МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРОННИХ ТА
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПРОЕКТУВАННЯ БАЗ ДАНИХ ДЛЯ КОРПОРАТИВНИХ ЗАСТОСУВАНЬ

Методичні рекомендації
до виконання курсового проекту з дисципліни
«Бази даних»
для здобувачів вищої освіти
першого (бакалаврського)
рівня вищої освіти за спеціальністю
121 – "Інженерія програмного забезпечення"

Обговорено і рекомендовано
на засіданні кафедри інформаційних технологій та програмної інженерії
протокол № 2 від 14.09.2020 р.

Чернігів 2020

Проектування баз даних для корпоративних застосувань. Методичні рекомендації до виконання курсового проекту для здобувачів вищої освіти першого (бакалаврського) рівня вищої освіти за спеціальності 121 – «Інженерія програмного забезпечення» /Укл.: І.В. Білоус, М.М. Войцеховська, О.О. Дружинін – Чернігів : НУ «Чернігівська політехніка», 2020. – 81 с., укр. мовою.

Укладачі: Білоус Ірина Володимирівна, кандидат технічних наук, завідувач кафедри інформаційних технологій та програмної інженерії;

Войцеховська Марія Михайлівна, викладач кафедри інформаційних технологій та програмної інженерії;

Дружинін Олександр Олександрович, викладач кафедри інформаційних технологій та програмної інженерії

Відповідальний за випуск: Войцеховська М.М., викладач кафедри інформаційних технологій та програмної інженерії

Рецензент: Риндич Євген Володимирович, доцент кафедри інформаційних та комп'ютерних систем, к.т.н.

ЗМІСТ

ВСТУП

Проектування бази даних (БД) – одне з найбільш складних і відповідальних завдань, пов'язаних із створенням корпоративного застосування (enterprise application). В результаті його вирішення мають бути визначені зміст БД, ефективний для усіх її майбутніх користувачів спосіб організації даних і інструментальні засоби управління даними. Основна мета проектування БД – це скорочення надмірності даних, що зберігаються, а отже, економія об'єму використовуваної пам'яті, зменшення витрат на багатократні операції оновлення надмірних копій та усунення можливості виникнення протиріч через зберігання в різних місцях відомостей про один і той же об'єкт.

Корпоративний застосунок є програмним застосунком, призначеним для управління даними великого об'єму і їх обробки за бізнес правилами, що дозволяє принести певні переваги корпорації (підприємству) при його впровадженні.

Корпоративними застосунками **не** є засоби обробки тексту, регулювання витрати палива в автомобільному двигуні, управління ліфтами та устаткуванням телефонної станції, автоматичного контролю хімічних процесів, а також операційні системи, компілятори, ігри і так далі.

Корпоративний застосунок зазвичай має на увазі необхідність довготривалого (іноді впродовж десятиліть) зберігання даних. Дані часто здатні пережити декілька поколінь програм, призначених для їх обробки, апаратних засобів, операційних систем і компіляторів.

Безліч користувачів звертаються до даних паралельно. Як правило, їх кількість не перевищує сотні, але для систем, розміщених в середовищі Web, цей показник зростає на декілька порядків.

При великих об'ємах даних в застосунку має бути передбачений зрозумілий призначений для користувача інтерфейс.

Корпоративні застосунки рідко існують в ізоляції. Зазвичай вони вимагають інтеграції з іншими системами, побудованими в різний час із застосуванням різних технологій.

Корпоративні застосунки, як правило, є складними програмними системами.

1 АРХІТЕКТУРА КОРПОРАТИВНОГО ЗАСТОСУНКУ. РОЗШАРУВАННЯ СИСТЕМИ

Концепція шарів (layers) – одна із загальновживаних моделей, використовуваних розробниками програмного забезпечення для розділення складних систем на простіші частини. У архітектурі комп'ютерних систем, наприклад, розрізняють шари коду на мові програмування, функцій операційної системи, драйверів пристроїв, наборів інструкцій центрального процесора і внутрішньої логіки чіпів. У середовищі мережевої взаємодії протокол FTP працює на основі протоколу TCP, який, у свою чергу, функціонує "поверх" протоколу IP, розташованого "над" протоколом Ethernet.

Описуючи систему в термінах архітектурних шарів, зручно сприймати складові її підсистеми у вигляді "листкового пирога". Шар більш високого рівня користується службами, що надаються шаром, що пролягає нижче, але той не "обізнаний" про наявність сусіднього верхнього шару. Більше того, зазвичай кожен проміжний шар "приховує" нижній шар від верхнього: наприклад, шар 4 користується послугами шару 3, який звертається до шару 2, але шар 4 не знає про існування шару 2 (не у кожній архітектурі шари настільки "непроникні", але у більшості випадків саме так). На рисунку 1.1 представлена архітектура багатошарового застосунку. Багатошарові застосунки складаються з наступних шарів:

- шар бізнес логіки (Domain) – шар, що описує основні функції застосування, призначені для досягнення поставленої перед ним мети;
- шар сервісів (Service layer) – шар служб визначає межі застосунку і безліч операцій, що надаються їм для інтерфейсних клієнтських шарів коду;
- шар доступу до даних (Datasource layer) або шар інтеграції – завдання цього шару полягає в тому, щоб забезпечити можливість взаємодії застосування з різними компонентами інфраструктури (наприклад, СУБД) для виконання необхідних функцій;
- шар представлення (Presentation layer) – шар, що охоплює все, що має відношення до спілкування користувача з системою.

Розшарування системи на шари надає цілий ряд переваг:

- окремий шар можна сприймати як єдине самодостатнє ціле, не особливо піклуючись про наявність інших шарів;
- можна вибирати альтернативну реалізацію базових шарів;
- залежність між шарами можна звести до мінімуму;
- кожен шар є вдалим кандидатом на стандартизацію;
- створений шар може служити основою для декількох різних шарів більш високого рівня.

Рисунок 1.1 – Багатошарова архітектура корпоративного застосунку

Схема розшарування має і певні недоліки:

- модифікація одного шару часом пов'язана з необхідністю внесення каскадних змін до інших шарів;
- наявність надмірних шарів нерідко знижує продуктивність системи.

Проте найважче при використанні архітектурних шарів – це визначення вмісту і меж відповідальності кожного шару.

2 ПРОЕКТУВАННЯ БАЗИ ДАНИХ

Під проектуванням БД можна розуміти процес створення проекту БД, що призначений для підтримки функціонування підприємства і сприяє досягненню його цілей.

При проектуванні БД, як правило, виконуються три основні етапи:

1) концептуальне (інфологічне) проектування;
2) логічне проектування;
3) фізичне проектування.

2.1 Концептуальне проектування

Концептуальне проектування БД розпочинається із створення концептуальної моделі предметної області (ПрО) підприємства, повністю незалежної від будь-яких деталей реалізації. До останніх відносяться вибір СУБД, склад програм застосунку, використовувана мова програмування, конкретна апаратна платформа, питання продуктивності і будь-які інші фізичні особливості реалізації.

Концептуальна модель ПрО є описом структури і динаміки ПрО, характеру інформаційних потреб користувачів в термінах, зрозумілих користувачеві і не залежних від реалізації БД. Цей опис виражається в термінах не окремих об'єктів ПрО і зв'язків між ними, а їх типів, пов'язаних з ними обмежень цілісності і тих процесів, які призводять до переходу ПрО з одного стану в інший.

На етапі побудови концептуальної моделі ПрО в загальному випадку повинно бути виконано наступне:

- визначення типів сутностей;
- визначення типів зв'язків;
- визначення атрибутів і зв'язування їх з типами сутностей і зв'язків;
- визначення доменів атрибутів;
- визначення атрибутів, що є потенційними і первинними ключами;
- перевірка моделі на відсутність надмірності;
- перевірка відповідності концептуальної моделі конкретним призначеним для користувача транзакціям.

Розглянемо основні підходи до створення концептуальної моделі ПрО.

2.1.1 Функціональний підхід до проектування бази даних

Цей метод реалізує принцип руху «від завдань» і застосовується як комплекс завдань, для обслуговування яких створюється корпоративне застосування. В цьому випадку можна виділити мінімальний необхідний набір об'єктів предметної області, які мають бути описані.

2.1.2 Предметний підхід до проектування баз даних

Предметний підхід до проектування БД застосовується в тих випадках, коли у розробників є чітке уявлення про саму ПрО і про те, яку саме інформацію вони хотіли б зберігати у БД, а структура запитів не визначена або визначена не повністю. Тоді основна увага приділяється дослідженню ПрО і найбільш адекватному її відображенню у БД з урахуванням найширшого спектру інформаційних запитів до неї. У опис предметної області в цьому випадку включаються такі об'єкти і взаємозв'язки, які найбільш характерні і найбільш суттєві для неї.

2.1.3 Проектування з використанням методу «сутність-зв'язок»

Метод «сутність-зв'язок» (Entity-Relationship, ER-method) є комбінацією двох попередніх. Етап концептуального проектування розпочинається з моделювання ПрО. Проектувальник розбиває її на ряд локальних областей, кожна з яких (у ідеалі) включає інформацію, достатню для забезпечення запитів окремої групи майбутніх користувачів або рішення окремої задачі (підзадачі). Кожне локальне представлення моделюється окремо, потім вони об'єднуються.

Вибір локального представлення залежить від масштабів ПрО. Зазвичай вона розбивається на локальні області так, щоб кожна з них відповідала окремому зовнішньому застосунку і містила 6-7 сутностей.

Результатом проектування методом «сутність-зв'язок» є ER-модель. ER-модель представляється за допомогою ER-діаграм, які є графічною нотацією для абстрагування даних у вигляді сутностей, взаємозв'язків і атрибутів.

Сутність – збиральне поняття, деяка абстракція реально існуючого об'єкту, процесу або явища, про яке необхідно зберігати інформацію. Сутності бувають як фізично існуючі (наприклад, Викладач або Постачальник), так і абстрактні (наприклад, Іспит або Діагноз).

Для сутностей розрізняють тип сутності і екземпляр. Тип характеризується ім'ям і списком властивостей, а екземпляр – конкретними значеннями властивостей.

Типи сутностей можна класифікувати як сильні і слабкі. Сильні сутності існують самі по собі, а існування слабких сутностей залежить від існування сильних. Наприклад, читач бібліотеки – сильна сутність, а абонемент цього читача – слабка, яка залежить від наявності відповідного читача. Слабкі сутності називають підлеглими (дочірніми), а сильні – базовими (основними, батьківськими).

Для кожної сутності вибираються властивості (атрибути). Розрізняють:

– ідентифікуючі і описові атрибути. Ідентифікуючі атрибути мають унікальне значення для сутностей цього типу і є потенційними ключами. Вони дозволяють однозначно розпізнавати екземпляри сутності. З потенційних ключів вибирається один первинний ключ (primary key, PK). В якості PK зазвичай вибирається потенційний ключ, по якому частіше відбувається звернення до екземплярів запису. Крім того, PK повинен включати до свого складу мінімальну необхідна для

ідентифікації кількість атрибутів. Інші атрибути називаються описовими і містять в собі дані, що описують властивості;

- складені і прості атрибути. Простий атрибут складається з одного компонента, його значення неділиме. Складений атрибут є комбінацією декількох компонентів, що можливо, належать різним типам даних (наприклад, ПІБ або адреса). Рішення про те, використати складений атрибут або розбивати його на компоненти, залежить від характеру його обробки і формату призначеного для користувача представлення цього атрибуту;
- однозначні і багатозначні атрибути (можуть мати відповідно одне або багато значень для кожного екземпляра суті);
- основні і похідні атрибути. Значення основного атрибуту не залежить від інших атрибутів. Значення похідного атрибуту обчислюється на основі значень інших атрибутів (наприклад, вік студента обчислюється на основі дати його народження і поточної дати).

Специфікація атрибуту складається з його назви, вказівки типу даних і опису обмежень цілісності – безлічі значень (чи домена), які може приймати цей атрибут.

Далі здійснюється специфікація зв'язків усередині локального представлення. Зв'язки можуть мати різний змістовний сенс (семантику). Розрізняють зв'язки типу «сутність-сутність», «сутність-атрибут» і «атрибут-атрибут» для відносин між атрибутами, які характеризують одну і ту ж суть або один і той же зв'язок типу «сутність-сутність».

Кожен зв'язок характеризується ім'ям, обов'язковістю і мірою.

Ім'я зв'язку – це дієслівна фраза, що характеризує відношення між батьківською і дочірньою сутностями.

Міра (потужність) зв'язку – це відношення числа сутностей, що беруть участь в утворенні зв'язку. Наприклад, «один до одного», «один до багатьох», «багато до багатьох».

Розрізняють факультативні і обов'язкові зв'язки. Якщо знову породжений об'єкт одного типу має бути пов'язаним з об'єктом іншого типу, то між цими типами об'єктів існує обов'язковий зв'язок. Інакше зв'язок є факультативним. Для факультативного зв'язку його міра може дорівнювати нулю, тобто екземпляр сутності можна зв'язати з 0, 1 або декількома екземплярами іншої сутності. Для обов'язкового зв'язку міра не може дорівнювати нулю.

Розглянемо приклад побудови концептуальної моделі для ПО «Деканат ВНЗ». Припустимо, що для деканату одного з факультетів учбового закладу вимагається розробити корпоративний застосунок для автоматизації аналізу і обліку даних про студентів, що навчаються на факультеті, і їх успішності в поточній сесії. В даному випадку основним об'єктом ПрО є студент, тому виділимо набір властивостей цього об'єкту. Студент характеризується атрибутами:

- номер залікової книжки;
- прізвище ім'я і по батькові;

- дата народження;
- стать;
- курс;
- номер учбової групи;
- спеціальність;
- кількість іспитів, що складаються в сесію;
- оцінки, отримані на іспитах.

Приведений набір властивостей-характеристик об'єкту «студент», природно, може бути розширений.

Необхідно передбачити наступні обмеження на інформацію:

- курси мають значення від 1 до 6;
- в учбовій групі може вчитися від 15 до 32 студентів;
- номер учбової групи змінюється від 001 до 999;
- кількість іспитів, що складаються в сесію, може змінюватися від 2 до 7;
- оцінка по іспиту може бути від 0 до 100 балів.

Передбачається, що з цим корпоративним застосунку працюватимуть співробітники деканату, і вони повинні мати можливість вирішувати з його допомогою наступні завдання:

- введення і редагування даних про студентів;
- перегляд даних про студентів у відсортованому вигляді;
- визначення чисельності контингенту в кожній учбовій групі, на кожному курсі і в цілому на факультеті;
- підготовка до виведення на друк списків студентів;
- обчислення середнього балу кожного студента і виведення списків студентів по групах в алфавітному порядку прізвищ або по убуванню середнього балу;
- обчислення середнього балу в кожній учбовій групі, на кожному курсі і в цілому по факультету;
- визначення відсотка успішності в кожній учбовій групі, на кожному курсі і в цілому по факультету;
- перегляд і підготовка до виведення списків боржників (з вказівкою кількості заборгованостей) по курсах і групах;
- перегляд і підготовка до виведення результатів складання іспиту студентами заданої групи по заданому предмету;
- визначення успішності студентів факультету по заданому предмету;
- перегляд і підготовка до виведення списків викладачів, що читають дисципліни на заданій кафедрі або групам заданої спеціальності;
- перегляд і підготовка до виведення списків студентів, що претендують на отримання стипендії, з відміткою про те, яка стипендія покладається: звичайна або підвищена;
- організація пошуку студентів по прізвищу або частині прізвища.

Таким чином, з приведеного аналізу окрім об'єкту «студент» можна виділити наступні сутності:

- кафедра;

– спеціальність;
– група;
– студент;
– іспит;
– дисципліна;
– викладач.

Між сутностями встановлені такі зв'язки:

– СТУДЕНТ (вчитися в) ГРУПА (міра зв'язку N :1);
– СПЕЦІАЛЬНІСТЬ (відноситься до) КАФЕДРА (міра зв'язку N :1);
– ВИКЛАДАЧ (працює на) КАФЕДРА (міра зв'язку N :1);
– ГРУПА (навчається по) СПЕЦІАЛЬНІСТЬ (міра зв'язку N :1);
– ІСПИТ (здає) СТУДЕНТ (міра зв'язку N :1);
– ІСПИТ (проводиться по) ДИСЦИПЛІНА (міра зв'язку N :1);
– ІСПИТ (приймає) ВИКЛАДАЧ (міра зв'язку N :1).

На рисунку 2.1 приведена ER-діаграма в нотації П. Чена, яка представляє концептуальну модель ПрО «Деканат ВНЗ». Базові сутності і обов'язкові зв'язки виділені напівжирним шрифтом.

Ідентифікуючі атрибути виділені підкресленням.

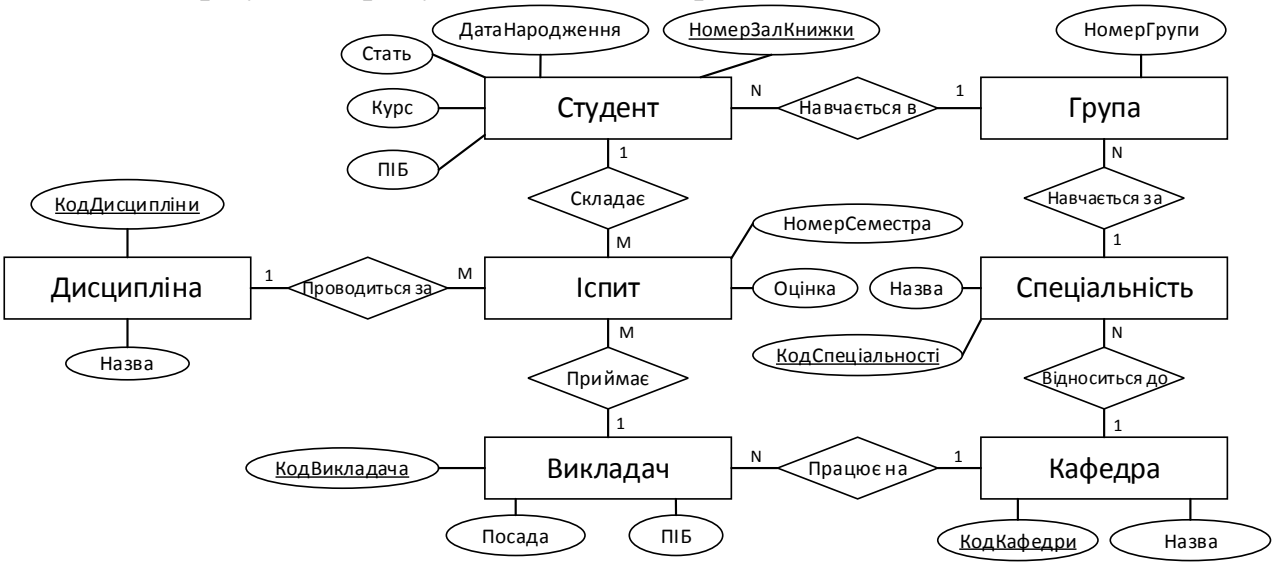

Рисунок 2.1 – Концептуальна модель ПрО «Деканат ВНЗ»

2.2 Логічне проектування бази даних

На етапі логічного проектування розробляється логічна структура БД, що відповідає логічній моделі ПрО. Рішення цієї задачі істотно залежить від моделі даних, підтримуваної вибраної СУБД.

Результатом виконання цього етапу є схеми БД концептуального і зовнішнього рівнів архітектури, описані на мовах визначення даних (DDL, Data Definition Language), підтримуваних цією СУБД.

2.2.1 Проектування реляційних баз даних

Як правило, при використанні СУБД в корпоративних застосунках, після етапу концептуального моделювання даних виконується перетворення ER-моделі в реляційну модель, тобто сутностям і зв'язкам ER-моделі ставляться у відповідність відношення, або таблиці.

Для наведеного вище прикладу ПрО «Деканат ВНЗ» її логічна модель може виглядати так, як показано на рисунку 2.2.

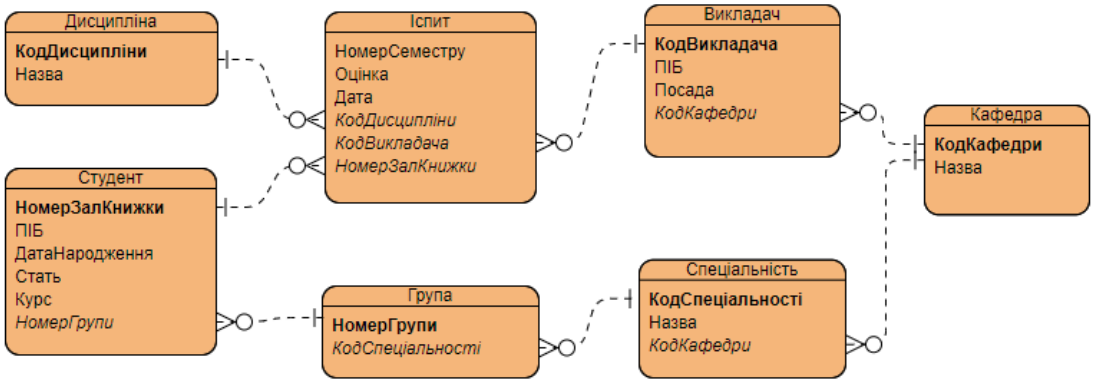

Рисунок 2.2 – Схема БД ПрО «Деканат ВНЗ»

Зв'язок типу 1 : N (один-до-багатьох) між відношеннями реалізується через зовнішній ключ. Ключ вводиться для того відношення, до якого здійснюється множинний зв'язок. Наприклад, Іспит, Студент, Група, Спеціальність, Кафедра, Викладач.

Зв'язок типу N : M (багато-до-багатьох) між відношеннями реалізується через допоміжне відношення, яке є з'єднанням первинних ключів відповідних відношень.

Кожне реляційне відношення відповідає одній сутності (об'єкту ПрО) і в нього вносяться усі атрибути сутності. Для кожного відношення необхідно визначити первинний ключ і зовнішні ключі (якщо вони є). У тому випадку, якщо базове відношення не має потенційних ключів, вводиться сурогатний первинний ключ, який не несе смислового навантаження і служить тільки для ідентифікації записів. Для наведеного прикладу сурогатним ключем буде атрибут «код_кафедри» для відношення «Кафедра», для відношення «Дисципліна» – «код_дисципліни».

Потенційним ключем відношення «Студент» є атрибут «номер_зал_книжки». Оскільки значення цього атрибуту обов'язкові і унікальні, то його слід вибрати в якості первинного ключа. Викладача можна ідентифікувати по атрибуту «код_викладача», який є його табельним номером. Ідентифікація групи проводитися по атрибуту «номер_групи».

Потенційними ключами відношення «Іспит» являються комбінації первинних ключів відповідних базових відношень «Дисципліна», «Студент», «Викладач».

Опис відношень приведені в таблицях 2.1–2.7. Для кожного відношення вказані атрибути з їх внутрішньою назвою, типом і довжиною.

Типи даних позначаються так: N – числовий, C – символьний, D – дата.

Таблиця 2.1 – Опис відношення Дисципліна (Subject)

Зміст поля	Ім'я поля	Тип даних (довжина)	Примітка
Код дисципліни	id_subj	N	первинний ключ
Назва	name_subj	C (20)	обов'язкове поле

Таблиця 2.2 – Опис відношення Іспит (Exam)

Зміст поля	Ім'я поля	Тип даних (довжина)	Примітка
Номер семестру	num_sem	N	первинний ключ
Оцінка	mark	C (20)	обов'язкове поле
Дата іспиту	data	D	обов'язкове поле, значення за умовчанням поточна дата
Код дисципліни	num_subj	N	зовнішній ключ (до Дисципліна)
Код викладача	num_teach	N	зовнішній ключ (до Викладач)
Номер залікової книжки студента	num_zach	N	зовнішній ключ (до Студент)

Таблиця 2.3 – Опис відношення Викладач (Teacher)

Зміст поля	Ім'я поля	Тип даних (довжина)	Примітка
Код викладача	num_teach	N	первинний ключ
ПІБ	fio	C (30)	обов'язкове поле
Посада	post	C (30)	обов'язкове поле
Код кафедри	num_kaf	N	зовнішній ключ (до Кафедра)

Таблиця 2.4 – Опис відношення Кафедра (Kaf)

Зміст поля	Ім'я поля	Тип даних (довжина)	Примітка
Код кафедри	num_kaf	N	первинний ключ

Назва	name	C (100)	обов'язкове поле

Таблиця 2.5 – Опис відношення Спеціальність (Spec)

Зміст поля	Ім'я поля	Тип даних (довжина)	Примітка
Код спеціальності	num_spec	N	первинний ключ
Назва	name	C (50)	обов'язкове унікальне поле
Код кафедри	num_kaf	N	зовнішній ключ (до Кафедра)

Таблиця 2.6 – Опис відношення Група (Groups)

Зміст поля	Ім'я поля	Тип даних (довжина)	Примітка
Номер групи	num_grp	N	первинний ключ
Код спеціальності	num_spec	N	зовнішній ключ (до Кафедра)

Таблиця 2.7 – Опис відношення Студент (Student)

Зміст поля	Ім'я поля	Тип даних (довжина)	Примітка
Номер залікової книжки	num_zach	N	первинний ключ
ПІБ	fio	C (50)	обов'язкове поле
Дата народження	date_brth	D	обов'язкове поле
Стать	sex	C (1)	обов'язкове поле
Курс	course	N	обов'язкове поле, значення за замовчанням 1
Номер групи	num_grp	N	Зовнішній ключ (до Група)

На рисунку 2.3 приведена уточнена схема БД.

Якщо отримані відношення ненормалізовані, то для них необхідно провести процедуру нормалізації. У наведеному прикладі її виконувати не треба.

2.2.1.1 Обмеження посилальної цілісності

Для зовнішніх ключів необхідно визначити каскадні обмеження посилальної цілісності. За допомогою цих обмежень можна визначати дії, які СУБД робитиме, коли користувач спробує видалити або відновити ключ, на

який вказують ще існуючі зовнішні ключі. Допустимі варіанти: RESTRICT, CASCADE, SET NULL, SET DEFAULT.

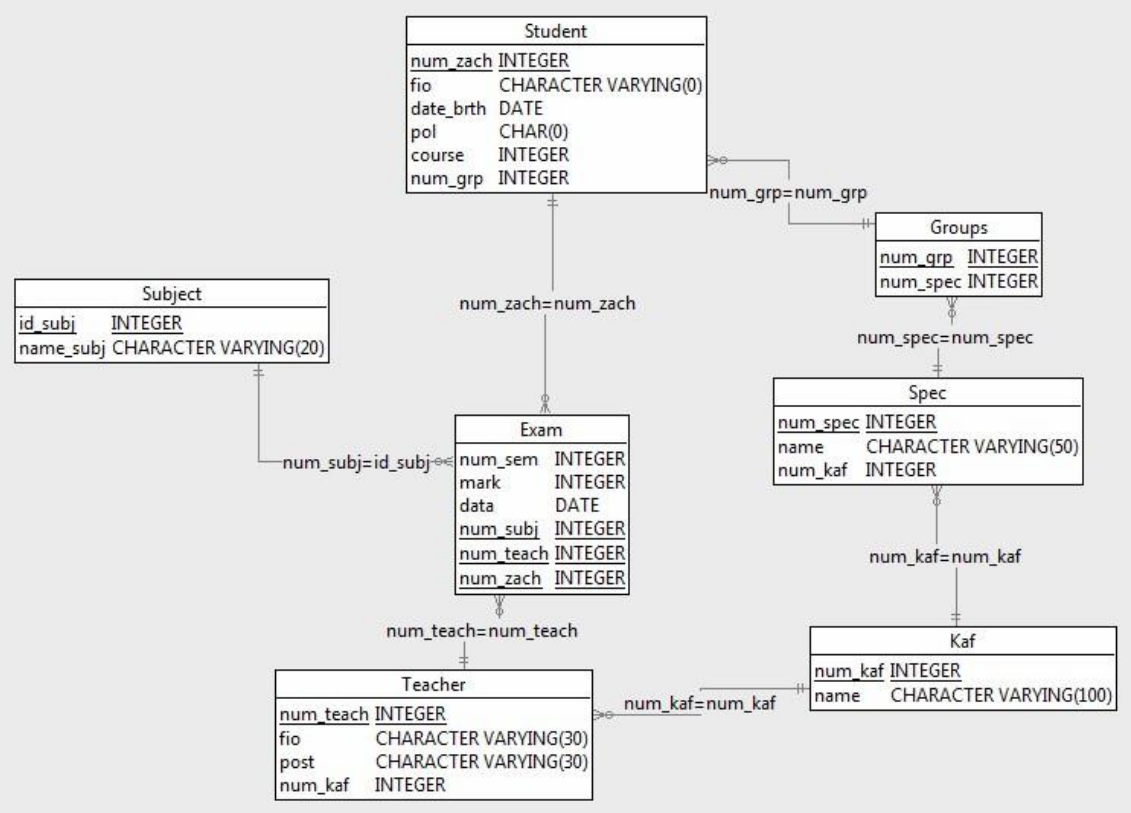

Рисунок 2.3 – Уточнена схема БД предметної області «Деканат ВНЗ»

RESTRICT – забороняє видаляти рядок (чи змінювати значення первинного ключа посилального рядка) батьківської таблиці, якщо на цей рядок є посилання з дочірньої таблиці.

CASCADE – видалення рядка у батьківській таблиці призводить до видалення усіх пов'язаних з нею рядків дочірньої таблиці. Зміна значення посилального ключа у батьківській таблиці призводить до відповідної зміни значень зовнішніх ключів дочірньої таблиці.

SET NULL – видалення рядка (чи зміна значення посилального ключа) батьківської таблиці призводить до установки в значення NULL усіх зовнішніх ключів дочірньої таблиці, які посилаються на видалені (чи змінені) значення посилального ключа батьківської таблиці.

SET DEFAULT – видалення рядка (чи зміна значення посилального ключа) батьківської таблиці призводить до установки в значення за умовчанням усіх зовнішніх ключів дочірньої таблиці, які посилаються на видалені (чи змінені) значення посилального ключа батьківської таблиці.

2.2.1.2 Додаткові обмеження цілісності

Перерахуємо обмеження цілісності, які не вказані в таблицях 2.1–2.7:

1) поле course таблиці Student може набувати значень від 1 до 6;
2) в учбовій групі може вчитися від 15 до 32 студентів;
3) область значень атрибуту sex відношення Student – символи 'ч' і 'ж';

4) відношення Exam не має первинного ключа, але комбінація значень (num_subj, num_teach, num_zach) унікальна;
5) номер учбової групи змінюється від 001 до 999;
6) кількість іспитів, що складаються в сесію, може змінюватися від 2 до 7;
7) оцінка по іспиту може бути від 0 до 100 балів.

Обмеження 2 не можна реалізувати в схемі відношень. У реальних БД подібні обмеження цілісності реалізуються програмно (через зовнішнє застосування або спеціальну процедуру контролю даних).

2.2.1.3 Групи користувачів і права доступу

При проектуванні БД необхідно визначити групи користувачів, що мають доступ до даних БД. Користувачами БД в наведеному прикладі можуть бути: адміністратор БД, декан, секретар деканату і викладач. Опишемо для кожної групи користувачів права доступу до кожної таблиці і до кожного поля (атрибуту).

Адміністратор БД має доступ до усіх даних (по запису), може змінювати структуру бази даних і зв'язку між відношеннями, встановлює права доступу для усіх інших груп.

Декан має доступ до усіх даних по запису і читанню.

Секретар деканату має доступ по читанню до усіх стосунків БД і доступ по запису до відношень Student, Groups.

Викладач має доступ по читанню до таблиць Student (fio, num_grp, num_zach) і доступ по запису до таблиці Exam. Так само, він може проглянути список дисциплін (name_subj) і викладачів (fio), які їх читають. Якщо з однієї або декількох таблиць користувачеві доступні не усі поля, то зручно створити представлення і встановити до нього доступ по читанню (чи запису у разі представлення, що модифікується).

Перераховані права доступу зручно представити у вигляді таблиці:

Таблиця 2.8 – Права доступу користувачів до таблиць і представлень БД

5555	Student	Groups	Spec	Kaf	Teacher	Exam	Subject	s_view	s_t_view
Адміні-стратор	S, U, D, I	S, U, D, I	S, U, D, I	S, U, D, I	S, U, D, I	S, U, D, I	S, U, D, I	-	-
Декан	S, U, D, I	S, U, D, I	S, U, D, I	S, U, D, I	S, U, D, I	S, U, D, I	S, U, D, I	-	-
Секретар	S, U, D, I	S, U, D, I	S	S	S	S	S	-	-

Викладач	-	-	-	-	-	S, U, D, I	-	S	S

У таблиці S – select, U – update, D – delete, I – insert.

2.3 Фізичне проектування бази даних

Етап фізичного проектування полягає в зв'язуванні логічної структури БД і фізичного середовища зберігання з метою найбільш ефективного розміщення даних, тобто відображенні логічної структури БД в структуру зберігання. Вирішується питання розміщення даних, що зберігаються, в просторі пам'яті, вибору ефективних методів доступу до різних компонент «фізичної» БД. Результати цього етапу документуються у формі схеми зберігання на мові визначення даних (DDL). Прийняті на цьому етапі рішення роблять визначальний вплив на продуктивність системи.

Нижче наводяться фрагменти опису схеми БД на DDL.

Відношення Student (студенти) :

```
CREATE TABLE Student  (
        num_zach INTEGER NOT NULL, fio CHARACTER VARYING (30) NOT NULL
        , date_brth DATE NOT NULL
        , sex CHAR (1) NOT NULL check (sex in ('ж','м'))
        , course INTEGER DEFAULT 1 NOT NULL
        , num_grp INTEGER NOT NULL
        , PRIMARY KEY (num_zach)
        ,CONSTRAINT "FK_Студент_1" FOREIGN KEY (num_grp) REFERENCES Groups (num_grp)
);
```

Відношення Groups (групи) :

```
CREATE TABLE Groups  (
num_grp INTEGER NOT NULL   , num_spec INTEGER NOT NULL
        , PRIMARY KEY (num_grp)
        ,  CONSTRAINT "FK_Група_1" FOREIGN KEY (num_spec) REFERENCES Spec
);
```

Інші відношення описуються аналогічно.

Однією з найважливіших складових проекту бази даних є розробка засобів захисту БД. Захист даних має два аспекти: захист від збоїв і захист від несанкціонованого доступу. Для захисту від збоїв розробляється стратегія резервного копіювання. Для захисту від несанкціонованого доступу кожному користувачеві доступ до даних надається тільки відповідно до його прав доступу.

Права доступу користувачів, описані в п. 2.2.1.3, надаються за допомогою команди GRANT. Розглянемо для прикладу права секретаря деканату secretary, який приймає і обслуговує замовлення. Права доступу до стосунків Student, Groups, Spec, Kaf, Teacher, Exam, Subject можуть бути описані таким чином:

```
grant select, delete, insert, update on Student, Groups to secretary;
grant select on Spec, Kaf, Teacher, Exam, Subject to secretary;
```

Для реалізації часткового доступу до відношення ***Students*** слід створити відповідне представлення і надати доступ до цього представлення:

```
create view s_view (num_zach, fio, num_grp) as
select num_zach, fio, num_grp from Student;
grant select on s_view to teacher;
```

Проаналізувавши завдання з пункту 2.1.3, які повинні вирішувати співробітники деканату за допомогою корпоративного застосунку, можна зробити висновок, що для підвищення ефективності роботи з даними необхідно створити індекси для усіх зовнішніх ключів (і усіх первинних ключів, якщо вибрана СУБД не створює їх автоматично), і полів, часто використовуваних при вибірці даних. Наведемо приклади створення індексів :

```
create index s_fio on Students (fio);
create index s_num_grp on Students (num_grp);
create index subj_name on Subject (name_subj);
create unique index e_uni_ind on Exam (num_subj, num_teach, num_zach);
```

3 ПРОЕКТУВАННЯ І РОЗРОБКА ШАРУ БІЗНЕС-ЛОГІКИ НА МОВІ JAVA

Логіка домена (бізнес-логіка або логіка предметної області) описує основні функції застосування, призначені для досягнення поставленої перед ним мети. До таких функцій відносяться обчислення на основі даних, що вводяться і, перевірка усіх елементів даних і обробка команд, що поступають від шару представлення, а також передача інформації шару джерела даних.

Часто у рамках застосунку передбачають декілька варіантів реалізації кожної з трьох категорій логіки. Наприклад, застосунок, орієнтований на використання як інтерфейсних засобів товстого клієнта, так і командного рядка, може (і, ймовірно, повинно) бути оснащене двома відповідними версіями логіки представлення. З іншого боку, різним базам даних можуть відповідати численні шари джерел даних. На окремі "пакети" може бути поділений навіть шар бізнес-логіки (скажімо, за ситуації, коли алгоритми розрахунків залежать від типу джерела даних).

Хоча три основні шари – представлення, бізнес-логіка і джерело даних – можна виявити у будь-якому корпоративному застосунку, спосіб їх розділення залежить від міри складності цього застосунку. Простий сценарій добуття порції інформації з бази даних і відображення її в контексті Web-сторінки можна описати однією процедурою. Але все одно варто виділити в нім три шари – нехай навіть, як в цьому випадку, розподіливши функції кожного шару по різних підпрограмах. При ускладненні застосунку це дало б можливість рознести код шарів по окремих класах, а пізніше розбити безліч класів на пакети. Форма розшарування може бути довільною, але у будь-якому корпоративному застосунку шари мають бути ідентифіковані.

Найскладнішим в роботі над бізнес-логікою являється, ймовірно, вибір того, що саме і як слід відносити до того або іншого шару.

3.1 Взаємодія між архітектурними шарами

Іноді шари організовують так, щоб бізнес-логіка повністю приховувала джерело даних від представлення. Частіше, проте, код представлення може звертатися до джерела даних безпосередньо. Хоча такий варіант менш бездоганний з теоретичної точки зору, в практичному відношенні він нерідко зручніший і доцільніший: код представлення може інтерпретувати команду користувача, активізувати функції джерела даних для витягання відповідних порцій інформації з бази даних, звернутися до засобів бізнес-логіки для аналізу цієї інформації і здійснення необхідних розрахунків і тільки тоді відобразити відповідну картинку на екрані.

Окрім необхідності розділення на шари, існує правило, що стосується взаємовідношення шарів: залежність бізнес-логіки і джерела даних від рівня представлення не допускається. Іншими словами, в програмному тексті застосунку не повинно бути викликів функцій представлення з коду бізнес-логики або джерела даних. Зв'язок між бізнес-логікою і джерелом даних, проте,

не такий однозначний і багато в чому визначається вибором типових рішень для архітектури джерела даних.

Бізнес-логіка зазвичай схильна до частих змін, тому дуже важлива можливість простої модифікації і тестування цього шару коду. Звідси слідує наполеглива необхідність знижувати міру залежності моделі предметної області від інших шарів системи.

3.2 Реалізація бізнес-логіки

У своїх найгірших проявах бізнес-логіка буває надзвичайно складною, з множиною правил і умов, що обумовлюють різні варіанти використання і особливості поведінки системи. Доцільніше використати об'єктно-орієнтований підхід для створення бізнес-логики, створюючи мережу взаємозв'язаних об'єктів, кожен з яких представляє деяку осмислену суть. Одні об'єкти покликані імітувати елементи даних, якими оперують в цій області, а інші повинні формалізувати ті або інші бізнес-правила. Функції тісно поєднуються з даними, якими вони маніпулюють.

Об'єктно-орієнтована модель предметної області часто нагадує схему відповідної бази даних, хоча між ними все ще залишається безліч відмінностей. У моделі предметної області змішуються дані і функції, допускаються багатозначні атрибути, створюються складні мережі асоціацій і використовуються зв'язки спадкоємства.

3.3 Виклик процедур, що зберігаються, в Java

Виклик процедур, що зберігаються, з Java застосунку практично нічим не відрізняється від виконання звичайного SQL запиту на вибірку даних. Інтеграція з JDBC – це перевага для процедур, що зберігаються, оскільки для того, щоб її викликати із застосунку, не треба змінювати класи або використовувати будь-які конфігураційні файли.

JDBC підтримує виклик процедур, що зберігаються, за допомогою класу CallableStatement. Цей клас є підкласом класу PreparedStatement. Рядок, який подається в якості параметра методу prepareCall(), – це специфікація виклику процедури. Вона визначає ім'я процедури, що викликається, і символи '?', які визначають необхідні параметри. Можливі два варіанти специфікації. Якщо функція:

1. Повертає значення

{?= call <procedure - name>[(<arg1>,<arg2>, ...)]}

2. Не повертає значення

{call <procedure - name>[(<arg1>,<arg2>, ...)]}

При виконанні процедур, що зберігаються, можливе виникнення помилкових ситуацій, в цьому випадку буде згенеровано виключення SQLException.

Розглянемо варіанти виклику процедур, що зберігаються.

Приклад 1. Функція з декількома вхідними параметрами і не повертає значень: Додати відомості про нового студента: insStudent (st_no integer, st_name character varying)

```
connection.setAutoCommit (false);
CallableStatement proc = connection.prepareCall ("{call insStudent ( ?, ? ) }"); proc.setInt (1, 958977); proc.setString (2, "Іванов"); proc.execute ();
```

У цій функції використовується два параметри. Для привласнення параметрам значень використовуються методи setInt (1, 958977) для завдання значення типу int першому параметру і setString (2, "Іванов") для завдання значення типу String другому параметру.

Приклад 2. Функція не має вхідних параметрів і повертає одне значення: Підрахувати суму витрат на стипендію усіх студентів — getSumStipAll ().

```
connection.setAutoCommit (false);
CallableStatement proc = connection.prepareCall ("{ ? = call getSumStipAll ()}");
proc.registerOutParameter (1, Types.REAL); proc.execute ();
float sum = proc.getFloat (1);
```

За допомогою методу registerOutParameter () реєструється SQL-тип вихідного параметра. У Java замість типу real використовується тип float, і тому для отримання результату функції використовується метод getFloat ().

Приклад 3. Функція з одним вхідним параметром і повертає одне значення: Підрахувати суму витрат на стипендію усіх студентів групи X, де X - параметр, що задає назву групи : getStipGroup (gr_name character varying)

```
connection.setAutoCommit (false);
CallableStatement proc = connection.prepareCall ("{ ? = call getStipGroup (?) }"); proc.registerOutParameter (1, Types.REAL); proc.setString (2, "ПІ-171"); proc.execute ();
float sum = proc.getFloat (1);
```

У цій функції використовується один параметр. Для його встановлення використовується метод setString (2, "ПІ-171"), де значення 2 вказує, що другому параметру процедури, що зберігається, тобто першому параметру функції getStipGroup необхідно присвоїти строкове значення.

Приклад 4. Функція з одним вхідним параметром і повертає безліч елементів (записів) таблиці : getStudGroup (gr_name character varying).

```
connection.setAutoCommit (false);
CallableStatement proc = connection.prepareCall ("{ ? = call getStudGroup (?) }"); proc.registerOutParameter (1, Types.OTHER); proc.setString (2, "ПІ-171"); proc.execute ();

ResultSet rs (ResultSet) proc.getObject (1); while (rs.next ()){
    String name = rs.getString (3);      float stip = proc.getFloat (4);
    System.out.println ("Студент " + name + " стипендія " + stip);
} rs.close ();
```

Результатом цієї функції є безліч рядків з таблиці Student. Безліч рядків представляється об'єктом класу ResultSet. За допомогою методу registerOutParameter (1, Types.OTHER) реєструється тип вихідного параметра як Types.OTHER. У Java для зберігання об'єктів цього типу використовується клас Object. Після виклику процедури, що зберігається, необхідно провести низхідне перетворення до типу ResultSet. Далі використовується стандартний спосіб обходу записів об'єкту ResultSet.

4 ПРОЕКТУВАННЯ І РОЗРОБКА ШАРУ СЕРВІСІВ

4.1 Призначення шару. Переваги і недоліки використання

Корпоративні застосунки зазвичай мають на увазі застосування різного роду інтерфейсів до даних, що зберігаються, і логіки, що реалізовується, – завантажувачів даних, інтерфейсів користувача, шлюзів інтеграції і так далі. Незважаючи на відмінності в призначенні, подібні інтерфейси часто потребують одних і тих же функцій взаємодії із застосунком для маніпулювання даними і виконання бізнес-логіки. Функції можуть бути дуже складними і здатними включати транзакції, що охоплюють численні ресурси, а також операції по координації реакцій на дії. Опис логіки взаємодії в кожному окремо взятому інтерфейсі користувача зв'язаний з багатократним повторенням одних і тих же фрагментів коду.

Шар сервісів визначає межі застосунку і безліч операцій, що надаються їм для інтерфейсних клієнтських шарів коду. Він інкапсулює бізнес-логіку застосунку, управляє транзакціями і координує реакції на дії.

Багато проектувальників люблять розносити бізнес-логіку по двох категоріях: логіка домена (domain logic) має справу тільки з предметною областю як такою (прикладом можуть служити стратегії обчислення зарахованого доходу за контрактом), а логіка застосунку (application logic) описує сферу відповідальності застосунку (скажімо, повідомляє користувачів і сторонні застосунки про протікання процесу обчислення доходів).

Перевагою використання шару сервісів є можливість визначення набору загальних операцій, доступних для застосування різним клієнтам системи, таким як веб-інтерфейс і застосування товстого клієнта, і координація відгуків застосування на виконання кожної операції.

4.2 Реалізація шару сервісів

Існує декілька способів реалізації шару сервісів, і багато в чому вони залежать від способів реалізації бізнес-логіки. Для забезпечення можливості підключення різних клієнтів і для розділення логіки домена від логіки застосунку шар сервісів реалізується як безліч "товстих" класів, які безпосередньо утілюють в собі логіку застосунку, але за бізнес-логікою звертаються до класів домена. Операції, що надаються клієнтам шару сервісів, реалізуються у вигляді сценаріїв, що створюються групами в контексті класів, кожен з яких визначає деякий фрагмент відповідної логіки. Подібні класи формують "служби" застосування (у назвах службових типів прийнято вживати суфікс "Service"). Шар сервісів і містить в собі ці прикладні класи. Нижче наведений приклад класу шару сервісів (лістинг 4.1).

Лістинг 4.1 – Приклад класу шару сервісів

```
class StudentServiceImpl {
function void moveStudent (Student s, Group g) {
Transactio tx = new Transaction ();
tx.start ();
// Знайти поточну групу
// Видалити студента з поточної групи
// Додати студента в групу g
tx.commit ();
}
}
```

4.3 Визначення необхідних служб і операцій

Встановити, які операції мають бути розміщені в шарі сервісів, зовсім не складно. Це визначається потребами клієнтів шару сервісів, першою (і найбільш важливою) з яких зазвичай являється призначений для користувача інтерфейс. Як відомо, він призначений для підтримки варіантів використання застосунку, тому в якості відправної точки для визначення набору операцій шару сервісів повинен розглядатися призначений для користувача інтерфейс застосунку.

Більшість варіантів використання корпоративних застосунків складають прості операції CRUD (create, read, update, delete – створити, прочитати, відновити, видалити) над об'єктами домена: створити екземпляр, зчитати колекцію екземлярів або відновити. На практиці варіанти використання CRUD практично завжди повністю відповідають операціям шару сервісів.

Створення, оновлення або видалення в застосунку об'єкту домена, не кажучи вже про перевірку правильності його вмісту, вимагає відправки повідомлень користувачам або іншим інтегрованим застосуванням.

Координацією відгуків і займаються операції шару сервісів.

5 ПРОЕКТУВАННЯ І РОЗРОБКА ШАРУ ІНТЕГРАЦІЇ

5.1 Призначення шару

Роль шару інтеграції (шару джерела даних або шар доступу до даних) полягає в тому, щоб забезпечити можливість взаємодії застосунку з різними компонентами інфраструктури для виконання необхідних функцій. Головна складова подібної проблеми пов'язана з підтримкою діалогу з базою даних – у більшості випадків реляційною. Однією з найсерйозніших причин успіху реляційних систем є підтримка ними SQL – найбільш стандартизованої мови комунікацій з базою даних.

5.2 Реалізація шару інтеграції

Спосіб реалізації шару інтеграції залежить від взаємодії бізнес-логіки з базою даних. Вибір, який робиться на цьому етапі, має далекосяжні наслідки і відмінити його буває важко або навіть неможливо. Тому він заслуговує найбільш ретельного осмислення. Нерідко подібними рішеннями якраз і обумовлюються варіанти компонування бізнес-логіки.

Розумніше відособити код SQL від бізнес-логіки, розмістивши його в спеціальних класах. Вдалий спосіб організації подібних класів полягає в "копіюванні" структури кожного об'єкта бази даних в окремому класі, який формує шлюз, що підтримує можливості звернення до таблиці. Тепер основному коду застосунку немає необхідності що-небудь "знати" про SQL, а усі SQL-операції зосереджуються в компактній групі класів.

Вдаліший варіант полягає в тому, щоб ізолювати модель предметної області від бази даних, поклавши на проміжний шар усю повноту відповідальності за відображення об'єктів домена в об'єкти бази даних. Подібний перетворювач даних обслуговує усі операції завантаження і збереження інформації, що ініціюються бізнес-логікою, і дозволяє незалежно варіювати як модель предметної області, так і схему бази даних. Це найбільш складне з архітектурних рішень, що забезпечують відповідність між об'єктами застосунку і реляційними структурами, але його безперечна перевага полягає в повному відособленні двох шарів.

5.3 ORM

На сьогодні Java розробники можуть скористатися вже наявними засобами: серіалізація, JDBC, засоби об’єктно-реляційного відображення, об'єктні бази даних і EJB. Кожен з цих засобів має свої сфери застосування і, отже, деякі недоліки. JDO дозволяє усунути ці недоліки і забезпечує велику прозорість.

Серіалізація – це вбудований Java-механізм, що забезпечує перетворення об'єктів в послідовність байт, які можуть бути збережені у файл або передані по

мережі. Серіалізація дуже проста у використанні, але і дуже обмежена. При використанні серіалізації об'єкт зберігається як одне ціле. Вона не підтримує транзакції, а також використання одного і того ж серіалізованого об'єкту в різних потоках або програмах без виникнення конфліктів між ними.

Java Database Connectivity (JDBC). JDBC є набором класів і інтерфейсів, що забезпечують роботу з реляційними базами даних. В порівнянні з серіалізацією JDBC має безліч переваг: забезпечує роботу з великою кількістю даних, підтримує транзакції, спільний доступ до інформації і підтримує власну мову запитів SQL. Але при цьому JDBC має і свої недоліки. Так, наприклад, JDBC набагато складнішим у використанні, ніж серіалізація. JDBC спочатку не призначався для зберігання об'єктів і, отже, робить неможливим використання об’єктно-орієнтованого програмування для тієї частини коду, яка забезпечує роботу зі сховищем даних.

Об'єктно-реляційне відображення. Деякі компанії розробили власні засоби відображення об'єктних структур на реляційні бази даних. Такі продукти дозволяють використати об'єктно-орієнтовані концепції для розробки, але вони мають настільки різні API, що не представляється можливим використати продукт однієї компанії замість іншої.

Об'єктні бази даних. Об'єктні бази даних були спеціально розроблені для зберігання об'єктів і повністю вписуються в концепцію об'єктно-орієнтованого програмування. Object Database Management Group (ODMG) була створена для розробки єдиного API для роботи з такими базами. Проте багато постачальників баз даних все ще не наважуються перейти з реляційної системи, що добре себе зарекомендувала, на об'єктно-орієнтовану. Так само менше число засобів аналізу даних доступно для об'єктних баз і дуже велика кількість даних вже збережена в реляційних базах. З цих причин, а також з безлічі інших, об'єктні бази даних поки що не знайшли такого широкого застосування, на яке сподівалися їх творці.

Enterprise Java Beans (EJBs). EJB є компонентами, які зберігають свій стан в реляційній базі даних і забезпечують об'єктно-орієнтоване відображення постійних даних. На відміну від продуктів об'єктно-реляційного відображення, EJB мають жорстку специфікацію, що робить можливим використання продуктів від різних постачальників. На жаль, EJB стандарт обмежений в об'єктно-орієнтованому відношенні. Він не підтримує наслідування, поліморфізм і тому подібне. Також EJB компоненти вимагають великих витрат для їх написання і часто спеціального програмного забезпечення для їх роботи.

На сьогодні існують різні фреймворки, що використовують цю техніку програмування. Ось деякі з них: Hibernate, iBATIS, Java Data Objects (JDO), JPOX, Cayenne, TopLink.

При організації ORM з використанням різних технологій необхідно створити файли відображення об'єктів; створити конфігураційні файли, в яких вказуються файли ресурсів, джерела даних, підтримка транзакцій і так далі.

У файлах відображення можна виділити наступні відмінності між технологіями:

− у Hibernate і JDO описується об'єкт моделі, з вказівкою полів таблиць і зв'язків між об'єктами;

− у iBATIS об'єкт формується за допомогою SQL-запиту, тобто файл складається з набору SQL-запитів.

Організація зберігання об'єктів приведена на рисунку 5.1.

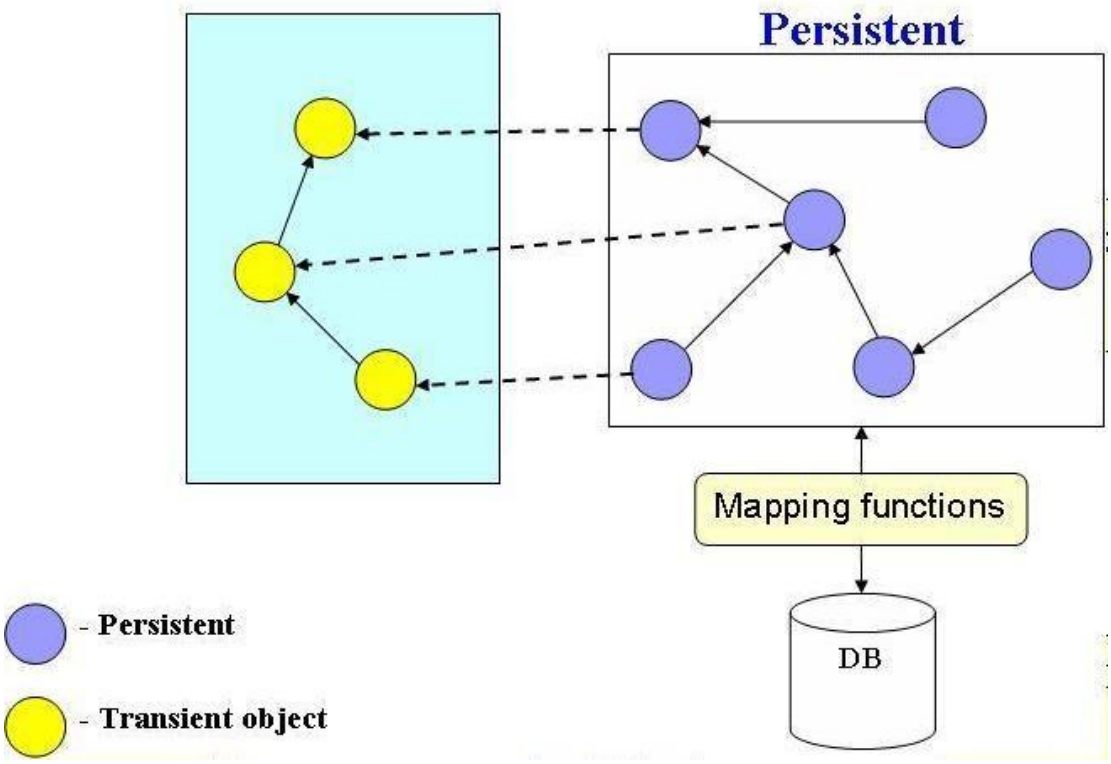

Рисунок 5.1 – Структура зберігання об'єктів в ORM

При зверненні до БД, отримані дані відображаються в об'єкти моделі і потрапляють в "сховище довгоживучих (постійних) об'єктів" (persistent storage). Застосуванню передаються "тимчасові об'єкти" (transient objects) – екземпляри постійних об'єктів моделі. При наступному зверненні, пошук проводитися в сховищі довгоживучих об'єктів, потім у базі.

6 ПРОЕКТУВАННЯ І РОЗРОБКА ШАРУ ПРЕДСТАВЛЕННЯ

6.1 Представлення даних в Web

Існує дві основні форми представлення програми Web-сервера – сценарій (script) і сторінка сервера (serverpage).

Сценарій складається з функцій або методів, призначених для обробки запитів HTTP. Типовими прикладами можуть служити сценарії CGI і сервлети Java. Завдання формування коду HTML за допомогою команд потокового виведення не дуже привабливе для програмістів, а не програмістам вона взагалі не під силу, хоча вони із задоволенням взялися б за Web-дизайн за допомогою інших інструментів. Це природним чином підводить до моделі сторінок сервера, де функції програми зводяться до повернення порції текстових даних. Сторінка містить текст HTML з "вкрапленнями" виконуваного коду. Подібний підхід, що реалізовується, наприклад, в PHP, ASP і JSP, особливо зручний, якщо потрібна незначну додаткову обробку тексту з урахуванням реакції користувача.

Оскільки модель сценаріїв краще підходить для інтерпретації запитів, а схема сторінок сервера – для форматування відповідей, цілком розумно застосовувати їх спільно. Насправді це досить стара ідея, уперше реалізована в призначених для користувача інтерфейсах на основі типового рішення модель-представлення-контролер.

Вхідний контролер приймає запит і витягає з нього інформацію. Потім він передає бізнес-логіку належному об'єкту моделі, який звертається до джерела даних і виконує дії, передбачені в запиті, включаючи збір інформації, необхідної для відповіді. Після закінчення функцій він передає управління вхідному контролеру, який, аналізуючи отриманий результат, приймає рішення про вибір варіанту представлення відповіді. Управління і відповідні дані передаються представленню. Взаємодія вхідного контролера і представлення частенько здійснюється не у вигляді прямих викликів, а за посередництва деякого об'єкту HTTP-сеансу, який служить для передачі даних в обох напрямах.

6.2 Технологія JSP

Технологія JSP-сторінок (Java Server Pages – JSP) дозволяє без зусиль створювати web-вміст, у якого є як статична, так і динамічна компоненти.

Основними характеристиками JSP-технології є:

- мова розробки JSP-сторінок, що є текстовими документами, які описують процес обробки запиту і конструювання відповіді;
- конструкції для дістування доступу до об'єктів на стороні сервера;
- механізми, що визначають розширення для JSP-мови.

JSP-сторінкою є документ з текстовою основою, що містить два типи тексту: статичні шаблонні дані, що виражаються за допомогою будь-якого

формату на текстовій основі, такого як HTML, SVG, WML, і XML, а також JSP-елементи, які створюють динамічний вміст.

6.2.1 Життєвий цикл JSP- сторінки

JSP-сторінка обслуговує запити так само, як це робить сервлет. Таким чином, життєвий цикл і більшість можливостей JSP-сторінок (зокрема, динамічні аспекти) визначаються технологією Java Servlet. Коли запит відображається на JSP-сторінку, він обробляється спеціальним сервлетом, який спочатку перевіряє, чи не є сервлет JSP-сторінки старше, ніж сама JSP-сторінка. Якщо це так, він транслює JSP-сторінку в клас сервлета і компілює клас. Однією з переваг процесу розробки JSP-сторінок в порівнянні з розробкою сервлетов є те, що процес створення JSP-сторінок здійснюється автоматично.

6.2.1.1 Трансляція і компіляція

Під час фази трансляції кожен тип даних в JSP-сторінці обробляється окремо. Шаблонні дані перетворяються в програмний код, який відправлятиме дані в потік, що повертає дані клієнтові. JSP-елементи обробляються таким чином:

− директиви використовуються для управління Web-контейнером в процесі трансляції і виконання їм JSP-сторінки;

− елементи сценаріїв вставляються в клас сервлета JSP-сторінки (див. розділ JSP-елементи сценаріїв).

− елементи форми <jsp: XXX ... /> конвертуються у виклики методу JavaBean-компонентів або виклики Java Servlet API.

Для JSP-сторінки з ім'ям pageName, джерело сервлета JSP-сторінки зберігається у файлі:

<JWSDP_HOME>/work/StandardEngine/localhost/context_root/page Name$jsp.java

Наприклад, джерело сторінки index (index.jsp) для прикладу "локалізація дати" буде названа:

<JWSDP_HOME>/work/StandardEngine/localhost/date/index$jsp.java

Обидві фази, як трансляція, так і компіляція, можуть викликати помилки, які простежуються тільки при першому запиті до сторінки. Якщо помилка зустрічається під час трансляції сторінки (наприклад, у разі, коли транслятор зустрічає погано сформований JSP-елемент), сервер повертає виключення ParseException, а початковий файл класу сервлета буде порожнім або неповним. Останній неповний рядок вкаже на неправильний JSP-елемент.

Якщо помилка зустрічається під час компіляції JSP-сторінки (наприклад, у разі синтаксичної помилки в скриптлеті), сервер поверне виключення JasperException і повідомлення, що включає ім'я сервлета JSP-сторінки з рядком, де була знайдена помилка.

Якщо сторінка була трансльована і відкомпілювалася, сервлет JSP-сторінки більшою мірою дотримується життєвого циклу сервлета, який описаний в розділі Життєвий цикл сервлета.

1) Якщо екземпляра сервлета JSP-сторінки не існує, контейнер:

a) Завантажує клас сервлета JSP-сторінки.
b) Створює екземпляр класу сервлета.
c) Ініціалізує екземпляр сервлета, шляхом виклику методу jspInit.

2) Контейнер викликає метод _jspService, передаючи об'єкт запиту і відповіді.

Якщо контейнеру вимагається видалити сервлет JSP-сторінки, він викликає метод jspDestroy.

6.2.1.2 Виконання

Можна контролювати різні параметри виконання JSP-сторінок за допомогою директив page.

6.2.1.2.1 Використання буферизації

При виконанні JSP-сторінки виведення, записане в об'єкт відповіді, автоматично заноситься у буфер. Можна встановити розмір буфера за допомогою наступної директиви page :

<%@ page buffer="none|xxxkb" %>

Більший розмір буфера дозволяє записати більший об'єм вмісту. Таким чином, JSP-сторінці забезпечується більше часу для установки відповідних кодів і заголовків статусу або ж для відправки іншому web-ресурсу. Менший розмір буфера зменшує завантаження пам'яті сервера і дозволяє клієнтові швидше отримувати дані.

6.2.1.2.2 Використання спільного доступу до об'єктів

З JSP-сторінок, що запускаються як багатопотокові сервлети можна отримати доступ до об'єктів, до яких застосовані ці умови. Можна вказати web-контейнеру, яким чином слід відправляти численні запити клієнтів, використовуючи наступну директиву page :

<%@ page isThreadSafe="true|false" %>

Коли значенням isThreadSafe є true, web-контейнер встановлює більш високий пріоритет відправки численних паралельних клієнтських запитів до JSP-сторінці.

Це значення використовується за умовчанням. Використовуючи true, повинно гарантуватись, що належним чином синхронізувано доступ до усіх загальних об'єктів, визначених на рівні сторінки. Цей рівень включає об'єкти, створені в оголошеннях, JavaBean-компоненти із сторінкою в якості зони дії і атрибути об'єкту зони дії page.

Якщо значенням isThreadSafe є false, запити вирушають по черзі (тобто, по одному) в тому порядку, в якому вони були отримані, а контролювати доступ до об'єктів рівня сторінки при цьому не вимагається. Проте, повинна гарантуватись правильна синхронізація доступу до атрибутів об'єктів зони дії application або session, а також до JavaBean-компонентам із застосуванням або сесією, вказаними в якості зони дії.

6.2.1.2.3 Обробка помилок

Під час виконання JSP-сторінки може виникнути будь-яка кількість виняткових ситуацій. Можна задати, щоб web-контейнер у разі виникнення

виняткової ситуації передавав управління сторінці помилок. Для цього необхідно включити наступну директиву page в початок JSP-сторінки :

<%@ page errorPage="file_name" %>

Наприклад, можна вказати наступну директиву:

<%@ page errorPage="errorpage.jsp"%>

На початку сторінки за допомогою наступної директиви page вказано, що errorpage.jsp служить сторінкою помилок :

<%@ page isErrorPage="true|false" %>

Ця директива робить доступним для сторінки помилок об'єкт виключення (типу javax.servlet.jsp.JspException) так, щоб програміст міг витягати, інтерпретувати і можливо навіть відображати інформацію про причину виникнення виняткової ситуації на сторінці помилок.

Примітка: Можна також визначати сторінки помилок для WAR-файла, який містить JSP-сторінку. Якщо сторінки помилок визначені як для WAR, так і для JSP-сторінки, спочатку з'являється сторінка помилок JSP-сторінки.

6.2.1.2.4 Ініціалізація і завершення роботи JSP-сторінки

Можна настроїти процес ініціалізації шляхом перевизначення методу jspInit інтерфейсу JspPage. Це дозволить JSP-сторінці прочитувати постійно ці конфігурації, ініціалізувати ресурси і здійснювати будь-яку іншу одноразову діяльність. Звільняють ресурси, використовуючи метод jspDestroy.

З причини того, що корпоративний компонент спільно використовується усіма JSP-сторінками, його необхідно ініціалізувати при запуску застосуну, а не в кожній окремій JSP-сторінці. Технологія Java Servlet забезпечує для цієї мети події життєвого циклу застосування і класи-слухачі.

6.2.2 Створення статичного вмісту

Статичний вміст створюється в JSP-сторінці так само, як якби це була звичайна сторінка, що містить тільки текстовий формат даних. Статичний вміст може бути виражений у будь-якому форматі, призначеному для форматування текстів, приміром, HTML, WML і XML. Форматом, визначеним за умовчанням, є HTML.

За бажання можна використати і інший формат тексту. Для цього включите в початок вашої JSP-сторінки директиву page з атрибутом contentType, якому в якості значення слід встановити тип формату. Наприклад, якщо ви хочете, щоб ваші дані були виражені у форматі WML (wireless markup language – мова розмітки для безпровідних систем), вимагається включити наступну директиву:

<%@ page contentType="text/vnd.wap.wml"%>

Реєстр імен типів вмісту визначається організацією IANA (Агентство по виділенню імен і унікальних параметрів протоколів Internet).

6.2.3 Створення динамічного вмісту

Динамічний вміст створюється шляхом звернення до об'єктів мови програмування Java з елементів сценаріїв.

6.2.3.1 Використання об'єктів в JSP- сторінках

З JSP-сторінки можна отримати доступ до різних об'єктів, включаючи корпоративні компоненти і JavaBean-компоненти. JSP-технологія автоматично організовує доступ до певних об'єктів так, що можна створювати об'єкти-застосунки і звертатися до них з JSP-сторінки.

6.2.3.1.1 Неявні об'єкти

Неявні об'єкти створюються web-контейнером і містять інформацію, що відноситься до конкретного запиту, сторінки або застосування. Багато хто з таких об'єктів визначається технологією Java Servlet, що лежить в основі JSP-технології. У таблиці 6.1 представлені неявні об'єкти.

Таблиця 6.1 – Неявні об'єкти

Змінна	Клас	Пояснення
application	javax.servlet.ServletContext	Контекст для сервлета JSP сторінки і будь-якого web компонента, що міститься в тому ж самому застосунку.
config	javax.servlet.ServletConfig	Інформація про ініціалізацію сервлета JSP сторінки.
exception	java.lang.Throwable	Доступно тільки із сторінки помилок (error page).
out	javax.servlet.jsp.JspWriter	Вихідний потік.
page	java.lang.Object	Екземпляр сервлета JSP сторінки, оброблювальний поточний запит. Зазвичай не використовується авторами JSP сторінок.
pageContext	javax.servlet.jsp.PageContext	Контекст JSP сторінки. Містить єдиний API для управління різними контекстними атрибутами, описаними в Sharing Information. Цей API широко використовується, коли реалізують tag handlers (програми обробки тегів)
request	підтип javax.servlet.ServletRequest	Запит, що запускає виконання JSP сторінки.
response	підтип javax.servlet.ServletResponse	Відповідь, що повертається клієнтові. Зазвичай не використовується авторами JSP сторінок.

Змінна	Клас	Пояснення
session	javax.servlet.http.HttpSession	Об'єкт-сесія клієнта.

6.2.3.1.2 Об'єкти застосунку

По можливості, режим роботи застосунку слід інкапсулювати в об'єкти так, щоб розробники сторінок могли сфокусуватися на аспектах представлення. Об'єкти можуть створюватися розробниками, які є професіоналами в програмуванні Java і в діставанні доступу до баз даних і інших служб. Існує чотири способи для створення і використання об'єктів в JSP-сторінці.

1) Змінні екземпляра і класу, що відносяться до класу сервлета JSP-сторінки, створюються в оголошеннях і доступні в скриптлетах і виразах.
2) Локальні змінні класу сервлета JSP-сторінки створюються і використовуються в скриптлетах і виразах.
3) Атрибути об'єктів зони дії створюються і використовуються в скриптлетах і виразах.
4) JavaBean-компоненти можуть створюватися і бути доступними за допомогою модернізованих JSP-елементів. Ці елементи обговорюються в главі JavaBean-компонентів в JSP-сторінках. Можливе створення JavaBean-компонент в оголошенні або скриптлеті і виклик методів JavaBean-компонента в скриптлеті або виразі.

6.2.3.2 JSP-елементи сценаріїв

JSP-елементи сценаріїв використовуються для створення об'єктів і організації до них доступу, визначення методів і управління потоком контролю. З причини того, що однією з цілей JSP-технології є відділення статичних шаблонних даних від програмного коду, потрібного для динамічної генерації вмісту, рекомендується помірне використання сценаріїв JSP. Велику частину роботи, що вимагає використання сценаріїв, можна усунути шляхом використання замовлених тегів.

JSP-технологія дозволяє контейнеру підтримувати будь-яку мову написання сценаріїв, що має можливість викликати Java-об'єкти. Якщо замість мови java, встановленої за умовчанням для створення сценаріїв, ви бажаєте використати іншу мову, задайте її в директиві page на початку JSP-сторінки :

<%@ page language="scripting language" %>

Унаслідок того, що елементи сценаріїв конвертуються у вирази мови програмування в класі сервлета JSP-сторінки, необхідно імпортувати усі класи і пакети, використовувані цією JSP-сторінкою. Якщо сторінка написана на мові java, імпортуйте клас або пакет за допомогою директиви page :

<%@ page import="packagename .*, fully_qualified_classname" %>

Наприклад:

<%@ page import="java.util.*, stu.jtap2.labs.lab2.domain.*" %>

6.2.3.2.1 Оголошення

JSP-оголошення використовується для оголошення змінних і методів на мові створення сценаріїв сторінки. Синтаксис для оголошення наступний:

<%! scripting language declaration %>

Коли мовою створення сценаріїв є Java, змінні і методи в JSP-оголошеннях стають оголошеннями класу сервлета JSP-сторінки.

Наприклад:

```
<%!    private StudentManager stm;
       public void jspInit (){
        ...    }
       public void jspDestroy (){
        ... }
%>
```

6.2.3.2.2 Скриплети

JSP-скриптлет використовується для зберігання якого-небудь фрагмента коду, дійсного для мови сценаріїв, використовуваної в цій сторінці. Синтаксис для скриптлета виглядає таким чином:

<%
scripting language statements
%>

Коли мовою сценаріїв є Java, скриптлет трансформується у фрагмент виразу мови Java і вставляється в метод service сервлета JSP-сторінки. Змінна мови програмування, створена в скриптлеті, доступна з будь-якої точки JSP-сторінки.

Наступний приклад містить скриптлет, що витягує ітератор зі списку студентів академ групи, а також скриптлет, який встановлює конструктор для циклічного проходу по усіх позиціях списку. Під час циклу JSP-сторінка витягує властивості об'єктів і форматує їх, використовую при цьому HTML-розмітку. Блок відкриває цикл while, тому HTML-розмітка передує скриптлету, що закриває блок.

Наприклад:

```
<%
  Iterator i = group.getItems ().iterator ();    while (i.hasNext ()) {
    Student item (Student) i.next ();
%>
  <tr>
    <td                align="right"                bgcolor="#ffffff">
<%=item.getFirstName ()%>
    </td>
    <td                align="right"                bgcolor="#ffffff">
<%=item.getFirstName ()%>
```

```
        </td>

    <!-- ... код сторінки -->

    <%
      } // while
    %>
```

6.2.3.2.3 Вирази

JSP-вираз використовується для вставки в потік даних, що повертається клієнтові, значення виразу мови сценаріїв, конвертованої в рядок. Коли мовою написання сценаріїв є Java, вираз перетворюється в оператор, що конвертує значення виразу в об'єкт String і вставляє його в неявний об'єкт out.

Синтаксис такого виразу виглядає таким чином:

<%= scripting language expression %>

Зверніть увагу, що використання крапки з комою в JSP-виразах заборонене, навіть в тих випадках, коли у такого ж виразу, що використовується в скриптлеті, вона є присутньою.

Наступний скриптлет витягає кількість студентів, що знаходяться в списку групи:

```
<% //
  int num = group.getNumberOfItems ();    if (num> 0) {
%>
  <font size="+2">
    У групі <%=num%>
    <%= (num==1 ? "Студент": "Студентів")%> <br/>
  </font>
<%
  }//if
%>
```

6.2.3.2.4 Включення вмісту в JSP-сторінку

Існує два механізми для включення іншого web-ресурсу в JSP-сторінку: використання директиви include і елементу jsp: include.

Директива include обробляється під час трансляції JSP-сторінки в клас сервлета.

Дією цієї директиви є вставка тексту, що міститься в іншому файлі, (або у статичному вмісті, або ж в іншій JSP-сторінці) в необхідну JSP-сторінку. Зазвичай директиву include використовують для включення в іншу сторінку вмісту шапки, авторської інформації або будь-якої іншої порції інформації, яку використовують багаторазово. Синтаксис для директиви include наступний:

<%@ include file="filename" %>

Наприклад, усі сторінки застосунку "книжковий магазин" включають файл banner.jsp, в якому знаходиться вміст шапки. Це здійснюється за допомогою наступної директиви:

<%@ include file="banner.jsp" %>

Крім того, сторінки bookstore.jsp, bookdetails.jsp, catalog.jsp, і showcart.jsp включають JSP-елементи, що створюють і руйнують компонент бази даних за допомогою цієї директиви:

<%@ include file="initdestroy.jsp" %>

У цього підходу немає обмежень, оскільки директива include статично поміщається в кожен файл, що повторно використовує ресурс, на який посилається директива.

Елемент jsp: include обробляється під час виконання JSP-сторінки. Дія include дозволяє включати в JSP-файл як статичні, так і динамічні ресурси.

Результати, отримані при включенні статичних і динамічних ресурсів, дещо відрізняються. Якщо ресурс статичний, його вміст вставляється в JSP-файл. Якщо ж ресурс є динамічним, то ресурсу, що включається, відправляється запит. Виконується сторінка, що включається, а потім результат включається у відповідь, JSP-сторінки. Синтаксис елементу jsp: include виглядає таким чином:

<jsp: include page="includedPage" />

Примітка: Tomcat не перевантажуватиме статично включену сторінку, що піддалася модифікаціям до тих пір, поки сторінка, що включає її, також не буде змінена.

Наприклад:

<jsp: include page="header.jsp"/>

6.2.3.2.5 Передача управління іншому web-компоненту

Механізм передачі управління іншому web-компоненту з JSP-сторінки використовує виконувані функції, що забезпечуються Java Servlet API. Доступ до цих функцій здійснюється з JSP-сторінки за допомогою елементу jsp: forward:

<jsp: forward page="/main.jsp" />

Увага! Якщо клієнтові вже повернули які-небудь дані, в елементі jsp : forward виникне виняткова ситуація IllegalStateException.

6.2.3.2.6 Елемент jsp: param

При виклику елементу include або forward, цільовій сторінці надається первинний об'єкт запиту. Якщо необхідно помістити в цю сторінку додаткові дані, за допомогою елементу jsp: param можливо додати в об'єкт запиту і інші параметри:

<jsp: include page="header.jsp">
<jsp: param name="param1" value="value1"/>
</jsp: include>

6.2.3.2.7 Включення аплета

Включати аплет або JavaBean-компонент в JSP-сторінку можна за допомогою елементу jsp: plugin. Цей елемент генерує HTML-код, в якому містяться відповідні конструкції (<object> чи <embed>). Вони залежать від взаємодії між клієнтом і браузером і приводять, при необхідності, до завантаження програмного забезпечення Java Plugin і компонента на стороні клієнта, а також до подальшого виконання будь-якого компонента на стороні сервера. Синтаксис елементу jsp: plugin виглядає таким чином:

<jsp: plugin type="bean|applet" code="objectCode" codebase="objectCodebase"
{ align="alignment" }
{ archive="archiveList" }
{ height="height" }
{ hspace="hspace" }
{ jreversion="jreversion" }
{ name="componentName" }
{ vspace="vspace" }
{ width="width" }
{ nspluginurl="url" }
{ iepluginurl="url" } >
{ <jsp: params>
{ <jsp: param name="paramName" value= paramValue" /> }+
</jsp: params> }
{ <jsp: fallback> arbitrary_text </jsp: fallback> } </jsp: plugin>

Тег jsp: plugin замінюється або тегом <object>, або тегом <embed>. Це залежить від клієнта, що здійснює запит. Атрибути тега jsp: plugin забезпечують ці конфігурації для представлення елементу, як того вимагає версія plugin. Атрибути nspluginurl і iepluginurl задають URL-адресу, з якої цей plugin можна завантажити.

Елементи jsp: param задають параметри аплета або JavaBean-компонента. Якщо plugin не можна запустити (чи теги <object> і <embed> не підтримуються клієнтом, або з якої-небудь іншої причини), елемент jsp: fallback вказує на те, щоб вміст був видимий у браузері клієнта.

Якщо ж plugin запускається, але аплет або JavaBean-компонент не можуть бути знайдені або ж не запускаються, користувачеві відобразиться спеціальне повідомлення, вбудоване в plugin (найімовірніше, це буде спливаюче вікно, що сповіщає про виняткову ситуацію ClassNotFoundException).

6.2.3.3 Розширення JSP-мови

За допомогою JavaBean-компонентів, що використовуються спільно із скриплетами, можна виконати безліч різних, динамічним чином оброблюваних завдань. Вони включають організацію доступу до баз даних, використання корпоративних служб, таких як e-mail і каталоги, а також управління потоками. Проте одним з недоліків скриптлетів є те, що вони ускладнюють підтримку JSP-сторінок. З іншого боку, JSP-технологія забезпечує механізм, що

називається замовлені теги, який дозволяє інкапсулювати динамічно виконувані функції в об'єкти, доступні через розширення JSP-мови.

Замовлені теги передають JSP-сторінкам переваги іншого рівня компонентного представлення.

Замовлені теги упаковуються і розподіляються в одиниці, що називаються бібліотекою тегів. Синтаксис замовлених тегів не відрізняється від синтаксису, що використовується для JSP-елементів, тобто <prefix: tag> є формою написання замовленого тега. Проте, елемент тега prefix визначається користувачем бібліотеки тегів, а tag-розробником самого тега.

6.3 JSTL – Java Server Pages Standard Tag Library

JSTL – стандартна бібліотека тегів JSP. Вона розширює специфікацію JSP, додаючи у бібліотеку JSP теги спеціального призначення, призначені для розбору XML даних, умовної обробки, створення циклів, обробки колекцій, роботи з базами даних і підтримки інтернаціоналізації.

JSTL є альтернативою такому виду вбудованої в JSP логіки, як скриплети, тобто прямі вставки Java коду. Використання стандартизованої безлічі тегів прийнятніше, оскільки отримуваний код легше підтримувати і простіше відділяти бізнес-логіку від логіки відображення.

Ця бібліотека тегів включає широкий спектр спеціалізованих функцій, які більшість авторів JSP вважали украй корисними і необхідними на етапі створення специфікація цієї бібліотеки.

Для роботи з даними в JSTL використовується мова виразів (EL – Expression Language), що є спрощеним об'єднанням JavaScript і XPath. Багато з можливостей мови виразів, що отримали підтримку в стандартній бібліотеці тегів, були включені в наступну специфікацію JSP, а саме JSP 2.0 або JSR-152. Але JSTL залишається незалежною бібліотекою тегів.

Інженерам і розробникам JSP сторінок має сенс використати саме стандартну бібліотеку тегів виходячи з наступного:

- бібліотеку легко вивчити і вона надає широкий спектр функціональності;
- розробники JSP-сторінок, а також використовувані ними засоби розробки можуть легко генерувати сторінки в усіх JSP-споріднених контейнерах, використовуючи стандартні теги;
- стандартні теги, створені експертною групою, відповідають запитам великої безлічі співтовариств розробників;
- теги ретельно протестовані і готові до застосування;
- зменшення вартості навчання досягається шляхом надання цільових учбових матеріалів, а також спрощеної переносимості і супроводження JSP-сторінок і застосунків.

Більшість контейнерів JSP 1.1 (і вище) вже надають бібліотеку тегів, яку можуть використати їх клієнти. В процесі розробки JSP сторінок існує необхідність в інкапсуляції функціональності. Якщо функціональність інкапсульована, JSP-розробники можуть використати спеціалізовані теги, не

маючи великих знань в Java і не докладаючи яких-небудь інших зусиль із програмування. Інкапсуляція також дозволяє повторно використати загальну функціональність усередині одного застосування або декількох застосувань. Кожен призначений для користувача тег пишеться, тестується і відлагоджується тільки один раз. JSTL інкапсулює найбільш необхідну загальну функціональність, якої зазвичай потребують автори JSP сторінок.

JSTL складається з чотирьох різних бібліотек:

- бібліотека ядра;
- маніпуляцій з XML даними;
- дії для роботи з БД;
- інтернаціоналізації і форматування.

Найчастіше застосовними тегами є теги бібліотеки ядра. Бібліотека ядра розділена на чотири функціональні області:

- дії загального призначення, які використовуються для роботи зі змінними на JSP-сторінках. Ці дії також включають дії з обробки помилок і виключень;
- умовні дії, що використовуються для умовних обчислень на JSP сторінках;
- ітераційні дії, що забезпечують можливість роботи з колекціями об'єктів;
- дії, що забезпечують можливість роботи з URL-ресурсами на сторінках.

JSTL – це універсальний інструмент, який забезпечує гнучкість і максимальну ефективність розробки JSP-сторінок.

6.3.1 Бібліотека тегів

У JSP-технології теги можуть створювати і модифікувати об'єкти. Роблячи це, теги зазвичай впливають на існуючий потік виведення, виконуючи деяку логіку застосування. Окрім стандартних тегів, найбільш помітним доповненням до специфікації JSP було введення механізму для інкапсуляції інших типів динамічної функціональності в призначені для користувача теги, які є розширеннями мови JSP.

Призначений для користувача тег – це елемент мови JSP, визначений користувачем. Коли сторінка JSP містить призначений для користувача тег, транслюється в сервлет, тег конвертується в операції на об'єкті, що називається обробник тега. Web-контейнер потім викликає ці операції, коли сервлет сторінки JSP виконується.

Призначені для користувача теги зазвичай поширюються у формі бібліотеки тегів, яка визначає набір пов'язаних призначених для користувача тегів і містить об'єкти, які реалізують теги.

Однією з переваг використання призначених для користувача тегів по порівнянню, наприклад, з JavaBeans являється те, що користувач дістає доступ до контексту і об'єктів зони видимості JSP. До таких об'єктів відносяться запит, відповідь, сесія і атрибути.

Призначеними для користувача тегами слід користуватися з ряду причин:

- дії можна робити більше виборчими завдяки наявності атрибутів, що передаються із сторінки;
- призначені для користувача дії мають доступ до усіх об'єктів, доступних на JSP-сторінках;
- призначені для користувача дії можуть модифікувати відповідь, генеровану сторінкою;
- призначені для користувача дії можуть кооперуватися один з одним, використовуючи змінні або JavaBeans;
- призначені для користувача дії можуть бути інкапсульованими одна в одну, таким чином, організовуючи складні взаємодії усередині JSP сторінок.

Завдання, які можуть бути виконані призначеними для користувача тегами, включають обробку форм, обробку XML, ітерації колекцій, доступ до баз даних і інших корпоративних служб, таким як e-mail і каталоги, забезпечення безпеки і контролю над процесом виконання. До появи призначених для користувача тегів компоненти JavaBeans в зв'язці із скриплетами були основним механізмом для виконання такого роду обчислень усередині JSP-сторінок. Використання JavaBeans має декілька недоліків. Передусім, сторінки стають складнішими. Також потрібно знання мови програмування Java і специфікації JavaBeans. JSP-сторінки стають важчими для супроводу за наявності скриплетів, розподілених по усій сторінці.

Призначені для користувача теги усувають ці проблеми шляхом абстрагування функціональності. Завдяки цьому вони можуть бути повторно використані більш ніж в одному застосуванні.

Бібліотеки тегів JSP створюються розробниками, які є професіоналами в мові програмування Java, і експертами в доступі до даних і інших служб, і використовуються проектувальниками Web-додатків, які можуть фокусуватися на проблемах презентації замість того, щоб сушити голову над тим, як звертатися до корпоративних служб.

6.3.2 Приклад застосування призначених для користувача тегів

Для створення бібліотеки тегів, передусім, необхідно визначити файл дескриптора бібліотеки тегів і необхідно створити обробник тега.

Обробник тега – це Web-об'єкт, що викликається контейнером, для виконання призначеного для користувача тега під час виконання JSP-сторінки, яка викликає тег.

Дескриптор бібліотеки тегів – це XML документ, який описує бібліотеку тегів. TLD містить інформацію як про бібліотеку в цілому, так і про кожен тег, що міститься у бібліотеці. Файли TLD використовуються JSP-контейнером для перевірки правильності тегів.

6.3.2.1 Дескриптор бібліотеки тегів, файл TLD

Приклад файлу дескриптора бібліотеки тегів складатиметься з одного тега, званого HelloTag. Тег HelloTag виводитиме повідомлення «hello», що персоналізується, в JspWriter, який є потоком виведення для JSP-сторінки.

Файл TLD приведений в лістингу 6.1.

Лістинг 6.1 – Файл TLD

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN" "http://java.sun.com/j 2ee/dtds/web - jsptaglibrary_1_2.dtd">
<taglib>
    <tlib-version>1.0</tlib - version>
    <jsp-version>2.0</jsp - version>
    <short-name>Application Tag Library</short - name>
    <description>
Ця бібліотека тегів містить один тег, а також визначення тега
    </description>
     <tag>
  <name>hello</name>
    <tag-class>jstlm.tags.HelloTag</tag - class>
    <body-content>empty</body - content>
     <description>
Цей тег друкуватиме повідомлення Hello, що персоналізується, Атрибути: name - ім'я того, кому адресовано вітання.
    </ description >
  <attribute>
   <name>name</name>
   <required>false</required>
       <rtexprvalue>true</rtexprvalue>
  </attribute>
    </tag>
</taglib>
```

Наступним кроком є створення обробника тега.

6.3.2.2 Обробник тега

Тег визначається в класі обробника. Цим класом є клас jstlm.tags.HelloTag, на який міститься посилання в елементі <tag-class> дескриптора бібліотеки.

Клас TagSupport є базовим класом для простих тегів, які або не мають тіла, або мають порожнє тіло. Він знаходиться в пакеті javax.servlet.jsp.tagext.

Клас TagSupport реалізує інтерфейс Tag і містить стандартну функціональність, необхідну для простих тегів.

Java-файл обробника тега HelloTag приведений в лістингу 6.2.

Лістинг 6.2 – Java-файл обробника тега HelloTag

```
public final class HelloTag extends TagSupport {
private String name = null;
  public int doStartTag () throws JspException {
                try {
                        if (name == null){
                          pageContext.getOut ().write ("Hello World");
                        } else {
                                pageContext.getOut                    ().write
("Hello"+name);
                        }
                } catch (java.io.Exception ioe){
                        throw new JspTagException (ioe.getMessage ());
                }
                return (SKIP_BODY);
          }
          public String getName (){
                return (this.name);
          }
          public void setName (String name) {
           this.name = name;
          }
          public void release (){
          super.release ();
                name = null;
          }
  }
```

У прикладі необхідно реалізувати методи doStartTag() і release() інтерфейсу Tag. Інші методи матимуть реалізацію, що надається за умовчанням класом TagSupport. Контейнер викликає настановний метод (setter method) для кожного атрибуту, що зустрічається в призначеному для користувача тегу. Для цього визначені методи setName () і getName () для роботи з атрибутами. При трасляції JSP-сторінки в сервлет відбувається виклик методу doStartTag() нашого обробника, де відбувається перевірка атрибуту і здійснюється виведення повідомлення.

Реалізація методу release() потрібна для звільнення ресурсів, які використовує обробник тега. Метод release () виконується, коли JSP-контейнер закінчує працювати з тегом.

6.3.2.3 Використання призначеного для користувача тега

Маючи реалізацію обробника тега і опис бібліотеки тегів можна використати тег в JSP-сторінці.

Перед використанням тега необхідно використати директиву taglib.

Ця директива має наступну форму:

<%@ taglib uri="." prefix="." %>

Атрибут uri може бути абсолютним або відносним URL посиланням на файл-дескриптор бібліотеки тегів.

Атрибут prefix визначає префікс, який використовуватимуться перед ім'ям тега.

Використання тега HelloTag в JSP- сторінці приведене в лістингу 6.3.

Лістинг 6.3 – Використання тега HelloTag в JSP-сторінці

```
<%@ taglib uri="/jstlm/tags/hello - taglib.tld"        prefix="jstlm"%>
<html>
      <head>
       <title> Hello Sample </title>
      </head>
      <body>
            <h1>
                  <jstlm: hello name="Andrew"/>
            </h1>
            <h2>
                   <jstlm: hello/>
            </h2>
      </body>
</html>
```

Директива taglib в JSP файлі прикладу використовує відносний URL, який вказує на ім'я файлу дескриптора бібліотеки, і визначає префікс «jstlm» для тегів бібліотеки.

6.3.2.4 Стандартна бібліотека тегів

Використання JSTL усуває необхідність написання своїх власних TLD-файлів, обробників тегів і створення своїх власних бібліотек для великого кола стандартних дій, що використовуються при написанні JSP сторінок.

Приклад призначеного для користувача тега HelloTag можна замінити тегом, що надається бібліотекою JSTL.

JSP-сторінка з використанням JSTL для виведення персоналізованого повідомлення «hello» приведена в лістингу 6.4.

Лістинг 6.4 – JSP-сторінка з використанням JSTL для виведення персоналізованого повідомлення «hello»

```
<%@ page contentType="text/html; charset=UTF - 8" %>
<%@ taglib uri="http://java.sun.com/jstl/ea/core" prefix="c"%>
<html>
      <head>
       <title>Hello Sample</title>
```

```
</head>
<body>
 <h1>
 Hello <c: out value="${param.name}" default="World"/>
 </h1>
</body>
</html>
```

Тег «out» витягає з HTTP- запиту значення параметра «name», яке потім виводиться у вікно браузеру. Якщо це значення рівне «null», то автоматично використовуватиметься значення за умовчанням «World».

Застосовуючи бібліотеку JSTL, була отримана ідентична функціональність наведеного прикладу без необхідності створення призначеного для користувача тега.

6.3.3 Основи JSTL

6.3.3.1 Бібліотеки тегів входять в JSTL

Бібліотека JSTL містить велику кількість дій. Усі дії розбиті на чотири функціональні категорії:

- бібліотека ядра;
- маніпуляцій з XML даними;
- дії для роботи з БД;
- інтернаціоналізації і форматування.

У таблиці 6.2 перераховані чотири функціональні категорії доступні в JSTL.

Таблиця 6.2 – Функціональні категорії доступні в JSTL

Функціональна область	URI	Префікс тегів
Бібліотека ядра	http://java.sun.com/jstl/core	c
Маніпуляції з XML даними	http://java.sun.com/jstl/xml	x
Інтернаціоналізація і форматування	http://java.sun.com/jstl/fmt	fmt
Дії для роботи з БД	http://java.sun.com/jstl/sql	sql

6.3.3.2 Зони видимості на JSP- сторінках

JSP-сторінка може мати доступ, створювати і модифікувати об'єкти на стороні сервера. Ці об'єкти можуть бути видимими для стандартних і призначених для користувача дій. Зона видимості об'єкту, описує, які сутності можуть мати доступ до цього об'єкту. Існують наступні зони видимості об'єктів : «page» (JSP-сторінка), «request» (запит), «session» (поточна сесія) і «application» (застосунок).

Доступ до об'єктів із зоною видимості «page» можливий тільки з JSP-сторінки, на якій вони були створені. Після того, як відповідь відіслана клієнтові або запит перенаправлений, усі посилання на об'єкт звільняються.

Доступ до об'єктів із зоною видимості «request» можливий з усіх сторінок, що оброблюють один і той же запит. Це означає, що, якщо запит перенаправляється, об'єкт все ще знаходиться в зоні видимості. Посилання на об'єкти із зоною видимості request зберігаються в неявно створеному об'єкті «request». Коли запит оброблений, усі посилання на об'єкт звільняються.

Доступ до об'єктів із зоною видимості «session» можливий з усіх сторінок, що оброблюють запити і відносяться до однієї і тієї ж сесії, а саме до тієї, в якій вони були створені. Усі посилання на об'єкт звільняються після того, як сесія закривається. Посилання на об'єкт із зоною видимості «session» зберігаються в об'єкті «session».

Доступ до об'єктів із зоною видимості «application» можливий із сторінок, що оброблюють запити, у рамках того застосунку, в якому ці об'єкти були створені. Посилання на об'єкти із зоною видимості «application» зберігаються в об'єкті «application». Об'єкт «application» є контекстом сервлета, що отримується з об'єкту конфігурації сервлета.

Зони видимості об'єктів представлені на рисунку 6.1.

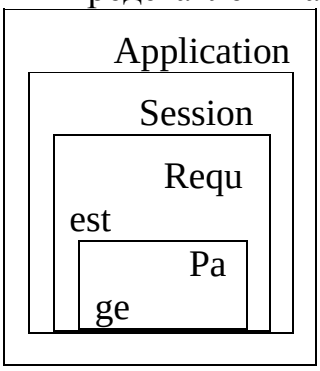

Рисунок 6.1 – Зони видимості об'єктів

Дії можуть діставати доступ до об'єктів, використовуючи іменований атрибут об'єкту PageContext. Екземпляр PageContext доступний через неявно створений об'єкт pageContext. Цей об'єкт надає доступ до усіх атрибутів, пов'язаних з PageContext JSP-сторінки. Ці атрибути включають такі об'єкти, як HttpServletRequest, HttpSession і ServetContext, а також їх власні атрибути.

6.3.4 Використання мови виразів

Мова виразів (MB) дозволяє, використовуючи простіший синтаксис, робити складніші речі з даними в застосунку. Нині MB в JSTL може тільки використовуватися зі значеннями атрибутів тегів, передусім в діях, визначених у бібліотеці JSTL. У специфікації JSP 2.0, MB можна використати усередині JSP-сторінки. Використання MB в JSP-сторінці, показано в лістингу 6.5.

Лістинг 6.5 – Використання MB в JSP-сторінці

```
<%@ page language="java" contentType="text/html; charset=ISO -8859-1" pageEncoding="ISO-8859-1"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/ jstl/core"%>
<%@page import="domain.DBWorker"%>
<%@page import="domain.TestClass"%>
<%@page import="java.util.Collection"%>
<html>
<body>
<jsp: useBean id="testClass" scope="page" class="domain.TestClass"/>
<jsp: setProperty name="testClass" property="*"/>
<c: if test="${not empty param.id}">
    <%
      DBWorker.addElem (testClass);
    %>
</c: if>
<%
Collection<TestClass> listObj = DBWorker.getAll ();
pageContext.setAttribute ("listObj", listObj);
%>
<table border=1>
    <tr>
```

```
0
1       <td> id </td>
2       <td> name </td>
3       <td> address </td>
4       </tr>
5   <c: forEach var="item" items="${listObj}">
6       <tr>
7       <td><c: out value="${item.id}"/></td>
8       <td>${item.name}</td>
9       <td>${item.address}</td>
0       </tr>
1   </c: forEach>
2   </table>
3   </body>
4   </html>
```

У умовному тегу <c: if>, рядки 11-14, ми можемо використати MB в атрибуті «test» для визначення наявності в запиті атрибуту з ім'ям «id». Якщо атрибут є присутнім і значення не порожнє – виконується тіло тега <c: if> (елемент буде доданий у базу даних). У рядках 15-18 створюється колекція існуючих об'єктів, яка додається в область видимості сторінки - pageContext.

Використовуючи ітераційний тег <c: forEach> об'єкти колекції «listObj» виводяться в таблицю. Для виведення значень об'єктів використовується тег <c: out>, рядок 27, або мова виразів JSP, рядки 28-29.

Використання ідентифікатора «listObj», в MB, ідентично виклику методів екземпляра класу PageContext для отримання об'єктів, наприклад findAttribute («listObj»).

Ідентифікатор може знаходитися у будь-якій із зон видимості JSP. Ці зони видимості включають «page», «request», «session» і «application». Якщо ідентифікатор не знайдений ні в одній із зон видимості, тоді повертається значення «null».

6.3.4.1 Неявно створювані об'єкти, доступні в МВ

Існує дещо неявно створюваних об'єктів, доступних в МВ. Ці об'єкти дозволяють діставати доступ до будь-яких змінних, що зберігаються в цій зоні видимості JSP. Ці об'єкти включають «pageScope», «requestScope», «sessionScope» і «applicationScope». Усі Scopeобъекты реалізують інтерфейс Map і зв'язують імена атрибутів з їх значеннями.

У таблиці 6.3 перераховані усі неявно створювані об'єкти доступні в МВ.

Таблиця 6.3 – Неявно створювані об'єкти МВ

Ім'я об'єкту	Опис
pageScope	Колекція усіх змінних в зоні видимості page
requestScope	Колекція усіх змінних в зоні видимості request
sessionScope	Колекція усіх змінних в зоні видимості session
applicationScope	Колекція усіх змінних в зоні видимості application
param	Параметри запиту для цієї сторінки
paramValues	Усі значення для цього параметра
header	Заголовки, що посилаються веб-браузером
headerValues	Усі значення для цього заголовка
initParam	Параметр ініціалізації контексту
cookie	Значення cookie
paramContext	Надає доступ до інформації сторінки

Використовуючи неявно створені об'єкти param і paramValues, можна отримати доступ до параметрів HTTP-запиту. Це також відноситься до заголовної інформації запиту, якщо використовуються об'єкти header і headerValues.

Об'єкти param і header реалізують інтерфейс Map, який зв'язує імена параметрів і заголовків відповідно зі значеннями типу String.

Використання об'єктів param і header аналогічно використанню методів ServletRequest.getParameter (String name) і ServletRequest.getHeader (String name).

Об'єкти paramValues і headerValues також реалізують Map і зв'язують імена параметрів і імена заголовків з масивом значень типу String[].

Об'єкт initParam надає доступ до параметрів ініціалізацій контексту, а cookie дає можливість отримати cookies, що містяться в запиті.

Неявно створений об'єкт pageContext надає доступ до усіх властивостей, пов'язаних з контекстом JSP-сторінки, такими як об'єкти: HttpServletRequest, ServletContext і HttpSession, а також до їх властивостей.

Приклади використання неявно створених об'єктів :

- ${pageContext.request.servletPath} повертає шлях сервлета, отриманий з HttpServletRequest;
- ${sessionScope.loginld} повертає атрибут loginld зони видимості session або null, якщо атрибут не знайдений;
- ${param.id} повертає значення типу String параметра «id» або «null», якщо параметр не знайдений;
- ${paramValues.id} повертає масив String[], що містить усі значення параметра «id» або «null», якщо він не знайдений. Використання paramValues особливо корисно, коли параметр може мати декілька значень, наприклад, отриманих з форми з множинним вибором.

6.3.4.2 Отримання інформації із структур даних

MB надає два способи доступу до структур даних, оператори "". і [ ]. Використання цих операторів полегшує доступ до інкапсульованих даних.

Використання нотації "." є скороченою формою доступу до властивості об'єкту. У лістингу 6.5, рядки 27-29, оператор "." використовується для набуття значень полів id, name, address об'єкту item. Також можна отримати доступ до цих полів, використовуючи, оператор [ ]. Наприклад, вираз ${item.id} еквівалентний ${item.["id"]}.

Оператор [ ] також використовується для доступу до елементів колекцій. Колекції включають: списки, асоціативні масиви і просто масиви.

Тип індексу залежить від типу колекції, що використовується.

6.3.4.3 Оператори MB

MB підтримує оператори відношення, арифметичні і логічні оператори.

Оператори відношення включають: empty ==, !=, <, >, <=, >=, not, eq, ne, lt, gt, le і ge. Останніх шість операторів було введено для того, щоб усунути необхідність використання символьних сутностей XML. Символьні сутності доводиться використовувати, якщо усередині XML-елементу є присутніми такі символи, як, наприклад, кутові дужки "<".

Префіксний оператор «empty» наданий для тестування, значення рівне «null» або порожнє, тобто «», лістинг 6.5, рядок 10.

Арифметичні оператори включають складання (+), віднімання (-), множення (*), ділення (/ чи div) і залишок від ділення (% чи mod). Логічні оператори включають &&, || і !. Вони являються логічними «і», «або» і логічним запереченням.

Приклади використання операторів у виразах, приведені в таблиці 6.4.

Таблиця 6.4 – Приклади використання операторів у виразах

Вираз	Опис	Результат
${1 == 1}	Оператор «рівно»	«Істина»
${l != 1}	Оператор «не рівно»	«Брехня»
${1 <= 1}	«Менше» або «рівно»	«Істина»
${l le 1}	«Менше, ніж» або	«Істина»
${l == 1 || 2 > 3}	Складене порівняння «істина» або «брехня»	«Істина»
${1 == 1 and 2 > 3}	Складене порівняння «істина» і «брехня»	«Брехня»
${6 * 5 == 30}	Використовується множення. Результат (30 ==30)	«Істина»
${param.name == 'Me'}	Перевірка, чи дорівнює параметр name рядку "Me"	«Істина», при виконанні page.jsp?name=Me
${empty param.name)	Перевірка значення параметра запиту name. Якщо значення “null" або "" результатом буде «Істина»	«Брехня», при виконанні page.jsp?name=Me

6.3.4.4 Автоматичне приведення типів

MB, підтримує приведення між різними типами об'єктів і примітивними типами даних. JSTL автоматично визначає відповідні перетворення типів і значення за умовчанням. Наприклад, параметр String із запиту буде приведений до потрібного об'єкту або примітивного типу даних.

Автоматичне приведення типів приведене в лістингу 6.6.

Лістинг 6.6 – Виконання приведення типів

```
Створення змінної
<c: set var="myInteger" value="${param.month}"/>
Виведення значення змінної помноженої на 2
<c: out value="${mylnteger * 2}"/>
```

Тег <c: set> здійснює створення змінної myInteger. Необхідно використати її значення у виразі як число. Якщо параметр stringValue, передається в запиті маючи тип String, значення змінної буде правильним,

оскільки String буде перетворене в потрібний тип, коли змінна використовуватиметься. Якщо значення параметра неможливо було правильно синтаксично розібрати і привести до числа, наприклад, значення рівне «Four», а не 4, тоді буде викинуто виключення.

6.3.5 Робота з діями Ядра

Бібліотека Ядра підрозділяється на чотири функціональні області:

- дії загального призначення, які використовуються для маніпуляцій змінними на JSP- сторінках. Ці дії загального призначення також включають дії з обробки помилок;
- умовні дії, що використовуються для умовних обчислень на JSP-сторінках;
- ітераційні дії, які допомагають здійснювати ітерації з колекціями об'єктів;
- дії, що відносяться до роботи з URL-ресурсами на JSP-сторінках.

6.3.5.1 Теги загального призначення

Існує чотири теги загального призначення:

- тег <c: out>;
- тег <c: set>;
- тег <c: remove>;
- тег <c: catch>.

Виведення інформації в поточний потік виведення

Тег <c: out> використовується для виведення інформації в поточний потік виведення JSP. Синтаксис тега <c: out> приведений в таблиці 6.5.

Таблиця 6.5 – Синтаксис тега <c: out>

Без тіла тега
<c: out value="value" [escapeXml="{true|false}"] [default="defaultValue"] />
З тілом тега
<c: out value="value" [escapeXml="{true|false}"]> default value </c: out>

Опис атрибутів тега <c: out> приведений в таблиці 6.6.

Таблиця 6.6 – Опис атрибутів тега <c: out>

Атрибут	Тип	Опис
value	Object	Вираження
escapeXml	boolean	Включає те, що перекодувало символів <,>,&,’,” у символьні сутності XML

default	Object	Значення за умовчанням, якщо результат виразу рівний «null»

Тег <c: out> виконує вираз вказаний в атрибуті «value» і виводить результат в поточний об'єкт JspWriter. Значення виведення за умовчанням може бути вказане за допомогою атрибуту «default», використовуючи, тег без його тіла. Або можна вказати значення за умовчанням в тілі тега.

Тег <c: out> може здійснювати перекодовування символів «<», «>», «&» у символьні сутності XML, якщо значення атрибуту «escapeXml» рівне «true». Це означає, що символ «<» буде автоматично перекодований в «<».

Значення символьних сутностей XML, які використовуються для того, щоб перекодовувати символи, показані в таблиці 6.7.

Таблиця 6.7 – Символьні сутності XML

Символ	Сутність XML
<	<
>	>
&	&
‘	'
‘’	"

Використання мови виразів в тегу <c: out> наведено в лістингу 6.7. У цьому прикладі відбувається автоматичний приведення типів, рядок '1' буде приведений до типу int.

Лістинг 6.7 – Використання мови виразів в тегу <c: out>

```
<c: out value="${5 + 5 + '1'}"/>
```

Вивід:
11

Використання значення за умовчанням приведений в лістингу 6.8. У цьому прикладі змінна «variable» не визначена, результат виразу рівний «null». Для виведення тег <c: out> візьме значення за умовчанням, в першому тегу значення задане через атрибут «default», в другому випадку значення за умовчанням визначене в тілі тега.

Лістинг 6.8 – Використання значення за умовчанням, результат виразу

${variable} рівний «null»

```
<c: out value="${variable}" default="Variable is null"!/>
```

```
<c: out value="${variable}">
Variable is null
</c: out>
```

Вивід:
Variable is null!

Перекодування символів «<», «>», «&» у символьні сутності XML наведено в лістингу 6.9. При значенні «true» атрибуту «escapeXml» символи «<», «>» будуть замінені сутностями. Текст не буде відображений курсивом у вікні браузеру. У другому випадку текст буде відображений курсивом.

Лістинг 6.9 – Перекодування символів «<», «>», «&» у символьні сутності XML

```
<%
    String text = new String ("<I> Hello World</I>");
pageContext.setAttribute ("text", text);
%>
<c: out value="${text}" escapeXml="true">
<c: out value="${text}" escapeXml="false">
```

Вивід:
1) <I&gt Hello World </I>
2) Hello World

Установка змінних

Тег <c: set> використовується для установки значення контекстної змінної у будь-якій зоні видимості JSP, а також він може використовуватися для установки значення полів указаного об'єкту.

Синтаксис тега <c: set> приведений в таблиці 6.8,

Таблиця 6.8 – Синтаксис тега <c: set>

Завдання значення контекстній змінній в зоні видимості JSP
<c: set value="value" var="varName" [scope="{**page**|request|session|application}"]/>
Завдання значення змінній, використовуючи тіло тега
<c: set var="varName" [scope="{**page**|request|session|application}"]> Тіло тега </c: set>
Завдання значення поля змінною, використовуючи атрибут
<c: set value="value" target="target" property="propertyName"/>
Завдання значення поля змінною, використовуючи тіло тега
<c: set target="target" property="propertyName"> Тіло тега </c: set>

Опис атрибутів тега <c: set> приведений в таблиці 6.9.

Таблиця 6.9 – Опис атрибутів тега <c: set>

Атрибут	Тип	Опис
value	Object	Вираження
var	String	Ім'я контекстної змінної, якою буде задано значення, отримане при виконанні виразу в атрибуті «value». Тип змінної буде встановлений таким же, як і тип результату виразу
scope	String	Зона видимості змінної
target	Object	У цьому атрибуті вказується цільовий об'єкт, полю якого буде задано значення, отримане при виконанні виразу в атрибуті «value». Цільовий об'єкт повинен наводиться до JavaBean і мати настановний метод для поля вказаного в атрибуті «property». Чи мати тип java.util.Map
property	String	Ім'я поля цільового об'єкту

Для завдання значення змінній необхідно використати атрибути «var», який задає ім'я змінної, і «value», який задає значення. Встановлювати значення атрибуту «value» можна, використовуючи MB або тіло тега.

При використанні цільового об'єкту, цільовий об'єкт повинен мати тип JavaBean або java.util.Map. Якщо цільовий об'єкт – це JavaBean, то він повинен містити відповідні настановні і отримуючі методи (getter/setter methods). Якщо цільовий об'єкт рівний «null» і об'єкт не має типу Map або JavaBean, або JavaBean не має відповідних настановних і отримуючих методів для полів, то викидається виключення. Ім'я цільового об'єкту і його поле задається атрибутами «target» і «property».

Зоною видимості змінної за умовчанням є «page», але зона видимості змінної може бути задана використовуючи атрибут «scope».

Варто відмітити, що не усі змінні можуть бути змінені. Наприклад, можна отримати значення параметра запиту, використовуючи тег <c: out value="${param.name}"/>. Але при зміні значення параметра, використовуючи тег <c: set var="param.name" value="new value"/>, значення залишиться старим.

Це відбувається відповідно до специфікації сервлетів, де неможливо змінювати значення параметрів, оскільки немає методу ServletRequest.setParameter(). Значення параметрів спеціально зроблені незмінними.

Для завдання значення змінній, використовуючи конкатенацію результатів виразів необхідно використати тіло тега. Використання тега <c: set> для завдання значення змінній наведено в лістингу 6.10. Перший тег визначає змінну «text», зі значенням 10, і зоною видимості «page». У другому тегу змінна створюється в зоні видимості «session». У третьому тегу значення визначається в тілі тега.

Лістинг 6.10 – Використання тега <c: set> для завдання значення змінній

```
<c: set value="${5+5}" var="text"/>

<c: set value="${5+5}" var="text" scope="session"/>

<c: set var="text">      ${5 + 5}
</c: set>
```

Використання тега <c: set> для завдання значення полю цільового об'єкту наведено в лістингу 6.11. У цьому прикладі тег встановлює значення «1980» для поля «year», цільового об'єкту «item».

Лістинг 6.11 – Використання тега <c: set> для завдання значення полю цільового об'єкту

```
<c: set value="1980" target="item" property="year"/>

<c: set target="item" property=" year">
        1980
</c: set>
```

Видалення змінних

Тег <c: remove> використовується для видалення контекстної змінної у будь-якій зоні видимості JSP.

Синтаксис тега <c: remove> приведений в таблиці 6.12.

Таблиця 6.12 – Синтаксис тега <c: remove>

```
<c: remove var="varName" [scope="{page|request|session|application}"]/>
```

Опис атрибутів тега <c: remove> приведений в таблиці 6.13.

Таблиця 6.13 – Опис атрибутів тега <c: remove>

Атрибут	Тип	Опис
var	String	Ім'я контекстної змінної, яка буде видалена
scope	String	Зона видимості змінної

Якщо атрибут «scope» не заданий, змінна видаляється згідно з методом PageContext.removeAttribute (varName).

Якщо зона видимості задана, змінна видаляється відповідно до методу PageContext.removeAttribute (varName, scope).

Обробка виключень

Тег <c: catch> є новим способом обробки виключень в JSP. Ця дія може обробляти помилки будь-якого тега, а також декількох тегів, якщо ці теги ув'язнені усередині блоку <c: catch>. Ця дія не призначена для заміни сторінок помилок JSP (JSP error pages), воно було введене для виборчої обробки виключень.

Виключення, які не є критичними, можуть відловити тегом <c: catch> і таким чином можуть бути краще оброблені.

Тег <c: catch> відловлює Java.lang.Throwable виключення, викинуте будь-якою з вкладених дій.

Синтаксис тега <c: catch> приведений в таблиці 6.14, опис атрибутів приведений в таблиці 6.15.

Таблиця 6.14 – Синтаксис тега <c: catch>

```
<c: catch [var="varName"]>
        Вкладені дії
</c: catch>
```

Опис атрибутів тега <c: catch> приведений в таблиці 6.15.

Таблиця 6.15 – Опис атрибутів тега <c: catch>

Атрибут	Тип	Опис
var	String	Ім'я контекстної змінної, для зберігання виключення викинутого з вкладених дій. Тип змінної ідентичний типу викинутого виключення.

Коли викидається виключення, воно зберігається в змінній із зоною видимості «page», ім'я змінної визначається атрибутом «var» тега.

Якщо змінна задана, а виключень немає, вона видаляється.

Завдання імені змінній атрибутом «var» необов'язково, якщо цей атрибут не вказаний, виключення ловиться, але ніде не зберігається.

Використання обробки виключень приведене в лістингу 6.12.

Лістинг 6.12 – Використання обробки виключень

```
<c: catch var="urlError">
        <!-- Вкладені теги -->
        <c: import url="test.jsp"/>
</c: catch>

<c: if test="${not empty urlError}">
        <b> Файл не знайдений </b> <br>
        Інформація про помилку
        <c: out value="${urlError}"/>
</c: if>
```

Використовуючи вираз ${not empty urlError}, можна визначити, сталася помилка або ні. Якщо помилка існує, здійснюється виведення повідомлення про помилку.

Тег <c: catch> не призначений для заміни механізму errorPage, який існує в специфікації JSP. Цей тег призначений для обробки менш критичних помилок.

6.3.5.2 Умовні дії

Бібліотека JSTL підтримує прості і умовні взаємовиключні дії.

Використовуючи прості і умовні взаємовиключні дії, можливо, створювати прості сторінки з дуже складною логікою роботи. Для організації умовних дій на JSP-сторінках, найчастіше, застосовуються скриплети. Використання скриплетів в JSP-сторінках перетворює їх на безлад і зменшує їх читаність. Умовні дії бібліотеки JSTL полегшують умовну обробку на JSP-сторінці, вони дозволяють замінити скриплети роблячи код зрозумілішим і читабельним.

Умовними діями є: тег <c: if> для простих умов і комбінація тегів <c: choose>, <c: when> і <c: otherwise> для обробки взаємовиключних умов.

Прості умови, тег <c: if>

Дія <c: if> є простою умовою.

Синтаксис тега < c: if > приведений в таблиці 6.16.

Таблиця 6.16 – Синтаксис тега <c: if>

Проста умова, без використання тіла тега
<c: if test="testCondition" var="varName" [scope="{page|request|session|application}"]/>
Проста умова, використовуючи тіло тега
<c: if test="testCondition" [var="varName"] [scope="{page|request|session|application}"]> Тіло тега </c: if>

Опис атрибутів тега <c: if> приведений в таблиці 6.17.

Таблиця 6.17 – Опис атрибутів тега <c: if>

Атрибут	Тип	Опис
test	boolean	Тестова умова на МВ. Якщо в результаті виконання вираження атрибут набуває значення «true,» відбувається виконання тіла дії
var	String	Ім'я контекстної змінної, для зберігання результату виразу атрибуту «test». Тип контекстної змінної «boolean»
scope	String	Зона видимості змінної

Тег умовних дій <c: if> має атрибут «test», який представляє умову виконання. Атрибут «test» може бути заданий з використанням МВ. Якщо в результаті виконання виразу атрибут «test» набуває значення «true» відбувається виконання тіла дії. Якщо умова «test» набуває значення «false», тоді вміст тіла пропускається.

Результат виконання вираження атрибуту «test» можна експортувати в змінну, що має задану зону видимості, вказавши атрибути «var» і «scope».

Використання тега умовних дій <c: if> наведено в лістингу 6.13.

Лістинг 6.13 – Використання тега умовних дій <c: if>

<jsp: useBean id="testClass" scope="page" class="domain.TestClass"/> <jsp: setProperty name="testClass" property="*"/> **<c: if test="${not empty param.id}">**

<%

DBWorker.addElem (testClass);

```
    %>
</c: if>
```

В умовному тегу <c: if>, ми можемо використати МВ в атрибуті «test» для визначення наМВності в запиті атрибуту з ім'ям «id». Якщо атрибут є присутнім і значення не порожнє – виконується тіло тега <c: if> (елемент буде доданий у базу даних).

Взаємовиключні умови, теги <c: choose>, <c: when> і <c: otherwise>

Теги <c: choose>, <c: when> і <c: otherwise> використовуються для створення умовних дій, що взаємно виключаються.

При використанні взаємовиключних умовних дій обробка тіла тега здійснюється тільки у одного з ряду альтернативних дій. Використання цих тегів аналогічно використанню структури «switch/case/default» в мові Java.

Синтаксис тегів приведений в таблиці 6.18.

Таблиця 6.18 – Синтаксис тегів взаємовиключних умов

Тег <c: choose>
<c: choose> Тіло тега Вкладені теги <when> і тег <otherwise> </c: choose>
Тег <c: when>
<c: when test="testCondition"> Тіло тега </c: when>
Тег <c: otherwise>
<c: otherwise> Тіло тега </c: otherwise>

Опис атрибутів тега <c: when> наведено в таблиці 6.19.

Таблиця 6.19 – Опис атрибутів тега <c: when>

Атрибут	Тип	Опис
test	boolean	Тестова умова на МВ. Якщо в результаті виконання виразу атрибут набуває значення «true» відбувається виконання тіла дії, при цьому виконання інших вкладених тегів <c: when> пропускається

При використанні умовних дій <c: if>, тег <c: if> завжди обробляє своє тіло тега, якщо результатом умови «test» є значення «true». При виконанні дій

<c: when>, завжди виконується тільки перший тег чия умова «test» набуває значення «true», інші теги пропускаються.

Дій <c: when> може бути необмежена кількість.

Використання дії <c: otherwise> необов'язково, але якщо ця дія використовується, то вона має бути єдиним і останнім тегом, вкладеним в тег <c: choose>.

Тіло тега <c: otherwise> обробляється, якщо жодна з дій <c: when> не набуває значення «true».

Використання взаємовиключних умов приведене в лістингу 6.14.

Лістинг 6.14 – Використання взаємовиключних умов

```
<c: choose>
  <c: when test="${month==1}">
            January
  </c: when>
  <c: when test="${month==2}">
            February
  </c: when>
  <c: when test="${month==3}">
            March
  </c: when>
…
…
  <c: when test="${month==12}">
            December
  </c: when>
  <c: otherwise>
            Invalid month.
  </c: otherwise>
</c: choose>
```

У цьому прикладі ми можемо використати тег <c: choose> для виведення текстової назви місяця по його номеру, який заданий змінній «month».

6.3.5.3 Ітератори, дії для роботи з колекціями

Ітераційні дії бібліотеки JSTL спрощують доступ до елементів колекцій.

Колекції або контейнери – це класи що дозволяють зберігати і проводити операції з безліччю об'єктів. Колекції використовуються для збереження, отримання, маніпулювання даними і забезпечують агрегацію одних об'єктів іншими. Термін колекція має абсолютно інше значення у бібліотеці JSTL, чим в мові Java. У бібліотеці JSTL велика безліч різних типів об'єктів можуть функціонувати як колекції.

Наступні типи класів визначають термін колекції у бібліотеці JSTL :

- будь-який клас, який реалізує інтерфейс java.util.Collection;
- масиви об'єктів, які мають примітивний тип даних;

- будь-який клас, який реалізує інтерфейс java.util.Iterator;
- будь-який клас, який реалізує інтерфейс java.util.Enumeration;
- будь-який клас, який реалізує інтерфейс java.util.Map.

Усі класи, які належать «Java Collections API», можуть бути використані для роботи з ітераційними тегами бібліотеки JSTL. Ці класи включають: LinkedList, ArrayList, Vector, Stack, Set і Map.

Одним з достоїнств бібліотеки JSTL є те, що можна використати рядки з даними розділеними заданими роздільниками. Наприклад, список значень «Monday, Tuesday, Wednesday, Thursday, Friday» може бути оброблений як колекція днів тижня, в якому кожен елемент має тип «String».

Бібліотека JSTL підтримує два теги, які призначені для роботи з колекціями. Для перебору елементів колекції застосовується тег <c: forEach>. Для токенезації рядків застосовується тег <c: forTokens>.

Дія <c: forEach> у циклі виконує тіло тега для кожного елементу колекції, який знаходиться в заданому діапазоні.

Дія <c: forToken> призначена для синтаксичного розбору і ітерацій над токенами, які розділені заданими роздільниками.

Ітераційні дії тег <c: forEach>

Синтаксис тегів приведений в таблиці 6.20.

Таблиця 6.20 – Синтаксис тега <c: forEach>

Перебір усіх елементів колекції
<c: forEach [var="varName"] items="collection" [varStatus="varStatusName"] [begin="begin"] [end="end"] [step="step"]> Тіло тега </c: forEach>
Перебір елементів колекції в заданому діапазоні
<c: forEach [var="varName"] [varStatus="varStatusName"] begin="begin" end="end" [step="step"]> Тіло тега </c: forEach>

Опис атрибутів тега <з: forEach> наведено в таблиці 6.21.

Таблиця 6.21 – Опис атрибутів тега <c: forEach>

Атрибут	Тип	Опис
var	String	Ім'я контекстної змінної, для зберігання поточного елементу колекції. Тип контекстної змінної заданий типом елементу колекції. Якщо не заданий атрибут «items», то атрибут ення індексу поточного елементу, в діапазоні заданого атрибутами «begin» і «end».
items	Типи колекцій описані вище.	Колекція елементів
varStatus	String	Ім'я контекстної змінної, для зберігання поточного статусу ітерації. Тип контекстної змінної «javax.servlet.jsp.jstl.core.LoopTagStatus«
begin	int	Індекс першого елементу колекції, з якого розпочинається перебір. Індекс першого елементу колекції дорівнює нулю.
		Якщо атрибут не заданий, перебір здійснюється з нульового індексу
end	int	Індекс останнього елементу колекції, на якому закінчується перебір. Якщо атрибут не заданий, перебір здійснюється до останнього елементу колекції
step	int	Крок перебору.

Продовження Таблиці 6.21

Дія <c: forEach> використовує атрибут «items» для встановлення колекції об'єктів. Якщо атрибут «items» заданий і має значення не рівне «null», тіло тега виконується циклічно над кожним елементом колекції. Якщо атрибут «items» не заданий або має значення «null», тоді колекція вважається як би порожньою, але при цьому тіло тега виконується задане атрибутами «begin» і «end» кількість разів.

Поточний елемент колекції експортується як змінна, задана атрибутом «var». Змінна має успадковану від колекції зону видимості. Для більшості колекцій тип змінної визначається типом об'єкту колекції.

Якщо об'єкт колекції має тип java.util.Map, тоді поточний елемент матиме тип java.util.Map.Entry.

Цей тип має наступні публічні поля:

– «key» – ключ елементу в об'єкті Map;
– «value» – елемент, що відповідає цьому ключу.

Якщо колекція задана масивом, елементи якого мають примітивний тип даних, то при переборі кожен елемент буде автоматично перетворений в клас-обгортку, наприклад: «int» в «Integer», «float» в «Float» і т. д.

Ітераційні теги також можуть експортувати об'єкт, який зберігає інформацію про статус ітерації. Для завдання змінної статусу ітерації використовується атрибут «varStatus». Змінна задана цим атрибутом має тип «LoopTagStatus», пакет javax.servlet.jsp.jstl.core. Опис методів інтерфейсу «LoopTagStatus» приведений в таблиці 6.22.

Таблиця 6.22 – Опис методів інтерфейсу «LoopTagStatus»

Тип	Метод	Опис
Integer	getBegin ()	Повертає значення атрибуту «begin», або значення «null», якщо атрибут не заданий
Integer	getEnd ()	Повертає значення атрибуту «end», або значення «null», якщо атрибут не заданий
Integer	getStep ()	Повертає значення атрибуту «step», або значення «null», якщо атрибут не заданий
int	getCount ()	Повертає кількість виконаних ітерацій
Object	getCurrent ()	Повертає поточний елемент ітерації
int	getIndex ()	Повертає поточний індекс елементу
boolean	isFirst ()	Повертає значення «true» якщо виконується перша ітерація
boolean	isLast ()	Повертає значення «true» якщо виконується остання ітерація в циклі

Усі ітераційні теги підтримують завдання меж перебору елементів колекцій. Атрибути «begin» і «end» визначають межі. Якщо атрибут «begin» визначений, то його значення має бути більше або дорівнює нулю. Якщо атрибут «end» визначений, то його значення має бути більше або дорівнює значенню атрибуту «begin».

Атрибут «items» не обов'язковий для використання в тегу <c: forEach>. Якщо атрибут «items» не визначений, то тоді значення поточного елементу встановлюється рівним значенню цілочисельного поточного індексу. Таким чином, можливо, реалізувати аналог циклу «for» в мові Java.

Атрибут «step» дозволяє описувати цикли із заданим кроком. Якщо атрибут «step» визначений, то він має бути більший або дорівнює одиниці. Використання тега <c: forEach> наведено в лістингу 6.15.

Лістинг 6.15 – Використання тега <c: forEach>

```
<%
    Collection<TestClass> listObj = DBWorker.getAll (); pageContext.setAttribute ("listObj", listObj);
%>
<table border=1>
    <tr>
    <td> id </td>
    <td> name </td>
    <td> address </td>
    </tr>
    <c: forEach var="item" items="${listObj}">
        <tr>
        <td><c: out value="${item.id}"/></td>
        <td>${item.name}</td>
        <td>${item.address}</td>
        </tr>
    </c: forEach>
```

У цьому прикладі створюється колекція існуючих об'єктів, яка додається в область видимості сторінки pageContext.

Використовуючи ітераційний тег <c: forEach> об'єкти колекції «listObj» виводяться в таблицю. Для виведення значень об'єктів використовується тег <c: out>.

7 ЗАВДАННЯ НА КУРСОВИЙ ПРОЕКТ

7.1 Загальна частина для усіх варіантів. Обов'язкові вимоги

Обов'язковою умовою є реалізація серверної частини корпоративного застосунку в середовищі серверів БД на вибір: PostgreSQL, MySQL, MS SQL Server, Oracle, а клієнтської частини у вигляді Web-застосунку та (або) мобільного застосунку. Обов’язково має бути виконано розшарування застосунку як мінімум на два шари – доступу до даних та представлення.

Курсовий проект може бути виконаний декількома учасниками (2-3 в залежності від складності завдання за погодженням з керівником). В такому випадку обов’язково за одним учасником закріплюється як мінімум один шар, це має бути зазначено в окремому розділі пояснювальної записки під назвою «Список авторів». Прізвища всіх виконавців з особистим підписом мають бути зазначені і в титульному аркуші, і в технічному завданні, і в списку авторів.

Як тематика курсового проекту може бути використана одна на вибір з індивідуальних завдань, що наведені в пункті 7.2 цих методичних вказівок, або власна тематика за згоди керівника.

Вимоги до організації бази даних

1. Побудувати концептуальну модель предметної області (ПрО), що містить не менш 4-х об'єктів, пов'язаних між собою взаємозв'язками типу 1:M і M : N, у вигляді діаграми «сутність-зв'язок» і дати її опис.
2. Побудувати логічну модель ПрО і виконати її нормалізацію (не менше 5-ти таблиць) з вказівкою:
   - первинних і зовнішніх ключів;
   - типів цих атрибутів;
   - індексів.
3. На рівні структури БД реалізувати засоби забезпечення цілісності даних:
   - унікальність і обов'язковість введення первинних ключів;
   - підтримка посилальної цілісності для зовнішніх ключів (при видаленні і оновленні);
   - значення атрибутів за умовчанням і обов'язковість введення значень атрибутів.
4. На рівні бізнес-логіки реалізувати забезпечення обмежень на значення цих атрибутів виду : "інтервал", значення", що "перераховує, і "порівняння значень двох атрибутів однієї таблиці".
5. Реалізувати запити до БД на пошук, оновлення і видалення даних;
6. Реалізувати не менш 2-х процедур, що зберігаються. Продемонструвати їх виклик в Java.
7. Реалізувати не менш 3-х тригерів (для Insert, Update і Delete).
8. Сформувати декілька варіантів представлень.
9. Реалізувати призначення, перевірку і відміну прав доступу.

10. Реалізацію надійності збереження даних: механізми реплікації та (або) кластеризації даних.

Вимоги до призначеного для користувача інтерфейсу

Обов'язкова наявність форм для перегляду, додавання, редагування і видалення записів усіх таблиць.

Окрім цього, в кожному завданні вказані додаткові форми, які треба реалізувати.

Вимоги до пояснювальної записки

Основні пункти змісту пояснювальної записки і їх опис наводяться в таблиці 7.1.

Таблиця 7.1 – Зміст пояснювальної записки

Пункт змісту	Опис
ВСТУП	Актуальність теми, необхідність в створенні програмної системи (корпоративного застосунку).
1. АНАЛІЗ ВИРІШУВАНОЇ ЗАДАЧІ	
1.1 Аналіз предметної області	Детальний опис предметної області. Містить опис сутностей, з якими проходитиме робота, взаємозв'язок сутностей між собою.
1.2 Мета і завдання системи	Приклад: «Метою системи управління завданнями є автоматизація видачі і контролю завдань, обліку співробітників і відділів». Проте однієї пропозиції буде мало, проявляйте креативність, студент повинен уміти думати і викладати свої думки.
1.3 Призначення системи	Хто користуватиметься системою, чому вона потрібна, що вона дозволяє зробити краще, ніж інші. Сюди якнайкраще підійде Діаграма варіантів використання системи (Use Case) або словесний опис.
1.4 Вимоги до системи	Необхідно описати детальніше те коротке завдання на курсову роботу, яке було отримано (повний опис усій бізнес логіки і процесів). Вітається самостійний опис завдання.
2 ПРОЕКТУВАННЯ	

Продовження Таблиці 7.1

Пункт змісту	Опис
2.1Вибір інструментальних засобів розробки системи	
2.1.1 Сервер баз даних	Слід провести порівняння тих серверів баз даних, які можна було б використати для виконання цієї роботи. З них вибрати якийсь один і описати причину вибору (чому вибрали).
2.1.2 Технології реалізації системи	Слід коротко описати використовувані технології і що вони дозволяють: – J2EE: Java Server Pages 2.0 – Java Servlet 2.5 – Java Server Pages Standard Tag Library – JPA
2.2 Проектування архітектури системи	Для написання цього пункту слід звернутися до розділу 1 цих методичних вказівок.
2.2.1 Проектування шару бізнес-логіки і бізнес-правил	
Визначення об'єктів системи	Опис класів домена, визначення полів і методів.
Проектування бізнес-правил	Словесний опис алгоритмів, схеми алгоритмів. Вітається використання або діаграми діяльності (Activity), або діаграми станів (State).
2.2.2 Проектування шару доступу до даних	
Проектування фабрики DAO	Опис структури DAO. В ході виконання курсового проекту треба було розібратися з готовою реалізацією цього шару і переробити на свою реалізацію. У цьому пункті необхідно описати те, що вийшло і привести діаграму класів.
Проектування інтерфейсів DAO	Діаграма класів. Опис кожного інтерфейсу і його призначення.
2.2.3 Проектування шару відображення	Детальний опис нижченаведених пунктів. Винести діаграми, які здавалися як етап курсового проекту.
Проектування веб-сторінок	

Продовження Таблиці 7.1

Пункт змісту	Опис
Навігація між сторінками	
3 РОЗРОБКА	
3.1 Розробка бази даних системи	
3.1.1 Розробка схеми бази даних	Схема бази даних і опис зв'язків між таблицями, первинних і зовнішніх ключів.
3.1.2 Забезпечення цілісності даних	Слід описати обмеження цілісності згідно п.3-4 підрозділи «Вимоги до організації бази даних».
3.1.3 Розробка базових запитів	Визначити і дати повний опис базових запитів на вибірку, оновлення і видалення даних. Привести їх лістинги на SQL.
3.1.4 Вибір індексів	Визначити які поля стосунків (таблиць) використовуватимуться як індекси. Обґрунтувати вибір. Реалізація індексів на SQL.
3.1.5 Розробка процедур, що зберігаються, і тригерів	Дати повний опис процедур, що зберігаються, і тригерів БД. Привести реалізацію на SQL.
3.1.6 Розробка представлень	Дати повний опис представлень БД. Привести реалізацію на SQL.
3.1.7 Організація захисту даних	Визначити права груп користувачів системи на доступ до об'єктів БД. Привести реалізацію на SQL.
3.1.8 Організація надійності даних	Обгрунтувати архітектуру реплікації або (та) кластеризації даних, навести лістинг файлів налаштувань відповідних серверів в додатки
3.1.3 Об’єктно-реляційне відображення	Слід описати основні структури використовувані у файлах sql-мапінга: власне опис відображення об'єктів на таблиці, вхідні і вихідні параметри при описі SQL запитів. Повний лістинг файлів мапінга винести в додатки.
3.2 Розробка модулів системи	У розділі проектування наводяться діаграми класів, а в розділах розробки наводяться класи реалізації на Java і їх опис.
3.2.1 Розробка модулів шару бізнес логіки і бізнес правил	Код реалізації обмежень цілісності БД на Java з детальними коментарями. Код реалізації викликів процедур, що зберігаються, на Java з

Продовження Таблиці 7.1

Пункт змісту	Опис
Продовження Таблиці 7.1	ітарями.
3.2.2 Розробка модулів шару доступу до даних	
3.2.3 Розробка модулів шару відображення	
ВИСНОВКИ	Написати, що було зроблено в результаті виконання курсового проекту, що не вдалося зробити, як можна удосконалити застосунок.
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ	Мінімум 10 джерел
ДОДАТКИ	У додатки винести:
	1. Лістинг класів домена 2. Лістинг класів DAO (інтерфейси + реалізація, фабрика) 3. SQL-скрипт бази даних. 4. Файли налаштувань реплікації (кластеризації) 4. Файли з SQL-мапінгом 5. Лістинг JSP сторінок 6. Скріншоти призначеного для користувача інтерфейсу 7. ANT-скрипт (build.xml) для проекту збірки
ДОДАТОК А	
…	
ДОДАТОК N	

Усі матеріали, викладені у курсовому проекті, мають відповідати принципам академічної доброчесності.

7.2 Індивідуальні завдання

7.2.1 Університет

Об'єкти: група; студент; викладач; предмет; оцінка.

Бізнес-логіка: підрахунок середнього балу для студента при введенні. При відрахуванні студента відправляти йому повідомлення на пошту. Відрахування проводитися автоматично, коли у студента чотири незадовільні оцінки.

Форми: Список студентів по середньому балу.

7.2.2 Футбольний чемпіонат

Об'єкти: країна; клуб; гравці; матчі.

Бізнес-логіка: після кожного матчу проставляти рахунок клубу. Посилати повідомлення на пошту президентові клубу. У разі програшу – відсилання усім гравцям повідомлення про зменшення зарплати в цьому місяці.

Форми: таблиця чемпіонату.

7.2.3 Всеукраїнська мережа супермаркетів

Об'єкти: місто; супермаркет; продавці; продукти; постачальники.

Бізнес-логіка: при зменшенні кількості продуктів менше критичної маси автоматично відсилається замовлення (поштою) постачальникові продукту.

Форми: постачальники продукту по збільшенню ціни.

7.2.4 Фірма екстремального спуску з гір

Об'єкти: гора; засіб спуску (сноуборд, лижі...); заходи після спуску (душ, чай, заспокійливе); клієнт, рахунок.

Бізнес-логіка: підрахунок загальної вартості спуску з гори, зняття цих грошей з рахунку і відправка повідомлення клієнтові.

Форми: TOP10 активних клієнтів за останній місяць.

7.2.5 Корпорація з продажу земельних ділянок на планетах сонячної системи

Об'єкти: планета; клієнт; ділянка; розклад рейсів на планету; засіб пересування.

Бізнес-логіка: підраховувати залишок вільного місця на планеті після видалення площі усіх ділянок. При знищенні планети астероїдом або захопленні позаземними цивілізаціями (зміна статусу планети) – відсилання повідомлення усім власникам ділянки на планеті.

Форми: виведення клієнтів планети по убуванню площі ділянок.

7.2.6 Мережа зоопарків

Об'єкти: місто; зоопарк; тварина; робітник; послуги, які робітник може робити тварині (чищення, миття, розчісування, годування).

Бізнес-логіка: при додаванні нової тварини зв'язувати її з працівником, у якої найменша кількість тварин такого виду. Також посилати працівникові повідомлення на пошту.

Форми: TOP10 працівників місяця.

7.2.7 Форум

Об'єкти: рубрика; тема; повідомлення; користувачі, список некультурних слів.

Бізнес-логіка: фільтрація доданих повідомлень згідно з довідником некультурних слів. Відправка повідомлення користувача про неприйняття повідомлення. Дозволяти створювати тему користувачам, якщо у нього більше 10 повідомлень.

Форми: TOP10 активних користувачів.

7.2.8 Міжнародна компанія «Апельсин+»

Об'єкти: країна; фрукт; одержувач; спосіб доставки (вартість за кілометр); доставка (відстань).

Бізнес-логіка: не додавати інформацію про доставку, якщо час зберігання фруктів менше ніж час доставки. Розрахунок вартості доставки. Відправляти повідомлення одержувача про підтвердження доставки.

Форми: TOP10 популярних фруктів. TOP10 країн, в які поставляється найбільша кількість фруктів.

7.2.9 Національна компанія «Пилорама»

Об'єкти: лісник (поставляє дерево); тип дерева; постачання (лісником); склад; продукція, що випускається; одержувач товару.

Бізнес-логіка: не дозволяти змішувати в постачаннях хвойні і листяні породи деревини. На кожні 10 кубометрів деревини лісник додає 11-й кубометр безкоштовно. Відправляти повідомлення про підтвердження одержувачеві.

Форми: найбільш використовувана деревина, найбільш активні лісники.

7.2.10 Їдальня «Радянська»

Об'єкти: блюда; напої; продукти; меню на день; клієнт; замовлення.

Бізнес-логіка: не дозволяти змішувати м'ясні і рибні блюда в одному замовленні. У одному замовленні не брати блюда з оселедця і молочні блюда. Після формування меню відправляти інформацію про меню клієнтам поштою.

Форми: найбільш активні клієнти, блюдо, що найбільш замовляється.

7.2.11 Фірма з продажу легких наземних екологічно-чистих засобів пересування «Машина майбутнього»

Об'єкти: засіб пересування (велосипед, самокат, ...); деталь; постачальник; замовлення транспортного засобу; клієнт.

Бізнес-логіка: підрахунок вартості транспортного засобу (вартість деталей + вартість складання), виведення кількості транспортних засобів, які можуть бути проведені з доступних деталей. Відправляти повідомлення клієнтові про замовлення.

Форми: таблиця велосипедів, які можна зібрати з доступних деталей, з описом деталей.

7.2.12 Програма по контролю діяльності секретних агентів в зарубіжних країнах

Об'єкти: країна; агент; документи, які передані; завербований агентом персонал.

Бізнес-логіка: не дозволяти вербувати агентові більше восьми людей (за ними стає важко стежити). При переході кого-небудь з персоналу в статус «розсекречений» відправляти повідомлення (якщо можна – зашифроване) в штаб-квартиру про загрозу розкриття агента.

Форми: TOP10 агентів по кількості переданих документів.

7.2.13 Програма для контроля МНС рибалок, що дрейфують на крижинах

Об'єкти: море; крижина; рибалка; корабель; мореплавство.

Бізнес-логіка: підрахунок кількості рибалок, які залишаються на крижині після порятунку деякої кількості рибалок кораблем. При додаванні інформації про крижину, що дрейфує, посилати повідомлення в мореплавство з інформацією про кількість рибалок.

Форми: TOP10 крижин, з найбільшою кількістю рибалок.

7.2.14 Фірма прокату весільних суконь і аксесуарів

Об'єкти: сукня; фата; квіти; замовлення; клієнт.

Бізнес-логіка: не об'єднувати в одному замовленні жовтий і червоний колір. Якщо клієнт з Японії – не пропонувати білі квіти. Посилати підтвердження на пошту клієнтові з інформацією про замовлення.

Форми: TOP10 найбільш використовуваних суконь.

7.2.15 Туристична фірма

Об'єкти: країна; готель; номер; додаткові послуги (екскурсія, сафарі, відвідування театру, місяць-парк); клієнт; замовлення.

Бізнес-логіка: клієнтам із США не пропонувати тури в ісламські країни. Підрахунок вартості повного замовлення. Посилати підтвердження про замовлення клієнтові.

Форми: найбільш відвідувані готелі.

7.2.16 Перукарня

Об'єкти: перукар; зачіска; додаткові послуги; клієнт; замовлення.

Бізнес-логіка: не пропонувати додаткових послуг «гоління» жінкам. При додаванні замовлення посилати повідомлення перукаря.

Форми: список замовлень на день.

7.2.17 Дендропарк

Об'єкти: ліс (може бути декілька в одному дендропарку), дерево, лісник, школа.

Бізнес-логіка: якщо дерев якогось виду стає менше критичної кількості, відправляти повідомлення школі про необхідність надання учнів для посадки дерев.

Форми: найбільш корисні школи (виконали найбільшу кількість замовлень), стан лісів з інформацією про дерева.

7.2.18 Фірма по розведенню акваріумних рибок

Об'єкти: акваріум; риба; риба-екземпляр (з інформацією про вагу, довжину, масу); корм; замовлення на рибу; клієнт.

Бізнес-логіка: не дозволяти тримати в одному акваріумі хижих і не хижих риб. Відправляти повідомлення клієнтів про появу нового виду риб.

Форми: наявність рибок по акваріумах, TOP10 видів риб, що замовляються.

7.2.19 Громадська організація «Тимурівець»

Об'єкти: місто; агент-тимурівець; потенційний клієнт; вид допомоги; допомога (місце, час, вид допомоги).

Бізнес-логіка: клієнтам, вік яких менше 60 років не робити вид допомоги «перевести через дорогу». Не виконувати певні види робіт в деякі пори року (збір яблук зимою). Якщо за день виконані більше 10 добрих справ – відправляти повідомлення на центральний офіс міста про виконання плану.

Форми: TOP10 активних агентів-тимурівців.

7.2.20 Служба порятунку домашніх тварин

Об'єкти: місто; тварина; заява про зникнення; знахідка тварини; клієнт.

Бізнес-логіка: при знаходженні тварини, усім хто давав заяву про зникнення такого виду тварини посилати повідомлення на пошту. Видаляти тварину з бази, якщо її не забрали впродовж 30 днів.

Форми: пошук в знайдених тваринах екземплярів за якими-небудь параметрами.

7.2.21 Електронний пісенник

Об'єкти: виконавці; пісні; коментарі подружок; користувачі; стиль пісні (довідник).

Бізнес-логіка: змінювати колір фону сторінки при кожному додаванні коментаря. При додаванні коментаря – повідомлення користувачеві.

Форми: TOP10 подружок по кількості коментарів.

7.2.22 Корпорація по боротьбі з полтергейстом

Об'єкти: агент; вид полтергейсту; місто; замовлення на розслідування.

Бізнес-логіка: не вибирати агентів-новачків на складні види полтергейстів. Не дозволяти агентам виконувати більше 3 замовлень в тиждень. При додаванні заявки на розслідування полтергейсту відправляти повідомлення усіх агентів, які спеціалізуються по цьому виду полтергейсту.

Форми: статистика виконаних/невиконаних завдань по співробітниках, статистика замовлень і виконаних завдань по містах.

7.3.23 Програма по контролю за випадками зустрічі з НЛО

Об'єкти: місто, учасник події; тип НЛО; подія.

Бізнес-логіка: при вступі заявки про подію в однаковий час і в тому ж місці – створювати одну заявку, приєднуючи учасників події. При додаванні події на поштову адресу центрального офісу міста посилати повідомлення.

Форми: найбільш активні очевидці НЛО. НЛО, що найчастіше з'являється.

7.2.24 Аукціон

Об'єкти: товар; клієнт, які купує; лот; рубрика; проданий товар.

Бізнес-логіка: не дозволяти виставляти ціну за лот менше тієї, яка вже є зараз. При закритті аукціону автоматично додавати товар в проданий і відправляти поштове повідомлення користувача.

Форми: найдорожчі продані товари.

7.2.25 Доставка піци

Об'єкти: піцерія, піца, продукти, замовлення, машина, додаткові блюда.

Бізнес-логіка: Не дозволяти створювати замовлення на якийсь час, яке вже зайняте. При замовленні 10 піц, 11 безкоштовно. Відправляти повідомлення піцерії при оформленні замовлення.

Форми: найпопулярніші піци, поточний стан машин доставки (вільна, зайнята).

7.2.26 Дитячий оздоровчий табір "Відпочинь"!

Таблиця: заїзд дітей, загони, діти, вожаті, заходи.

Бізнес-логіка: В загоні не має бути більше 30 дітей. Для кожного загону є обмеження у віці. За перемогу в кожному заході загін отримує бали, і у кінці заїзду визначається самий кращий загін. При зміні статусу дитини (прибув, виїхав), відправляти повідомлення батьків на пошту.

Форми: Список заходів на заїзд, Список балів кожного загону.

7.2.27 Військкомат

Об'єкти: військкомат (кількість необхідних призовників), працівник військкомату; бригада по пошуку призовників; студент; заявка на виїзд до студента.

Бізнес-логіка: не дозволяти додавати у бригаду працівників, які є однофамільцями студентів. Не дозволяти формувати бригади взимку і літом (відсутність заклику). При упійманні студента бригадою, відправляти повідомлення на поштову адресу батьків.

Форми: Відстежування набору призовників (скільки залишилося).

Кількість студентів, спійманих кожною бригадою (топ 10 бригад).

7.2.28 Авіакомпанія "Зліт-посадка"

Об'єкти: літаки, міста, пілоти, замовлення, рейс.

Бізнес-логіка: Не дозволяти оформляти замовлення за відсутності вільних місць. У рейсі задається вартість польоту і пілоти. Не дозволяти пілотам літати частіше 4 раз на тиждень. При відміні рейсу, відправляти повідомлення клієнтам, адреси яких вказані в замовленнях.

Форми: найчастіше відвідувані міста.

7.2.29 Рибнагляд

Об'єкти: риб-інспектор, човен риб-інспектора, довідники риб і снастей, потенційні браконьєри, акт про упіймання.

Бізнес-логіка: Один човен дозволяється комплектувати 2-ма - 4-ма риб-інспекторами. Автоматично розраховувати вартість збитку природі, виходячи з

маси і виду спійманої риби, і використовуваних снастей. При складанні акту відправляти копію його міської адміністрації, суду, родичам браконьєра.

Форми: топ 10 риб-інспекторів і браконьєрів.

7.2.30 Агентство з продажу квартир

Об'єкти: заявки на продаж, заявки на купівлю, агенти з продажу, нотаріуси, угоди.

Бізнес-логіка: При складанні угоди агентом автоматично призначати вільного нотаріуса для оформлення документів. При вступі заявки на продаж квартири відправляти повідомлення усіх клієнтів, які подали заявку на купівлю квартири з схожими характеристиками, і навпаки.

Форми: список квартир, що продаються, складений на основі заявок на продаж; Топ10 агентів, які уклали найбільше угод.

8 РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Дейт К. Введение в системы баз данных: пер. с англ. / К.Дж. Дейт 8-е издание – М.: Вильямс, 2017. – 1328 с.
2. Гайдаржи В.І. Бази даних в інформаційних системах / В.І. Гайдаржи, І. В. Ізварін. – Університет "Україна", 2018. – 418 с.
3. Грабер М. SQL: пер. с англ. / Дж. Колби, П. Уилтон. – М.: Лори, 2016. – 643 с.
4. Селко Дж. SQL для профессионалов. Программирование / Дж.Селко, 2-е издание. – М: Лори, 2009. – 442 с.
5. Riggs S. PostgreSQL 11 Administration Cookbook / Riggs S., Ciolli G., Meesala S.K. – PacktPublishing, 2019. – 600 p.
6. Schwartz B. High Performance MySQL. Third edition / Schwartz B., Zaitsev P., Tkachenko V. – O'Reilly Media, 2012. – 826 p.
7. Ben-Gan I. Microsoft SQL Server 2012 T-SQL Fundamentals (Developer Reference) 1st Edition. – Microsoft Press, 2012. – 442 p.
8. Фаулер Мартин. Архитектура корпоративных программных приложений. – М.: «Вильямс», 2008. – 544с.
9. Janssen Thorben. Hibernate Tips: More than 70 solutions to common Hibernate problems – Thoughts on Java, 2018. – 256 p.
10. Фаулер М. Шаблоны корпоративных приложений. : пер. с англ. – М.: Диалектика, 2019. – 544 с.
11. Мельник Р. Програмування веб-застосувань (фронт-енд та бек-енд) – Л.: Львівська політехніка, 2018. – 248 с.
12. Мухамедзянов Р. Р. Java. Серверные приложения. – М.: "СОЛОН-Пресс", 2010. – 336 с.
13. Хо К. Spring 5 для профессионалов / Хо К., Харроп Р., Шефер К. : пер. с англ. – М.: Вильямс, 2016. – 1120 с.

ДОДАТОК А

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання курсового проекту з дисципліни «Бази даних»
ПІБ ……., ЗВО гр. ПІ-___
ПІБ ……..., ЗВО гр. ПІ-___
Тема роботи: БД "Супермаркет"

Передбачувані технічні та експлуатаційні результати роботи:

Програмна система управління супермаркетами і їх товарами представляє собою корпоративний додаток, що дозволяє отримувати інформацію про товари, продавців, склади і постачальників.

Розробка БД повинна включати:

а) побудову концептуальної моделі предметної області, що включає не менш 3-х об'єктів, пов'язаних між собою взаємозв'язками типу 1: М, у вигляді діаграми «сутність-зв'язок»;

б) побудову нормалізованої логічної моделі предметної області (не менше 3-х таблиць) із зазначенням: первинних і зовнішніх ключів, типів даних атрибутів;

в) реалізацію на рівні структури БД засобів забезпечення цілісності даних: унікальність і обов'язковість введення первинних ключів; підтримка посилальної цілісності для зовнішніх ключів; значення атрибутів за замовчуванням і обов'язковість введення заданих атрибутів;

г) реалізацію базових запитів до БД на пошук і додавання даних;

д) реалізацію бізнес-логіки зі сторони серверу БД: тригерів, збережених процедур та представлень;

є) реалізацію розділеного доступу до БД зі сторони серверу БД: користувачі та їх права;

ж) реалізацію надійності збереження даних: механізми реплікації та кластеризації даних.

Взаємодія з БД має бути реалізована в архітектурі клієнт-сервер з наступними відокремленими рівнями:

1) доступу до даних з використанням технологій об’єктно-реляційного маппінгу;
2) обробки даних (бізнес-логіки);
3) сервісів;
4) інтерфейсу.

Обсяг текстової та графічної документації

Пояснювальна записка до проекту обсягом 25-30 сторінок друкованого тексту формату А4 і програмна документація на систему обсягом 35-40

сторінок друкованого тексту формату А4. Обсяги текстової інформації можуть бути скориговані в процесі роботи за погодженням з керівником.

Планові терміни по етапах:
1) Узгодження завдання – __.__.____
2) Аналіз предметної області. Розробка БД – __.__.____
3) Прототип призначеного для користувача інтерфейсу – __.__.____
4) Модулі та бізнес логіка – __.__.____
5) Інтерфейс користувача – __.__.____
6) Оформлення роботи – __.__.____

Демонстрація працездатності програмного проекту до – __.__.____

Плановий термін захисту проекту– __.__.____

Керівник роботи: _____ ПІБ

Виконавці роботи: _____ ПІБ
_____ ПІБ

Дата видачі завдання
«__» _____ 20__ р

ДОДАТОК Б

СПИСОК АВТОРІВ

ПІБ співавтора	Опис частини роботи	Підпис
ЗВО гр. ПІ-__ ПІБ	Проектування діаграми використання системи. Проектування та реалізація бази даних, Проектування та реалізація шару доступу до даних. Проектування та реалізація шару бізнес-логіки. Оформлення звіту: вступ; реферат; розділ 1; розділ 2.1, 2.2, 2.3.1, 2.3.2; розділ 3.1, 3.2.1, 3.2.2.	
ЗВО гр. ПІ-__ ПІБ	Вибір засобів розробки та проектування архітектури системи. Проектування та реалізація шару відображення. Оформлення звіту: розділ 2.3, 2.3.4; розділ 3.2, 3.2.3; висновки.	