайт повинен давати можливість вводити інформацію про програмне забезпечення, яке тестується, знайдені помилки, зміну їхнього статусу, детальний опис відтворення та історію виправлення. Користувач може призначати помилку іншому користувачеві, змінювати її статус, створювати нову, а також шукати помилки за відповідальним користувачем, автором, програмним модулем тощо.

12. Розробити базу даних для сайту обслуговування мережі ресторанів. Сайт дає можливість отримати інформацію про наявні ресторани, їх адресу, назву, місткість та ін. Для кожного ресторану визначено множину столиків і місць, а також рівні клієнтів (користувачів сайту). Постійні клієнти можуть мати картки різних класів, які дозволяють їм бронювати столики певних категорій у відповідних ресторанах. Користувач сайту може виконувати бронювання столиків у ресторанах на певний період, відповідно до своїх можливостей, а також виконувати пошук вільних місць та можливостей бронювання.

13. Розробити базу даних для сайту проведення змагань і шахів. Сайт дає можливість зареєструватися шаховим клубам, які мають назву і контактну інформацію та до яких належать гравці. Організатори шахового турніру створюють турнір, задаючи назву, місце проведення, період проведення та контактні дані (адресу та телефони). В рамках кожного турніру проводяться ігри між гравцями, результати яких повинні відображатися на сайті. Користувачем такої системи є організатор турніру, який може створювати і змінювати клуби та гравців. Іншим типом користувачів сайту є відвідувачі, які можуть переглядати поточний хід змагань та їх результати і в особистому, і клубному заліку, або ж розраховувати загальний рейтинг гравців чи клубів.

14. Розробити базу даних для сайту з торгівлі запчастинами. Сайт дає можливість вибирати типи механізмів і виконувати пошук запчастин за серійними номерами, виробниками чи певними властивостями. Деталі можуть містити довільну кількість характеристик і підходити до різних механізмів. Користувач

може додавати або змінювати механізми, виробників, запчастини, а також виконувати пошук запчастин за їх властивостями.

15. Розробити базу даних для сайту управління проєктним портфелем. Сайт дає можливість компанії зареєструватися, зареєструвати користувачів та відстежувати стан проєктів через відслідковування окремих завдань у рамках проєкту. Кожне завдання має короткий опис, повний опис, стан, автора та користувача, який відповідає за нього в поточний момент. По завданнях ведеться журнал активності користувачів, який фіксує зміни в кожному з атрибутів завдання. Проєкт може змінюватися залежно від зміни стану всіх його завдань. Користувач може створювати нові проєкти, завдання, міняти їх та виконувати пошук за активним користувачем, проєктом, компанією тощо.

16. Розробити базу даних для сайту продажу квитків для подій. Події реєструються користувачами, які мають контактну інформацію (адресу та телефон), а також номери банківських рахунків. Для кожної події встановлюються місця проведення з назвами, адресами та типами квитків. Окрім того, для кожної події створюються квитки відповідних типів, які можуть використовуватися і для цілої події, і для певних місць її проведення. Користувачі сайту можуть шукати події за місцем чи датою проведення або за вартістю квитків і та ін.

17. Розробити базу даних для сайту соціальної мережі. Соціальна мережа підтримує реєстрацію користувачів зі збереженням усіх їхніх деталей (ім’я, прізвище, дата народження, місце проживання, телефони, сайти/email/skype тощо), місця і періоди перебування, місця і періоди навчання, роботи, служби, приєднані файли (зображення, фільми, аудіо), які можна пов’язувати із місцями з деталей. Окрім того кожен користувач має можливість розміщувати свої повідомлення на власній сторінці, отримувати на повідомлення «лайки» та коментарі, а також додавати інших користувачів у друзі або в «чорний список». Додатково користувач повинен мати змогу шукати нових друзів за довільними критеріями.

18. Розробити базу даних для вебсайту системи постачання. Замовники реєструються на сайті та формують замовлення наявних товарів певних постачальників. Для реєстрації замовник повинен вказати свою назву, контактні дані (адреси та телефони), банківські рахунки та відповідальних осіб. Замовник може обирати товари за найменуваннями, виробниками або іншими властивостями (колір, матеріал, розміри та ін.) Після вибору,

відбувається формування замовлення, для якого автоматично підбивається загальна вартість та підбирається спосіб доставки.

19. Розробити базу даних для сайту пошуку дитячих гуртків. Сайт містить інформацію про гуртки, яка включає назву, опис, контактні дані (адреси та телефони), контактних осіб, рейтинги гуртків та відгуки користувачів сайту, які повинні реєструватися, щоб залишити відгук. Будь-який користувач може виконувати пошук сайтів за частинами адрес, рейтингами тощо.

20. Розробити базу даних для системи обліку працівників компанії. Система веде облік працівників – деталі їх біографії, історія зміни посад та/або відділів. Окрім того, компанія виконує фіксацію переміщення працівників за допомогою RFID міток. Фіксуються час і приміщення в яке заходить працівник та з якого він виходить. До того ж частина обладнання так само оснащена RFID мітками, для яких також збирається інформація про переміщення між приміщеннями компанії. Користувач системи може додавати нові RFID мітки в систему, прив’язувати їх до працівників або обладнання, а також виконувати пошук переміщень тощо.

21. Розробити базу даних для системи автоматизації страхової компанії. Система веде облік застрахованого майна та осіб. Кожна операція страхування містить інформацію про страхувальника, майно, період страхування та суму. Страхувальник описується іменем (ПІБ), персональними даними (дата і місце народження, стать), та контактною інформацією (адреса і телефони). Страхуватися можуть і предмети, і люди. В кожному випадку об’єкт страхування має множину властивостей, які дозволяють його описати. Користувач може ініціювати процес страхування, а також виконувати пошук за угодами за сумами страхування, страхувальниками, предметами страхування тощо.

22. Розробити базу даних для системи автоматизації шкільної бібліотеки. Система веде облік читачів, які реєструються в бібліотеці і можуть позичати книги. Для читачів зберігається прізвище, ім’я, контактні дані (адреси та телефони) та рейтинг (наскільки вчасно повертають позичені книги). Крім того база даних містить інформацію про книги, які зберігаються в бібліотеці, їх статус (доступна, кому позичена, не видається, в ремонті тощо) Бібліотекар може додавати нових читачів, книги і їх деталі, а також здійснювати пошук книг за авторами, видавництвами, назвами, статусом та ін

23. Розробити базу даних для сайту онлайн енциклопедії. Сайт містить наукові статті з різних розділів знань. Кожна стаття, крім назви, вступу, змісту,

основної частини та ілюстрацій, містить множину тегів (ключових слів) та пов’язаних статей. До того ж сайт веде облік користувачів, які виконують редагування, та містить історію змін у статтях. Будь-який користувач сайту може вносити зміни в наявні статті, чи створювати нові, а також шукати їх за тегами або ключовими словами в назві, вступі до статті. Крім того, користувачі можуть переглядати історію змін у статтях.

24. Розробити базу даних для сайту зберігання та онлайн-перегляду фільмів. Сайт містить інформацію про фільми, що розміщені як великі двійкові об’єкти в базі даних. Крім фільмів (назва, рік випуску, актори, студії, короткий зміст та ілюстрації), сайт містить відгуки користувачів на кожен фільм, які формують його рейтинг. Ба більше, зареєстровані користувачі можуть додавати додаткову інформацію до фільму та змінювати наявну. Також можна виконувати пошук фільмів за назвами, акторами, студіями, роками випуску та ін.

25. Розробити базу даних для системи автоматизації технологічних процесів. Система дозволяє збирати дані з датчиків, розміщених на пристроях, розташованих у різних місцях. Пристрої описуються назвою, адресою та координатами, а також містять перелік датчиків, кожен з яких має множину певних властивостей, а також множину каналів обробки даних. Для кожного каналу збираються дані з встановленою частотою. Користувач системи має можливість створювати та змінювати пристрої, підключати чи відключати датчики, змінювати канали збору даних на датчиках, а також виконувати пошук зібраних даних щодо каналів даних (для всіх пристроїв), пристроїв (каналів даних) тощо.

26. Розробити базу даних для системи автоматизації складу. Він може містити окремі приміщення з стелажами, які мають полиці та в певний спосіб ідентифіковані позиції на полицях. Окрім того, склад повинен володіти інформацією про номенклатуру товарів, які зберігаються у ньому та зберігати історію їх руху. Користувач має змогу задавати нові товари, які з’являються на складі і формувати завдання для розміщення нових товарів чи пошуку наявних. До того ж він може аналізувати наявність певних товарів на складі, їх залишки та потребу в поновленні запасів.

ЛАБОРАТОРНА РОБОТА № 2
«Вивчення поняття домени бази даних. Створення таблиць бази даних»

Тема: *«Вивчення поняття домени бази даних. Створення таблиць бази даних».*

Мета роботи: Ознайомлення з поняттям домен, його створенням і використанням. Створення таблиць бази даних відповідно до розробленої на лабораторній №1 концептуальної моделі «сутність–зв'язок» (ER-діаграми).

Теоретичний матеріал

Перелік розділів та понять, з якими необхідно ознайомитись для виконання завдання лабораторної роботи:

1. Встановлення сервера PostgreSQL (postgresql.org).
2. Типи даних.
   2.1. Числові типи.
   2.2. Грошові типи.
   2.3. Символьні типи.
   2.4. Двійкові типи даних.
   2.5. Типи дати / часу.
   2.6. Логічний тип.
3. Бітові рядки.
4. Створення бази даних (CREATE DATABASE).
5. Домени (CREATE DOMAIN).
6. Підмова опису даних (DDL).
   6.1. Створення таблиці (CREATE TABLE).
   6.2. Видалення таблиці (DROP).
   6.3. Зміна структури таблиці (ALTER).
7. Підмова зміни даних (DML).
   7.1. Додавання рядків у таблицю (INSERT).
   7.2. Зміна даних у таблиці (UPDATE).
   7.3. Видалення рядків із таблиці (DELETE).
8. Відображення зв'язків між таблицями за допомогою первинних та зовнішніх ключів (Primary key і Foreign key).
   8.1. Набори атрибутів. Ключові атрибути (потенційні ключі).

8.2. Зв’язки.

8.2.1. Види бінарних зв’язків.

8.2.2. Багатосторонні зв’язки. Їх перетворення в бінарні.

8.3. Підкласи в ER-моделі.

Література та перелік електронних ресурсів, необхідних для ознайомлення з теоретичним матеріалом теми, наведено у списку рекомендованих джерел до теми.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Встановити сервер PostgreSQL.
3. Створити власну базу даних.
4. Створити домен(и).
5. Створити таблиці бази даних згідно з розробленої на лабораторній роботі № 1 власної концептуальної моделі «сутність–зв’язок» (ER-діаграми) з відображенням зв’язків між таблицями за допомогою первинних та зовнішніх ключів. Увести не менше 5-ти кортежів даних у кожну таблицю.
6. Оформити звіт про виконання лабораторної роботи, який має містити:
   - титульну сторінку (див. додаток А);
   - тему, мету та завдання лабораторної роботи;
   - навести знімки екрану зі схемою створеної бази даних та розкритими вузлами Domain та Tables (усі таблиці). В таблицях розкрити Columns і Constraints.
7. Завантажити в канал «БД та ІС. Лабораторна робота» своєї команди в Microsoft Teams.

Контрольні питання

1. Наведіть приклади скалярних типів даних.
2. Які типи даних використовують для оголошення грошових даних?
3. Дайте визначення поняття домена.
4. Підмова даних є сукупністю …
5. Які оператори відносяться до мови оголошення даних?
6. Які оператори відносяться до мови обробки даних?
7. Напишіть оператор створення таблиці **Товар** зі стовпчиками **Назва** і **Кількість**.
8. Змініть таблицю **Товар** додавши до неї стовпчик **Ціна**.
9. Напишіть оператор додавання у таблицю **Товар** рядка з назвою товару ***«Деталь1»*** кількістю 10 і ціною 200.

10. Напишіть для таблиці **Товар** оператор зменшення ціни товару на 50% для товару ***«Деталь1»***.
11. Напишіть для таблиці **Товар** оператор видалення рядків з товарами яких залишилось 0.

Список рекомендованих джерел до теми

1. Garcia-Molina H. Database Systems: 2nd Edition / H. Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. – Pearson Education Inc. – 2009.– 1203 p.
2. Date C. J. An Introduction to Database Systems (8th Edition) / C. J. Date. – 8th ed. – Pearson Education Inc. – 2004. – 1040 p.
3. Groff J. R. SQL The Complete Reference / James R. Groff, Paul N. Weinberg, Andy Oppel. – 3rd Edition. – McGraw Hill Professional. – 2008. – 912 p.
4. PostgreSQL. Documentation. [Електронний ресурс]. – Доступний з: https://www.postgresql.org/docs/ .
5. PostgreSQL Data Types. [Електронний ресурс]. – Доступний з: https://www.postgresqltutorial.com.
6. SQL Database Tutorial W3C. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/sql/sql_create_db.asp .

ЛАБОРАТОРНА РОБОТА № 3
«Обмеження цілісності даних в SQL даних»

Тема: *«Обмеження цілісності даних в SQL даних».*

Мета роботи: Ознайомлення з поняттями обмеження цілісності даних в SQL, їх створення і використання.

Теоретичний матеріал

Типи даних – це спосіб обмеження виду даних, які можуть зберігатися в таблиці. Однак для багатьох застосувань такі обмеження занадто грубі. Наприклад, стовпець, що містить ціну товару, повинен, мабуть, приймати лише додатні значення. Але не існує стандартного типу даних, який приймає лише додатні числа. Інша проблема полягає в тому, що ви можете обмежити дані стовпців стосовно інших стовпців або рядків. Наприклад, у таблиці, що містить інформацію про товар, має бути лише один рядок для кожного коду товару.

З цією метою SQL дозволяє визначати обмеження для стовпців та таблиць. Якщо користувач намагається зберігати дані у стовпці, що порушує обмеження, виникає помилка. Це застосовується, навіть якщо це значення за замовчуванням.

Перелік розділів та понять, з якими необхідно ознайомитися для виконання завдання лабораторної роботи:

1. Обмеження цілісності даних в SQL.
   - 1.1. Обмеження-перевірки CHECK.
   - 1.2. Обмеження NOT NULL.
   - 1.3. Обмеження унікальності UNIQUE.
   - 1.4. Первинні ключі PRIMARY KEY.
   - 1.5. Зовнішні ключі FOREIGN KEY.

Література та перелік електронних ресурсів, необхідних для ознайомлення з теоретичним матеріалом теми, наведено у списку рекомендованих джерел до теми.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Проаналізувати наявні обмеження цілісності даних у створених таблицях та додати відсутні (особливо звернувши увагу на природні ключі). Проілюструвати знімками екрану з командами створення обмежень.
3. Оформити звіт про виконання лабораторної роботи, який має містити:

- титульну сторінку (див. додаток А);
- тему, мету та завдання лабораторної роботи;
- навести знімки екрану з отриманими результатами.

4. Завантажити в канал «БД та ІС. Лабораторна робота» своєї команди в Microsoft Teams.

Контрольні питання

1. Перерахуйте відомі вам обмеження цілісності даних.
2. Що означає обмеження цілісності даних первинний ключ (Primary Key)?
3. Що означає обмеження цілісності даних зовнішній ключ (Foreign Key)?
4. Що означає обмеження цілісності даних UNIQUE?
5. Поясніть обмеження цілісності даних NOT NULL.
6. Наведіть приклад обмеження цілісності даних CHECK.

Список рекомендованих джерел до теми

1. Garcia-Molina H. Database Systems: 2nd Edition / H. Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. – Pearson Education Inc. – 2009.– 1203 p.
2. Date C. J. SQL and Relational Theory: How to Write Accurate SQL Code (3rd edition) / C. J. Date. — O'Reilly Media, Inc. – 2015. – 563 p.
3. PostgreSQL Tutorial. Section 13. PostgreSQL Constraints. [Електронний ресурс]. – Доступний з: https://neon.tech/postgresql/tutorial

ЛАБОРАТОРНА РОБОТА № 4
«Вивчення поняття запитів мови SQL»

Тема: *«Вивчення поняття запитів мови SQL».*

Мета роботи: Вивчення створення і використання запитів мови SQL.

Теоретичний матеріал

Перелік розділів та понять, з якими необхідно ознайомитися для виконання завдання лабораторної роботи: Мова SQL.

1. Синтаксис оператора SELECT.
   1.1. Операнд FROM.
   1.2. Операнд WHERE.
   1.3. Операнди GROUP BY та HAVING.
   1.4. Операнд DISTINCT.
2. Запити з кількома відношеннями.
   2.1. Декартів добуток та з'єднання JOIN.
3. Об'єднання, перетин та різниця запитів (UNION, INTERSECT і EXCEPT).
4. Операнди ORDER BY та LIMIT.
5. Підзапити.

Література та перелік електронних ресурсів, необхідних для ознайомлення з теоретичним матеріалом теми, наведено у списку рекомендованих джерел до теми.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Написати кілька запитів SELECT відповідно до свого варіанту завдання. Обов'язково використати JOIN, фільтр WHERE (з якимось з операторів BETWEEN, LIKE, IS NULL), агрегатні функції і GROUP BY та HAVING.
   Також вміти застосувати множинні оператори UNION, INTERSECT, EXCEPT
   Один з запитів написати з використанням підзапиту у вигляді Correlated Subquery або за допомогою операторів IN, ANY, ALL чи EXISTS)
3. Оформити звіт про виконання лабораторної роботи, який має містити:
   - титульну сторінку (див. додаток А);
   - тему, мету та завдання лабораторної роботи;
   - навести знімки екрана з отриманими результатами.
4. Завантажити в канал «БД та ІС. Лабораторна робота» своєї команди в Microsoft Teams.

Контрольні питання

1. Як розшифровується абревіатура SQL?
2. Які підмови даних мови SQL ви знаєте?
3. Чи SQL може працювати з об’єктно-орієнтованими та різними частково структурованими даними?
4. Який вираз використовують, щоб вибрати певні рядки таблиці?
5. За допомогою якої команди можна сполучати між собою таблиці зі спільними даними?
6. Який пріоритет вищий: виконання умов, указаних у директиві ON, чи умов у виразі WHERE команди JOIN?

Список рекомендованих джерел до теми

1. Garcia-Molina H. Database Systems: 2nd Edition / H. Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom .– Pearson Education Inc. – 2009. – 1203 p.
2. Date C.J. SQL and Relational Theory: How to Write Accurate SQL Code (3rd edition) / C. J. Date. — O’Reilly Media, Inc. – 2015. – 563 p.
3. PostgreSQL Tutorial. [Електронний ресурс]. – Доступний з: https://neon.tech/postgresql/tutorial

ЛАБОРАТОРНА РОБОТА № 5
«Індекси в SQL»

Тема: *«Індекси в SQL».*

Мета роботи: Ознайомлення з індексами, їх створенням і використанням.

Теоретичний матеріал

Для розуміння необхідності та користі індексів в SQL уявімо собі процес обробки найпростішого запиту виду SELECT * FROM R, зверненого до відношення R, записи якого «розкидані» по всьому диску. Системі доведеться перевіряти кожен блок даних у вторинному сховищі, сподіваючись, що заголовки блоків і записів містять достатню інформацію для того, щоб визначити, де початок кожного запису, де він завершується і до якого відношення належить. Краще рішення полягає в резервуванні для кожного відношення певної кількості блоків, можливо, навіть цілих циліндрів диска. Тепер у процесі пошуку блоків вмісту відношення системі вдасться уникнути, передусім, необхідності сканування всього простору дискового сховища. Проте подібний спосіб організації даних не може спростити задачу обробки дещо складнішого запиту – наприклад, SELECT * FROM R WHERE a=10. У цьому випадку є важливими створення і вибір індексів відношення, що дають змогу пришвидшити пошук таких кортежів, певні компоненти яких відповідають заданій умові.

Розглянемо наступний приклад. Припустимо, що у нас є така таблиця:

```
CREATE TABLE  phonebook ( id serial PRIMARY KEY, phone integer,
                        name varchar [50] );
```

і додаток виконує багато подібних запитів:

SELECT name FROM **phonebook** WHERE phone = константа; Якщо система не буде заздалегідь підготовлена, їй доведеться сканувати всю таблицю phonebook рядок за рядком, щоб знайти всі відповідні записи. Коли таблиця містить велику кількість рядків, а цей запит повинен повернути лише кілька (можливо, один або нуль), таке сканування, очевидно, неефективне. Але якщо створити в системі індекс за полем phone, вона зможе знаходити рядки набагато швидше. Можливо, для цього їй знадобиться пройти всього кілька рівнів у дереві пошуку.

Такий підхід часто використовується в технічній літературі: терміни і поняття, які можуть представляти інтерес, збираються в алфавітному покажчику в кінці книги. Читач може переглянути цей покажчик доволі швидко і потім перейти відразу до відповідної сторінки, замість того, щоб перегортати всю книгу в пошуках потрібного матеріалу. Так само, як завдання автора передбачити, що саме будуть шукати в книзі читачі, завдання програміста баз даних - заздалегідь визначити, які індекси будуть корисні.

Створити індекс для стовпця phone розглянутої раніше таблиці можна за допомогою наступної команди:

```
CREATE INDEX phonebook_phone_index ON phonebook (phone);
```

Ім’я індексу phonebook_phone_index може бути довільним, головне, щоб воно дозволяло зрозуміти, для чого цей індекс. Для видалення індексу використовується команда DROP INDEX. Додавати і видаляти індекси можна в будь-який час.

Коли індекс створений, ніякі додаткові дії не потрібні: система сама буде оновлювати його в разі зміни даних у таблиці і сама буде використовувати його в запитах, де, на її думку, це буде ефективніше, ніж сканування всієї таблиці.

Індекси можуть бути корисні також при виконанні команд UPDATE і DELETE з умовами пошуку. Крім того, вони можуть застосовуватися в пошуку зі з’єднанням. Тобто індекс, визначений для стовпця, який бере участь в умові з’єднання, може значно прискорити запити з JOIN.

Створення індексу для великої таблиці може займати багато часу. За замовчуванням PostgreSQL дозволяє паралельно зі створенням індексу виконувати читання таблиці (оператори SELECT), але операції запису (INSERT, UPDATE і DELETE) блокуються до закінчення побудови індексу. Для виробничого середовища це обмеження часто буває неприйнятним. Хоча є можливість дозволити запис паралельно зі створенням індексів, при цьому потрібно враховувати низку застережень.

Після створення індексу система має підтримувати його в стані, відповідному даним таблиці. З цим пов’язані неминучі пов’язані витрати при зміні даних. Отож, індекси, які використовуються в запитах нечасто або взагалі ніколи, повинні бути видалені. Перевірка використання індексу для окремого запиту виконується за допомогою команди EXPLAIN. Вона відображає план виконання, який планувальник PostgreSQL генерує для наданого оператора. План виконання показує, як таблиця(і), на яку посилається інструкція, скануватиметься, шляхом звичайного послідовного сканування, сканування індексу, тощо, і, якщо посилаються на кілька таблиць, які алгоритми об’єднання використовуватимуться для об’єднання необхідних рядків із кожної вхідної таблиці.

Найважливішою частиною відображення є приблизна вартість виконання оператора, яка є припущенням планувальника про те, скільки часу буде потрібно для виконання оператора (вимірюється в одиницях вартості, які є довільними, але, зазвичай, вказується в одиницях вибірки сторінок диска, тобто seq_page_cost традиційно встановлюється такими, що дорівнюють 1.0, а інші параметри вартості встановлюються відносно нього). Фактично відображаються два числа: початкова вартість перед поверненням першого рядка та загальна вартість повернення всіх рядків. Для більшості запитів важлива загальна вартість, але в

таких контекстах, як підзапит у EXISTS планувальник вибере найменшу початкову вартість замість найменшої загальної вартості (оскільки виконавець однаково зупиниться після отримання одного рядка). Крім того, якщо ви обмежуєте кількість рядків для повернення за допомогою пропозиції LIMIT, планувальник робить відповідну інтерполяцію між витратами кінцевої точки, щоб визначити, який план дійсно найдешевший.

Перевірити точність розрахунків планувальника можна за допомогою EXPLAIN ANALYZE, що викликає фактичне виконання оператора, а не тільки планування. Потім на дисплей додається статистика фактичного часу виконання, включаючи загальний час, витрачений на кожен вузол плану (в мілісекундах) і загальну кількість рядків, які він фактично повернув. Найважливіше, на що, зазвичай, слід звернути увагу, це те, чи отримана кількість рядків наближена до реальності.

Важливо! Майте на увазі, що оператор насправді виконується, коли використовується EXPLAIN ANALYZE. Хоча EXPLAIN відкине будь-який вихід, що поверне SELECT, інші побічні ефекти оператора відбуватимуться як зазвичай. Якщо треба використовувати EXPLAIN ANALYZE для операторів INSERT, UPDATE, DELETE, CREATE TABLE AS або EXECUTE, не дозволяючи команді впливати на ваші дані, оформіть код як транзакцію:

```
BEGIN;
EXPLAIN ANALYZE sql_statement;
ROLLBACK;
```

Щоб допомогти планувальнику запитів визначити найефективніші плани виконання запитів можна скористатися командою ANALYZE, що збирає статистичні дані про вміст таблиць у базі даних і зберігає результати в pg_statistic системного каталогу. Наприклад:

```
ANALYZE Cars;
```

для збору статистики таблиці Cars.

Перевірка використання індексу програми без запуску ANALYZE є марною справою, бо за відсутності будь-якої реальної статистики планувальник припускає деякі значення за замовчуванням, які мабуть будуть неточними.

Команди EXPLAIN та ANALYZE є розширенням PostgreSQL і відсутні у стандарті SQL.

Перелік розділів та понять, з якими необхідно ознайомитися для виконання завдання лабораторної роботи (спочатку ознайомтесь з п. 9 Контроль використання індексів – це розділ 11.12 в [4].):

1. Типи індексів.
2. Складені (багато стовпчикові) індекси.

3. Індекси і ORDER BY.
4. Об’єднання декількох індексів.
5. Унікальні індекси.
6. Індекси за виразами.
7. Часткові індекси.
8. Сканування тільки індексу і покриття індексів.
9. Контроль використання індексів.

Література та перелік електронних ресурсів, необхідних для ознайомлення з теоретичним матеріалом теми, наведено у списку рекомендованих джерел до теми.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Створити вказані індекси до вибраних таблиць, попередньо додавши, за можливості, якомога більшу кількість кортежів – **індекс для одного стовпчика, складений (багатостовпчиковий) індекс, унікальний, індекс за виразами, частковий індекс.**
3. Підібрати запити для демонстрації використання індексів, застосовуючи контроль використання індексів (EXPLAIN – для відображення лише плану виконання, або EXPLAIN ANALYZE – що викликає фактичне виконання оператора з отриманням статистики фактичного часу виконання, а не тільки планування) і продемонструвати це відповідними знімками екрана.
4. Оформити звіт про виконання лабораторної роботи, який має містити:
   - титульну сторінку (див. додаток А);
   - тему, мету та завдання лабораторної роботи;
   - навести знімки екрану з отриманими результатами.
5. Завантажити в канал «БД та ІС. Лабораторна робота» своєї команди в Microsoft Teams.

Контрольні питання

1. Яка перевага використання індексів?
2. Типи індексів.
3. Які індекси використовуються для **sequential file?**
4. У якому випадку можна вважати, що індекс вдалий?
5. Багаторівневі індекси.
6. Які структури індексів застосовують до багатовимірних даних?

Список рекомендованих джерел до теми

1. Garcia-Molina H. Database Systems: 2nd Edition / H. Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. – Pearson Education Inc. – 2009. – 1203 p.
2. Date C. J. SQL and Relational Theory: How to Write Accurate SQL Code (3rd edition) / C. J. Date. – O'Reilly Media, Inc. – 2015. – 563 p.
3. Groff J. R. SQL The Complete Reference / James R. Groff, Paul N. Weinberg, Andy Oppel. – 3rd Edition. – McGraw Hill Professional. – 2008. – 912 p.
4. PostgreSQL. Documentation. PostgreSQL 15. [Електронний ресурс]. – Доступний з: https://www.postgresql.org/docs/15/indexes.html.
5. PostgreSQL Tutorial. PostgreSQL Indexes. [Електронний ресурс]. – Доступний з: https://www.postgresqltutorial.com/postgresql-indexes.
6. SQL Tutorial W3C. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/sql/default.asp .

ЛАБОРАТОРНА РОБОТА № 6
«Нормалізація відношень бази даних»

Тема: *«Нормалізація відношень бази даних».*

Мета роботи: Ознайомлення з поняттям нормалізації відношень бази даних та власне самим процесом нормалізації.

Теоретичний матеріал

Метою проєктування бази даних на концептуальному рівні є створення такої логічної моделі даних, яка дозволяє якнайточніше представлення сутностей предметної області, зв'язків між ними та необхідних обмежень. Тобто, потрібно визначити відповідну множину відношень, що містять ці дані. Для цього використовують висхідний або низхідний методи проєктування.

При висхідному проєктуванні спочатку створюють одне *універсальне* відношення зі всіма даними, а потім для виділення окремих відношень застосовують метод нормалізації даних, який починається зі встановлення зв'язків між атрибутами. Звичайно, що за наявності кількох десятків атрибутів – це доволі трудомісткий процес.

Тому, на практиці, зазвичай, застосовують низхідний метод проєктування, при якому використовується модель «сутність–зв'язок» у вигляді ER-діаграми, а нормалізація застосовується лише як методу перевірки правильності отриманого рішення.

Незалежно від того, як створюється схема відношень часто можна покращити якість проєктування, шляхом реалізації певних типів обмежень. Одним з яких є обмеження унікальності, що називається *функціональною залежністю*. Реалізація цього обмеження дуже важлива, бо дає змогу вирішити основну проблему реляційної моделі БД, а саме надлишковість даних, яка, своєю чергою призводить до аномалій поновлення даних – фактично зникання постійних (для предметної області) даних під час поновлення або ж появи надлишкових кортежів під час загалом коректних сполучень. По суті, функціональна залежність є зв'язком типу «багато до одного» між множинами атрибутів усередині змінної відношення. Тут і надалі термін відношення розглядається як *змінна відношення* з множиною всіх допустимих значень своїх даних, на відміну від *значення змінної відношення*, що являє собою конкретне значення змінної відношення в певний момент часу.

Отже, в сенсі змінної відношення функціональна залежність є обмеженням цілісності.

Означення для випадку множини всіх допустимих значень змінної відношення:

Нехай R – змінна відношення, а X та Y – довільні множини атрибутів з R. Y функціонально залежить від X (X→Y) тоді і лише тоді, коли для будь-якого допустимого значення змінної відношення R, значення атрибутів X відношення R зв'язане з точно одним значенням атрибутів Y відношення R. Тобто, для будь-якого допустимого значення змінної відношення R, якщо два довільні кортежі з R збігаються за значеннями атрибутів X, то вони збігаються і за значеннями атрибутів Y. Ліва і права частини запису називаються відповідно – *детермінантом* та залежною частиною. І детермінант, і залежна частина є множинами. Якщо X є потенційним ключем R, то усі атрибути Y змінної відношення R повинні бути функціонально залежні від X. Якщо існує залежність A→B і A не є потенційним ключем, то можна говорити про наявність надлишковості в R. Надмножина потенційного ключа називається *суперключем*. Суперключ володіє властивістю унікальності, але не обов'язково є нескоротним. Звісно, потенційний ключ – це частковий випадок суперключа. Якщо SK – суперключ для змінної відношення R, а A – атрибут R, то в R обов'язково існує функціональна залежність SK → A.

Функціональна залежність є *тривіальною*, тоді і лише тоді коли права частина її символічного запису є підмножиною лівої частини.

Цілі нормалізації:

- позбавлення аномалій при модифікації даних (insert, update, delete);
- мінімізація реструктурування бази даних при додаванні нових типів;
- забезпечення нейтральності відношень до статистики запитів із бігом часу (запити виконуються однаково незалежно від змін у базі даних).

Ймовірно, що відношення знаходиться в певній нормальній формі, якщо задовольняється певна сукупність умов. Для спрощення припускаємо, що кожна змінна відношення тут і надалі має лише один потенційний ключ який є первинним.

Перша нормальна форма (1НФ)

Відношення відповідає 1НФ тоді, коли є ключовий атрибут, значення атрибутів тільки елементарні (неподільні, атомарні) та нема груп атрибутів, що повторюються.

Наприклад.

Відношення Staff (**empl id**, name, phone1, phone2), де name може збігатися, а **empl id** є PrimaryKey (PK)

empl_id	Name	phone1	phone2
1	Юрій Василенко	2134567800	3214567890
2	Андрій Петренко	1234567777	NULL
3	Юрій Петренко	4512367888	NULL

Щоб привести відношення до 1НФ потрібно замінити атрибут name на два атомарні first_name і last_name, а також об'єднати групу повторюваних атрибутів phone1 і phone2 в один атрибут phone.

Для цього проведемо декомпозицію на 2 відношення *без втрат*, зберігши зв'язки між атрибутами за допомогою зовнішнього ключа FK_empl_id.

empl_id	first_name	last_name
1	Юрій	Василенко
2	Андрій	Петренко
3	Юрій	Петренко

FK_empl_id	**phone**
1	2134567800
1	3214567890
2	1234567777
3	4512367888

Друга нормальна форма (2НФ)

Відношення знаходиться в 2НФ, якщо виконуються обмеження 1НФ і кожен неключовий атрибут функціонально повно залежить від первинного ключа (зокрема і складеного).

Наприклад.

Відношення Supply (**maker, detail_id**, weight).

Як бачимо, є функціональна залежність ваги від частини PK деталі detail_id -> weight (в загальному випадку вага може й збігатися в різних деталях). Це призводить до аномалій оновлення даних. Наприклад, при видаленні кортежу з вагою певної деталі можемо також втратити інформацію про виробника, якщо він постачав лише цю деталь. Насправді, видно, що в початковій таблиці фактично було дві сутності – постачальник товару та сам товар з його характеристиками (вага).

Тому, робимо декомпозицію на 2 відношення *без втрат*, зберігши зв'язки між атрибутами.

maker	**FK_detail_id**	weight
1	5	100
2	5	100
1	3	100

maker	**FK_detail_id**
1	5
2	5
1	3

FK_detail_id	weight
5	100
3	100

Третя нормальна форма (ЗНФ)

Відношення знаходиться у ЗНФ, якщо виконуються обмеження 2НФ і всі *неключові* атрибути відношення взаємно незалежні і повністю залежать від первинного ключа, тобто кожний *неключовий* атрибут нетранзитивно залежить лише від ключа.

Розглянемо приклад відношення Рейси з літаками та кількістю місць у них.

Flights (**flight**, plane, seats**)**

flight	Plane	seats
BRU	A320	180
LTN	B-737	180
VIE	E-195	124
AMS	A320	180

Як бачимо є функціональна залежність неключових атрибутів plane->seats, а також залежність plane від PK flight, тобто маємо транзитивну залежність **flight** -> plane -> seats, яка може призводити до аномалій оновлення даних. Наприклад, при видаленні кортежу з певним рейсом, можемо також втратити інформацію про певний літак, якщо він був задіяний лише на цьому рейсі.

Тому, робимо декомпозицію на 2 відношення шляхом винесення залежного неключового атрибута seats в окрему таблицю і копіювання в неї детермінанта plane функціональної залежності plane -> seats, де він стає PK.

flight	plane
BRU	A320
LTN	B-737
VIE	E-195
AMS	A320

plane	seats
A320	180
B-737	180
E-195	124

Використання штучних ключів ID лише ускладнює ситуацію з нормалізацією, а також збільшує кількість JOIN під час роботи з даними.

flight_id	flight	FK_plane_id
1	BRU	1
2	LTN	2
3	VIE	3
4	AMS	1

plane_id	plane	seats
1	A320	180
2	B-737	180
3	E-195	124

Як бачимо в таблиці Plane теж є функціональна залежність атрибутів plane -> seats, але тут атрибут plane є потенційним ключем.

Якщо говорити неформально, то нормалізація полягає у створенні для кожної сутності з її атрибутами окремої таблиці.

Нормальна форма Бойса – Кодда (НФБК)

Відношення знаходиться в НФБК, тоді і лише тоді коли детермінант кожної функціональної залежності є потенційним ключем. Якщо це правило не виконується, то, щоб привести вказане відношення до НФБК його слід розділити на два відношення шляхом двох операцій проєкції на кожну функціональну залежність детермінант, якої не є потенційним ключем:

1. Проєкція без атрибутів залежної частини такої функціональної залежності.
2. Проєкція на всі атрибути цієї функціональної залежності.

Визначення НФБК не потребує жодних умов попередніх нормальних форм. Якщо проводити нормалізацію послідовно, то в переважній більшості випадків при досягненні 3НФ автоматично будуть задовольнятися вимоги НФБК.

3НФ не збігається з НФБК лише тоді, коли *одночасно* виконуються такі три умови:

1. Відношення має 2 або більше потенційних ключів.
2. Ці потенційні ключі складені (містять більш ніж один атрибут).

3. Ці потенційні ключі перекриваються, тобто мають щонайменше один спільний атрибут.

Четверта нормальна форма (4НФ)

Відношення знаходиться в 4НФ, якщо воно є в НФБК і в ньому відсутні багатозначні залежності, які не є функціональними залежностями. Четверта нормальна форма стосується відношень, в яких є повторювані набори даних. Декомпозиція, заснована на функціональних залежностях, не призводить до виключення такої надмірності. В цьому випадку використовують декомпозицію, засновану на багатозначних залежностях. Багатозначна залежність є узагальненням функціональної залежності і розглядає відповідності між множинами значень атрибутів. Приведення відношення до 4НФ дозволяє виключити тип аномалій оновлення. Для приведення відношення з НФБК до 4НФ слід виконати проєкції вихідного відношення на пари атрибутів, що створюють багатозначні залежності. 4НФ є окремим випадком 5НФ, коли повна декомпозиція повинна бути з'єднанням рівно двох проєкцій. Доволі не просто підібрати реальну таблицю, яка перебувала б у 4НФ, але не була б у 5НФ. Ця нормальна форма має більший інтерес для теоретичних досліджень, ніж для практики проєктування баз даних.

П'ята нормальна форма (5НФ)

У всіх розглянутих до цього моменту випадках нормалізація відношення відбувається декомпозицією одного відношення на два. Іноді нормалізувати відношення шляхом декомпозиції на два відношення без втрат не вдається, але є можливість декомпозиції вихідного відношення без втрат на більшу кількість відношень, тобто $n > 2$. Якщо в процесі природнього з'єднання отриманих відношень порівняно з вихідним відношенням генеруються зайві кортежі, то така декомпозиція характеризується залежністю з'єднання.

У відношенні R (X, У, ..., Z) відсутня залежність з'єднання *(X, У, ..., Z), в тому і тільки в тому випадку, коли R відтворюється без втрат шляхом з'єднання своїх проєкцій на X, У, ..., Z.

Відношення знаходиться в п'ятій нормальній формі, якщо воно не містить залежностей з'єднання.

Шоста нормальна форма (6НФ)

Таблиця знаходиться у 6НФ, якщо вона є у 5НФ та задовольняє вимогу відсутності нетривіальних залежностей. Зазвичай 6НФ ототожнюють з доменно-ключовою нормальною формою**.**

Доменно-ключова нормальна форма (ДКНФ)

Це нормальна форма, що використовується в нормалізації баз даних і вимагає щоб база даних не містила жодних інших обмежень, крім обмежень доменів і обмежень ключів. Обмеження домену – обмеження, яке наказує використовувати для певного атрибуту значення тільки з деякого заданого домену. Обмеження по своїй суті є заданням переліку (або логічного еквіваленту переліку) допустимих значень типу і оголошенням про те, що вказаний атрибут має даний тип. Обмеження ключа – обмеження, яке стверджує, що деякий атрибут чи комбінація атрибутів є потенційним ключем. Будь-яка змінна відношення, що знаходиться в ДКНФ, обов'язково є в 5НФ. Але не будь-яку змінну відношення можна привести до ДКНФ.

Неформальний процес нормалізації.

1. Відношення у 1НФ розбивають на такі проєкції, які дозволяють ліквідувати нескоротні функціональні залежності. В результаті отримуємо 2НФ.
2. Отримані відношення у 2 НФ розбиваються на проєкції, які дозволять ліквідувати транзитивні функціональні залежності. Отримуємо 3НФ.
3. Отримані відношення у 3НФ розбиваються на проєкції, у яких ліквідовуються функціональні залежності, де детермінанти не є потенційними ключами. Отримуємо НФБК.
4. Відношення у НФБК розбиваються на проєкції, у яких ліквідовано усі багатозначні залежності, що не є функціональними. Отримуємо 4НФ.
5. Відношення у 4 НФ розбиваємо на проєкції у яких ліквідовано залежності сполучення, що не визначаються потенційними ключами. Отримуємо 5НФ.

На кожному етапі декомпозиція має відбуватися без втрат і зі збереженням залежностей.

Як зазначається в [3], нормалізація корисна, але її не можна вважати панацеєю, в чому легко переконатися, розглянувши її цілі і те, наскільки добре вона їм відповідає:

• прийти до структури, яка б «добре» представляла реальний світ, тобто була б інтуїтивно зрозумілою і хорошою основою для майбутнього розширення;

• зменшити надлишковість даних;

• у такий спосіб уникнути деяких аномалій при оновленні;

• спростити формулювання і перевірку деяких обмежень цілісності.

Перелік понять, які необхідно опрацювати, використовуючи лекційний матеріал та додаткову літературу, для виконання завдання лабораторної роботи:

1. Поняття функціональної залежності
   1.1. Суперключ.

1.2. Детермінант і залежна частина.
1.3. Тривіальна, нетривіальна залежності.
1.4. Повна функціональна залежність.
1.5. Транзитивна функціональна залежність.
2. Багатозначні залежності.
3. Надлишковість даних і аномалії поновлення даних у БД.
4. Нормалізація відношень.

Література та перелік електронних ресурсів, необхідних для ознайомлення з теоретичним матеріалом теми, наведено у списку рекомендованих джерел до теми.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Проаналізувати створені відношення (таблиці) бази даних на відповідність нормальним формам. За необхідності провести нормалізацію відношень до певної форми за допомогою декомпозиції . Пояснити необхідність проведеної нормалізації або чому її не проводили, а залишили відношення без змін. Проілюструвати скрінами.
3. Оформити звіт про виконання лабораторної роботи, який має містити:
- титульну сторінку;
- тему, мету та завдання лабораторної роботи;
- пояснення виконаних дій та скріни екрана з отриманими результатами.
4. Завантажити в канал «БД. Лабораторна робота» своєї команди в Teams.

Контрольні питання

1. Наведіть означення функціональної залежності для значення змінної відношення.
2. Наведіть формулювання 1НФ.
3. Наведіть формулювання 2НФ.
4. Наведіть формулювання 3НФ.
5. Наведіть формулювання НФ Бойса – Кодда.

Список рекомендованих джерел до теми

1. Garcia-Molina H. Database Systems: 2nd Edition / H. Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. – Pearson Education Inc. – 2009.– 1203 p.
2. Date C. J. An Introduction to Database Systems (8th Edition) / C. J. Date. – 8th ed. – Pearson Education Inc. – 2004. – 1040 p.
3. Date C.J. SQL and Relational Theory: How to Write Accurate SQL Code (3rd edition) / C. J. Date. — O'Reilly Media, Inc. – 2015. – 563 p.

ЛАБОРАТОРНА РОБОТА № 7
«Користувацькі функції на мові запитів SQL»

Тема: *«Користувацькі функції на мові запитів SQL».*

Мета роботи: Ознайомлення з поняттям користувацьких функцій на мові запитів SQL, їх створенням та застосуванням.

Теоретичний матеріал

У PostgreSQL представлені функції чотирьох видів:

- функції на мові запитів (функції, написані на SQL);
- функції на процедурних мовах (функції, написані, наприклад, на
- PL/pgSQL, PL/Tcl або PL/Python);
- внутрішні функції;
- функції на мові C.

Функції будь-яких видів можуть приймати як аргументи (параметри) базові типи, складені типи або їх поєднання та повертати, як абсолютні значення базового чи складеного типу, і як сукупність рядків (тобто кілька рядків).

Функції SQL виконують довільний список операторів розділених крапкою з комою і повертають результат останнього запиту в списку. Після останнього оператора крапки з комою може не бути. Якщо функція не оголошена такою, що повертає void, останній оператор має бути SELECT, INSERT, UPDATE, або DELETE з використанням оператора RETURNS.

У простому випадку буде повернуто перший рядок результату останнього запиту. Також можна оголосити функцію SQL, яка повертає сукупність рядків, вказавши як тип повернення SETOF деякий_тип, або оголошуючи її з оператором RETURNS TABLE (стовпці). В цьому випадку будуть повернуті всі рядки результату останнього запиту. Окрім запитів SELECT, функції можуть містити команди, які змінюють дані (INSERT, UPDATE і DELETE), а також інші SQL-команди. Але в SQL-функціях не можна використовувати команди управління транзакціями, наприклад COMMIT, SAVEPOINT і деякі допоміжні команди, як VACUUM. Останньою командою має бути SELECT чи команда з RETURNING, що повертає результат з типом повернення функції. Якщо треба визначити функцію SQL, що виконує дії, але не повертає значень, то можна оголосити її тип void. Наприклад, функція видаляє рядки з від'ємною зарплатою з таблиці Employees:

```
CREATE FUNCTION Clean_Employees() RETURNS void AS
$$ DELETE FROM Employees
WHERE Salary < 0;
```

```
$$ LANGUAGE SQL;
Використання функції:
SELECT Clean_Employees();
```

Синтаксис команди CREATE FUNCTION вимагає, щоб тіло функції було записане як рядкова константа. Найзручніше заключати її в знаки долара ($$), бо якщо застосовувати звичайний синтаксис з апострофами, то доведеться дублювати апострофи (') і обернену скісну риску (\) в тілі функції.

Важливо пам'ятати, що перед тим, як почнеться виконання команд SQL-функції, здійснюється розбір її тіла. Тому, коли SQL-функція містить команди, що модифікують системні каталоги, наприклад, CREATE TABLE, то дія таких команд не буде проявлятися на стадії аналізу наступних команд цієї функції. Так, команди

```
CREATE TABLE Foos (...);
INSERT INTO Foos VALUES(...);
```

не будуть працювати як очікується, якщо їх упакувати в одну SQL-функцію, бо таблиця Foos ще не буде існувати на момент розбору команди INSERT (тому під час аналізу INSERT отримаємо повідомлення про помилку, бо ще нема такої таблиці, адже виконання команд відбувається після їх аналізу).

У таких випадках замість SQL-функції використовують функції на мові PL/pgSQL (збережені процедури).

Перелік понять, з якими необхідно ознайомитись, використовуючи лекційний матеріал та додаткову літературу, для виконання завдання лабораторної роботи:

1. Користувацькі функції на мові запитів SQL.
   1.1. Аргументи SQL-функцій.
   1.2. Функції SQL з базовими типами.
   1.3. Функції SQL зі складними типами.
   1.4. Функції SQL з вихідними параметрами.
   1.5. Функції SQL зі значеннями аргументів за замовчуванням.
   1.6. Функції SQL, які повертають рядок таблиці.
   1.7. Функції SQL, які повертають підмножину рядків таблиці (SETOF).
   1.8. Функції SQL, які повертають таблиці (TABLE).

Електронні ресурси, необхідні для ознайомлення з теоретичним матеріалом теми, наведено у списку рекомендованих джерел до теми.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Відповідно до свого завдання написати не менше 3-х користувацьких функцій на мові запитів SQL. Виконання функцій і результати проілюструвати знімками екрана, а також поясненнями роботи функції. Функції мають бути різних типів - з базовими типами чи складними типами, які повертають рядок таблиці чи всю таблицю (TABLE) тощо.
3. Оформити звіт про виконання лабораторної роботи, який має містити:
   - титульну сторінку (див. додаток А);
   - тему, мету та завдання лабораторної роботи;
   - навести знімки екрана з отриманими результатами.
4. Завантажити в канал «БД та ІС. Лабораторна робота» своєї команди в Microsoft Teams.

Контрольні питання

1. Назвіть види функцій представлених у PostgreSQL.
2. Які типи даних можуть приймати функції як аргументи?
3. Які оператори можуть використовуватися у функції SQL?
4. Які особливості синтаксису функції SQL?
5. Поясніть послідовність виконання функцій SQL.
6. Наведіть приклад функцій SQL з базовими типами та складними типами.
7. Наведіть приклад функції SQL з використанням значень за замовчуванням для вхідних і вихідних аргументів.
8. Наведіть приклад функцій SQL, які повертають підмножину рядків таблиці.
9. Наведіть приклад функцій SQL, які повертають таблиці.
10. Наведіть приклад поліморфної функції SQL.

Список рекомендованих джерел до теми

1. PostgreSQL. Documentation. PostgreSQL 15. Query Language (SQL) Functions [Електронний ресурс]. – Доступний з: https://www.postgresql.org/docs/15/xfunc-sql.html.

ЛАБОРАТОРНА РОБОТА № 8
«Віртуальні таблиці SQL»

Тема: *«Віртуальні таблиці SQL».*

Мета роботи: Ознайомлення з поняттям Віртуальні таблиці SQL, їх створення та застосування.

Теоретичний матеріал

Відношення, визначене за допомогою команди CREATE TABLE фізично існує в базі даних і може знаходитись в одному і тому ж стані необмежено довго, доки не буде змінено командами INSERT, DELETE, UPDATE або не буде видалено за допомогою DROP. В SQL підтримується можливість визначення відношень і іншої категорії, які, на відміну від «звичайних» відношень, не створюються на фізичному рівні. Такі відношення заведено називати віртуальними таблицями (virtual tables) або представленнями (VIEWS).

До віртуальних таблиць можна звертатись із запитами, а в деяких випадках – і з командами модифікації.

Представлення дають можливість «перевизначити» структуру бази даних, даючи можливість кожному користувачеві бачити свою власну структуру і свою частину вмісту бази даних. Представлення можна використовувати майже скрізь, де можна використовувати базові таблиці. І доволі часто представлення створюються на базі інших представлень.

- Представлення – це віртуальна таблиця, створена на основі запиту. Як і реальна таблиця, вона містить рядки і стовпці даних, проте дані, видимі в представленні, насправді є результатами запиту.
- Представлення може бути простою підмножиною рядків (горизонтальне) і стовпців (вертикальне) однієї таблиці, може резюмувати вміст таблиці (згруповане представлення) або містити дані з двох або більше таблиць (поєднане представлення).
- В інструкціях SELECT, INSERT, DELETE і UPDATE до представлення можна звертатися як до звичайної таблиці. Однак більш складні представлення модифікувати не можна, вони доступні тільки для читання.
- Представлення зазвичай використовуються для спрощення видимої структури бази даних і запитів, а також для захисту деяких рядків і стовпців від несанкціонованого доступу.
- Матеріалізовані представлення можуть підвищити ефективність роботи бази даних у разі високої активності запитів і низької активності оновлень. Дещо спрощений вигляд команди створення представлення:

```
CREATE OR REPLACE  VIEW  ім'я [ ( ім'я _стовпця [, ...] ) ]
    AS запит
    WITH  [ CASCADED | LOCAL ] CHECK OPTION
```
,

де *ім'я* – ім'я створюваного представлення (можливо, доповнене схемою); *ім'я_стовпця* – необов'язковий список імен, які призначаються стовпцям представлення. Якщо відсутній, імена стовпців формуються з результатів запиту; *запит* – команда SELECT або VALUES, яка видає стовпці і рядки представлення;

```
    WITH  [ CASCADED | LOCAL ] CHECK OPTION
```

– ця вказівка керує поведінкою змінюваних представлень. Якщо вона присутня, при виконанні операцій INSERT і UPDATE з цим представленням буде перевірятися, чи задовольняють нові рядки умову, що визначає представлення (тобто, перевіряється, чи будуть нові рядки видно через це представлення). Якщо вони не задовольняють умові, операція не буде виконана. Якщо вказівка CHECK OPTION відсутня, команди INSERT і UPDATE зможуть створювати в цьому представленні рядки, яких не буде видно в ньому.

Підтримуються наступні ***варіанти перевірки***:

- LOCAL

Нові рядки перевіряються лише за умовами, наведеними безпосередньо в самому представленні. Будь-які умови, визначені в нижчих базових представленнях, не перевіряються (якщо тільки в них немає вказівки CHECK OPTION).

- CASCADED

Нові рядки перевіряються за умовами даного представлення та всіх нижчих базових. Якщо вказано CHECK OPTION, а LOCAL й CASCADED не вказано, мається на увазі вказівка CASCADED.

Вказівку CHECK OPTION не можна використовувати з рекурсивними представленнями, а також CHECK OPTION підтримується тільки для змінюваних представлень, які не мають тригерів INSTEAD OF і правил INSTEAD.

Приклади деяких представлень:

1. Представлення, що містить назву офісу та його місцезнаходження

```
CREATE VIEW Officeinfo AS
      SELECT Office, City, Region
FROM Offices;
```
.

2. Представлення, що містить сумарні дані про замовлення по кожному службовцю

```
CREATE VIEW Ord_By_Rep (who, how_many, total, low, high, average)
AS
  SELECT rep, COUNT(*), SUM(amount), MIN(amount), MAX(amount),
                                                  AVG(amount)
     FROM Orders
     GROUP BY rep;
```

3. Використання представлення у запиті, що виводить ім’я, число замовлень, загальну вартість замовлень і середню вартість замовлення по кожному службовцю

```
SELECT name, how_many, total, average
FROM Salesreps, Ord_By_Rep
WHERE who = empl_num    ORDER BY total DESC;
```

Доступ до таблиць, задіяних у представленні, визначається правами власника представлення. У деяких випадках це дозволяє організувати безпечний, але обмежений доступ до базових таблиць. Однак не всі представлення можуть бути захищеними (див. Документація PostgresSQL13 Розділ 40.5 Rules and Privileges). Функції, що викликаються в представленні, виконуються так, наче вони викликаються безпосередньо із запиту, що звертається до представлення. Тому користувач представлення повинен мати права, необхідні для виклику всіх функцій, задіяних у представленні.

Змінюване представлення має задовольняти наступні умови:

- Представлення повинно мати тільки один запис у своєму FROM списку, який має бути таблицею або іншим представленням, яке можна оновлювати;
- Визначення представлення не повинно містити WITH, DISTINCT, GROUP BY, HAVING, LIMIT або OFFSET речення на верхньому рівні;
- Визначення представлення не повинно містити набір операцій UNION, INTERSECT або EXCEPT на верхньому рівні;
- Список вибору представлення не повинен містити жодних агрегатів, віконних функцій або функцій, що повертають набір.

Переваги представлень

Використання представлень в базах даних різних типів може виявитися корисним в найрізноманітніших ситуаціях. У базах даних на персональних комп’ютерах їх застосовують для зручності, бо це дозволяє спрощувати запити до бази даних. У промислових базах даних вони відіграють головну роль у створенні власної структури бази даних для кожного користувача і гарантуванні її безпеки.

Основними перевагами представлень є:

- Безпека. Кожному користувачеві можна дозволити доступ до невеликого числа представлень, що містять тільки ту інформацію, яку йому дозволено знати. Отож, можна здійснити обмеження доступу користувачів до інформації, що зберігається.
- Простота запитів. За допомогою представлення можна отримати дані з декількох таблиць і представити їх як одну таблицю, перетворюючи в такий спосіб запит до багатьох таблиць в однотабличний запит до представлення.
- Структурна простота. За допомогою представлень для кожного користувача можна створити власну структуру бази даних, визначивши її як множину доступних користувачеві віртуальних таблиць.
- Захист від змін. Представлення може повертати несуперечливий і незмінний образ структури бази даних, навіть якщо базові таблиці діляться, реструктуруються або перейменовуються. Однак визначення представлення має бути оновлено, коли перейменовуються таблиці або стовпці, що лежать у його основі.
- Цілісність даних. Якщо доступ до даних або введення даних здійснюється за допомогою представлення, СКБД може автоматично перевіряти, чи виконуються певні умови цілісності.

Недоліки представлень

Поряд із зазначеними перевагами, представлення мають і три суттєві недоліки.

- Продуктивність. Представлення створює лише видимість існування відповідної таблиці, і СКБД доводиться перетворювати запит до представлення в запит до базових таблиць. Якщо представлення відображає багато-табличний запит, то простий запит до представлення стає складним об’єднанням і на його виконання може знадобитися багато часу. Однак це пов’язано не з тим, що запит звертається до представлення, будь-який погано побудований запит може викликати проблеми з продуктивністю. Річ у тім, що складність запиту ховається в представленні, так що користувачі не уявляють, який обсяг роботи може викликати навіть, здається, простий запит.
- Керованість. Представлення, як і всі інші об’єкти баз даних, повинні бути керовані. Якщо розробники і користувачі баз даних зможуть безконтрольно створювати представлення, то робота адміністратора бази даних суттєво ускладниться. А надто в тому випадку, коли створюються представлення, в основі яких є інші представлення, які, своєю чергою, можуть бути засновані на інших представленнях. Чим більше рівнів між

базовими таблицями і представленнями, тим складніше вирішувати проблеми з представленнями, які можуть виникнути в такій системі.

- Обмеження на оновлення. Коли користувач намагається оновити рядки представлення, СКБД повинна перетворити запит у запит на оновлення рядків базових таблиць. Це можливо для простих представлень; складніші представлення оновлювати не можна, вони доступні тільки для вибірки.

Зазначені недоліки означають, що не варто без потреби застосовувати представлення, особливо багаторівневі, і використовувати їх замість базових таблиць.

У кожному конкретному випадку необхідно враховувати зазначені переваги і недоліки представлень.

Перелік понять, з якими необхідно самостійно ознайомитися, використовуючи лекційний матеріал та додаткову літературу, для виконання завдання лабораторної роботи:

1. Віртуальні таблиці SQL (представлення).
   1.1. Створення представлень (CREATE VIEW).
   1.2. Типи представлень (горизонтальне, вертикальне, змішане, згруповане, поєднане).
   1.3. Змінювані представлення. Умови модифікації представлення.
   1.4. Контроль над модифікацією представлення (CHECK OPTION).
2. Матеріалізовані представлення.
3. Видалення представлення (DROP VIEW).

Література та перелік електронних ресурсів, необхідних для ознайомлення з теоретичним матеріалом теми, наведено у списку рекомендованих джерел.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Відповідно до свого завдання написати не менше 3-х представлень SQL. Виконання і результати проілюструвати знімками екрану, а також поясненнями їх роботи. Представлення мають бути різних типів – змінювані (з використанням CHECK OPTION), незмінювані, матеріалізовані.
3. Оформити звіт про виконання лабораторної роботи, який має містити:
   - титульну сторінку (див. додаток А);
   - тему, мету та завдання лабораторної роботи;
   - пояснення виконаних дій та знімки екрана з отриманими результатами.

4. Завантажити в канал «БД. Лабораторна робота» своєї команди в Microsoft Teams.

Контрольні питання

1. Що означає поняття Представлення (VIEWS) та для чого вони застосовуються?
2. Які умови має задовольняти змінюване представлення?
3. Поясніть зміну змінюваного представлення, якщо представлення це віртуальна таблиця, то що ж змінюється?
4. Поясніть для чого в представленні вказується CHECK OPTION.
5. Що таке матеріалізоване представлення?
6. Назвіть переваги представлень.
7. Які недоліки властиві представленням?

Список рекомендованих джерел до теми

1. Garcia-Molina H. Database Systems: 2nd Edition / H. Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. – Pearson Education Inc. – 2009.– 1203 p.
2. PostgreSQL. Documentation. PostgreSQL 15. CREATE VIEW define a new view. [Електронний ресурс]. – Доступний з: https://www.postgresql.org/docs/15/sql-createview.html.
3. PostgreSQLTutorial. PostgreSQL Views. [Електронний ресурс. – Доступний з: https://www.postgresqltutorial.com/postgresql-views/.

ЛАБОРАТОРНА РОБОТА № 9
«Збережені процедури СКБД PostgreSQL»

Тема: *«Збережені процедури СКБД PostgreSQL».*

Мета роботи: Вивчення поняття збережених процедур СКБД PostgreSQL v.11 .

Теоретичний матеріал

Процедура представляє собою об’єкт бази даних, подібний до функції. Відмінність полягає в тому, що

- процедура не повертає значення, і тому для неї не визначається тип повернення (хоча є спосіб повернути значення через вхідні параметри типу inout);
- процедура викликається явно, оператором CALL, тоді як функція викликається у складі запиту або команди DML;
- починаючи з 11 версії в PostgreSQL представлені збережені процедури, які підтримують транзакції, тобто можна розпочати транзакцію, зафіксувати або відкотити її (на відміну від визначених користувачем функцій, що не можуть виконувати транзакції).

Усе, що зазначено про створення користувацьких функцій, може бути застосовано і до процедур, за винятком того, що для процедур використовується команда CREATE PROCEDURE, не визначається тип результату, і до них не належать деякі властивості, наприклад, строгість (strict).

Загалом збережена процедура – це іменований набір операторів SQL та певної процедурної мови, який зберігається на сервері у скомпільованому вигляді. Отже, процедура – це блок з однієї або більше команд. Це може бути не тільки один запит, а ціла програма, з власною логікою (оператори IF, цикли). Процедура може приймати заздалегідь певні змінні і використовувати їх у своїх розрахунках, завдяки чому, результат роботи процедури може бути динамічним і залежатиме від певних умов і/або стану одержуваних значень змінних. У процедурі можна:

- використовувати оператори, які виконують будь-які операції DML у базі даних (вибірка, додавання, зміна або видалення даних), включаючи ;
- можливість виклику інших вбудованих процедур;
- використовувати вхідні параметри;
- повертати статус виконання для відображення вдалого або
- помилкового виконання;
- повертати декілька значень у формі вихідних параметрів.

Збережена процедура дозволяє згрупувати блок обчислень і послідовність запитів ***усередині*** сервера бази даних, в такий спосіб, ми отримуємо силу процедурної мови і простоту використання SQL за значної економії пов'язаних витрат на клієнт-серверну взаємодію:

- виключаються додаткові звернення між клієнтом і сервером;
- проміжні непотрібні результати не передаються між сервером і клієнтом;
- є можливість уникнути численних аналізів одного запиту.

У результаті це призводить до значного збільшення продуктивності порівняно з додатком, що не використовує збережені процедури.

Але, крім переваг, є ще деякі застереження:

- повільніша розробка програмного забезпечення, оскільки PL/pgSQL вимагає спеціалізованих навичок, якими не володіє багато розробників;
- складно керувати версіями і налагоджувати їх;
- ускладнюється переносимість до інших систем керування базами даних.

Перелік розділів та понять, з якими необхідно ознайомитися для виконання завдання лабораторної роботи (документація PostgreSQL v.11 і вище):

Розділ 43. PL/pgSQL – процедурна мова SQL.

1. Огляд.
   1.1. Переваги використання PL/pgSQL .
   1.2. Підтримувані типи даних аргументів і значень.
2. Структура PL/pgSQL.
3. Оголошення.
   3.1 Оголошення параметрів функції.
   3.2 ALIAS (нове_ім'я ALIAS FOR старе_ім'я).
   3.3 Спадкування типів даних (змінна %TYPE).
   3.4 Типи кортежів (ім'я ім'я_таблиці%ROWTYPE).
   3.5 Тип record (ім'я RECORD).
4. Вирази.
5. Основні оператори.
   5.1 Присвоювання.
   5.2 Виконання команди, що не повертає результат (PERFORM запит).
   5.3 Виконання запиту, що повертає один рядок (STRICT).
   5.4 Виконання динамічно сформованих команд (EXECUTE).
   5.5 Статус виконання команди (GET DIAGNOSTICS / FOUND).
   5.6 Не робити нічого (NUUL).

6. Керуючі структури (галуження).
   6.1 Команди для повернення значення з функції.
   6.2 Завершення процедури.
   6.3 Виклик процедури (CALL).
   6.4 Умовні оператори.
   6.5 Прості цикли.
   6.6 Цикл за результатами запиту.
   6.7 Цикл за елементами масиву.
   6.8 Обробка помилок (EXCEPTION).
   6.9 Отримання інформації про місце виконання (GET DIAGNOSTICS stack =PG_CONTEXT).
7. Курсори.
   7.1 Оголошення курсорних змінних.
   7.2 Відкриття курсору.
   7.3 Використання курсорів.
   7.4 Обробка курсору в циклі.
8. Повідомлення і помилки.
   8.1 Виведення повідомлень і помилок (RAISE).
   8.2 Налагоджувальні перевірки в функції PL/pgSQL (ASSERT).

Література та перелік електронних ресурсів, необхідних для ознайомлення з теоретичним матеріалом теми, наведено у списку рекомендованих джерел.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Розробити дві або три збережені процедури (або користувацькі функцію на процедурній мові PL/pgSQL для версій PostgreSQL нижче 11) відповідно до власних потреб роботи зі створюваною базою даних.
3. У збережених процедурах необхідно використати:
   - оголошення вхідних та вихідних параметрів функції, псевдоніми, змінні та константи;
   - присвоювання змінній типу RECORD та команди PERFORM, STRICT або EXECUTE на вибір;
   - керуючі структури (IF/CASE/FOR … на вибір);
   - обов’язково команди обробки помилок EXCEPTION та виведення повідомлень RAISE;
   - обов’язково використати курсор.
4. Оформити звіт про виконання лабораторної роботи, який має містити:

- титульну сторінку (див. додаток А);
- тему, мету та завдання лабораторної роботи;
- короткий перелік та опис використаних при створенні збереженої процедури команд, структур, понять;
- навести знімки екрана з кодом збереженої процедури та отриманим результатом її виконання.

5. Завантажити в канал «БД. Лабораторна робота» своєї команди в Microsoft Teams.

Контрольні питання

1. Функції яких видів представлені у PostgreSQL?
2. Що означає поняття збережена процедура?
3. Які відмінності процедури від функції?
4. Які переваги та застереження використання збережених процедур?
5. Який наступний оператор збереженої процедури буде виконано після виконання оператора EXCEPTION?
6. Що означає поняття «курсор» і для чого застосовується.
7. Поясніть порядок явного оголошення та використання курсору.
8. Наведіть приклад неявного використання курсору.
9. Які типи повідомлень можна застосовувати в операторі RAISE?

Список рекомендованих джерел до теми

1. PostgreSQL. Documentation. [Електронний ресурс]. – Доступний з: https://www.postgresql.org/docs/ .
2. PostgreSQL PL/pgSQL. [Електронний ресурс. – Доступний з: https://www.postgresqltutorial.com/postgresql-plpgsql/ .
3. PostgreSQL PL/pgSQL. PostgreSQL CREATE PROCEDURE. [Електронний ресурс. – Доступний з: https://www.postgresqltutorial.com/postgresql-plpgsql/postgresql-create-procedure/.

ЛАБОРАТОРНА РОБОТА № 10
«Транзакції в СКБД PostgreSQL»

Тема: *«Транзакції в СКБД PostgreSQL».*

Мета роботи: Ознайомлення з використанням транзакцій, їх розробкою та застосуванням, рівнями ізоляцій та механізмом управління одночасним доступом у СКБД PostgreSQL.

Теоретичний матеріал

Транзакції та їх властивості

Транзакції – це фундаментальне поняття в усіх СКБД. Суть транзакції в тому, що *вона об'єднує послідовність дій в одну операцію* – ***«все або нічого»***. Проміжні стани всередині послідовності невидимі іншим транзакціям і якщо щось завадить успішно завершити транзакцію, жоден із результатів цих дій не збережеться в базі даних. Важливими властивостями транзакцій є:

a. Атомарність – гарантує, що транзакція завершується за принципом «все або нічого».
b. Узгодженість – гарантує, що зміни даних, записаних у базу даних, мають бути коректними та відповідати попередньо визначеним правилам.
c. Ізоляція – визначає, як (коли) зміни даних виконані транзакцією будуть видимі для інших транзакцій.
d. Довговічність – гарантує, що результати транзакції, які були зафіксовані, зберігатимуться в базі даних постійно.

Наприклад, розглянемо базу даних банку, в якій міститься інформація про рахунки клієнтів, а також загальні суми по відділеннях банку. Припустимо, що ми хочемо перевести 100 доларів з рахунку Аліси на рахунок Боба. Відповідні SQL-команди можна записати так:

```
UPDATE Accounts
SET Balance = Balance - 100.00
WHERE Name = 'Alice';

UPDATE Branches
SET Balance = Balance -
100.00
WHERE Name = (
    SELECT Branch_name
    FROM Accounts
    WHERE Name = 'Alice');

UPDATE Accounts
SET Balance = Balance + 100.00
WHERE Name = 'Bob';

UPDATE Branches
SET Balance = Balance + 100.00
WHERE Name = (
    SELECT Branch_name
    FROM Accounts
    WHERE Name = 'Bob');
```

Точний зміст команд тут не важливий, важливо лише те, що для виконання цієї доволі простої операції було потрібно декілька окремих дій. Водночас з погляду банку *необхідно, щоб всі ці дії виконалися разом або не виконалися зовсім*. Якщо Боб отримає 100 доларів, але вони не будуть списані з рахунку Аліси, пояснити це помилкою системи, безумовно, не вдасться. І навпаки, Аліса навряд чи буде задоволена, якщо вона переведе гроші, а до Боба вони не дійдуть. Нам потрібна гарантія, що якщо щось завадить виконати операцію до кінця, жодна з дій не залишить сліду в базі даних. І ми отримуємо цю гарантію, об'єднуючи дії в одну *транзакцію*. Отже, транзакція *атомарна,* якщо відповідно до інших транзакцій вона або виконується і фіксується повністю, або не фіксується зовсім.

Нам також потрібна гарантія *довговічності* змін, тобто, що після завершення і підтвердження транзакції системою баз даних, її результати справді зберігаються і не будуть втрачені, навіть якщо незабаром відбудеться аварія. Наприклад, якщо ми списали суму і видали її Бобу, ми повинні виключити можливість того, що сума на його рахунку відновиться, як тільки він вийде за двері банку. Транзакційна база даних гарантує, що *всі зміни записуються* в постійне сховище (наприклад, на диск) *до того*, як транзакція вважатиметься завершеною.

Інша важлива характеристика транзакційних баз даних – *ізоляція*, пов’язана з атомарністю змін: коли одночасно виконується безліч транзакцій, кожна з них не бачить незавершені зміни, зроблені іншими. Наприклад, якщо одна транзакція підраховує баланс по відділеннях, буде неправильно, якщо вона підрахує витрати у відділенні Аліси, але не врахує прихід у відділенні Боба, або навпаки. Тому властивість транзакцій "все або нічого" має визначати не тільки, як зміни зберігаються в базі даних, але і як їх видно в процесі роботи. *Зміни, зроблені відкритою транзакцією, невидимі для інших транзакцій, поки вона не буде завершена, а потім вони відразу стають видні всі*.

Управління транзакціями

У PostgreSQL транзакція визначається набором SQL-команд, оточеним командами `BEGIN` і `COMMIT`. Отож, наша банківська транзакція повинна була б виглядати так:

```
BEGIN;
UPDATE
  Accounts
SET
  Balance = Balance - 100.00
WHERE
  Name = 'Alice';
COMMIT;                                                    .
```

Якщо в процесі виконання транзакції ми вирішимо, що не хочемо фіксувати її зміни (наприклад, тому що виявилося, що баланс Аліси став негативним), ми можемо виконати команду `ROLLBACK` замість `COMMIT`, і всі наші зміни будуть скасовані.

PostgreSQL насправді відпрацьовує кожен SQL-оператор як транзакцію. Якщо ви не вставите команду `BEGIN`, то кожен окремий оператор буде неявно оточений командами `BEGIN` і `COMMIT` (в разі успішного завершення). Групу операторів, оточених командами `BEGIN` і `COMMIT` іноді називають *блоком транзакції*.

Операторами в транзакції можна також керувати на більш детальному рівні, використовуючи *точки збереження*. Вони дозволяють вибірково скасовувати деякі частини транзакції і фіксувати всі інші. Визначивши точку збереження за допомогою `SAVEPOINT`, за необхідності ви можете повернутися до неї за допомогою команди `ROLLBACK TO`.

Усі зміни в базі даних, що відбулися після точки збереження і до моменту відкату, скасовуються, але зміни, зроблені раніше, зберігаються. Коли ви повертаєтеся до точки збереження, вона продовжує існувати, тому ви можете відкочуватися до неї кілька разів. З іншого боку, якщо ви впевнені, що вам не

доведеться відкочуватися до певної точки збереження, її можна видалити, щоб система вивільнила ресурси.

При видаленні або відкаті до точки збереження всі точки збереження визначені після неї, автоматично знищуються.

Усе це відбувається в блоці транзакції, тому в інших сеансах роботи з базою даних цього не видно. Виконані дії стають видимі для інших сеансів, тільки коли ви фіксуєте транзакцію, а скасовані дії не видно взагалі ніколи.

Повернувшись до банківської бази даних, припустимо, що ми списуємо 100 доларів з рахунку Аліси, додаємо їх на рахунок Боба, і раптом виявляється, що гроші потрібно було перевести Wally. В цьому разі ми можемо застосувати точки збереження:

```
BEGIN;
UPDATE Accounts SET Balance = Balance - 100.00
WHERE Name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE Accounts SET Balance = Balance + 100.00
WHERE Name = 'Bob';
- помилкова дія ... ігнорувати її і використовувати
рахунок
Wally
ROLLBACK TO my_savepoint;
UPDATE Accounts SET Balance = Balance + 100.00
WHERE name = 'Wally';
COMMIT;
```

.

Цей приклад, звичайно, трохи надуманий, але він показує, як можна управляти виконанням команд в блоці транзакцій, використовуючи точки збереження. Більше того, `'ROLLBACK TO'` - *єдиний спосіб* повернути контроль над блоком транзакцій, які опинилися в перерваному стані через помилки системи, беручи до уваги можливості повністю скасувати її і почати знову.

У процедурах, що викликаються командою `CALL`, а також в анонімних блоках коду (в команді `DO`) можна завершувати транзакції, виконуючи `COMMIT` і `ROLLBACK`. Після завершення транзакції цими командами нова буде розпочато автоматично, тому окремої команди `START TRANSACTION` немає. (Зауважте, що команди `BEGIN` і `END` в PL / pgSQL мають інший сенс). Наприклад:

```
CREATE PROCEDURE transaction_testl ()
LANGUAGE plpgsql
AS $$ BEGIN
    FOR i IN 0..9 LOOP
        INSERT TNTO testl (a) VALUES (i)
```

```
        IF i% 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
$$;

CALL transaction_testl ();
```

Управління транзакціями можливо тільки у викликах `CALL` або `DO` в коді верхнього рівня, або у вкладених `CALL`, або `DO` без інших проміжних команд. Наприклад, у стеку викликів

```
CALL proc1()→ CALL proc2()→ CALL proc3()
```

друга і третя процедури можуть управляти транзакціями. Але в стеці

```
CALL proc1()→ SELECT func2()→ CALL proc3()
```

остання процедура позбавлена цієї можливості через проміжного SELECT.

Циклам із курсором притаманні деякі особливості. Наприклад:

```
CREATE PROCEDURE transaction_test2 ()
LANGUAGE plpgsql
AS $$
DECLARE
   r RECORD;
BEGIN
   FOR r IN SELECT * FROM test2 ORDER BY x LOOP
      INSERT INTO test1 (a) VALUES (r.x);
      COMMIT;
   END LOOP;
END;
$$;

CALL  transaction test2 ();
```

Зазвичай курсори автоматично закриваються при фіксуванні транзакції. Однак курсор, створюваний всередині циклу подібним чином, автоматично перетворюється в утримуваний курсор першою командою `COMMIT` або `ROLLBACK`. Це означає, що курсор повністю обчислюється при виконанні першої команди

`COMMIT` або `ROLLBACK`, а не для кожного чергового рядка. Водночас він автоматично видаляється після циклу, так що це відбувається майже непомітно для користувача.

Команди управління транзакціями не допускаються в циклах з курсором, якими керують запити, що не лише читають, а й модифікують дані (наприклад, `UPDATE ... RETURNING`).

Транзакція не може завершуватися всередині блоку з обробниками винятків `EXCEPTION`.

Перелік розділів та понять, з якими необхідно ознайомитись для виконання завдання лабораторної роботи (за документацією до PostgreSQL Documentation PostgreSQL15 https://www.postgresql.org/docs/15/mvcc.html):

Розділ 13. Контроль паралельності

1. Вступ
2. Ізоляція транзакцій
   2.1. Рівень ізоляції Read Committed
   2.2. Рівень ізоляції Repeatable Read
   2.3. Рівень ізоляції Serializable

Література та перелік електронних ресурсів необхідних для ознайомлення з теоретичним матеріалом теми наведено у списку рекомендованих джерел.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Розробити кілька транзакцій відповідно до власних потреб роботи зі створюваною базою даних. У транзакціях необхідно продемонструвати:
   a. команди початку та фіксації транзакції;
   b. використання точки збереження SAVEPOINT;
   c. відкат виконання транзакції ROLLBACK TO;
3. На прикладі паралельного виконання двох транзакцій заданого рівня ізоляції пояснити їх роботу з погляду видимості змін під час виконання та після фіксації.
4. Оформити звіт про виконання лабораторної роботи, який має містити:
   a. титульну сторінку (див. додаток А);
   b. тему, мету та завдання лабораторної роботи;
   c. короткий опис роботи створених транзакцій;
   d. навести знімки екрана з кодом транзакцій, видимості змін даних під час їх виконання та після фіксації отриманих результатів.
5. Завантажити файл *.pdf у в канал «БД. Лабораторна робота» своєї команди в Microsoft Teams.

Контрольні питання

1. Поясніть у чому суть поняття транзакції.
2. Назвіть властивості транзакцій.
3. Що означає атомарність транзакції?
4. Що означає узгодженість транзакції?
5. Що означає ізоляція транзакції?
6. Що означає довговічність транзакції?
7. Що таке точка збереження транзакції?
8. Що таке відкат виконання транзакції?
9. Які є рівні ізоляції транзакцій?
10. На основі якого механізму реалізовано рівні ізоляції транзакцій?
11. Що означає поняття «брудне читання»?
12. Що означає поняття «неповторюване читання»?
13. Що означає поняття «фантомне читання»?
14. Що означає поняття «аномалія серіалізації»?

Список рекомендованих джерел до теми

1. PostgreSQLTutorial. PostgreSQL Transaction. [Електронний ресурс]. – Доступний з: *http://www.postgresqltutorial.com/postgresql-transaction/* .
2. PostgreSQL Documentation. PostgreSQL 15. Concurrency Control [Електронний ресурс]. – Доступний з: *https://www.postgresql.org/docs/15/mvcc.html* .

ЛАБОРАТОРНА РОБОТА № 11
«Мова XML та її використання в СКБД PostgreSQL»

Тема: *«Мова XML та її використання в СКБД PostgreSQL».*

Мета роботи: Ознайомлення з конструкціями мови XML та її використанням в СКБД PostgreSQL, зокрема, зі створенням XML даних та XML-документів, перетворенням таблиць реляційної бази в XML-документ.

Теоретичний матеріал

XML або Розширювана мова розмітки - це мова розмітки, яка визначає набір правил для кодування документів у форматі, який зручний для читання і створення документів і людиною, і машиною. Мова називається розширюваною, бо нею не фіксується розмітка (теги), яка використовується в документах: розробник може створювати розмітку відповідно до потреб конкретної предметної області, аби лише вона відповідала синтаксичним правилам мови. Була створена W3C [1] для подолання обмежень HTML – мови розмітки гіпертексту, який є основою для всіх вебсторінок. Проблема з HTML полягає в тому, що його теги говорять браузеру, *як відобразити* цю інформацію, вони не вказують браузеру, *що це за інформація.*

Як і HTML, XML заснована на SGML – стандартній узагальненій мові розмітки. Хоча SGML використовується у видавничій галузі впродовж десятиліть (ще з 60-х рр.), її складність сприйняття злякала багатьох, що хотіли використати її (SGML інколи жартома розшифровують як «Sounds great, maybe later» – звучить чудово, можливо, згодом). І, що важливо, XML був розроблений з урахуванням інтернету.

Отже, XML – це програмно- та апаратно-незалежний інструмент для зберігання та транспортування даних.

- *XML розшифровується як розширювана мова розмітки.*
- *XML – мова розмітки, схожа на HTML.*
- *XML був розроблений для зберігання та транспортування даних.*
- *XML був розроблений так, щоб він був самоописовим.*
- *XML – це рекомендація W3C.*

Створювані файли або екземпляри документа, складаються з тегів і тексту, причому тегам у XML можна присвоїти деяке значення, що допоможе правильно

[1] (World Wide Web Consortium – це міжнародна спільнота, де організації-члени, штатний персонал та громадськість працюють разом над розробкою веб стандартів під керівництвом вебвинахідника та директора Тіма Бернерса-Лі).

розуміти документ при читанні або обробляти його в електронному вигляді. Наприклад, можна легко отримати з документа поштовий індекс, відомий як <postal-code> елемент, розмістивши вміст, оточений тегами <postal-code> та </postal-code>.

Чим більше описових тегів, тим більше частин документа можна ідентифікувати. Одна з переваг розмітки полягає в тому, що в разі втрати комп'ютерної системи, роздруковані дані однаково залишаються читабельними завдяки тегам.

У XML можна створювати свої власні теги, що дозволяє точно представляти фрагменти даних. Документи можна не просто розділяти на абзаци і заголовки, а й виділяти будь-які фрагменти всередині документа. Щоб це було ефективно, потрібно визначити кінцевий перелік своїх елементів і дотримуватися його.

Елементи можна визначати в Описі типу документа (Document Type Definition - DTD), що називається DTD схемою документа.

Існують три терміни, що використовуються для опису частин документа XML: *теги*, *елементи* та *атрибути*. Ось зразок документа, який ілюструє ці терміни:

```
<address>
    <name>
        <title>Mrs.</title>
   <first-name>Mary</first-name>
     <last-name>McGoon</last-name>
    </name>
    <street>
        1401 Main Street
    </street>
    <city state="NC">Anytown</city>
    <postal-code>
        34829
    </postal-code>
</address>
```

.

- Тег – це текст між лівою кутовою дужкою (<) та правою кутовою дужкою (>). Існують початкові теги (наприклад <name>) та кінцеві теги (наприклад </name>).
- Елементом є початковий тег, кінцевий тег і все, що знаходиться між ними. У поданому прикладі <name> елемент містить три дочірні елементи: <title>, <first-name> і <last-name>.
- Атрибут – це пара ім'я="значення" всередині *початкового* тегу елемента. У цьому прикладі state є атрибут <city> елемента.

- Основними поняттями XML є «правильно сформований» (well formed) та коректний (valid) документ.

Правильно сформований документ відповідає всім синтаксичним правилам XML. Документ, що не є правильно сформований, не може називатися XML-документом. Сумісний синтаксичний аналізатор[1] (conforming parser) не повинен обробляти такі документи. Правильно сформований XML-документ повинен мати:

- лише один кореневий елемент;
- непорожні елементи розмічено початковим та кінцевим тегами;
- порожні елементи можуть помічатися «закритим» тегом, наприклад <IAmEmpty/>. Така пара еквівалентна <IAmEmpty></IAmEmpty>;
- один елемент не може мати декілька атрибутів з однаковим іменем. Значення атрибутів знаходяться або в одинарних ('), або у подвійних (") лапках;
- теги можуть бути вкладені, але не можуть перекриватися. Кожен некореневий елемент мусить повністю знаходитись в іншому елементі;
- документ має складатися тільки з правильно закодованих дозволених символів множини Юнікоду. Єдиними кодуваннями, які обов'язково має розуміти XML-процесор, є UTF-16 та UTF-8. Фактичне та задекларованe кодування (character encoding) документа мають збігатися. Воно може бути задекларовано ззовні у заголовку «Content-Type» при передачі за протоколом HTTP, або в самому документі використанням явної розмітки на початку документа. В разі відсутності інформації про кодування, документ має бути в кодуванні UTF-8 (або його підмножині ASCII).

Документ називається **коректним**, якщо він є правильно сформований, містить посилання на граматичні правила та повністю відповідає обмеженням, вказаним у цих правилах (DTD або XML Schema).

Переваги застосування XML:

- *XML спрощує обмін даними*.

Використовуючи XML, різні організації або навіть різні частини однієї організації створюють єдину утиліту, яка перетворює їх внутрішні формати даних у XML і навпаки.

- *XML дозволяє створювати смарткод (розумний код)*.

Оскільки документи XML можуть бути структуровані так, щоб ідентифікувати кожну важливу інформацію (а також взаємозв'язки між частинами),

[1] Синтаксичним аналізатором називається програма або компонент, що читає XML-документ, проводить синтаксичний аналіз, та відтворює його структуру. Якщо синтаксичний аналізатор перевіряє документ на валідність, то такий аналізатор називають валідатором.

можна написати код, який може обробляти ці XML-документи без втручання людини.

- *XML дозволяє інтелектуальний пошук*.

Незважаючи на те, що пошукові системи постійно вдосконалювалися протягом багатьох років, все ж доволі часто трапляються помилкові результати пошуку. Якщо ви шукаєте HTML-сторінки на когось з іменем «Chip», ви також можете знайти сторінки з шоколадними чіпсами, комп'ютерними чіпами, деревними трісками та безліччю інших марних збігів. Пошук XML-документів за <firstname> елементами, які містять текст Chip, дасть вам набагато кращий набір.

Перелік розділів та понять, з якими необхідно ознайомитися для виконання завдання лабораторної роботи (за документацією PostgreSQL https://www.postgresql.org/docs/15/functions-xml.html):

1. XML-функції PostgreSQL
2. Створення XML-контенту.
3. Умови з XML.
4. Обробка XML.
5. Відображення таблиць у XML .

Література та перелік електронних ресурсів, необхідних для ознайомлення з теоретичним матеріалом теми, наведено у списку рекомендованих джерел.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Створити XML-документ (схеми DTD чи XML створювати не потрібно), який містить інформацію з 3-х чи 4-х зв'язаних таблиць реляційної бази даних.

   Для створення документа не використовувати функцій PostgreSQL відображення таблиць table_to_xml чи всієї бази даних database_to_xml в XML. Натомість, використати SELECT з функціями створення XMLконтенту: xmlelement, xmlroot, xmlforest, xmlagg та ін.
3. Використати on-line **XML FORMATTER** для форматування та *синтаксичний валідатор* для перевірки чи правильно сформований створений XML-документ.
4. Оформити звіт про виконання лабораторної роботи, який має містити:
   a. титульну сторінку (див. додаток А);
   b. тему, мету та завдання лабораторної роботи;
   c. короткий перелік та опис створення XML документа;
   d. навести знімки екрану з кодом XML документа та результатами форматування та перевірки його синтаксичним валідатором.
5. Завантажити в канал "БД. Лабораторна робота" своєї команди в Microsoft Teams.

Контрольні питання

1. Що означає абревіатура XML? 2. Які три терміни використовуються для опису вмісту документа XML?
2. Що означає поняття «правильно сформований» (well formed) документ?
3. Що означає поняття «коректний» (valid) документ?
4. Для чого в PostgreSQL при створенні XML-контенту використовується функція xmlagg?
5. Для чого в PostgreSQL при створенні XML-контенту використовується функція xmlforest?
6. Що таке простір імен XML і навіщо в його оголошенні вказується URI?

Список рекомендованих джерел до теми

1. Fawcett J. Beginning XML (5-edition) /Joe Fawcett, Danny Ayers, Liam R. E. Quin. – Wiley. – 2012. – 864 p.
2. W3schools: XML Tutorial. [Електронний ресурс]. – Доступний з: *https://www.w3schools.com/xml/default.asp*.
3. PostgreSQL: Documentation. PostgreSQL 15. XML-функції PostgreSQL [Електронний ресурс]. – Доступний з: *https://www.postgresql.org/docs/15/functions-xml.html*.
4. XML Formatter. [Електронний ресурс]. – Доступний з: *https://www.liquidtechnologies.com/online-xml-formatter*.
5. Free Online XML Validator (Well formed) . [Електронний ресурс]. – Доступний з: *https://www.liquid-technologies.com/online-xml-validator*.

ЛАБОРАТОРНА РОБОТА № 12
«DTD-схема XML-документа»

Тема: *«DTD-схема XML-документа».*

Мета роботи: Практичне ознайомлення з конструкціями DTD-схеми XML документа, її створенням і використанням.

Теоретичний матеріал

Основними поняттями XML є правильно сформований (well formed) та коректний (valid) документ:

- **правильно сформований** документ відповідає всім синтаксичним правилам XML. Документ, що не є правильно сформований, не може називатись XML-документом;
- документ називається **коректним** *(valid)*, якщо він є правильно сформований і містить посилання на граматичні правила та повністю відповідає обмеженням, вказаним у цих правилах описаних у схемі - DTD, XML Schema або RELAX NG.

Є кілька способів визначення елементів, які використовуються для представлення даних у XML-документі.

Першим було використання схеми **DTD** (Document Type Definition), яка дуже схожа на DTD для метамови SGML. DTD визначає елементи, які можуть з'являтися в XML-документі, порядок їх відображення, як вони можуть бути вкладені один в одного, та інші основні деталі структури документа XML.

Інший метод полягає у використанні **XML Schema**. Крім можливостей DTD, вона ще визначає **типи** даних і **більш складні** правила, ніж може DTD.

DTD (Document Type Definition)

Розглянемо зразок XML-документа, що описує поштову адресу США:

```
<address>
<name>
<title>Mrs.</title>
<first-name>Mary</first-name>
<last-name>McGoon</last-name>
</name>
<street>1401 Main Street</street>
<city>Anytown</city>
<state>NC</state>
<postal-code>34829</postal-code>
</address>
```

.

DTD-схема дозволяє задати структуру XML-документа. Наступний приклад DTD описує структуру XML-документа поштової адреси:

```
<!-- address.dtd -->
<!ELEMENT address (name, street, city, state, postal-code)>
<!ELEMENT name (title?, first-name, last-name)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT last-name (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT postal-code (#PCDATA)>
```

.

У даній DTD визначаються всі елементи, використані у XML-документі поштової адреси. А саме: наступні три основні речі:

- <*address*> елемент містить <name>, <street>, <city>, <state> і <postalcode>. Усі ці елементи *повинні* бути присутні і вказуватися *у зазначеному порядку*.
- <*name*> елемент містить необов'язковий <title> елемент (*знак питання* означає, що заголовок є *необов'язковим*), а потім <first-name> і <lastname> елементи.
- Усі інші елементи містять текст. #PCDATA означає, що парсер здійснюватиме аналіз даних символів - ви не можете включити інший елемент до цих елементів, вони мають містити лише текст. Вміст елементів, визначених як #CDATA парсером не аналізується.

Хоча DTD-схема доволі проста, вона дає зрозуміти, які поєднання елементів є допустимими. Наприклад, адресний документ, який містить <postalcode> елемент перед <state> елементом *не є допустимим*, і жоден документ, який не має <lastname> елемента, теж.

Порожні елементи оголошуються за допомогою ключового слова EMPTY:

`<!ELEMENT emp EMPTY>` визначає в XML-документі `<emp />` .

Елементи, оголошені за допомогою ключового слова ANY, можуть містити будь-яку комбінацію аналізованих даних:

`<!ELEMENT note ANY>`.

Також потрібно зауважити, що синтаксис DTD *відрізняється* від звичайного синтаксису XML (на відміну від XML Schema, яка сама по собі є XML). Незважаючи на різний синтаксис для DTD, її однаково можна помістити в XML-документ.

Якщо DTD оголошено всередині файлу XML, його потрібно загорнути всередину визначення <! DOCTYPE>:

```
<!DOCTYPE НазваКореневогоЕлементаXML […]>
```

.

Якщо DTD оголошено в зовнішньому файлі, визначення <! DOCTYPE> повинно містити посилання на файл DTD:

```
<!DOCTYPE note SYSTEM "note.dtd">
```

.

Отже, нашXML-документ разом з DTD-схемою матиме вигляд

```
<!DOCTYPE address [
<!ELEMENT address (name, street, city, state, postal-code)>
<!ELEMENT name (title?, first-name, last-name)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT last-name (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT postal-code (#PCDATA)>]>
<address>
<name>
<title>Mrs.</title>
<first-name>Mary</first-name>
<last-name>McGoon</last-name>
</name>
<street>1401 Main Street</street>
<city>Anytown</city>
<state>NC</state>
<postal-code>34829</postal-code>
</address>
```

.

Символи в DTD

Інує кілька символів, які використовуються в DTD, щоб вказати, як часто (або чи) щось може з'являтися в XML-документі. Наприклад:

```
<!ELEMENT address (name, city, state)>
```

<address> елемент *<u>повинен</u>* містити *<u>по одному</u>* <name>, <city> і <state> елементу в цьому ж порядку. Всі елементи необхідні. Кома розділяє список елементів.

```
<!ELEMENT name (title?, first-name, lastname)>
```

.

Знак `?` означає, що <name> елемент містить *<u>необов'язковий</u>* <title> елемент, за яким слідують обов'язкові <first-name> та <last-name> елементи. Знак

питання вказує, що необов'язковий елемент може з'явитися один раз або зовсім не з'явитися.

```
<!ELEMENT addressbook (address+)>
```

<addressbook> елемент містить один або кілька <address>елементів. Ви можете мати стільки <address> елементів, скільки вам потрібно, але має бути принаймні один.

```
<!ELEMENT private-addresses (address*)>
```

<private-addresses> елемент містить нуль або більше <address>елементів;

* вказує, що елемент може з'являтися будь-яку кількість разів, включаючи нуль.

<!ELEMENT name (title?, first-name, (middle-initial | middle-name)?, last-name)>

<name> елемент містить необов'язковий <title>елемент, за яким слідує <firstname> елемент, можливо, з подальшим або з <middle-initial> або <middle-name> елементом, за яким слідує <last-name> елемент. Тобто, обидва <middleinitial> і <middle-name> *необов'язкові*, і ви можете мати лише одне з двох.

Вертикальна риска позначає перелік варіантів; можна вибрати лише один елемент зі списку. Також цей приклад використовує дужки для групування певних елементів, а також знак питання до групи.

```
<!ELEMENT name ((title?, first-name, last-name) | (surname,
mothers-name))>
```

<name> елемент може містити одну з двох послідовностей: за бажанням <title>, за яким <first-name> і <last-name> **або** <surname> і <mothers-name>.

Визначення атрибутів

Для елементів XML-документа можна визначити атрибути XML. Використовуючи DTD, також можна:

- визначити, які атрибути потрібні;
- визначити значення за замовчуванням для атрибутів;
- перерахувати всі дійсні значення для даного атрибута.

Декларація атрибута має такий синтаксис:

```
<!ATTLIST element-name attribute-name attribute-type attribute-value>
```

Тип атрибута може бути одним із наступних:

CDATA	Значення – символьні дані
(en1|en2|..)	Значення повинно бути одне зі списку
ID	Значення – це унікальний ID
IDREF	Значення – ID іншого елемента
IDREFS	Значення – це список ID інших елементів

NMTOKEN	Значення – допустиме ім’я XML
NMTOKENS	Значення – це список допустимих імен XML
ENTITY	Значення – це сутність
ENTITIES	Значення – це список сутностей
NOTATION	Значення – це назва позначення
xml:	Значення – це заздалегідь задане значення xml.

Атрибут-значення може бути одним із наступних:

Значення	**Значення атрибута за замовчуванням**
#REQUIRED	атрибут обов’язковий;
#IMPLIED	атрибут необов’язковий;
#FIXED value	значення атрибута фіксоване.

Припустимо, потрібно змінити DTD, щоб зробити state атрибут <city> елемента. Ось як це зробити:

```
1. <!ELEMENT city (#PCDATA)>
2. <!ATTLIST city state CDATA #REQUIRED
```

.

Це визначає <city> елемент, як і раніше, але далі використовується ATTLIST декларація для переліку атрибутів елемента. Ім’я city всередині списку атрибутів повідомляє аналізатору, що ці атрибути визначені для <city> елемента. Назва state – це ім’я атрибута, ключові слова CDATA та #REQUIRED скажуть аналізатору, що state атрибут містить текст і він необхідний (якщо це необов’язково – CDATA #IMPLIED, якщо фіксоване значення – CDATA #FIXED "value").

Щоб визначити кілька атрибутів для елемента, застосовують ATTLIST так:

```
1. <!ELEMENT city (#PCDATA)>
2. <!ATTLIST city state CDATA #REQUIRED
3. postal-code CDATA #REQUIRED>
```

.

Цей приклад визначає stte і postal-code атрибути <city> елемента.

Нарешті, DTD дозволяє визначити значення за замовчуванням для атрибутів та перерахувати всі дійсні значення для атрибута:

```
1.<!ELEMENT city (#PCDATA)>
2.<!ATTLIST city state CDATA (AZ|CA|NV|OR|UT|WA) "CA">
```

.

Приклад вказує, що підтримуються лише адреси штатів Арізона (AZ), Каліфорнія (CA), Невада (NV), Орегон (OR), Юта (UT) та Вашингтон (WA), і за замовчуванням штат Каліфорнія. Отже, можна зробити дуже обмежену форму перевірки даних.

У XML немає правил щодо того, коли використовувати атрибути та дочірні елементи. Дані можуть зберігатися в дочірніх елементах або в атрибутах. Наприклад:

```
<person gender="female">
   <firstname>Anna</firstname>
   <lastname>Smith</lastname>
</person>

<person>
   < gender >female</ gender >
   <firstname>Anna</firstname>
   <lastname>Smith</lastname>
</person>
```

.

У першому прикладі стать є атрибутом. В останньому – це дочірній елемент елемента person. Деякі проблеми з атрибутами:

- атрибути не можуть містити декілька значень (дочірні елементи можуть);
- атрибути не легко розширюються (для майбутніх змін);
- атрибути не можуть описувати структури (дочірні елементи можуть);
- атрибути утруднюють зміни програмного коду;
- значення атрибутів непросто перевірити на DTD.

Якщо ви використовуєте атрибути як контейнери для даних, то отримуєте документи, які складно читати та підтримувати. Краще використовувати **елементи** для опису даних. Атрибути використовуйте лише для надання інформації, яка не стосується даних. Тобто метадані (дані про дані) повинні зберігатися як атрибути, а самі дані повинні зберігатися як елементи.

```
<messages>
    <note id="p501">
    <to>Tove</to>
     </note>
    <note id="p502">
       <to>Jani</to>
       <from>Tove</from>
    </note>
</messages>
```

id у цих прикладах – це лише лічильник або унікальний **id** для ідентифікації різних приміток у файлі XML, а не частина даних примітки.

Сутності

Сутності використовуються для визначення ярликів якихось характеристик. Вони можуть бути внутрішніми або зовнішніми. Формат оголошення внутрішньої сутності

```
<!ENTITY entity-name "entity-value">
```
.

Наприклад,

```
<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright W3Schools.">
```
.

У XML це відображатиметься як:

```
<author>&writer;&copyright;</author>
```
.

Формат оголошення зовнішньої сутності

```
<!ENTITY entity-name SYSTEM "URI/URL">
```
.

Наприклад,

```
<!ENTITY writer SYSTEM
"https://www.w3schools.com/entities.dtd">
<!ENTITY copyright SYSTEM
"https://www.w3schools.com/entities.dtd">
```
.

У XML це відображатиметься як:

```
<author>&writer;&copyright;</author>
```

Сутність складається з трьох частин: амперсанд (&), ім’я сутності та крапка з комою (;). Сутності розгортаються, коли документ аналізується аналізатором XML.

Наступні сутності попередньо визначені в XML:

Сутність	Символ
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	'

Деякі зауваження

Визначаючи схему документа слід пам'ятати про гнучкість щодо схеми документів XML. Розглянемо назву зразка та тип документа. На початку було чітко вказано, що це маються на увазі поштові адреси США. Якщо потрібно DTD або схему, яка визначає правила для інших типів адрес, то доведеться додати до неї набагато більшу складність. Так, <state> елемент може мати сенс в Австралії, але не у Великобританії. Канадська адреса може оброблятися зразком DTD у визначеннях типу документа, але додавання <province> елемента буде кращою ідеєю.

Отже, перед тим як визначити структуру документа XML, слід так само заздалегідь продумати свій DTD або схему, як і при розробці схеми бази даних або структури даних вашої програми. Чим більше майбутніх вимог можна передбачити, тим легше і дешевше буде їх виконати пізніше.

Також, на даний час в PostgreSQL тип xml не перевіряє значення що вводяться за схемою DTD, навіть якщо в них присутні посилання на DTD. Немає і вбудованої підтримки інших різновидів схем, наприклад XML Schema.

Перелік понять, з якими необхідно ознайомитись для виконання завдання лабораторної роботи:

1. Основні поняття XML (коректність та валідність)
2. Способи визначення структури та елементів в XML документі
3. Визначення структури, елементів та атрибутів у DTD схемі
4. Символи DTD схеми
5. Визначення атрибутів DTD схеми
   5.1 Типи атрибутів
   5.2 Значення атрибутів
6. Сутності

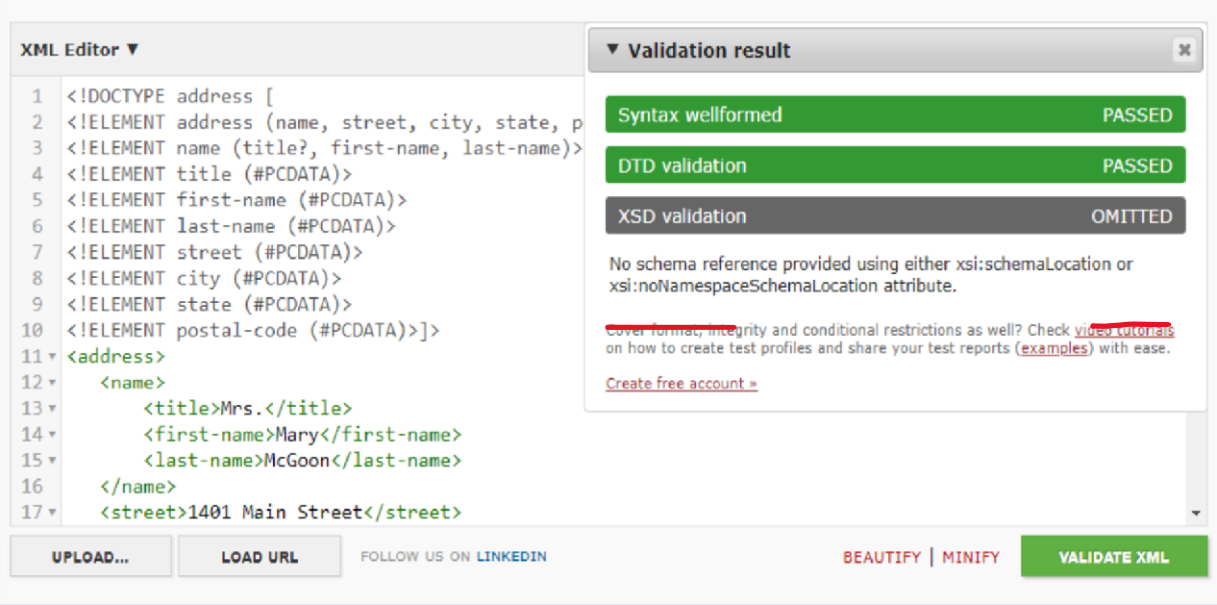
Література та перелік електронних ресурсів необхідних для ознайомлення з теоретичним матеріалом теми наведено у списку рекомендованих джерел.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Власноруч створити DTD схему для XML документа, який ви створили в попередній лабораторній роботі №10.
3. Використати on-line валідатор для перевірки валідності створеного XML документа разом з DTD схемою.
4. Оформити звіт про виконання лабораторної роботи, який має містити:
   - титульну сторінку (див. додаток А);
   - тему, мету та завдання лабораторної роботи;

- опис створення DTD схеми XML документа;
- навести знімки екрану з кодом DTD схеми та XML документа і результатом перевірки його валідатором наприклад TRUUGO (за деякий час може вимагати *безкоштовну* реєстрацію).

5. Завантажити в канал “БД. Лабораторна робота” своєї команди в Microsoft Teams.



Приклад валідації XML документа з DTD схемою:

Контрольні питання

1. Що дозволяє визначити схема XML документа?
2. Як описується складний елемент у DTD схемі?
3. Як описується простий елемент текстового типу в DTD схемі?
4. Які спеціальні символи використовуються в DTD схемі для вказання обмежень для елементів?
5. Як описуються атрибути елемента?
6. Яка є рекомендація стосовно використання атрибутів у XML документі?
7. Що означає в DTD схемі поняття “сутність”?

Список рекомендованих джерел до теми

1. Fawcett J. Beginning XML (5-edition) /Joe Fawcett, Danny Ayers, Liam R. E. Quin. – Wiley. – 2012. – 864 p.
2. W3schools: XML Tutorial. XML DTD. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/xml/xml_dtd.asp .
3. TRUUGO XML Validator . [Електронний ресурс]. – Доступний з:

https://www.truugo.com/xml_validator/ .

ЛАБОРАТОРНА РОБОТА № 13
«XML-схема XML-документа»

Тема: *«XML схема XML документа».*

Мета роботи: Ознайомлення з конструкціями XML Schema XML документа. Отримати практичні навички її створення і використання.

Теоретичний матеріал

Серед способів визначення елементів, які використовуються для представлення даних в XML документі найбільш поширений, на сьогодні, метод полягає у використанні XML Schema або XML Schema Definition (XSD). Схема може визначати всі структури документів, які можна помістити в DTD, а також може визначити типи даних і більш складні правила, ніж може DTD. Типи даних є контейнерами для різних видів вмісту, від тексту до цілих чисел і до дат. W3C розробив специфікацію схеми XML через кілька років після оригінальної специфікації XML.

Мова XML Schema визначена W3C у трьох частинах:

- Підручник, який дає ознайомлення з документами XML-схеми і орієнтований на швидке розуміння способів створення схем розташований за адресою https://www.w3.org/TR/xmlschema-0/ ;
- Стандарт для структур документів, який ілюструє, як визначити структуру XML-документів, розміщений за адресою https://www.w3.org/TR/2004/REC-xmlschema-1-20041028/structures.html ;
- Стандарт для типів даних, який визначає загальні типи даних та правила для створення нових типів розташований за адресою https://www.w3.org/TR/2004/RECxmlschema-220041028/datatypes.html .

За допомогою XML-схем є більше можливостей визначити, як виглядають коректні документи XML. Вони мають ряд переваг перед DTD:

• У схемах XML використовується синтаксис XML. Тобто, схема XML - це документ XML. Це означає, що можна обробити схему, як і будьякий інший документ. Наприклад, можна використати XSLT, який перетворює схему XML у вебформу в комплекті з автоматично сформованим кодом JavaScript, який перевіряє дані під час введення. І, звичайно, підтримка просторів імен.

• XML-схеми підтримують типи даних. Незважаючи на те, що DTD підтримують типи даних, але вони були розроблені з погляду публікації. XML-схеми підтримують крім типів даних з DTD (такі як ідентифікатори та

посилання на ідентифікатори), також цілі числа, числа з плаваючою комою, дати, час, рядки, URL-адреси та інші типи даних, корисні для обробки та перевірки даних. Це дозволяє

- простіше описати зміст документа
- простіше визначити обмеження щодо даних
- простіше перевірити правильність даних
- легше конвертувати дані між різними типами даних

• XML-схеми розширюються. Окрім типів даних, визначених у специфікації схеми XML, також можна створити свої власні типи та отримати нові типи даних на основі інших типів даних, використовувати повторно свою схему в інших схемах і посилатися на кілька схем в одному документі.

• XML-схеми підтримують встановлення обмежень даних. Наприклад, за допомогою XML-схем можна визначити, що значення будь-якого <state> атрибута не може бути довше двох символів або що значення будь-якого <postal-code> елемента повинно відповідати звичайному виразу [0-9]{5}([0-9]{4})?. З DTD не можна робити жодної з цих речей.

• Присутня підтримка DOM (Document Object Model).

Приклад XML-схеми

Щоб побудувати схему можна просто прослідкувати структуру XML документу та визначити кожний елемент, що зустрівся в ній. Цей метод є дуже простим, але є важливим для розуміння XML Schema.

Ось XML схема, яка відповідає DTD схемі попередньої лабораторної роботи. Але додає два обмеження: значення <state> елемента повинно мати лише два символа, а значення <postal-code> елемента повинно відповідати регулярному виразу [0-9]{5}(-[09]{4})?.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="address">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="name">
           <xsd:complexType>
             <xsd:sequence>
               <xsd:element name="title" type="xsd:string"/>
               <xsd:element name="first-name" type="xsd:string"/>
               <xsd:element name="last-name" type="xsd:string"/>
              </xsd:sequence>
           </xsd:complexType>
         </xsd:element>
```

```
         <xsd:element name="street" type="xsd:string" />
         <xsd:element name="city" type="xsd:string" />
         <xsd:element name="state">
            <xsd:simpleType>
               <xsd:restriction base="xsd:string">
                  <xsd:length value="2"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:element>
         <xsd:element name="postal-code">
             <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                     <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
                </xsd:restriction>
             </xsd:simpleType>
          </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Проте, побудована за допомогою цього методу схема складного документу може бути складною для читання та підтримки.

Альтернативний метод побудови, який застосовано в наведеному нижче прикладі, заснований на початковому визначенні всіх простих елементів та атрибутів і наступному посиланні на них за допомогою атрибута ref.

Хоча схема набагато довша, ніж DTD, вона краще визначає, як виглядає коректний документ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="title"            type="xsd:string"/>
<xsd:element name="first-name" type="xsd:string"/>
<xsd:element name="last-name"  type="xsd:string"/>
<xsd:element name="street"          type="xsd:string"/>
<xsd:element name="city"             type="xsd:string"/>
    <xsd:element name="address">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="name"/>
<xsd:element ref="street"/>
<xsd:element ref="city"/>
<xsd:element ref="state"/>
<xsd:element ref="postal-code"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

```
<xsd:element name="name">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="title" minOccurs="0"/>
<xsd:element ref="first-name"/>
<xsd:element ref="last-name"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="state">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:length value="2"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="postal-code">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:schema>
```

Визначення елементів у схемах

Схема XML визначає кілька елементів XML з <xsd:element> елементом. Перші два елементи, <address> та <name>, складаються з інших елементів. Елемент <xsd:sequence> визначає послідовність елементів, які містяться в кожному з них:

```
<xsd:element name="address">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="name"/>
<xsd:element ref="street"/>
<xsd:element ref="city"/>
<xsd:element ref="state"/>
<xsd:element ref="postal-code"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Як і у версії DTD, приклад схеми XML визначає, що <address> містить <name>, <street>, <city>, <state> і <postal-code> елемент у такому ж порядку. Але схема фактично визначає новий тип даних з <xsd:complexType> елементом. Це *складний* тип, оскільки містить інші елементи.

Більшість елементів містять текст. Тому просто оголошуємо новий елемент і надамо йому тип даних xsd:string. Це *простий* тип. Він не може містити інших елементів або атрибутів:

```
<xsd:element name="title"      type="xsd:string"/>
<xsd:element name="first-Name" type="xsd:string"/>
<xsd:element name="last-Name"  type="xsd:string"/>
<xsd:element name="street"     type="xsd:string"/>
<xsd:element name="city"       type="xsd:string"/>
```

Найпоширеніші вбудовані типи елементів є string, decimal, integer, boolean, date, time.

Визначення вмісту елементів у схемах

Приклад схеми визначає обмеження для вмісту двох елементів: Вміст <state> елемента повинен бути довжиною два символи, а вміст <postalcode> елемента повинен відповідати регулярному виразу [0-9]{5}(-[0-9]{4})?:

```
<xsd:element name="state">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="2"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="postal-code">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Для елементів <state> та <postal-code> схема визначає нові типи даних з обмеженнями. У першому випадку використовується <xsd:length> елемент, а в другому використовується <xsd:pattern> елемент для визначення регулярного виразу, якому цей елемент повинен відповідати. ***Визначення атрибута***

Синтаксис для визначення атрибута:

<xsd:attribute name="**xxx**" type="**yyy**"/> де **xxx** - це ім’я атрибута, а **yyy** вказує тип даних атрибута.

XML-схема має багато вбудованих типів даних атрибута. Найпоширеніші типи - string, decimal, integer, boolean, date, time.

Для елемента XML з атрибутом: <city country="US"/> відповідне визначення атрибута в схемі:

```
<xsd:element name="city">
     <xsd:complexType>
        <xsd:attribute name="country" type="xsd:string"/>
     </xsd:complexType>
</xsd:element>
```

Але для елемента XML з атрибутом:

<city country="US">**Anytown**</city> відповідне

визначення в схемі:

```
<xsd:element name="city"  >
   <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:string">
          <xsd:attribute name="country" type="xsd:string" />
        </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
</xsd:element>
```

Значення за замовчуванням і фіксовані значення для атрибутів

Атрибути можуть мати значення за замовчуванням або вказане фіксоване значення. Значення за замовчуванням автоматично призначається атрибуту, коли інше значення не вказано.

У наступному прикладі значенням за замовчуванням є "EN":

```
<xsd:attribute name="lang" type="xsd:string" default="EN"/>
```.

Фіксоване значення також автоматично призначається атрибуту, і ви не можете вказати інше значення.

У наступному прикладі фіксованим значенням є "UA":

```
<xsd:attribute name="lang" type="xsd:string" fixed="UA"/>
```.

Необов'язкові та необхідні атрибути

Атрибути за замовчуванням необов'язкові. Щоб вказати, що атрибут необхідний, використовується атрибут "required":

```
<xsd:attribute name="lang" type="xsd:string" use="required"/>
```.

Обмеження щодо вмісту

Якщо для елемента XML або атрибута визначений тип даних, він встановлює обмеження на вміст елемента або атрибута.

Якщо елемент XML має тип "xsd:date" і містить рядок типу "Hello World", елемент вважатиметься не валідним (тобто виникне помилка при валідації).

За допомогою XML-схем також можна додавати власні обмеження до своїх елементів та атрибутів XML. Ці обмеження ще називаються гранями або границями (facets) .

Обмеження для Datatypes

enumeration	Визначає список допустимих значень
fractionDigits	Задає максимальну кількість знаків після коми. Має дорівнювати або більше нуля
length	Визначає точне число символів або елементів списку. Має дорівнювати або більше нуля
maxExclusive	Визначає верхню межу для числових значень (значення повинно бути менше цього значення)
maxInclusive	Визначає верхню межу для числових значень (значення повинно бути менше або дорівнює цьому значенню)
maxLength	Задає максимальне число символів або елементів списку. Має дорівнювати або більше нуля
minExclusive	Задає нижні межі для числових значень (значення повинно бути більше, ніж це значення)
minInclusive	Задає нижні межі для числових значень (значення повинно бути більше або дорівнює цьому значенню)
minLength	Задає мінімальну кількість символів або елементів списку. Має дорівнювати або більше нуля
pattern	Визначає точну послідовність символів, які є прийнятними
totalDigits	Визначає точне число цифр. Повинно бути більше нуля
whiteSpace	Визначає, як обробляються пропуски (наступний рядок, табуляція, пропуски і повернення на початок рядка)

Перед тим, як визначити структуру документа XML, слід заздалегідь продумати схему, як і коли б ви розробляли схему бази даних або структуру даних вашої програми. Чим більше майбутніх вимог можна передбачити, тим легше і

дешевше буде їх виконати пізніше. На даний час в PostgreSQL немає вбудованої підтримки XML Schema.

Перелік розділів та понять, з якими необхідно ознайомитись для виконання завдання лабораторної роботи:

1. Основні поняття XML (правильно сформований та коректність)
2. Два способи визначення структури та елементів в XML документі.
3. Визначення структури та елементів у XML Schema.
4. Прості та складні типи елементів.
5. Визначення атрибутів.
6. Обмеження (грані) XML Schema.

Література та перелік електронних ресурсів необхідних для ознайомлення з теоретичним матеріалом теми наведено у списку рекомендованих джерел.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Використовуючи Альтернативний метод створити XML Schema XML документа, який містить інформацію та структуру двох-трьох зв'язаних таблиць реляційної бази даних. Обов'язково використати обмеження (restriction або extension) та границі (facets)/ .
3. Використати on-line валідатор для перевірки валідності створеного XML документа разом з XML Schema.
4. Оформити звіт про виконання лабораторної роботи, який має містити:
   - титульну сторінку (див. додаток А);
   - тему, мету та завдання лабораторної роботи;
   - короткий перелік та опис створення XML Schema XML документа;
   - навести знімки екрану з кодом XML Schema та XML документа та результатом перевірки його валідатором.
5. Завантажити в канал "БД. Лабораторна робота" своєї команди в Microsoft Teams.

Контрольні питання

1. Які є два способи визначення структури та елементів в XML документі?
2. Як визначаються прості та складні типи елементів?
3. Які особливості визначення атрибутів?
4. Які є обмеження (facets) для елемента XML або атрибута?

Список рекомендованих джерел до теми

1. Fawcett J. Beginning XML (5-edition) /Joe Fawcett, Danny Ayers, Liam R. E. Quin. – Wiley. – 2012. – 864 p.
2. W3C Recommendation XML Schema Part 0: Primer Second Edition. [Електронний ресурс]. – Доступний з: https://www.w3.org/TR/xmlschema-0/ .
3. W3schools: XSD Introduction. XML Schema Tutorial. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/xml/schema_intro.asp.

ЛАБОРАТОРНА РОБОТА № 14
«Мова виразів XPath»

Тема: *«Мова виразів XPath».*

Мета роботи: Вивчення мови виразів XPath та створення виразів XPath для ефективнішого використання XSLT та XQuery.

Теоретичний матеріал

Мова XPath визначена W3C наступними документами:

- Документ XML Path Language (XPath) 3.1 W3C Recommendation 21 March 2017, що визначає специфікацію мови XPath, яка дозволяє обробляти значення, що відповідають моделі даних, визначеній у XQuery and XPath Data Model (XDM) 3.1.
- Документ XQuery and XPath Functions and Operators 3.1. W3C Recommendation 21. March 2017, що визначає, крім іншого, функції та оператори на вузлах та послідовностях вузлів.

Назва мови XPath походить від її найбільш відмінної риси, виразу шляху, який забезпечує засіб ієрархічної адресації вузлів у XML-дереві. Основна мета XPath - звернення до вузлів дерев XML, іншими словами для навігації по елементах та атрибутах у документах XML.

Результатом виразу XPath може бути набір вузлів (node-set) з вхідних документів, атомарні значення, такі як цілі числа, рядки (string) та булеві (boolean) true чи false.

XPath використовує компактний не-XML-синтаксис (щоб полегшити використання XPath в межах значень URI та атрибутів XML).

XPath отримав свою назву завдяки використанню позначень шляху, як в URL-адресах для навігації по ієрархічній структурі XML-документа.

XPath містить понад 200 вбудованих функцій.

XPath був визначений одночасно з XSLT (1999 рік).

Спочатку XPath було розроблено для підтримки XSLT та XPointer (мова вказівника XML, що використовується для XLink, XInclude тощо). Сьогодні **XPath** також **використовує XQuery**. Взагалі, будь-який вираз, який є синтаксично дійсним і успішно виконується в XPath 3.1 та XQuery 3.1 , поверне однаковий результат в обох мовах. Є ***кілька винятків*** із цього правила:

- оскільки XQuery розширює заздалегідь визначені посилання на сутності та посилання на символи, а XPath цього не робить, вирази, що містять їх, дають різні результати на двох мовах. Наприклад, значення літерального рядка "&" є & в XQuery, і & в XPath;

- якщо ввімкнено режим сумісності XPath 1.0, XPath поводиться порізному від XQuery в кількох випадках, які перелічені в H.3.2 Incompatibilities when Compatibility Mode is false.
    - Система типів XPath 3.1 заснована на XML-схемі. Це визначається реалізацією, чи базується система типів на основі [XML Schema 1.0] або [XML Schema 1.1] .

1. Основні поняття

Основним будівельним блоком XPath є вираз, який є рядком символів. Мова містить декілька видів виразів, які можуть бути побудовані з ключових слів, символів та операндів. Взагалі операнди виразу - це інші вирази. XPath дозволяє виразам вкладатись із повною загальністю.

Як і XML, XPath є чутливою до регістру мовою. Ключові слова в XPath використовують малі символи та незарезервовані, тобто імена в виразах XPath можуть бути такими ж, як і ключові слова мови, за винятком наступних нефіксованих імен функцій: *array, attribute, comment, document-node, element, empty-sequence, function, if, item, map, namespace-node, node, processinginstruction, schema-attribute, schema-element, switch, text, typeswitch.*

Хоча ключові слова *switch* і *typeswitch* не використовуються в XPath, вони вважаються зарезервованими іменами функцій для сумісності з XQuery.

1.1. Порядок документа

Порядок, що називається порядком документа, визначається усіма вузлами, доступними під час обробки заданого виразу, які можуть складатися з одного або декількох дерев (документів або фрагментів). Стабільний порядок документа означає, що відносний порядок двох вузлів не зміниться під час обробки заданого виразу.

У дереві порядок документування задовольняє наступним обмеженням:

- Кореневий вузол - це перший вузол.
- Кожен вузол є перед усіма його дітьми та нащадками.
- Вузли простору імен відразу слідують за вузлом елемента, з яким вони асоціюються.
- Вузли атрибутів відразу слідують за вузлами простору імен елемента вузла, з яким вони асоціюються.
- Відносний порядок братів і сестер - це порядок, в якому вони відбуваються у children властивості свого батьківського вузла.
- Діти та нащадки трапляються до наступних братів і сестер.

Відносний порядок вузлів у різних деревах є стабільним, але залежним від реалізації, за умови наступного обмеження: якщо будь-який вузол у даному

дереві T1 знаходиться перед будь-яким вузлом у іншому дереві T2, то всі вузли у дереві T1 знаходяться перед усіма вузлами в дерево T2.

В моделі даних значення завжди є послідовністю. Послідовність являє собою упорядкований набір з нуля або більше елементів. Елемент є або атомарне значення, вузол або функція.

Вузол є екземпляром одного з семи видів вузлів: *елемент, атрибут, текст, простір імен, інструкція по обробці, коментар* та *документ*. Для прикладу розглянемо наступний документ XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <price>29.99</price>
  </book>

  <book>
    <title lang="en">Learning XML</title>
    <price>39.95</price>
  </book>
</bookstore>
```

Приклади вузлів у зазначеному документі XML:

<bookstore> (вузол кореневий елемент)
<price>29.99</price> (вузол елемент)
lang="en" (вузол атрибут)

Атомарні значення - це вузли, у яких немає дітей чи батьків.
Наприклад: 29.99, "en"

1.2. Взаємозв'язок вузлів

- Батьківський (Parent) — Кожен елемент та атрибут мають одного батьківського вузла. У наведеному вище прикладі елемент book - батько елемента title та price.
- Діти (Children) — Вузли елементів можуть мати нуль, одного або декількох дітей: title та price є дітьми елемента (book.
- Брати і сестри — Це вузли, які мають одного з батьків. У прикладі це назва та ціна.
- Предки — Батьківський вузол, батьківський батько тощо. У прикладі предками елемента title є елемент book та елемент bookstore.
- Нащадки — Діти вузла, діти дітей тощо. У прикладі нащадками елемента bookstore є елементи book, title та price

2. Синтаксис XPath

2.1. Вибір вузлів

XPath використовує вирази шляху для вибору вузлів у документі XML. Вузол вибирається шляхом слідування шляху або кроків. Нижче наведено найбільш корисні вирази шляху у короткій формі позначення: *nodename* вираз шляху, що складається лише з однієї назви вузла (напр. nodename) дозволяє вибрати всі вузли з цим іменем.

/ на початку виразу шляху - це абревіатура для початкового кроку. Значення цього початкового кроку в тому, що шлях повинен розпочатися в кореневому вузлі дерева, яке містить контекстний вузол.

// на початку виразу шляху означає встановлення початкового вузла послідовності яка містить корінь дерева, в якому знайдено контекстний вузол, плюс усі вузли, що походять від цього кореня. Ця послідовність вузлів використовується як вхід до наступних кроків у виразі шляху. Повне позначення descendantor-self::node()/ .

. Вибирає поточний вузол; повне позначення self::node() .

.. Вибір батька поточного вузла (на один крок назад/вверх); parent::node().

@ Вибір вузла атрибута; attribute:: .

Вираз **child::** може опускатися для **NAME**, але не для **COUNTRY**
`child::COUNTRY/child::NAME`

Наступні вирази ідентичні:

```
/child::book/child::author/child::name/attribute::lastName
/book/author/name/@lastName
```

Корінь документа завжди є контекстом за замовчуванням. Контекст - це поточний вузол або набір вузлів, щодо яких розраховується наступний крок. Нижче наведено деякі вирази шляху та результат виразів:

bookstore	Вибирає всі вузли з назвою bookstore
/bookstore	Вибирає кореневий елемент bookstore. Примітка. Якщо шлях починається з косої риски (/), він завжди представляє абсолютний шлях до елементу!
bookstore/book	Вибирає всі елементи book, які є дітьми елемента bookstore
//book	Вибирає всі елементи book незалежно від того, де вони знаходяться в документі
bookstore//book	Вибирає всі елементи book, які є нащадком елемента bookstore, незалежно від того, де вони знаходяться під елементом bookstore
//@lang	Вибирає всі атрибути, названі lang

Вираз шляху, який *починається з* " /" або " //", вибирає вузли, починаючи *з кореня дерева*, що містить елемент контексту; його часто називають *абсолютним* виразом шляху.

Відносний вираз шляху - це вираз шляху, який вибирає вузли в дереві, виконуючи ряд кроків, *починаючи з контекстного вузла* (який, на відміну від абсолютного виразу шляху, може бути будь-яким вузлом у дереві).

Кожне *непочаткове* виникнення "//" у виразі шляху фактично замінюється під час обробки виразу шляху. Наприклад, **bookstore//book** це скорочений термін, `child::bookstore /descendant-or-self::node()/child::book`

Так вибирають усіх **book** нащадків **bookstore** дітей, залишаючи послідовність кроків, розділених на "/". Але, вираз шляху **//book [1]** зовсім НЕ означає те ж саме, як вираз шляху **/descendant::book [1]**. Останній вибирає перший нащадковий **book** елемент; перший вибирає всі **book** елементи нащадків, які є першими **book** дітьми відповідних батьків.

Наступний приклад ілюструє використання відносних виразів шляху: `child::bookstore /child::book`

Вибирається дочірній елемент **book** дочірнього елемента **bookstore** елемента вузла контексту; тобто **book** - онука елемента контекстного вузла, який має **bookstore** батька. Приклади:

child::book	Вибирає всі вузли book, які є дітьми поточного вузла
attribute::lang	Вибирає атрибут lang поточного вузла
child::*	Вибирає всі дочірні елементи поточного вузла
attribute::*	Вибирає всі атрибути поточного вузла
child::text()	Вибирає всі текстові вузли дітей поточного вузла
child::node()	Вибирає всіх дітей поточного вузла
descendant::book	Вибирає всіх нащадків book поточного вузла
ancestor::book	Вибирає всіх предків book поточного вузла
ancestor-or-self::book	Вибирає всіх предків book поточного вузла - і поточний, також якщо це book вузол
child::*/child::price	Вибирає всіх price онуків поточного вузла

В обох випадках шлях розташування складається з одного або декількох кроків, кожен розділений косою рискою:

Абсолютний шлях розташування: /step/step/...

Відносний шлях розташування: step/step/...

Кожен крок оцінюється по відношенню до вузлів у поточному наборі вузлів. Крок складається з:

- вісь (визначає залежність дерева між вибраними вузлами та поточним вузлом),
- тест вузла (ідентифікує вузол у межах осі),
- нуль або більше предикатів (для подальшого уточнення вибраного набору вузлів).

Синтаксис кроку місцезнаходження: *axisname::nodetest[predicate]*

2.2. Предикати

Предикати використовуються для пошуку конкретного вузла або вузла, який містить конкретне значення. Предикати завжди вбудовані в квадратні дужки. Наприклад:

/bookstore/book[1]	Вибирає перший елемент book, який є дочірнім елементом bookstore.
/bookstore/book[last()]	Вибирає останній елемент book, який є дочірнім елементом bookstore.
/bookstore/book[last()-1]	Вибирає передостанній елемент book, який є дочірнім елементом bookstore.
/bookstore/book[position()<3]	Вибирає перші два елементи book, які є дітьми bookstore.
//title[@lang]	Вибирає всі елементи title, які мають атрибут з назвою lang .
//title[@lang='en']	Вибирає всі елементи title, які мають атрибут "lang" зі значенням "en".
/bookstore/book[price>35.00]	Вибирає всі елементи book елемента bookstore, які мають price елемент, значення якого перевищує 35,00 .
/bookstore/book[price>35.00]/title	Вибирає всі title елементи book елементів bookstore, які мають price елемент, значення якого перевищує 35,00 .

2.3. Вибір невідомих вузлів

Підстановочні символи XPath можна використовувати для вибору невідомих XML-вузлів:

*	відповідає будь-якому вузлу елемента,
@*	відповідає будь-якому вузлу атрибута,
node()	відповідає будь-якому вузлу будь-якого типу.

Наприклад:

/bookstore/*	вибирає всі дочірні вузли елемента bookstore ,
//*	вибирає всі елементи в документі,
//title[@*]	вибирає всі title елементи, які мають принаймі один атрибут будь-якого типу.

2.4. Вибір декількох шляхів

За допомогою | оператора в виразі XPath можна вибрати кілька шляхів.

//book/title | //book/price	Вибирає всі title ТА price елементи всіх елементів book .
//title | //price	Вибирає всі елементи title ТА price в документі .
/bookstore/book/title | //price	Вибирає всі title елементи book елемента bookstore елемента і всі price елементи в документі.

2.5. Oci XPath

Вісь представляє відношення до контекстного (поточного) вузла і використовується для пошуку вузлів відносно цього вузла на дереві.

AxisName	Result
ancestor	Вибирає всіх предків (батьків, бабусь і дідусів тощо) поточного вузла
ancestor-or-self	Вибирає всіх предків поточного вузла та самого поточного вузла
attribute	Вибирає всі атрибути поточного вузла
child	Вибирає всіх дітей поточного вузла
descendant	Вибирає всіх нащадків (дітей, онуків тощо) поточного вузла
descendant-or-self	Вибирає всіх нащадків поточного вузла та самого поточного вузла
following	Вибирає все в документі після закриття тегу поточного вузла
following-sibling	Вибирає всіх братів і сестер після поточного вузла
namespace	Вибирає всі вузли простору імен поточного вузла
parent	Вибирає батька поточного вузла
preceding	Вибирає всі вузли, що з'являються перед поточним вузлом у документі, крім предків, атрибутивних вузлів та вузлівпростору імен
preceding-sibling	Вибирає всіх братів і сестер перед поточним вузлом
self	Вибирає поточний вузол

Для осей, що найчастіше використовуються, існують скорочення:

attribute:: можна замінити на "@" ,
child:: часто просто опускають,
descendant:: можна замінити на ".//" ,
parent: можна замінити на «..» ,
self:: можна замінити на "." .

Перелік понять, з якими необхідно ознайомитись для виконання лабораторної роботи:

1. Основні поняття мови виразів XPath
   1.1. Порядок документа
   1.2. Взаємозв'язок вузлів
2. Синтаксис XPath
   2.1. Вибір вузлів
   2.2. Предикати
   2.3. Вибір невідомих вузлів

2.4. Вибір декількох шляхів
2.5. Осі XPath
2.6. Функції XPath

Література та перелік електронних ресурсів необхідних для ознайомлення з теоретичним матеріалом теми наведено у списку рекомендованих джерел.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Написати декілька виразів шляху XPath у відповідності до свого XML документа (обов'язково використати функції, наприклад count, sum, avg чи інші).
3. Використати on-line XPath Tester / Evaluator для перевірки коректності створених виразів шляху.
4. Оформити звіт про виконання лабораторної роботи, який має містити:
   - титульну сторінку (див. додаток А);
   - тему, мету та завдання лабораторної роботи;
   - короткий перелік та опис створених виразів шляху XPath;
   - навести знімки екрану з кодом та результатом їх перевірки XPath Tester.
5. Завантажити в канал "БД. Лабораторна робота" своєї команди в Microsoft Teams.

Контрольні питання

1. Що означає назва мови XPath?
2. Що таке порядок документа і яким обмеженням він має задовільняти?
3. Який взаємозв'язок вузлів XML документа?
4. Які вирази шляху у короткій формі використовуються для вибору вузлів?
5. Що таке абсолютний і відносний вирази шляху?
6. Наведіть приклад предикатів для пошуку конкретного вузла або вузла, який містить конкретне значення.
7. Наведіть приклад вибору невідомих XML-вузлів.
8. Як у виразі XPath можна вибрати кілька шляхів?

Список рекомендованих джерел до теми

1. Fawcett J. Beginning XML (5-edition) /Joe Fawcett, Danny Ayers, Liam R. E. Quin. – Wiley. – 2012. – 864 p.
2. W3schools: XPath Tutorial. [Електронний ресурс]. – Доступний з: *https://www.w3schools.com/xml/xpath_intro.asp* .

ЛАБОРАТОРНА РОБОТА № 15
«XSLT — мова перетворення XML-документів».

Тема: *«XSLT — мова перетворення XML-документів».*

Мета роботи: Ознайомлення з синтаксисом мови XSLT та перетворенням фрагмента XML документа з використанням XSLT.

Теоретичний матеріал

XSLT (XSL Transformations) це мова, призначена для перетворення XMLдокументів у документи іншого формату, наприклад HTML. Вона використовується спільно з XPath і поділяє ту саму модель даних, що і XPath. XSLT також використовує бібліотеку функцій та операторів, визначені в XQuery and XPath Functions and Operators. Мова XSLT визначена W3C в **XSL** Transformations (XSLT) Version 3.0 W3C Recommendation 8 June 2017 .

Як і XPath, XSLT є підмножиною XSL (eXtensible Stylesheet Language) - мова стилів таблиць для XML.

HTML використовує попередньо визначені теги. Значення та відображення кожного тегу добре зрозуміло. Для додавання стилів до елементів HTML використовуються каскадні таблиці стилів CSS.

XSL це фактично таблиці стилів для XML. XML не використовує попередньо визначені теги, і тому значення кожного тегу комп'ютеру наперед невідомо (наприклад, елемент <table> може вказувати таблицю HTML, предмет меблів чи щось інше - і браузери не знають, як це відобразити). Отже, XSL описує, як повинні відображатися елементи XML.

Але XSL - більше, ніж просто мова таблиці стилів. XSL складається з чотирьох частин:

1) XSLT - мова для перетворення XML-документів
2) XPath - мова для навігації в XML-документах
3) XSL-FO - мова для форматування XML-документів (завдяки модулю CSS3 Paged Media Module, W3C представив новий стандарт форматування документа і з 2013 року CSS3 пропонується як заміна XSL-FO)
4) XQuery - мова для запитів XML-документів

Перетворення мовою XSLT виражається у вигляді таблиці стилів. Таблиця стилів складається з одного або декількох добре сформованих XML документів, що відповідають просторам імен у Рекомендації XML.

Таблиця стилів, як правило, включає елементи, визначені XSLT, а також елементи, які не визначені XSLT. Елементи, визначені XSLT, розрізняються за допомогою простору імен http://www.w3.org/1999/XSL/Transform (див. 3.1

Простір імен XSLT). Таким чином, ця специфікація є визначенням синтаксису та семантики простору імен XSLT.

Термін таблиця стилів відображає той факт, що одна з важливих ролей XSLT полягає в додаванні інформації про стилізацію до вихідного документа XML, перетворюючи її в документ, що складається з об'єктів форматування XSL, або в інший презентаційно-орієнтований формат, такий як HTML, XHTML або SVG. Однак XSLT використовується для широкого спектру завдань трансформації, а не виключно для форматування та презентаційних програм.

Перетворення, виражене XSLT, описує правила перетворення вхідних даних у вихідні дані. Усі входи та виходи будуть екземплярами моделі даних XDM. У найпростішому і найпоширенішому випадку вхід - це XML-документ, який називається деревом-джерелом, а вихід - документом XML, який називається деревом результатів. Можлива також обробка декількох вихідних документів, генерування декількох результативних документів та обробка форматів, відмінних від XML.

Перетворення досягається набором правил шаблону. Правило шаблону асоціює шаблон, який зазвичай відповідає вузлам у вихідному документі, із конструктором послідовностей. У багатьох випадках оцінка конструктора послідовностей призведе до побудови нових вузлів, які можна використовувати для отримання частини дерева результатів. Структура результативних дерев може бути абсолютно різною від структури дерев-джерел. При побудові дерева результатів вузли з дерев джерел можуть бути відфільтровані та упорядковані, а також може бути додана довільна структура.

Такий механізм дозволяє застосовувати таблицю стилів для широкого класу документів, які мають схожі структури вихідного дерева. Таблиці стилів мають модульну структуру; вони можуть містити кілька пакетів, розроблених незалежно один від одного, і кожен пакет може складатися з декількох модулів таблиць стилів.

Отже, з допомогою XSLT можна додавати або видаляти елементи та атрибути до вихідного файлу або з нього. Також можна переставляти та сортувати елементи, виконувати тести та приймати рішення щодо того, які елементи ховати та відображати, та багато іншого.

У процесі перетворення XSLT використовує XPath для навігації по елементах та атрибутах у документа XML щоб визначити ті частини документа, які повинні відповідати одному або більше заздалегідь заданим шаблонам. Коли знайдено збіг, XSLT перетворить відповідну частину вхідного документа в документ результату.

1. Основні поняття XSLT

Для ознайомлення з основними поняттями XSLT розглянемо приклад перетворення наступного документа XML ("cdcatalog.xml") у XHTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
<cd>
  <title>Hide your heart</title>
  <artist>Bonnie Tyler</artist>
  <country>UK</country>
  <company>CBS Records</company>
  <price>9.90</price>
  <year>1988</year>
</cd>
...
</catalog>
```

Нехай створена нами таблиця стилів XSL ("cdcatalog.xsl") з шаблоном перетворення має наступний вигляд:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      <h2>My CD Collection</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
          <th>Title</th>
          <th>Artist</th>
        </tr>
        <tr>
          <td>...</td>
          <td>...</td>
        </tr>
      </table>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Щоб зв’язати таблицю стилів XSL з документом XML додамо посилання на таблицю стилів XSL до документа XML ("cdcatalog.xml"):

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
...
</catalog>
```

1.1. Декларація стиля

Повернемося до файлу таблиці стилів XSL ("cdcatalog.xsl").

Оскільки таблиця стилів XSL є документом XML, він завжди починається з декларації XML: <?xml version="1.0" encoding="UTF-8"?>.

Кореневим елементом, який оголошує документ як таблицю стилю XSL, є <xsl: stylesheet> або <xsl: transform> - обидва повні синоніми і можна використовувати будь-який з них. Правильний спосіб оголошення таблиці стилю XSL відповідно до рекомендації W3C XSLT:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

або:

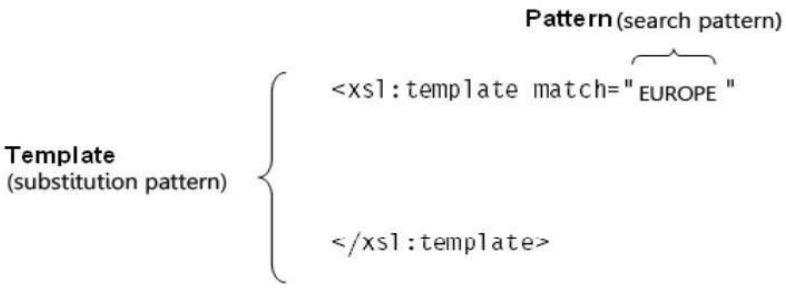
```
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Щоб отримати доступ до елементів, атрибутів та особливостей XSLT, ми повинні оголосити простір імен XSLT у верхній частині документа. xmlns: xsl = "http://www.w3.org/1999/XSL/Transform" вказує на офіційний простір імен W3C XSLT. Якщо ви користуєтеся цим простором імен, ви також повинні включити атрибут version = "1.0".

1.2. Шаблон перетворення

Таблиця стилів XSL складається з одного або декількох наборів правил, які називаються шаблонами. Правила шаблону - це правила для трансформації вихідного дерева в дерево результатів, тобто правила, які слід застосувати, коли вказаний вузол збігається .

Вони складаються з двох частин, одна з яких називається патерном (шаблон пошуку), а інша шаблоном (шаблон заміни), який виконується, коли шаблон пошуку має відповідний збіг у вихідному дереві. Склад шаблону наступний:



Для створення шаблонів використовується елемент <xsl: template>. В нашому прикладі він має вигляд <xsl:template match="/">. Атрибут match використовується для асоціації шаблону з XML-елементом. В даному випадку match="/" є виразом XPath, що пов'язує шаблон з коренем вихідного документа XML, тобто визначає весь документ.

Але в результаті використання наведеного шаблону перетворення жодні дані не будуть скопійовані з XML-документа на вихід. Ми отримаємо наступне:

My CD Collection

Title	Artist
…	…

1.3. Отримання значення XML-елемента <value-of>

Для того щоб отримати значення вибраного вузла та додати його до вихідного потоку перетворення потрібно використати елемент <xsl:valueof>. Так, якщо замість "…" у нашому прикладі записати

```
<xsl:value-of select="catalog/cd/title"/>
<xsl:value-of select="catalog/cd/artist"/>
```

Отримаємо

Title	Artist
Empire Burlesque	Bob Dylan

Атрибут select містить вираз XPath і був застосований, щоб вибрати лише певну частину дерева. Усі інші піделементи таким чином ігноруються.

Таким чином ми отримали значення title і artist лише для одного вузла CD.

1.4. Цикл у шаблоні перетворення <for-each>

Елемент <xsl: for-each> дозволяє робити цикл у шаблоні перетворення і може використовуватися для вибору кожного XMLелемента певного набору вузлів. Це робиться наступним чином:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
  <html>
   <body>
    <h2>My CD Collection</h2>
     <table border="1">
     <tr bgcolor="#9acd32">
       <th>Title</th>
       <th>Artist</th>
      </tr>
       <xsl:for-each  select="catalog/cd">
        <tr>
          <td><xsl:value-of  select="title | year"/>  <br /> </td>
          <td><xsl:value-of  select="artist"/> </td>
         </tr>
       </xsl:for-each>
     </table>
    </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

1.5. Сортування результатів виводу <sort>

Щоб сортувати вихід за прізвищами артистів, додамо елемент <xsl:sort> всередину елемента <xsl: for-every> у файл XSL:

```
<xsl:for-each select="catalog/cd">
      <xsl:sort select="artist"/>
      <tr>
     <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of  select="artist"/></td>
      </tr>
    </xsl:for-each>
```

1.6. Вивід за умовою <if>

Елемент <xsl: if> використовується для встановлення умови виводу вмісту файлу XML. Наприклад, щоб вивести лише назви та виконавця компакт-дисків, ціна яких перевищує 10 потрібно записати наступний код

```
<xsl:for-each select="catalog/cd">
      <xsl:if test="price &gt; 10">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
          <td><xsl:value-of select="price"/></td>
        </tr>
      </xsl:if>
    </xsl:for-each>
```

1.7.Вивід за однією з кількох умов <choose>

Щоб вивести за однією з кількох умов у файл XML, потрібно скористатися елементом <xsl:choose> разом з <xsl: when> та <xsl:otherwise>:

```
<xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <xsl:choose>
        <xsl:when test="price &gt; 10">
          <td bgcolor="#ff00ff">
          <xsl:value-of select="artist"/></td>
        </xsl:when>
        <xsl:otherwise>
          <td><xsl:value-of select="artist"/></td>
        </xsl:otherwise>
      </xsl:choose>
    </tr>
 </xsl:for-each>
```

Наведений вище код додасть рожевий фоновий колір у стовпчик «Виконавець», коли ціна компакт-диска вище 10.

1.8. Правило шаблону до поточного елемента або дочірніх вузлів поточного елемента

Елемент <xsl: apply-templates> застосовує правило шаблону до поточного елемента або дочірніх вузлів поточного елемента.

Якщо ми додамо атрибут "select" до елемента <xsl: apply-templates>, він обробить лише дочірні елементи, які відповідають значенню атрибута.

Також його можна використовувати щоб вказати в якому порядку повинні оброблятися дочірні вузли.

Наприклад:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>

<xsl:template match="cd">
  <p>
  <xsl:apply-templates select="title"/>
  <xsl:apply-templates select="artist"/>
  </p>
</xsl:template>

<xsl:template match="title">
  Title: <span style="color:#ff0000">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>

<xsl:template match="artist">
  Artist: <span style="color:#00ff00">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>

</xsl:stylesheet>
```

Результат використання шаблону:

```
My CD Collection
Title:  Empire Burlesque
Artist: Bob Dylan
Title:  Hide your heart
Artist: Bonnie Tyler
Title:  Greatest Hits
Artist: Dolly Parton
```

1.9. Нумерація <number>

За допомогою елемента <xsl: number> вузли можуть бути призначені послідовностям у дереві результатів. Таким чином, наприклад, глави, розділи або прості елементи можуть бути пронумеровані. Також можлива нумерація в різних рівнях і може бути визначено форматування виводу. Нумерація арабськими або римськими номерами або літерами може бути сформована за допомогою атрибуту формату. На практиці елемент має три загальні ознаки: кількість (count) , рівень (level) та формат (format). Атрибут count визначає, які вузли потрібно підрахувати, рівень визначає спосіб підрахунку та формат визначає відображення. Наприклад у вище наведеному прикладі можна пронумерувати назви дисків наступним чином

```
<xsl:template match="cd">
    <p>
       <xsl:number format="1" level="any"/>
       <xsl:apply-templates select="title"/>
       <xsl:apply-templates select="artist"/>
   </p>
</xsl:template>
```

На виході отримаємо

```
My CD Collection
Title:  Empire Burlesque
Artist: Bob Dylan
Title:  Hide your heart
Artist: Bonnie Tyler
Title:  Greatest Hits
Artist: Dolly Parton
```

Операція АБО декількох елементів може бути досягнута за допомогою "|"

```
 <xsl:template match="cd">
  <p>
    <xsl:apply-templates select="title | year"/>  <br />
    <xsl:apply-templates select="artist"/>
  </p>
</xsl:template>
```

отримаємо

```
My CD Collection
Title: Empire Burlesque
1985
Artist: Bob Dylan
```

Перелік розділів та понять, з якими необхідно ознайомитись для виконання завдання лабораторної роботи:

1. Основні поняття XSLT
   1.1. Декларація стиля
   1.2. Шаблон перетворення
   1.3. Отримання значення XML-елемента <value-of>
   1.4. Цикл у шаблоні перетворення <for-each>
   1.5. Сортування результатів виводу <sort >
   1.6. Вивід за умовою <if>
   1.7. Вивід за однією з кількох умов <choose>
   1.8. Правило шаблону до поточного елемента або його дочірніх елементів
   1.9. Нумерація <number>

Література та перелік електронних ресурсів необхідних для ознайомлення з теоретичним матеріалом теми наведено у списку рекомендованих джерел.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. Створити шаблон перетворення XSLT фрагмента свого XML документа з обов'язковим використанням <value-of>, <for-each>, <sort >, <if>/<choose>,<number>.
3. Використати довільний XSLT processor, наприклад freeformatter.com, для перевірки коректності роботи створеного шаблона перетворення.
4. Оформити звіт про виконання лабораторної роботи, який має містити:
   - титульну сторінку (див. додаток А);
   - тему, мету та завдання лабораторної роботи;
   - короткий перелік та опис створеного шаблона перетворення;
   - навести знімки екрану з кодом шаблона та результатом перетворення.
5. Завантажити в канал "БД. Лабораторна робота" своєї команди в Microsoft Teams.

Контрольні питання

1. Для чого призначена мова XSLT?
2. За допомогою якого механізму досягається перетворення XML документу?
3. Яка структура декларації стилю?
4. Що таке шаблон перетворення і з яких частин він складається?

5. Як отримати значення вибраного вузла та додати його до вихідного потоку перетворення?
6. Який оператор використовується для вибору кожного XML-елемента певного набору вузлів?
7. Як можна відсортувати та пронумерувати результати виводу?
8. Наведіть приклад встановлення умови виводу вмісту файлу XML.
9. Як здійснити вивід за однією з кількох умов?
10. Який елемент застосовує правило шаблону до поточного елемента або дочірніх вузлів поточного елемента?

Список рекомендованих джерел до теми

1. Fawcett J. Beginning XML (5-edition) /Joe Fawcett, Danny Ayers, Liam R. E. Quin. – Wiley. – 2012. – 864 p.
2. W3schools: XSLT Tutorial. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/xml/xsl_intro.asp.

ЛАБОРАТОРНА РОБОТА № 16
«XQuery - мова запитів XML-документів»

Тема: *«XQuery - мова запитів XML-документів».*

Мета роботи: Ознайомлення з синтаксисом мови XQuery та конструюванням запитів даних XML документа.

Теоретичний матеріал

Мова SQL, яка сьогодні є стандартом мови систем керування реляційними базами даних, була розроблена, щоб абстрагувати алгоритми, призначені для опису та перетворення даних, від деталей отримання та збереження цих даних. Це дозволяє створювати дуже складні програми транзакційної обробки даних, систем звітності, сховищ даних та бізнес аналітики.

Потужність і гнучкість SQL зробили реляційну модель домінуючим типом даних протягом останніх десятиліть. Але, хоч попередник реляційних баз даних – ієрархічні бази, якраз і занепали через наявність сильної залежності від механізмів доступу і збереження даних, зараз вони знову широко застосовуються завдяки використанню мови XML.

XML документи представляють відносини між сутностями, використовуючи ієрархію, тобто відносини предок-нащадок. І саме застосування XQuery для роботи з даними XML в одному або декількох документах, дозволяє абстрагуватися від механізмів збереження і доступу до даних. Тобто, XQuery - це мова для пошуку та маніпулювання будь-чим, що може бути представлено як дерево і відповідає моделі даних, визначеній у XQuery and XPath Data Model (XDM) 3.1 XQuery використовує XPath і одночасно її можна розглядати як розширення XPath. Це означає, що синтаксис XQuery схожий на XPath, і він не базується на основі XMLелементів, як XSLT.

XQuery також використовує спільну з XPath бібліотеку функцій та операторів, визначені в XQuery and XPath Functions and Operators . Мова XQuery визначена W3C в XQuery 3.1: An XML Query Language W3C Recommendation 21 March 2017.

Як і XPath, XQuery є підмножиною XSL (eXtensible Stylesheet Language) - мова стилів таблиць для XML. XSL складається з чотирьох частин:

1) XSLT - мова для перетворення XML-документів;
2) XPath - мова для навігації в XML-документах;
3) XSL-FO - мова для форматування XML-документів (завдяки модулю CSS3 Paged Media Module, W3C представив новий стандарт

форматування документа і з 2013 року CSS3 пропонується як заміна XSL-FO);

4) XQuery - мова для запитів XML-документів.

Взагалі, будь-який вираз, який є синтаксично дійсним і успішно виконується в XPath 3.1 та XQuery 3.1 , поверне однаковий результат в обох мовах. Є кілька винятків із цього правила:

- оскільки XQuery розширює заздалегідь визначені посилання на сутності та посилання на символи, а XPath цього не робить, вирази, що містять їх, дають різні результати на двох мовах. Наприклад, значення літерального рядка "&" є & в XQuery, і & в XPath;
- якщо ввімкнено режим сумісності XPath 1.0, XPath поводиться порізному від XQuery в кількох випадках, які перелічені в XML Path Language (XPath), версія 3.1 .

1. Основні поняття XQuery

XQuery можна використовувати для:

- знаходження інформації для використання у веб-сервісі
- створення зведених звітів
- перетворення XML-даних у XHTML
- пошуку відповідної інформації у веб-документах

1.1. Синтаксис

Деякі основні правила синтаксису:

- XQuery чутливий до регістру
- елементи, атрибути та змінні XQuery повинні бути дійсними іменами XML
- значення рядка XQuery може бути в одинарних або подвійних лапках
- змінна XQuery визначається з $, а потім ім'ям, наприклад, $bookstore
- коментарі XQuery обмежені **(:** … **:)**, напр. **(:** XQuery Comment **:)**

Запит XQuery складається з одного або декількох модулів з тілом запиту, що містить вираз, значення якого є результатом запиту. Вираз представлений у граматиці XQuery символом Expr :

```
Expr          :: =    ExprSingle (","
                       ExprSingle)*
ExprSingle             FLWOR_Expr
              :: =
```

| Quantified_Expr

| Switch_Expr

| Typeswitch_Expr

| If_Expr

| Try_CatchExpr

| Or_Expr

Оператором XQuery, який має найнижчий пріоритет є *оператор кома*, який використовується для об'єднання двох операндів для формування послідовності. Як показано вище, загальний вираз Expr може складатися з декількох операндів *ExprSingle*, розділених комами. Ім'я *ExprSingle* позначає вираз, який не містить *оператора коми* верхнього рівня (незважаючи на його ім'я, *ExprSingle* може означати послідовність, що містить більше одного елемента.)

ExprSingle використовується в різних місцях, де вираз не може містити коми верхнього рівня. Наприклад, кожен з аргументів виклику функції повинен бути *ExprSingle* , оскільки коми використовуються для розділення аргументів виклику функції.

Після коми, виразами, які мають нижчий пріоритет є

FLWORExpr,

QuantifiedExpr,

SwitchExpr,

TypeswitchExpr,

IfExpr,

TryCatchExpr, і OrExpr.

Для ознайомлення з синтаксисом та основними поняттями XQuery скористаємося прикладом наступного XML документа, де перелічені актори та фільми з їх участю, а також деякі атрибути цих фільмів :

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
 <video_template>
  <title/>
  <genre>
   <choice>action</choice>
   <choice>comedy</choice>
   <choice>drama</choice>
   <choice>musical</ch oice>
  </genre>
  <rating>
   <choice>G</choice>
   <choice>PG</choice>
```

```
   <choice>PG-13</choice>
   <choice>R</choice>
   <choice>NC-17</choice>
  </rating>
  <user_rating>
   <choice>1</choice>
   <choice>2</choice>
   <choice>3</choice>
  </user_rating>
  <summary/>
  <year/>
  <director/>
  <studio/>
  <runtime/>
  <vhs/>
  <vhs_stock/>
  <dvd/>
  <dvd_stock/>
 </video_template>
  <actors>
   <actor id="00000015">Anderson, Jeff</actor>
   <actor id="0000000f">Bonet, Lisa</actor>
   <actor id="00000006">Ford, Harrison</actor>
   <actor id="00000018">Ghigliotti, Marilyn</actor>
   <actor id="0000000c">Hackman, Gene</actor>
   <actor id="00000003">Jones, Tommy Lee</actor>
   <actor id="00000009">Smith, Will</actor>
   <actor id="0000001b">Spoonhauer, Lisa</actor>
   <actor id="00000012">Voight, John</actor>
  </actors>
  <videos>
   <video id="id1235">
    <title>The Fugitive</title>
    <genre>action</genre>
    <rating>PG-13</rating>
    <summary>
     Tommy Lee Jones and Harrison Ford are the hunter and the
hunted in this fastpaced story of a falsely convicted man who
escapes to find his wife's true killer.
    </summary>
    <year>1997</year>
    <director>Andrew Davis</director>
    <studio>Warner</studio>
    <user_rating>2</user_rating>
    <runtime>110</runtime>
    <actorRef>00000003</actorRef>
    <actorRef>00000006</actorRef>
    <vhs>13.99</vhs>
```

```
    <vhs_stock>206</vhs_stock>
    <dvd>14.99</dvd>
    <dvd_stock>125</dvd_stock>
   </video>
   <video id="id1244">
   <title>Enemy of the State</title>
   <genre>action</genre>
   <rating>R</rating>
   <summary>
    After a chance meeting with an old pal, Robert Deal finds
himself in posession of a disk that contains evidence of an
assassination plot by the  NSA.
   </summary>
   <year>1999</year>
   <director>Tony Scott</director>
   <studio>Buena vista</studio>
   <user_rating>3</user_rating>
   <runtime>113</runtime>
   <actorRef>00000009</actorRef>
   <actorRef>0000000c</actorRef>
   <actorRef>0000000f</actorRef>
   <actorRef>00000012</actorRef>
   <vhs>16.99</vhs>
   <vhs_stock>188</vhs_stock>
   <dvd>29.99</dvd>
   <dvd_stock>353</dvd_stock>
   </video>
    <video id="id1243">
    <title>Clerks</title>
    <genre>comedy</genre>
    <rating>R</rating>
    <summary>
     Spend a day at work with two friends and their quirky
customers at a video shop and convenience store.
    </summary>
    <year>1999</year>
    <director>Kevin Smith</director>
    <studio>Buena vista</studio>
    <user_rating>1</user_rating>
    <runtime>97</runtime>
    <actorRef>00000015</actorRef>
    <actorRef>00000018</actorRef>
    <actorRef>0000001b</actorRef>
    <vhs>16.99</vhs>
    <vhs_stock>188</vhs_stock>
   </video>
 </videos>
</result>
```

Припустимо, ми хочемо знайти заголовки всіх відео з актором, ім'я якого - Ліза. XPath запит виглядатиме так:

```
//video[actorRef=//actors/actor[ends-with(.,'Lisa')]/@id]/title
```

Тобто, його можна прочитати зліва направо як:

- Виберіть усі <video> елементи на будь-якому рівні
- Виберіть ті, у яких є actorRef елемент, значення якого дорівнює одному із наведених нижче значень:
- Виберіть усі <actors>елементи на будь-якому рівні
- Виберіть усі їх <actor>дочірні елементи
- Виберіть елемент лише в тому випадку, якщо його значення закінчується на "Lisa"
- Виберіть значення id атрибута
- Виберіть <title>дочірній елемент цих вибраних <video> елементів

Результат:

```
<title>Enemy of the State</title>
<title>Clerks</title>
```

Як бачимо, на такому рівні складності синтаксис XPath стає досить неприємним. Для такого роду запитів та для будь-якого складнішого синтаксису, власне, і підходить XQuery.

1.2. XQuery FLWOR вирази

Використовуючи SQL для останнього прикладу, фактично потрібно виконати з'єднання двох таблиць, таблиці відео та таблиці акторів. Це не зовсім те саме в XML, оскільки дані є ієрархічними, а не табличними, але XQuery дозволяє писати запити приєднання аналогічно звичному підходу SQL. Еквівалент виразу SELECT SQL називається виразом FLWOR, названим його п'ятьма пунктами: ***for, let, where, order by, return***. Приклад, як вираз FLWOR видає той самий результат:

```
let $doc := .
for $v in $doc//video,        $a  in $doc//actors/actor
where ends-with($a, 'Lisa')  and  $v/actorRef = $a/@id
return $v/title
```

Розберемо цей вираз FLWOR:

- Let просто оголошує змінну. Це оголошення включено, тому що можна по-різному встановити цю змінну; наприклад, можна ініціалізувати її як doc('videos.xml') або як результат якогось складного запиту, який знаходить певний документ у базі даних.

- for визначає дві змінні діапазону: одна стосується всіх відео, інша - всіх акторів. У сукупності вираз FLWOR обробляє всі можливі пари відео та акторів.
- Потім у where вибираються ті пари, які нас насправді цікавлять. Нас цікавить лише те, що в цьому відео є актор, і лише якщо ім'я актора закінчується на "Ліза".
- Нарешті return повідомляє системі, яку інформацію ми хочемо отримати. У цьому випадку ми хочемо назву відео.

Якщо уважно розглянути код запиту, можна помітити одну особливість XPath. А саме - більшість відеороликів матимуть більше одного актора. Тому вираз $v/actorRef вибирає кілька елементів. Правила для "=" оператора в XPath (а отже, і в XQuery) полягають у тому, що він порівнює *все* зліва з *усім* правим і повертає істину, якщо є хоча б одна відповідність. Фактично це робить неявне з'єднання. Можна уникнути використання цієї функції та написати запит у більш класичній реляційній формі як:

```
let $doc := .
for $v in $doc//video,
      $va in $v/actorRef,
      $a in $doc//actors/actor
where ends-with($a, 'Lisa')  and  $va eq $a/@id
return $v/title
```

Цього разу використали інший оператор рівності "***eq***", який дотримується більш звичайних правил, ніж = : він суворо порівнює *одне* значення зліва з *одним* значенням праворуч.

Таким чином у XQuery є два способи порівняння значень.

1. Загальні порівняння: =, !=, <, <=, >, >=
2. Порівняння величин: eq, ne, lt, le, gt, ge

Різниця між двома методами порівняння показана вище.

Щоб отримати результати у відсортованому порядку використовується **order by**. Припустимо, потрібно, щоб відео були впорядковані за роком їх виходу:

```
let $doc := .
for $v in $doc//video,
$a in  $doc//actors/actor
where ends-with($a, 'Lisa')  and  $v/actorRef =$a/@id
order by $v/year
return $v/title
```

for прив'язує змінну до кожного елемента, повернутого у виразі і призводить до ітерації.

Щоб зафіксувати певну кількість ітерацій, використовується ключове слово **to**:

```
for $x in (1 to 3)
return  <test>{$x}</test>
```

Результат:

```
<test>1</test>
<test>2</test>
<test>3</test>
```

Ключове слово **at** використовується для підрахунку ітерацій:

```
for $x at $i in //video/title
return <title>{$i}. {data($x)}</title>
```

Результат:

```
<title>1. The Fugitive</title>
<title>2. Enemy of the State</title>
<title>3. Clerks</title>
```

for і **let** операнди можуть з'являтися в будь-якому порядку, і будь-яку кількість разів. Вираз FLOWR є набагато більше, ніж те, що викладено - для отримання додаткової інформації ознайомтеся з навчальним посібником XQuery FLWOR http://www.stylusstudio.com/xquery_flwor.html.

Перелік розділів та понять, з якими необхідно ознайомитись для виконання завдання лабораторної роботи:

1. Основні поняття XQuery
   1.1 Синтаксис
   1.2 XQuery FLWOR вирази

Література та перелік електронних ресурсів необхідних для ознайомлення з теоретичним матеріалом теми наведено у списку рекомендованих джерел.

Порядок виконання роботи

1. Опрацювати теоретичний матеріал.
2. У відповідності до свого XML документа, написати декілька запитів XQuery застосувавши FLWOR вирази з обов'язковим використати функцій.
3. Для отримання максимального балу застосувати деякі з виразів QuantifiedExpr, SwitchExpr, TypeswitchExpr, IfExpr, TryCatchExpr або OrExpr.
4. Використати довільний XQuery processor, наприклад XQuery 3.0 Online Tester, для перевірки коректності роботи створених запитів XQuery.
5. Оформити звіт про виконання лабораторної роботи, який має містити:

- титульну сторінку (див. додаток А);

- тему, мету та завдання лабораторної роботи;
- короткий перелік та опис створеного шаблона перетворення;
- навести знімки екрану з кодом запитів XQuery та отриманими результатами.

6. Завантажити в канал “БД. Лабораторна робота” своєї команди в Microsoft Teams.

Контрольні питання

1. Для чого застосовується мова XQuery?
2. Які особливості синтаксису мови XQuery?
3. Що означає вираз FLWOR?
4. Наведіть приклад використання виразу QuantifiedExpr.
5. Наведіть приклад використання виразу SwitchExpr.
6. Як використовується вираз TryCatchExpr?

Список рекомендованих джерел до теми

3. Fawcett J. Beginning XML (5-edition) /Joe Fawcett, Danny Ayers, Liam R. E. Quin. – Wiley. – 2012. – 864 p.
4. Develop with XQuery: A better programming language for the database programmer. [Електронний ресурс]. – Доступний з: https://developer.ibm.com/technologies/web-development/.
5. W3schools: XQuery Tutorial. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/xml/xquery_intro.asp.

ПІСЛЯМОВА

У цьому практикумі розглянуто основні поняття та практичні питання роботи з базами даних. Наведено інформацію про процес розробки бази даних від спрощеної концептуальної моделі до фізичної реалізації. Описано можливості мови SQL для створення, запитування та модифікації даних. Розглянуто способи підтримки цілісності даних. Для пришвидшення швидкодії розглянуто поняття індексів та їхнє створення у SQL. Розглянуто тригери, збережені функції, view та курсори. Запропоновано читачу змоделювати одночасний доступ до даних, ознайомитися з рівнем ізоляції транзакцій та їхнім впливом на видимість даних. Останні лабораторні розглядають способи визначення і, потім, опрацювання XML за допомогою DTD, XML схем, мови виразів XPath та, врешті, XSLT та XQuery.

Сподіваємося, що цей лабораторний практикум буде корисним для студентів, що вивчають предмет “Бази даних та інформаційні системи”, а також для всіх, хто цікавиться цією галуззю знань. Бажаємо вам успіхів у подальшому вивченні та застосуванні отриманих знань у різних сферах діяльності на користь державі.

СПИСОК ЛІТЕРАТУРИ

1. Пасічник В. В. Організація баз даних та знань / В. В. Пасічник, В.А. Резніченко.– Київ.– Видавнича група BHV. – 2006. — 384 с.
2. 2019 Database Trends – SQL™ vs. NoSQL, Top Databases, Single vs. Multiple Database Use. [Електронний ресурс]. – Доступний з: https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databaseshttps://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/single-vs-multiple-database-use/ .
3. Date C. J. An Introduction to Database Systems (8th Edition) / C. J. Date. – 8th ed. – Pearson Education Inc. – 2004. – 1040 p.
4. Date C.J. SQL and Relational Theory: How to Write Accurate SQL Code (3rd edition) / C. J. Date. — O'Reilly Media, Inc. – 2015. – 563 p.
5. Develop with XQuery: A better programming language for the database programmer. [Електронний ресурс]. – Доступний з: https://developer.ibm.com/technologies/web-development/.
6. Fawcett J. Beginning XML (5-edition) /Joe Fawcett, Danny Ayers, Liam R. E. Quin. – Wiley. – 2012. – 864 p.
7. Garcia-Molina H. Database Systems: 2nd Edition / H. Garcia-Molina, Jeffrey Ullman, Jennifer Widom. – Pearson Education Inc. – 2009.– 1203 p.
8. Groff J. R. SQL The Complete Reference / James R. Groff, Paul N. Weinberg, Andy Oppel. – 3rd Edition. – McGraw Hill Professional. – 2008. – 912 p.
9. PostgreSQL Data Types. [Електронний ресурс]. – Доступний з: https://neon.tech/postgresql/postgresql-tutorial/postgresql-data-types.
10. PostgreSQL Documentation. PostgreSQL 15. Concurrency Control [Електронний ресурс]. – Доступний з: https: //www.postgresql.org/docs/15/mvcc.html .
11. PostgreSQL PL/pgSQL. [Електронний ресурс. – Доступний з: https: //www.postgresqltutorial.com/postgresql-plpgsql/ .
12. PostgreSQL PL/pgSQL. PostgreSQL CREATE PROCEDURE. [Електронний ресурс. – Доступний з: https://www.postgresqltutorial.com/postgresql-plpgsql/postgresql-create-procedure.
13. PostgreSQL Tutorial. PostgreSQL Indexes. [Електронний ресурс]. – Доступний з: https://www.postgresqltutorial.com/postgresql-indexes.
14. PostgreSQL. Documentation. [Електронний ресурс]. – Доступний з: https://www.postgresql.org/docs/ .

15. PostgreSQL. Documentation. PostgreSQL 15. Query Language (SQL) Functions [Електронний ресурс].– Доступний з: https://www.postgresql.org/docs/15/xfunc-sql.html.
16. PostgreSQL. Documentation. PostgreSQL 15. [Електронний ресурс]. – Доступний з: https://www.postgresql.org/docs/15/indexes.html
17. PostgreSQL. Documentation. PostgreSQL 15. CREATE VIEW – define a new view. [Електронний ресурс]. – Доступний з: https: //www.postgresql.org/docs/15/sql-createview.html .
18. PostgreSQL: Documentation. PostgreSQL 15. XML-функції PostgreSQL [Електронний ресурс]. – Доступний з: https: //www.postgresql.org/docs/15/functions-xml.html.
19. PostgreSQLTutorial. PostgreSQL Transaction. [Електронний ресурс]. – Доступний з: http://www.postgresqltutorial.com/postgresql-transaction/ .
20. PostgreSQLTutorial. PostgreSQL Views. [Електронний ресурс. – Доступний з: https://www.postgresqltutorial.com/postgresql-views/ .
21. SQL Database Tutorial W3C. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/sql/sql_create_db.asp .
22. SQL Tutorial W3C. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/sql/default.asp .
23. W3schools: XML Tutorial. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/xml/default.asp .
24. Liquid Technologies. XML Overview. [Електронний ресурс]. – Доступний з: https://www.liquid-technologies.com/Reference/Glossary/XML_Overview.html
25. Free Online XML Formatter. [Електронний ресурс]. – Доступний з: https://www.liquidtechnologies.com/online-xml-formatter .
26. Free Online XML Validator (Well formed). [Електронний ресурс]. – Доступний з: https://www.liquid-technologies.com/online-xml-validator
27. W3schools: XML Tutorial. XML DTD. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/xml/xml_dtd.asp .
28. W3schools: XML DTD. DTD Tutorial. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/xml/xml_dtd_intro.asp.
29. Liquid Technologies. DTD Overview. [Електронний ресурс]. – Доступний з: https://www.liquid-technologies.com/Reference/Glossary/DTD_Overview.html
30. TRUUGO XML Validator. [Електронний ресурс]. – Доступний з: https://www.truugo.com/xml_validator/ .

31. W3C Recommendation XML Schema Part 0: Primer Second Edition. [Електронний ресурс]. – Доступний з: https://www.w3.org/TR/xmlschema-0/.
32. W3schools: XML Tutorial. XML Schema. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/xml/xml_schema.asp.
33. W3schools: XSD Schema. XML Schema Tutorial. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/xml/schema_intro.asp .
34. W3schools: XPath Tutorial. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/xml/xpath_intro.asp .
35. W3schools: XQuery Tutorial. [Електронний ресурс]. – Доступний з https://www.w3schools.com/xml/xquery_intro.asp.
36. W3schools: XSLT Tutorial. [Електронний ресурс]. – Доступний з: https://www.w3schools.com/xml/xsl_intro.asp.

ДОДАТОК А

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Бази даних та інформаційні системи

ЛАБОРАТОРНА РОБОТА №__

Тема: *«_____».*

Виконав(ла):

Ст. *Прізвище Ім'я*

Група _____

202_