

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Теплоенергетичний факультет
Кафедра автоматизації проектування
енергетичних процесів і систем

Web-орієнтовані інформаційні системи

Методичні вказівки до вивчення дисципліни для студентів
напряму підготовки 6.050103 «Програмна інженерія»

Рекомендовано вченою радою ТЕФ НТУУ «КПІ»

Київ
НТУУ “КПІ”
2015

Web-орієнтовані інформаційні системи. Методичні вказівки до вивчення дисципліни для студентів напряму підготовки 6.050103 «Програмна інженерія». / Титенко С. В. – К.: НТУУ «КПІ» 2015. – 51с.

*Рекомендовано Вченою радою ТЕФ НТУУ „КПІ”
в якості електронного засобу навчання
(протокол № 10 від 27 квітня 2015 р.)*

Електронне навчальне видання

Web-орієнтовані інформаційні системи

Методичні вказівки до вивчення дисципліни для студентів
напряму підготовки 6.050103 «Програмна інженерія»

Автор Титенко Сергій Володимирович, канд. техн. наук.

Відповідальний редактор: С. О. Лук'яненко, завідувач кафедри АПЕПС, д-р. техн. наук, проф.

Рецензент: Богачков Ю. М., директор Інституту перепідготовки та підвищення кваліфікації НМК ІПО НТУУ «КПІ», канд. техн. наук.

© С. В. Титенко, 2015

Зміст

Зміст	3
Вступ	5
1. Системи керування контентом	6
Керування контентом.....	6
Архітектура CMS	7
2. Сторінка як ключова сутність інформаційної моделі контенту CMS ...	7
Структура веб-сторінки	7
Форми публікації сторінки як елемента контенту.....	9
Базова структура таблиці сторінок у базі даних сайту	10
Мовні версії контенту сторінки	10
3. Об'єктна модель CMS та реалізація публікації сторінки	11
4. Ієрархія контенту в CMS	13
Організація ієрархічного меню сайту	14
Алгоритм побудови ієрархічного меню сайту	15
Сторінки-контейнери.....	16
Типи сторінок-контейнерів	17
Керування сортуванням дочірніх сторінок	18
5. Ациклічний оргграф контенту та відношення псевдонімів в CMS	18
Відношення псевдонімів	19
Псевдоніми та розділ новин	21
6. Мова макрокоманд контенту в CMS	22
Реалізація парсингу макрокоманд в CMS.....	23
Реалізація обробки макрокоманд в CMS	24
Приклади макрокоманд в CMS.....	25
7. Використання універсального підходу в побудові CMS для інтернет-магазина	26
8. Фільтрація сторінок за специфічними параметрами в CMS.....	26
Реалізація фільтрації	27
Реляційна модель даних для реалізації фільтрів.....	28
Формування та обробка http-параметрів та URL.....	28
Генерація SQL-запиту.....	28
Особливості застосування логічних операцій «та/або» в SQL-запиті фільтра	29
Формування панелі вибору фільтрів.....	30
9. Конструювання нових типів контенту в CMS	30
Загальні елементи контенту	31

Інтеграція і розмежування спеціалізованих інформаційних об'єктів із загальними елементами контенту	31
Реляційна реалізація спеціалізованих інформаційних об'єктів CMS.....	33
10. Адміністративна частина сайту та автоматизація побудови інтерфейсу редагування БД.....	35
Структура класів бібліотеки автоматизованої побудови інтерфейсу редагування БД	35
Клас поля – Field	35
Клас таблиці – Table	36
Організація роботи застосунку з точки зору HTTP-параметрів.....	37
Особливості форми редагування запису	38
Підготовка скрипта та мета-опис таблиці	38
Режим перегляду таблиці	39
Обробка даних html-форми та генерація SQL-запитів.....	39
Lookur-поля вибору як засіб реалізації зв'язку один-до-багатьох в інтерфейсі редагування	40
Lookur-поля та реляційна модель	40
Реалізація lookur-поля в бібліотеці автоматизованої побудови інтерфейсу редагування БД.....	42
Автоматизована побудова інтерфейсу головний-детальний.....	43
11. Лабораторні роботи	45
Загальні вимоги до виконання лабораторних робіт	45
Лабораторна робота №1	45
Лабораторна робота №2	45
Лабораторна робота №3	46
Лабораторна робота №4	46
Лабораторна робота №5	47
Лабораторна робота №6	47
Лабораторна робота №7	47
Лабораторна робота №8	48
Лабораторна робота №9	48
Лабораторна робота №10	49
Лабораторна робота №11	49
Лабораторна робота №12	49
Лабораторна робота №13	50
Рекомендована література	50

Вступ

Веб-програмування охоплює широке коло професійних задач, пов'язаних з розробкою програмних систем, що функціонують у межах інфраструктури всесвітньої мережі WWW та її сервісів. Особливої уваги заслуговує задача проектування та побудови ефективних систем керування веб-контентом (CMS), які є фундаментом будь-яких веб-проектів, що в тій чи іншій мірі застосовують автоматизоване курування редагуванням та публікацією інформації для відвідувачів веб-ресурсів.

Одним з напрямків розвитку Всесвітньої мережі WWW є керування даними та контентом. Ця галузь містить такі напрямки як керування великими об'ємами даних, керування даними на основі хмарних обчислень, керування мультимедійними даними, а також є дотичною до Web-mining, задач кластеризації, класифікації та аналізу даних в Web, моделювання Web-контенту, Semantic Web тощо. Задача ефективного керування Web-контентом набула великої значущості для багатьох галузей, що виникли на базі інфраструктури WWW, серед яких інтернет-комерція, дистанційне навчання та освітні Web-ресурси, керування великими інформаційними порталами, розробка та підтримка корпоративних порталів, створення Web-ресурсів електронного урядування, підтримка персональних сайтів та блогів тощо.

Серед популярних програмних рішень, що використовуються для керування контентом слід зазначити такі CMS-системи: Drupal, Joomla, Wordpress, Plone та ін. Для дистанційного навчання часто використовуються системи Moodle та aTutor, а також MediaWiki як засіб створення освітніх вікі-проектів. Перевагою цих систем, що зумовило їх популярність, є відкритий програмний код, наявність великої кількості додаків та безкоштовна ліцензія на використання. Серед пропріетарних систем слід зазначити такі: Magento, 1-C Бітрікс (електронна комерція); Microsoft SharePoint, Adobe Business Catalyst (комплексні SaaS-системи); Blackboard (система дистанційного навчання).

Незважаючи на велике розмаїття програмних систем, що застосовуються для керування контентом, актуальним завданням залишається дослідження, розробка та вдосконалення засобів універсального керування інформаційними об'єктами Web-ресурсів, що дозволить спростити створення нових ресурсів різного призначення, а також забезпечить ефективні механізми їх супроводження та налаштування. У навчальному виданні розглянуто ключові засади створення сучасних Web-орієнтованих інформаційних систем.

1. Системи керування контентом

Система керування контентом (content management system, CMS) – програмне забезпечення, що дозволяє редагувати *контент* інформаційної системи (веб-сайту) за допомогою зручного інтерфейсу користувача та забезпечує публікацію контенту для відвідувачів разом із засобами навігації.

Контент – термін, що стійко закріпився в професійній термінології, пов'язаний з розробкою інформаційних веб-систем. Під **контентом** веб-сайту розуміють інформаційний *вміст*, що надається відвідувачам. Контент являє собою текст, гіпертекст, зображення, відео, аудіо, анімацію тощо.

CMS-система забезпечує дві ключові функції роботи з контентом: 1) з точки зору відвідувача веб-сайту CMS забезпечує *навігацію* та *подання* контенту сайту шляхом організації меню та інших навігаційних елементів, а також обробки запитів на отримання веб-сторінок сайту; 2) з точки зору редактора сайту CMS забезпечує зручні інтерфейси та механізми для додавання, збереження та *редагування* контенту сайту, звільняючи редактора від необхідності досконало володіти засобами розмітки веб-документів (HTML) та керування серверною структурою файлів та бази даних.

Серед інших функцій CMS – забезпечення багатокористувацького доступу до редагування контенту, підтримка реєстрації відвідувачів сайту, засоби коментування контенту тощо.

Керування контентом

Ключовим завданням CMS очевидно є *керування контентом*. Натомість, розробники нових CMS часто залишають ключове завдання системи без достатньої уваги та реалізують його виключно на механічному рівні, не створюючи достатні засоби автоматизації цього процесу. У загальному сенсі **керування контентом** – це сукупність процесів та технологій для збору, організації та публікації інформації у різних формах та на різних носіях. *Керування веб-контентом* покликано забезпечити життєвий цикл інформації, яка подається відвідувачам сайту.

Керування контентом веб-сайтів охоплює не тільки фізичні процеси розміщення веб-сторінок в інтернеті. Сучасне керування контентом вимагає розглядати контент як об'єкт моделювання, структурування і формалізації для організації його ефективного зберігання, публікації та подання кінцевим споживачам у зручному вигляді із зручними інтелектуалізованими засобами навігації. У зв'язку із значущою роллю керування контентом, першочергова увага розробників CMS повинна бути сконцентрована на розробці оптимальних моделей структурування та формалізації веб-контенту, виявленні та використанні закономірностей при переміщенні по контенту сайту, формалізації зв'язків між ділянками контенту та побудові зручних схем навігації. Подібні задачі вимагають глибшого погляду на контент веб-сайтів, аніж організація розміщення фізичного html-тексту із зображеннями

на сервері, та прокладають шлях від керування контентом до керування знаннями в веб-середовищі.

Архітектура CMS

Структура CMS визначається завданнями, які вирішує система для керування контентом. CMS складається з таких компонентів (рис.1):

- *модуль редагування* (адміністративна частина), що забезпечує виконання адміністративних функцій по редагуванню контенту сайту;
- контент у структурованому вигляді, що, як правило, включає *базу даних* для зберігання HTML-тексту веб-сторінок та сукупність упорядкованих медіа-файлів;
- *модуль подання*, ядро системи, що керує логікою подання контенту відвідувачам та забезпечує навігацію по контенту.

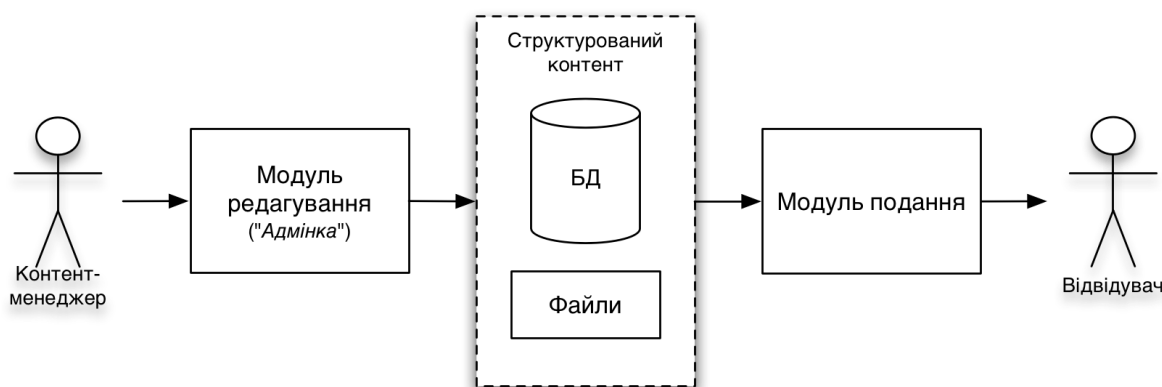


Рис. 1. Архітектура CMS.

2. Сторінка як ключова сутність інформаційної моделі контенту CMS

Ключовою інформаційною сутністю систем керування контентом сайтів є сторінка або елемент контенту. Як правило, в базі даних CMS-системи існує спеціальна таблиця, що представляє дану сутність, характеризуючи її певним набором полів. Основними полями загального елемента контенту є заголовок, html-текст, анотація та ін. В залежності від системи по-різному можуть бути реалізовані ієрархічні та структурні зв'язки між елементами контенту. В Drupal загальний елемент контенту позначається терміном Node (вузол), у Wordpress – Post (допис), в Joomla – матеріал.

Структура веб-сторінки

Кожна веб-сторінка сайту має унікальний *ідентифікатор* з точки зору відвідувача – свою *URL-адресу*.

Веб-сторінка сайту у загальному вигляді містить (рис.2):

- інформацію:
 - ефективний контент;
 - елементи навігації (меню, навігаційний шлях, додаткові посилання);
- елементи дизайну:
 - шапка сайту;
 - підвал сайту.

Ефективний контент веб-сторінки – це та інформація, заради якої дана сторінка була додана на веб-сайт, і та інформація, заради якої користувач здійснює перехід на цю сторінку. Поряд із ефективним контентом на сторінці може бути присутньою велика кількість допоміжної інформації (елементи дизайну, елементи навігації, реклама тощо). Саме ефективний контент сторінки відіграє роль логічного фрагменту інформації у загальній інформаційній структурі веб-сайту. Саме такі фрагменти та їх відношення перш за все потребують ефективного моделювання та подання в базі даних веб-системи.

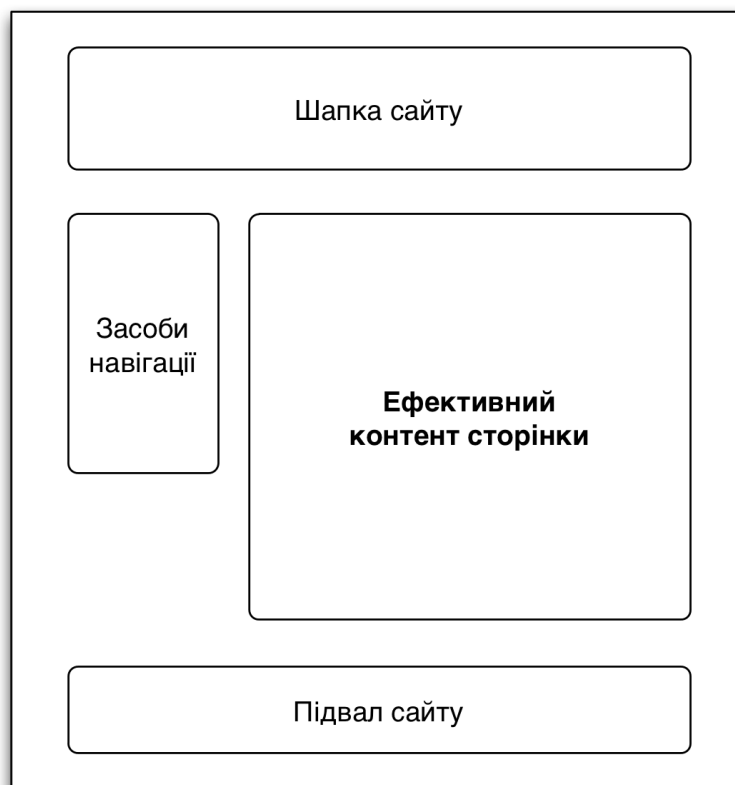


Рис. 2. Загальна структура веб-сторінки

Ефективний контент веб-сторінки сайту містить:

- заголовок;
- основний контент.

Також додатково може містити:

- анотацію;

- зображення-ілюстрацію для анотації;
- зображення-ілюстрацію основного контенту;
- короткий заголовок, якщо основний заголовок довгий (наприклад для меню);
- дати (дата створення, дата редагування тощо).

Такий набір атрибутів є базовим для більшості сторінок сайту, серед яких наступні:

- головна сторінка;
- сторінка розділу;
- інформаційна сторінка;
- стаття;
- новина;
- товар (має додаткові атрибути).

Таким чином ефективний контент веб-сторінки будемо сприймати як сутність системи, що підлягає формалізації в базі даних у якості елемента контенту або сторінки. Тож під терміном *сторінка* у межах інформаційно-логічної моделі системи сприйматимемо фрагмент інформації, що складається із згаданих вище атрибутів та може бути представленим на веб-сайті як веб-документ із власною URL-адресою.

Форми публікації сторінки як елемента контенту

Окрім окремого веб-документа елемент контенту або сторінка може публікуватися на веб-сайті також у таких формах:

- *посилання* в межах контенту деякої іншої сторінки; у такому випадку для публікації сторінки задіюється атрибут *URL* та, можливо, анотація як значення атрибуту *title* тега *<a>*;
- *посилання як елемент меню*; аналогічно у такому випадку для публікації сторінки задіюється атрибут *URL* та анотація як значення атрибуту *title* тега *<a>*;
- *блок анотованого посилання* – форма представлення посилання на сторінку, що містить заголовок-посилання (назва сторінки, на яку веде посилання), анотацію сторінки та ілюстрацію. Подібне оформлення є зручним та інформативним, тому часто зустрічається на багатьох веб-ресурсах (рис. 3). У випадку розміщення анотованого посилання у стрічці новин, блок посилання може також містити дату сторінки.

Мережа Internet та її сервіси

Мережа Internet



У 1986 році Internet зв'язувала менше 6000 комп'ютерів, причому усі вони розташовувалися на території США. Через п'ять років їх було вже 600,000, а в 1994 році їх стало більше мільйона

[Докладніше →](#)

Рис. 3. Приклад анотованого посилання

Базова структура таблиці сторінок у базі даних сайту

На основі поданої структури сторінки як елемента контенту можна описати базову структуру таблиці сторінок у БД сайту (рис.4).

Код сторінки (ключове поле), на його основі формується унікальний URL сторінки
Заголовок
Короткий заголовок
Анотація
Основний контент (html)
Ілюстрація основна
Ілюстрація для анотації
Дати

Рис. 4. Базова структура таблиці сторінок

Таблиця сторінок є ключовою таблицею БД CMS. По мірі розгляду завдань, які повинні вирішуватись CMS-системою, дана таблиця буде поповнюватись новими атрибутами.

Зауваження

Доцільним типом даних для коду сторінки є рядковий тип. Таким чином, кожна сторінка матиме унікальний текстовий ідентифікатор, наприклад *about*, *contact*, *products* тощо. Подібні ідентифікатори будуть утворювати зручні і зрозумілі URL-адреси сторінок, наприклад *site.com/?page=about* (за необхідності можна перетворити на *site.com/about* шляхом серверних та програмних налаштувань).

Мовні версії контенту сторінки

Багатомовність сайту реалізують різними способами – від спеціальних сутностей в БД, що містять переклад, до окремих гілок в ієрархії контенту, де нова мовна версія сайту – це просто інший розділ сайту. Подібні підходи не завжди ефективні з точки зору зручності використання та зручності реалізації.

Відповідно до моделі контенту, де сторінка – це основна сутність в БД, багатомовність зручно реалізувати на базі таблиці сторінок. Таким чином, для кожного з полів, що містять текстову інформацію створюється аналогічне поле для іншої мовної версії сайту, наприклад: *textUa*, *textRu*, *textEn* – поля, що містять основний html-контент для різних мовних версій. Подібний підхід дозволить використовувати ідентичну бізнес-логіку у роботі із сторінками не залежно від мовної версії.

3. Об'єктна модель CMS та реалізація публікації сторінки

Для реалізації CMS пропонується створити два базові класи – клас системи (ядро) та клас сторінки. Уся робота із сторінками сайту відбувається за допомогою класу сторінки. Клас ядра відповідає за загальну логіку роботи системи та обробку ключових запитів. Діаграма послідовності роботи системи показана на рис. 5.

Процес публікації сторінки містить наступні етапи:

1. Користувач ініціалізує запит до веб-сайту.
2. Ядро системи обробляє запит та видобуває код запитуваної сторінки.
3. По коду сторінки створюється об'єкт поточної сторінки (ОПС).
4. Під час ініціалізації ОПС звертається до бази даних для отримання відповідного запису із таблиці сторінок.
5. Ядро системи розпочинає процес публікації сторінки.
6. Відповідно до шаблону готується заголовок документа та навігаційні елементи.
7. Ядро системи звертається до метода публікації в ОПС.
8. Система завершує формування документа та надсилає користувачеві згенеровану веб-сторінку.

Особливості безпосередньої публікації документа повинні спиратися на роботу спеціальної підсистеми шаблонізації, що дозволяє розмежувати бізнес-логіку серверного коду від подання контенту засобами HTML.

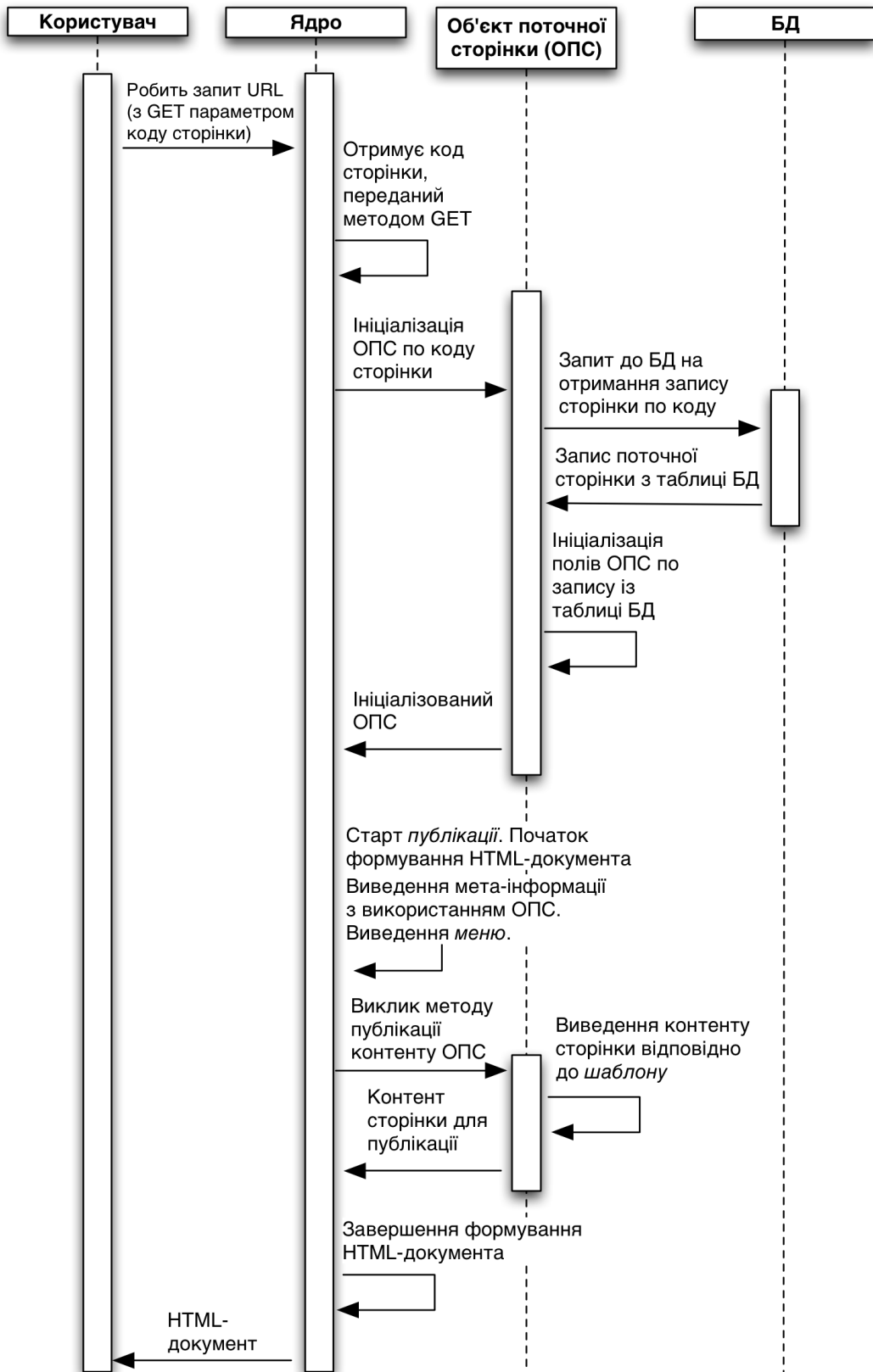


Рис.5. Діаграма послідовності CMS

4. Ієрархія контенту в CMS

Ієрархія – це розташування частин або елементів цілого в певному порядку від вищого до нижчого. Ієрархія – найзручніший і природно зрозумілий спосіб структурування речей, це ж справедливо і для елементів контенту сайту. Незважаючи на те, що ієрархічне структурування в Web викликає труднощі у випадках великої кількості інформації, тим не менше ієрархія, як базова структура, для організації контенту є зручним і практично доцільним способом структурування інформації веб-ресурсу.

Практичне значення ієрархічної структури контенту розповсюджується на обидві категорії користувачів CMS – на відвідувачів сайту і на редакторів.

Об'єктивно *навігація* по веб-ресурсу передбачає ієрархічно-подібну схему переходів від головної сторінки до розділу, від розділу до підрозділу і т.д. Такі шляхи є базовими маршрутами навігації. Звичайно окрім базових маршрутів є також допоміжні, асоціативні, упорядковані та ін. Натомість, наявність різноманітних шляхів навігації не применшує значущості ієрархічних схем переміщення по веб-ресурсу.

З іншого боку для редактора сайту наявність ієрархічно структурованого контенту спрощує внутрішню орієнтацію, пошук, редагування та додавання нового контенту в потрібне місце інформаційної структури ресурсу.

Таким чином, вважається доцільним зробити базовою структурою контенту *ієрархію сторінок*, що виражатиметься в *деревоподібній структурі* елементів контенту. Для реалізації такої моделі в реляційній БД CMS-системи немає необхідності створювати додаткові таблиці для збереження відношень між сторінками, як це часто робиться в розповсюджених системах. В даному випадку в *таблицю сторінок* необхідно додати поле, яке вказуватиме на батьківську сторінку. Таке відношення в БД прийнято називати зворотним (рис.6).

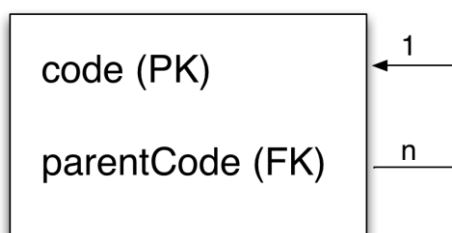


Рис. 6. Батьківське відношення в таблиці сторінок для реалізації дерева контенту

Тут *code* – код сторінки, ключове поле. *parentCode* – код батьківської сторінки, зовнішній ключ, що посилається на поле *code* цієї ж таблиці.

Зауваження

У випадку використання СКБД MySQL та типу таблиць MyISAM *зовнішній ключ* передбачається на логічному рівні застосунку, так як на

рівні БД як відомо даний тип таблиць не підтримує *foreign key*. Натомість таблиці InnoDB *foreign key* підтримують.

Подібний зв'язок дозволяє представити структуру сторінок у вигляді *дерева*, де у кожного елемента може бути лише один батьківський (рис.7).

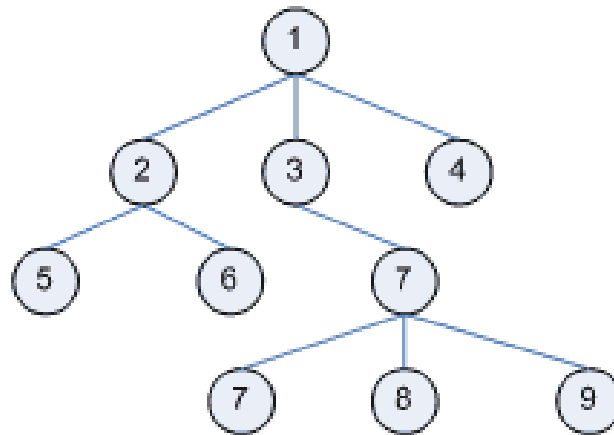


Рис. 7. Дерево контенту

Організація ієрархічного меню сайту

Навігаційна схема та схема меню сайту можуть значно відрізнятись від проекту до проекту. Натомість одним із найбільш зручних способів забезпечення навігації у великому інформаційному просторі веб-ресурсу є *ієрархічне меню*, що розкривається послідовно із *відвідуванням* наступного рівня вкладеності сторінок (рис.8).

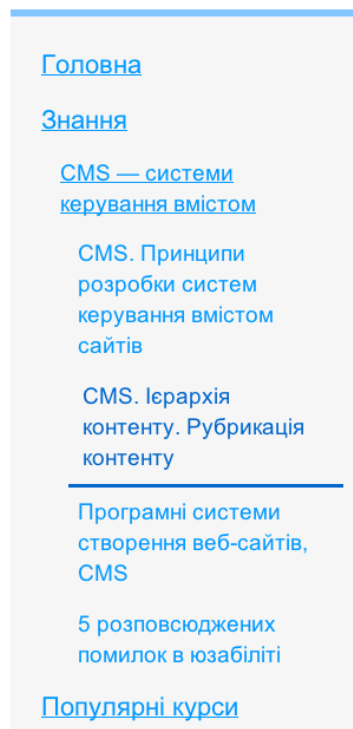


Рис. 8. Приклад ієрархічного меню сайту

Зауваження

Не слід плутати ієрархічне меню сайту, що відкривається після відвідування сторінки, із багаторівневим меню, що відкривається *при наведенні вказівника миші*. Другий із згаданих методів побудови меню є невдалим рішенням з точки зору зручності використання, так як вимагає від користувачів часом складних і незручних маніпуляцій для відкриття вкладених пунктів. Крім того, подібний підхід взято із практики побудови настільних застосунків, і він є неприродним для веб-навігації. У випадку веб-навігації користувач переміщується у інформаційному просторі, тоді як в середовищі настільного застосунку користувач викликає певні функції із меню. Тому послідовне відкриття розділів і підрозділів в меню *із завантаженням і відображенням сторінок* дає можливість користувачу відчувати напрямок руху в інформаційному середовищі, і тому вбачається більш доцільним для організації навігації на веб-сайті.

Алгоритм побудови ієрархічного меню сайту

Ієрархічне меню, про яке іде мова, працює відповідно до простого правила: *відкрито усіх пращурів поточного елемента, а також і сам поточний елемент*.

Поточний елемент меню – це такий елемент, який відповідає *поточній сторінці* сайту. Відкритий елемент меню – це такий елемент, для якого відображаються усі його дочірні елементи (звичайно один з дочірніх елементів також може бути відкритим).

Алгоритм програмної побудови такого меню:

1. Визначити *ланцюжок батьківських сторінок* для поточної сторінки – масив кодів сторінок-пращурів.
2. Рекурсивна функція *“Побудова меню”* з аргументом *“Сторінка”* (код сторінки):
 - 1) запит на всі *дочірні сторінки*;
 - 2) цикл по отриманим сторінкам:
 - якщо це поточна сторінка, вивести елемент меню як *поточний*;
 - інакше вивести звичайний елемент меню;
 - якщо сторінка міститься в *ланцюжку батьківських сторінок*, запустити функцію *“Побудова меню”* і передати дану сторінку у якості аргументу.
3. Запуск рекурсивної функції з параметром *“Сторінка-вершина дерева”* (*default*).

Кінець алгоритму.

Реалізація зсуву вправо. Для візуалізації ефекту вкладеності кожен наступний по глибині рівень повинен відображатися з деяким відступом праворуч (див. рис. 8). Реалізувати зсув можна різними способами. По-перше разом із сторінкою в рекурсивну функцію можна передавати рівень

вкладеності. Іншим способом є використання CSS, коли для блоку, що міститиме підпункти задається відступ.

Зауваження

Функція побудови меню повинна враховувати спосіб *сортування* того чи іншого розділу (чит. далі). У зв'язку з цим в п. 2.1 алгоритму в SQL-запиті слід задати відповідне сортування за допомогою клаузи *ORDER BY*.

Сторінки-контейнери

Сторінки-контейнери – сторінки, головним призначенням яких є подати список посилань (як правило список анотованих посилань) на *дочірні сторінки*. Власний контент таких сторінок, як правило, не має великого об'єму і є короткою анотацією до розділу або набору статей. Сторінки-контейнери займають друге місце після простих інформаційних сторінок по частоті застосування в веб-ресурсах (рис. 9).

Мова програмування PHP

• [Мережа Internet](#) • [PHP](#) • [Веб-програмування](#)

PHP (произносится пи-эйч-пи) — скриптовый язык программирования, созданный для генерации HTML-страниц на веб-сервере и работы с базами данных. Ныне поддерживается подавляющим большинством представителей хостингов. Входит в LAMP — «стандартный» набор для создания вебсайтов.

[Докладніше →](#)

Економічна теорія. Економіка

• [Економіка](#)

Вперше поняття "економіка" ввів грецький мислитель Арістотель (III ст. до н. е.). Описуючи організацію господарства в маєтку рабовласника, він фактично обґрунтував суть економіки як науки про домашнє господарство: грецьке "ойкос" означає дім, господарство, "номос" - вчення, закон

[Докладніше →](#)

Геоінформаційні системи (ГІС)

• [ГІС](#)

Геоінформаційна система — сучасна комп'ютерна технологія, що дозволяє поєднати модельне зображення території (електронне відображення карт, схем, космо-, аерозображень земної поверхні) з інформацією табличного типу (різноманітні статистичні дані, списки, економічні показники тощо).

[Докладніше →](#)

Рис. 9. Приклад сторінки-контейнера

До прикладів сторінок-контейнерів слід віднести такі розділи: категорії товарів, рубрики новин, тематичні розділи в блогах, сторінки тегів (міток) тощо.

Типи сторінок-контейнерів

Сторінки-контейнери можуть представляти дочірні сторінки у різних формах, основними з яких є наступні:

- 1) список анотованих посилань (рис.9);
- 2) плитка (рис. 10);

Керування формою сторінки-контейнера можна реалізовувати як на рівні адміністрування сайту, так і на рівні інтерфейсу відвідувача, в залежності від специфіки веб-проекта. З точки зору дизайну і зручності користування сайтом CMS система повинна передбачати керування основною формою подання даної конкретної сторінки-контейнера на рівні редактора сайту.

На рівні структури БД опис контейнерів реалізовується шляхом додавання спеціального поля в таблицю сторінок, наприклад *isContainer* та/або *containerType*. *isContainer* – прапор, що вказує, чи є контейнером сторінка, *containerType* – вказує тип відображення контейнера (список, плитка тощо).

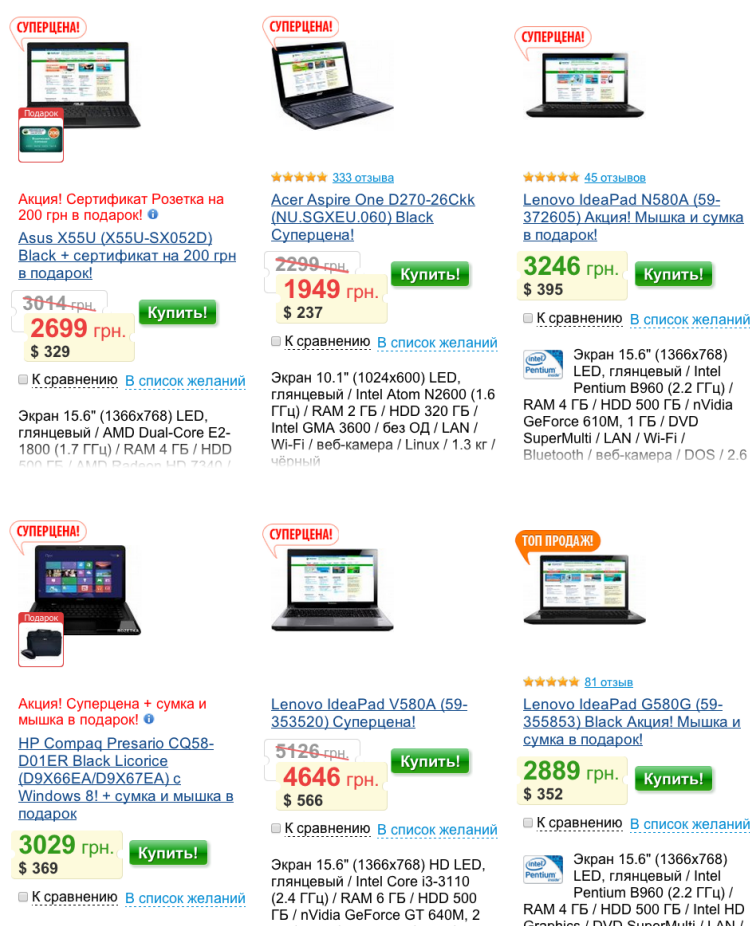


Рис. 10. Приклад сторінки-контейнера із відображенням дочірніх сторінок у вигляді плитки

Керування сортуванням дочірніх сторінок

Важливим елементом в реалізації контейнерів є порядок сортування дочірніх сторінок. Керуючи способом сортування, можна створювати розділи новин, категорії товарів, добірки статей тощо. Таким чином, важливим фактором в створенні ефективної схеми роботи з ієрархією контенту та контейнерами в CMS є можливість змінювати порядок сортування для різних розділів на сайті.

Для реалізації даної функціональної можливості на рівні БД слід додати спеціальне поле в таблицю сторінок, що вказуватиме на порядок сортування дочірніх сторінок, наприклад *sortBy*.

Основні типи сортування в контейнерах:

- по порядку визначеному редактором (за полем сортування);
- по даті (за спаданням, або за зростанням);
- по ціні (у випадку інтернет-магазину);
- за алфавітом по назві сторінки тощо.

Очевидно, для реалізації сортування у порядку визначеному редактором сторінка повинна отримати ще один атрибут – поле сортування, наприклад *place*. У випадку сортування сторінок по полю сортування запит на дочірні сторінки контейнера під час публікації міститиме клаузу *ORDER BY place*.

Таким чином, поле *sortBy* зберігатиме налаштування для керування вибіркою дочірніх сторінок з БД та порядком їх публікації в *контейнері* та в *меню*.

5. Ациклічний оргграф контенту та відношення псевдонімів в CMS

Тенденція примноження інформації ускладнює структурування інформаційних елементів у строгій деревоподібній формі. Крім цього строга ієрархія часто не задовольняє навігаційним вимогам веб-ресурсу. Потреба одночасної присутності одного і того ж елемента контенту в різних розділах сайту – досить розповсюджене і актуальне завдання. Прикладом цього може служити новина, яка по своєму змісту одночасно задовольняє різним розділам новин, товар, що підходить до різних категорій, деяка службова інформаційна сторінка сайту, яку доцільно розмістити в різних частинах сайту. Насправді з подібними ситуаціями ми часто зустрічаємося в побуті. В інтер'єрі – доступ до деяких речей організовується одночасно в різних частинах дому (телефон, календар, ручки та олівці), в магазинах – деякі типи товарів знаходяться одночасно в різних частинах залу і т.п.

Подібні об'єктивні неоднозначності вимагають удосконалення, розширення та розвитку моделі контенту CMS. В даному випадку мова іде про можливість одночасного розташування сторінки сайту в різних місцях ієрархії. Доцільною структурою для такого завдання є ациклічний

орієнтований граф (рис. 11). Ациклічний оргграф є узагальненням дерева. *Відмінністю від дерева* тут є те, що елемент структури тепер може мати більше, ніж одного батька.

Примітка

Серед іншого, ациклічний орієнтований граф використовується для структурування категорій Вікіпедії.

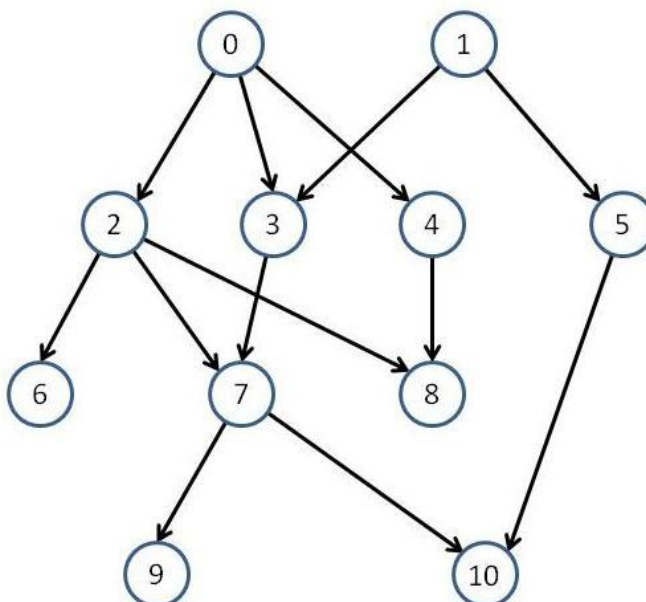


Рис. 11. Приклад ациклічного орієнтованого графу

Подібну структуру в реляційній БД можна реалізувати шляхом створення додаткової таблиці, що буде представляти матрицю суміжності графа (або список ребер). Основні поля таблиці представляють ребро графа – *код батьківської сторінки* та *код дочірньої сторінки*. Деякі CMS застосовують саме такий спосіб реляційного подання структури контенту. Натомість цей метод має ряд недоліків:

- 1) ієрархія як базова структура контенту не застосовується;
- 2) на рівні БД відбувається розмежування сутностей *сторінки* та *структури сторінок*, що ускладнює проект;
- 3) ускладнюються як інтерфейси, так і методи реалізації редагування контенту;
- 4) певною мірою ускладнюється програмна реалізація елементів навігації по сайту та збільшується навантаження на сервер БД.

Відношення псевдонімів

В структурі веб-сайту, як правило, абсолютна більшість ребер відповідають дереву, і лише частина ребер будуть описувати додаткові структурні відношення, характерні ациклічному оргграфу. Для подання такої структури зручно застосувати *відношення псевдонімів*, що дозволять зберегти базову деревоподібну структуру та нададуть можливість розміщати елементи

в різних місцях ієрархії. Для реалізації відношення не потрібно створювати нову сутність в БД у вигляді додаткової таблиці.

Сутність застосування псевдонімів полягає у встановленні відношень між двома елементами контенту, один з яких стає *псевдонімом* або логічним посиланням на елемент-джерело (рис. 12). Псевдонім дозволяє встановити для елемента-джерела додатковий батьківський зв'язок, і тим самим розмістити цей елемент в іншому місці ієрархії (з точки зору навігації по сайту).

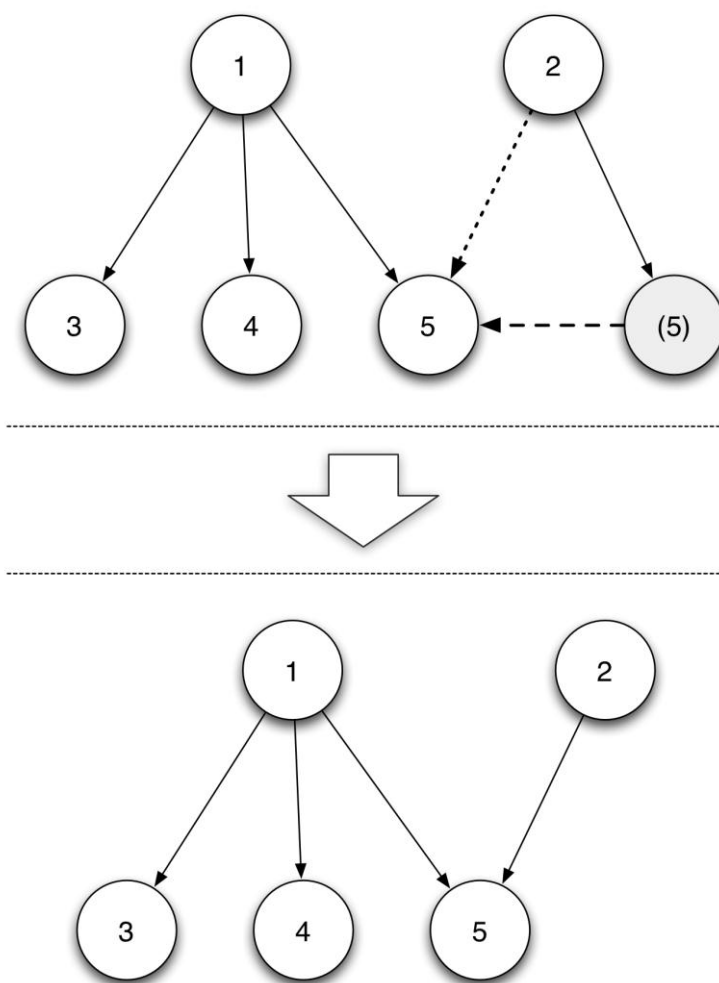


Рис. 12. Принцип роботи відношення псевдонімів

Таким чином, уся робота із контентом відбувається, як із деревоподібною структурою, а там, де зустрічаються псевдоніми відбувається автоматична підстановка даних із елементів-джерел.

Для реалізації псевдонімів потрібно здійснити наступні кроки:

- додати поле в *таблицю сторінок* (поле *aliasOf*); у випадку, якщо елемент є псевдонімом в дане поле буде записуватись код сторінки-джерела;
- на рівні *класу сторінки* на етапі ініціалізації передбачити перевірку: у випадку, якщо елемент є псевдонімом, автоматично

виконувати запит на отримання елемента-джерела із подальшою підстановкою даних із джерела в поля об'єкту сторінки.

Зауваження

Так як з точки зору *пошукової оптимізації* (SEO) наявність дублюючих сторінок на сайті є **негативним фактором**, логіку опрацювання псевдонімів слід будувати так, щоб URL псевдоніма безпосередньо не фігурував на сайті. Натомість на рівні ядра повинні відбутись автоматичні підстановки в URL адреси елемента-джерела. Це також стосується побудови меню та усіх інших елементів навігації. З точки зору ООП логіка роботи з псевдонімом повинна бути інкапсульована у класі сторінки. Таким чином, усі звернення до функцій та полів сторінки повинні відбуватись через даний клас, це ж стосується і процесу формування URL.

Зауваження

З урахуванням уникнення дублюючих сторінок та додаткових URL при кліку на елемент меню, який є псевдонімом, користувач потраплятиме в **інший розділ меню** сайту. Зважаючи на призначення псевдонімів та вимоги SEO таку поведінку системи слід вважати задовільною. Додаткові налаштування у вигляді збереження в cookie історії навігації слід використовувати обережно, пам'ятаючи про *ідемпотентність* GET-запитів, які використовуються як URL сторінки.

Таким чином, псевдонім не містить власної інформації, а використовує інформацію сторінки-джерела. Натомість, доцільно надати можливість зміни деякої інформації в псевдонімі. Наприклад, сторінка-джерело є товаром та містить таку назву “Кавоварка Philips 7X-234”. Сторінка-псевдонім у розділі новинок може містити такий заголовок для даного товару: “З’явилася довгоочікувана кавоварка від Philips!”, – усі інші поля будуть повторювати сторінку-оригінал.

Для реалізації цієї логіки в розділі ініціалізації слід виконувати перевірку для полів псевдоніма – якщо поле пусте, використати дані сторінки-джерела, в іншому випадку – залишити інформацію незмінною.

Псевдоніми та розділ новин

Досвід підтверджує ефективність застосування псевдонімів для побудови навігаційних схем сайту. Окрім власне моделювання ациклічного орграфу контенту псевдоніми дозволяють також застосовувати корисні схеми публікації контенту.

За допомогою псевдонімів можна з легкістю реалізувати *розділ новин*. На багатьох сайтах метою розділу новин є презентація нових матеріалів. Таким чином, контент-менеджер повинен створити як новий елемент контенту, так і відповідну йому новину. Така новина часто дублює частину контенту нової сторінки і врешті решт містить посилання на цю сторінку в

тексті. З точки зору відвідувача сторінка-новина не представляє жодної цінності і є лише додатковим навігаційним кроком до нового матеріалу на сайті. Подібний підхід робить розділ новин незручним як для контент-менеджера, так і для відвідувачів сайту.

У якості розділу новин доцільно представити колекцію псевдонімів, що представляють нові матеріали сайту. Самі матеріали можуть знаходитись при цьому в різних розділах. Додавання нового матеріалу для контент-менеджера не буде обтяжуватись створенням додаткової сторінки, а передбачатиме лише додавання псевдоніма в розділ новин. При публікації контейнера новин відвідувачі отримають ановані посилання нових матеріалів сайту, а при переході – відразу попадуть на цільові сторінки.

6. Мова макрокоманд контенту в CMS

Робота з контентом часто передбачає багаторазове виконання однотипних задач оформлення інформації. Серед таких задач можуть бути наступні:

- вставка ілюстрацій, що оформлюються певним однаковим стилем на всьому сайті;
- оформлення внутрішніх посилань на сторінки сайту, які не повинні залежати від технічних особливостей формування URL;
- створення блоку анованого посилання на іншу сторінку сайту у якості анонсу сторінки;
- оформлення певних однотипних логічних фрагментів контенту, що можуть повторюватись на різних сторінках сайту.

Виконання подібних задач вручну призводить до необхідності повторного формування html-коду та нагромодження в контенті сайту технічно-залежного коду (URL, посилання на фізичні каталоги зображень тощо).

Зручним розв'язком описаної проблеми є створення системи *макрокоманд контенту*, що дозволять зосередити логіку оформлення невеликих повторюваних фрагментів контенту в одному місці.

Приклад гіпотетичної макрокоманди для вставки зображення-ілюстрації:

```
<p> Lorem ipsum dolor sit amet. ~~img(some.jpg,left)~~ ut  
enim ad minim veniam. </p>
```

Інший варіант синтаксису:

```
<p> Lorem ipsum dolor sit amet. [[img some.jpg left]] ut  
enim ad minim veniam. </p>
```

Тут *img* – ім'я команди; *some.jpg, left* – аргументи команди; *~~* та *[[]]* – маркери початку та кінця команди.

Результат роботи команди:

```
<p> Lorem ipsum dolor sit amet.  
  
Ut enim ad minim veniam. </p>
```

Зауважимо, що даний приклад можна реалізувати засобами CSS, натомість часто засобів CSS не достатньо для зручного вирішення завдань повторної публікації невеликих фрагментів HTML.

Реалізація парсингу макрокоманд в CMS

Макрокоманди контенту в CMS – це спеціальна мова логічного керування контентом, оператори якої вбудовуються безпосередньо в html-текст веб-сторінки. Такі команди можуть використовуватись на рівні контент-менеджера або архітектора веб-сайту.

Алгоритм парсингу html-тексту сторінки для обробки макрокоманд:

1. Цикл поки в *Тексті* зустрічається *маркер початку* команди:
 - 1) отримати позицію першого входження *маркера початку* команди;
 - 2) отримати позицію *маркера кінця* команди;
 - 3) видобути текст до команди (*Текст1*) та текст після команди (*Текст2*) – рис. 13;
 - 4) видобути текст команди без маркерів;
 - 5) видобути *назву команди*;
 - 6) видобути *аргументи команди*;
 - 7) запустити обробку команди:
 - перевірити існування *функції-обробника* по імені команди;
якщо обробник відсутній, просто видалити текст команди з контенту;
 - запустити *функцію-обробник* і передати в неї *параметри* (масив параметрів);
функція повертає *результат* у вигляді рядка;
 - вставити на місце команди отриманий результат, рис. 13:
Текст = Текст1 + Результат + Текст2;

Кінець.

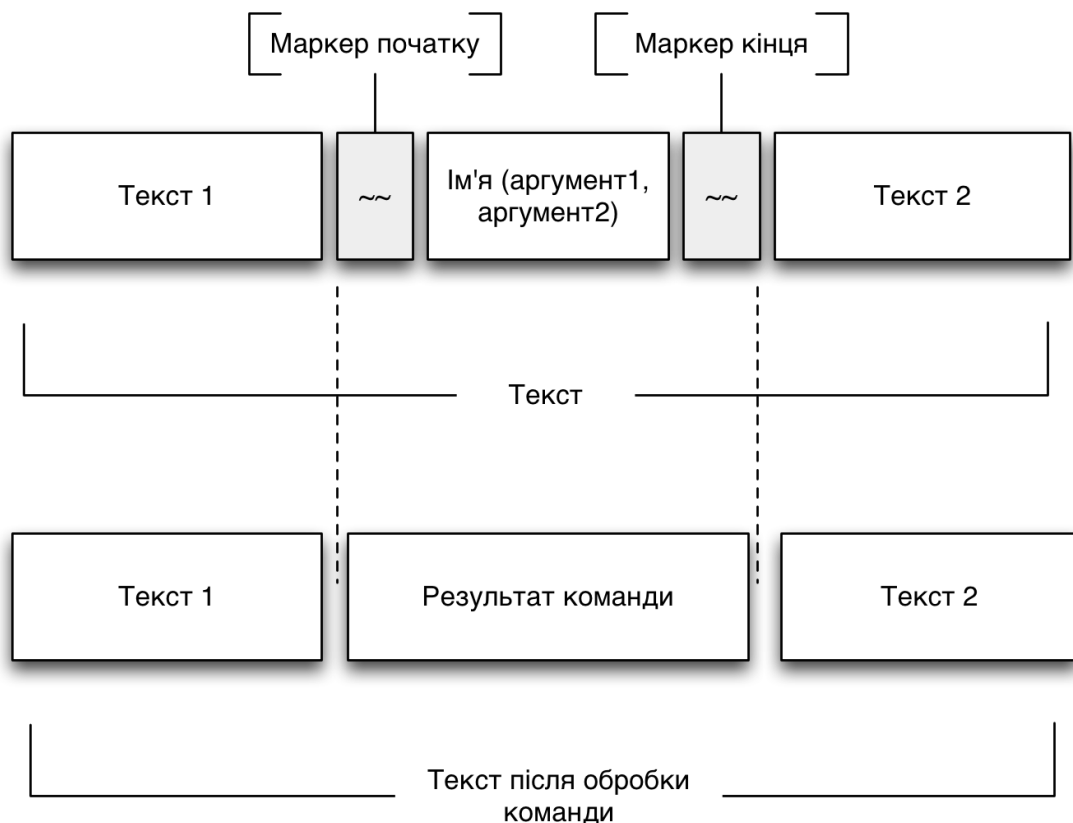


Рис. 13. Обробка макрокоманди

Зауваження

В PHP пошук маркерів команд слід виконувати за допомогою функції `strpos()` – пп.1, 1.1, 1.2 алгоритму.

Зауваження

Заміна тексту команди в контенті відбувається шляхом розбиття тексту на три частини: текст до команди, текст після команди та власне текст команди. Текст команди обробляється, та отриманий результат конкатенується з першою і третьою частинами тексту. В PHP для видобування частини рядку використовується функція `substr()` – п.1.3. Альтернативним способом парсингу макрокоманд є *регулярні вирази*.

Реалізація обробки макрокоманд в CMS

Після синтаксичного аналізу, коли отримано усю інформацію про команду, тобто її ім'я та аргументи, необхідно здійснити обробку команди. Для того, щоб забезпечити зручний механізм додавання нових команд в CMS визначення та запуск *функції-обробника* можна організувати автоматично по імені команди. Наприклад, для команди `img` слід здійснювати запуск функції `macro_img` і т.п.:

```
/**
 * $name – ім'я команди;
```

```

* $args – аргументи команди, масив
*/
$prefix = 'macro_';
$func_name = $prefix . $name;
if (function_exists($func_name))
    $result = $func_name($args);
...

```

Інший варіант обробки макрокоманд – автоматичний запуск *шаблону* із іменем, що будується на основі імені команди:

```

/**
* $name – ім'я команди;
* $args – аргументи команди, масив
* output() – функція підключення шаблону
*/
$folder = 'macros/';
$tpl_name = $folder . $name;
$result = output($tpl_name, $args);
...

```

Приклади макрокоманд в CMS

Макрокоманди допомагають створювати невеликі шаблони для вставки в текст веб-сторінок – так звані мікрошаблони. Серед типових завдань, що доцільно вирішувати за допомогою макрокоманд, наступні:

- внутрішнє посилання – команда, що у якості параметра приймає код сторінки:

```
~~link(about)~~Посилання на опис компанії~~endl~~
```

- блок анотованого посилання на внутрішню сторінку (анонс), параметр – код сторінки:

```
<h2>Оновлення сайту</h2>
~~announce (newpage)~~
```

- вставка зображення в потрібному стилі сайту, вхідний параметр – ім'я файлу зображення:

```
~~img(picct.jpg)~~
```

- Публікація дочірніх елементів деякої сторінки у вигляді списку чи плитки, параметр – код сторінки:

```
<h2>Сторінки розділу</h2>
~~list(products)~~
```

```
<h2>Сторінки розділу</h2>
~~tiles(products)~~
```

- Публікація сторінок із розділу новин у вигляді стрічки новин, параметр – код розділу новин; публікуються лише ті сторінки, що відповідають даті; застосовується сортування по даті; результат – список анотованих посилань на останні новини; цю макрокоманду зручно використовувати для головної сторінки сайту:

```
<h2>новини сайту</h2>  
~~lenta(news)~~
```

7. Використання універсального підходу в побудові CMS для інтернет-магазину

Щоб застосувати CMS з деревоподібною структурою контенту на базі єдиної таблиці БД для побудови інтернет-магазину достатньо додати до набору полів сторінки поле *Ціна (price)*.

Таким чином кожна сторінка, для якої ціна більша або рівна нулю, опрацьовується як товар. Для такої сторінки надаються засоби додавання до корзини, швидкого замовлення, публікується ціна. Для інформаційних сторінок значенням даного поля може бути -1. Такий підхід дозволяє організувати інтерфейс придбання в будь-якому місці сайту, у тому числі на сторінках акційних пропозицій та знижок.

8. Фільтрація сторінок за специфічними параметрами в CMS

Фільтри по характеристикам товарів є ефективним засобом навігації та пошуку необхідних товарів в інтернет-магазинах (рис.14). Натомість, реалізація такої функціональності вимагає цілого комплексу дій для моделювання даних та побудови алгоритмів пошуку та відображення інформації.

Беговые дорожки

Текущий фильтр ×

Назначение
 Профессиональная ×

Тип
 Электрическая ×

Конструкция
 Складные ×

[Убрать фильтр](#)

Фильтры


Назначение
 Профессиональная ×
 Полупрофессиональная
 Для дома

Тип
 Электрическая ×
 Механическая

Регулировка угла наклона
 Электрическая [5]
 Механическая


Конструкция
 Складные ×
 Нескладные

Максимальная скорость, км/ч
 12 и менее
 13-16
 17-20 [4]
 более 20 [11]




Беговая дорожка Circle M7200 (профессиональная)
Показания дисплея: Время, дистанция, пульс, калории, скорость, угол наклона полотна. Программы: 6 тренировочных, 2 пользовательские

34240 грн. [Купить](#)




Беговая дорожка Circle M7200 Luxury (профессиональная)
Программы: 6 тренировочных, 2 пользовательские. Сенсорные датчики измерения пульса + телеметрию "Быстрый доступ" изменения скорости: 5 клавиш

37440 грн. [Купить](#)




Беговая дорожка Johnson T7000 Pro профессиональная
Беговая дорожка Johnson T7000 Pro профессиональная. Продолжительная мощность двигателя: 3,2 л.с. (пиковая мощность 5,5 л.с.). Скорость: 0,8-20км/ч. Электроподъём бегового полотна: 0-12%

39120 грн. [Купить](#)



Беговая дорожка Matrix T50x-U профессиональная



Беговая дорожка Matrix T3x профессиональная

Рис. 14. Пример сторінки інтернет-магазину з фільтрами

Реалізація фільтрації

Клас *Фільтри* відповідає за обробку http-параметрів, підготовку структур даних для відображення панелі фільтрів, генерацію SQL-запиту (або його частини) для отримання сторінок, що задовольняють умовам фільтру.

Об'єкт сторінки-контейнера за необхідності ініціалізує та агрегує в собі об'єкт, що керує фільтрами даної сторінки-контейнера.

Реляційна модель даних для реалізації фільтрів

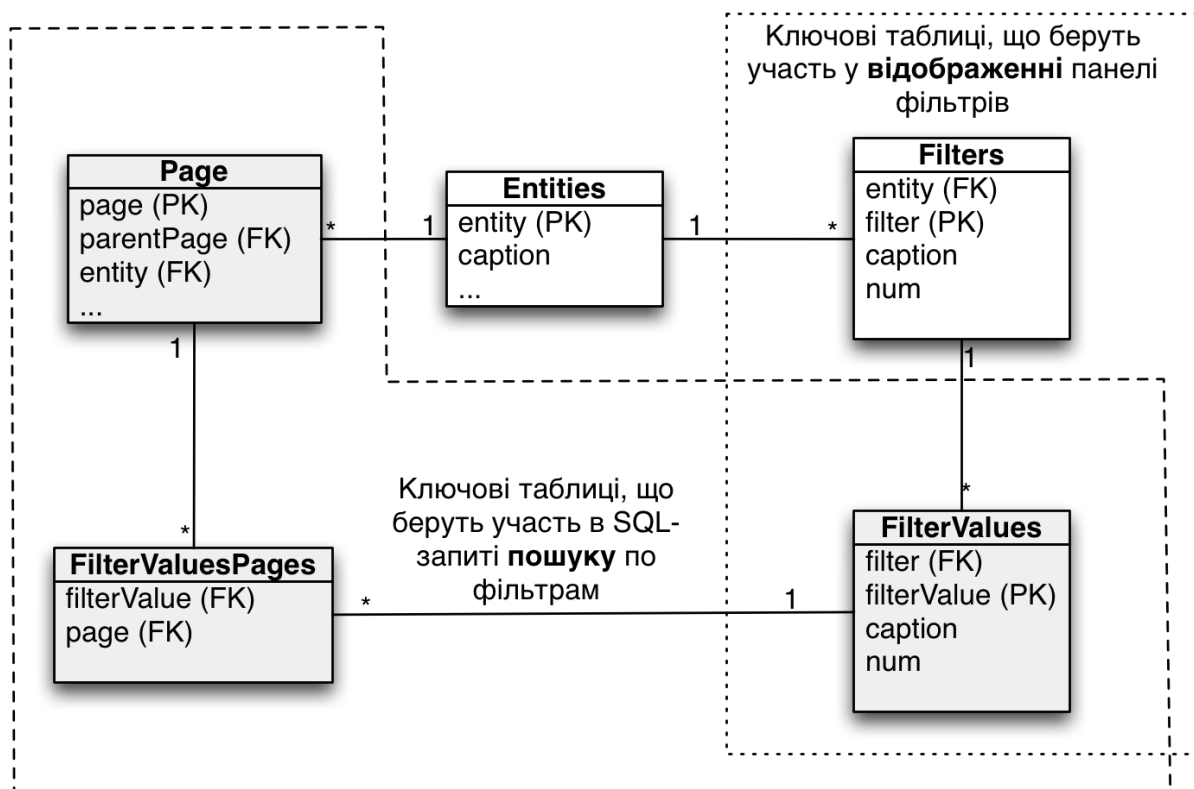


Рис. 15. Реляційна модель фільтрів

Формування та обробка http-параметрів та URL

Приклад:

http-запит: <http://www.site.com/?view=racecourses&f=26,41,46>

Коди фільтрів: 26, 41, 46.

В класі *Фільтрів* реалізувати функції *parseHttp()* та *makeHttp()*, що відповідно перетворюють параметри GET-запиту в масив значень фільтрів та навпаки.

Генерація SQL-запиту

Приклад запиту:

http-запит: <http://www.site.com/?view=racecourses&f=26,41,46>

Результуючий SQL-запит:

```
select pages.* from pages
```

```

inner join filterValuePages fv1 using (page)
inner join filterValuePages fv2 using (page)
inner join filterValuePages fv3 using (page)
where fv1.filterValue = '26'
and fv2.filterValue = '41'
and fv3.filterValue = '46'
and parentPage = 'racecourses'

order by price

```

Для реалізації таких запитів підготувати функцію класу *Фільтри*, що генерує клаузу *JOIN* – *getJoin()*, вхідний параметр – масив значень фільтра.

Цю функцію слід задіювати при генерації запита на дочірні елементи **контейнера-категорії товарів** в класі *Сторінка*.

Особливості застосування логічних операцій «та/або» в SQL-запиті фільтра

У випадку, коли користувач обирає значення фільтрів з різних секцій, такі фільтри формують набір умов “та”. Коли обрано значення фільтрів з однієї групи, такі значення об’єднуються за допомогою “або”.

Загальне правило: в середині групи “або”, за межами групи – “та”.

На рівні SQL-запиту логіку “або” можна реалізувати за допомогою оператора *IN (...)*. Наприклад користувач обрав значення фільтрів з такими ідентифікаторами: 26, 41, 42, при цьому 41 та 42 належать одній групі фільтру. SQL-запит буде наступним:

```

select pages.* from pages
inner join filterValuePages fv1 using (page)
inner join filterValuePages fv2 using (page)
inner join filterValuePages fv3 using (page)
where fv1.filterValue = '26'
and fv2.filterValue in ('41', '41')
and parentPage = 'racecourses'

order by price

```

Формування панелі вибору фільтрів

1. По налаштуванням сторінки-контейнера вибрати з БД усі фільтри та їх значення для даної категорії.
2. Відмітити поточні значення фільтрів.
3. Підрахувати доступну кількість сторінок для кожного значення фільтра:
 - додати до поточного набору фільтрів дане значення фільтра;
 - виконати запит на кількість сторінок `count(*)` для такої комбінації фільтрів, використовуючи для генерації запита функцію `getJoin()`;
4. Вивести посилання, сформоване за допомогою функції `makeHttp()` для даного значення фільтра, що буде враховувати вже відмічені фільтри.

9. Конструювання нових типів контенту в CMS

Незважаючи на велике розмаїття програмних систем, що застосовуються для керування контентом, актуальним завданням залишається розробка та вдосконалення засобів універсального керування інформаційними об'єктами Web-ресурсів, що дозволяє спростити створення нових ресурсів різного призначення, а також забезпечує ефективні механізми їх супроводження та налаштування. Зокрема актуальним є розв'язання наступних задач: створення спеціалізованих інформаційних об'єктів із користувацьким набором полів, керування відображенням таких об'єктів, сортування, фільтрація та навігація по сховищу створених об'єктів, інтеграція сховища інформаційних об'єктів із загальним сховищем контенту, керування ієрархією об'єктів, класифікація та групування об'єктів тощо.

Серед систем з відкритим кодом Drupal надає найширші можливості по виконанню вказаних задач. Недоліком системи є така організація структури даних та програмного коду, яка зумовлює велике навантаження на сервер баз даних. Це в свою чергу сповільнює швидкість генерації Web-сторінок ресурсу та вимагає великого ступеня кешування, що не завжди доцільно. Крім того в даній системі ускладнена робота по налаштуванню відображення інформаційних об'єктів, а також робота по загальному адмініструванню сайту.

Загалом проблема подання спеціалізованих інформаційних об'єктів в контексті CMS є дотичною до напрямку об'єктно-реляційного відображення ORM (Object-relational mapping), об'єктно-реляційних баз даних та інших напрямів, пов'язаних із моделюванням даних та знань.

Створення спеціалізованих інформаційних об'єктів із користувацьким набором полів в CMS є затребуваним завданням при побудові складних інформаційних веб-ресурсів.

Загальні елементи контенту

Ключовою інформаційною сутністю систем керування вмістом сайтів (CMS) є сторінка або елемент контенту. Як правило, в базі даних CMS-системи існує спеціальна таблиця, що представляє дану сутність, характеризуючи її певним набором полів. Основними полями загального елемента контенту є заголовок, html-текст, анотація та ін. В залежності від системи по-різному можуть бути реалізовані ієрархічні та структурні зв'язки між елементами контенту. В Drupal загальний елемент контенту позначається терміном Node (вузол), у Wordpress – Post (допис), в Joomla – матеріал. Доцільним вважається таке проектування системи, де сутністю, що відповідає за сторінки є множина елементів контенту $V=\{v_i\}$.

Інтеграція і розмежування спеціалізованих інформаційних об'єктів із загальними елементами контенту

Спеціалізовані інформаційні об'єкти веб-ресурсу можуть реалізовуватись двома шляхами: (1) як доповнення до набору полів загального елемента контенту – приєднання до контенту; (2) як окрема незалежна сутність, не пов'язана напряду з елементами контенту загального типу – розмежування контенту та спеціалізованих об'єктів (рис.16).

Приєднання до загального контенту вигідно використовувати у випадках, коли сутність, яку потрібно представити, є незначним розширенням сторінки загального типу і повинна грати роль деякого інформаційного елемента контенту, аналогічного сторінкам сайту. В Drupal це може бути реалізовано за допомогою функціональності Content types, що стала стандартною у версії 7 даної CMS. Wordpress реалізує згадані функції за допомогою додатка Custom Post Type. Для Joomla існує цілий набір розширень, що реалізують дану функціональність, найбільш відомим з яких є K2. K2 додає більше десятка таблиць в базу даних і фактично підмінює систему створення контенту в Joomla.

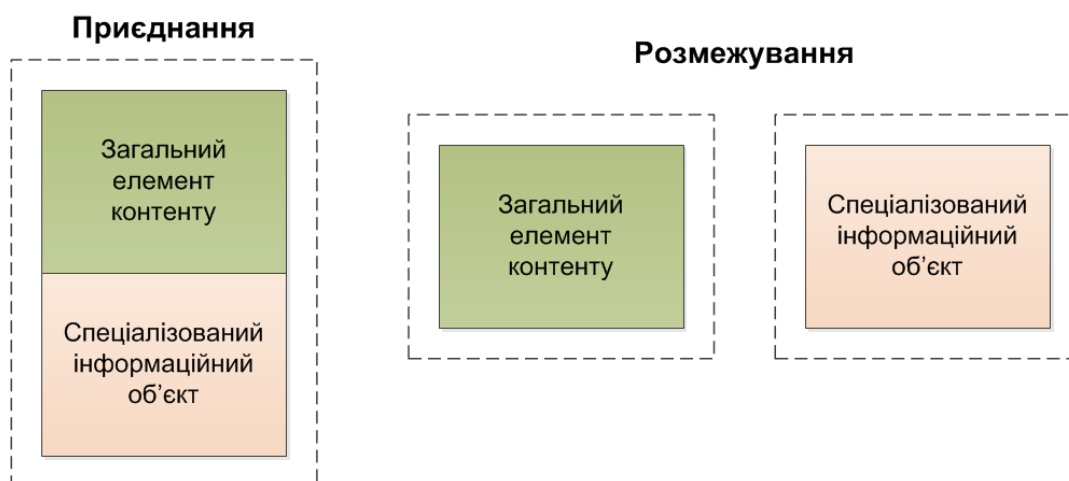


Рис. 16. Способи реалізації спеціалізованого інформаційного об'єкта у відношенні до загальних елементів контенту.

Розмежування загального контенту та спеціалізованих об'єктів в CMS передбачає наявність незалежних сутностей в БД системи, що відповідають за подання таких об'єктів. При цьому редагування даних об'єктів відбувається за допомогою окремого інтерфейсу, а набір полів не обтяжується стандартним набором полів загального елемента контенту. Спосіб відображення даних об'єктів на сайті повинен налаштовуватись за допомогою спеціальних засобів. Таким чином відбувається керування виглядом таких об'єктів, налаштування способів навігації по їх сховищу та інші спеціалізовані функціональні можливості.

Розмежування загального контенту та спеціалізованих об'єктів у згаданих CMS можна порівняти із розширенням стандартного набору полів для такої сутності CMS як елемент таксономії. Таксономія в математиці – це деревоподібна структура класифікацій деякого набору об'єктів. Таксономія в CMS використовується для різних типів класифікацій елементів контенту: для створення ієрархії розділів сайту, для організації міток (тегів), для організації меню та навігації, для тематичної класифікації контенту.

Drupal та Wordpress містять засоби створення таксономії (Wordpress використовує термін «категорії»). У цих системах на основі згаданих вище модулів для елементів таксономії можуть створюватись додаткові поля, у такому випадку окремі розділи таксономії можуть розглядатися як набори спеціалізованих інформаційних об'єктів.

Порівняно із стандартними елементами контенту, елементи таксономії не будуть містити значного переліку стандартних полів, тому такий метод опису спеціалізованих об'єктів можна розглядати як варіант розмежування контенту та відповідних об'єктів. Натомість слід зазначити, що такий спосіб не є строгим розмежуванням, і маніпуляції з новоствореними об'єктами будуть обтяжуватись стандартним шаром обробки, характерним для кожного елемента таксономії.

Реляційна реалізація спеціалізованих інформаційних об'єктів CMS

Робота по керуванню спеціалізованими інформаційними об'єктами в контексті CMS передбачає опис нового типу контенту як нової сутності в інформаційній системі, що повинно включати:

- *загальний опис* створюваної сутності, що включає такі характеристики як ім'я для програмного опрацювання та підпис для відображення користувачам;
- *опис набору полів*, які повинні характеризувати відповідні інформаційні об'єкти, що включає власне перелік полів, а також опис кожного окремого поля із зазначенням його імені, підпису, типу даних та ін. інформації;
- *створення інфраструктури для збереження* екземплярів новостворюваної сутності, тобто безпосередніх об'єктів із значеннями їх полів;
- *налаштування* способу відображення об'єктів на сайті, їх адміністрування, а також будь-яких інших засобів роботи з ними.

Таким чином спеціалізовані об'єкти в БД CMS-системи декомпонуються на такі сутності, які повинні певним чином обов'язково відображатися в системі:

- «тип об'єкту»;
- «поля об'єкту»;
- «об'єкти» (або екземпляри);
- «значення полів» об'єктів.

У випадку, коли для створення об'єкта застосовується спосіб *приєднання до контенту*, роль «об'єкту» виконуватиме сторінка загального типу, до якої буде приєднано спеціалізовані «значення полів» відповідно до «типу об'єкту». У випадку розмежування спеціалізованих об'єктів та контенту, «об'єкт» буде представлено окремою сутністю в базі даних.

Реляційна логіка та можливості сучасних SQL-систем допускають велике розмаїття способів реалізації спеціалізованих інформаційних об'єктів в контексті бази даних CMS. Основні способи подані на рис. 17 у вигляді прототипів інформаційно-логічних моделей реляційної бази даних CMS-системи. Деякі моделі вимагають застосування інструкцій DDL (створення таблиць) на етапі додавання нових типів об'єктів. Розглянемо кожну із поданих моделей.

1. *Готова об'єктно-реляційна надбудова* (рис.17., модель 1). Без застосування DDL-інструкцій. Недоліком моделі є використання однієї таблиці для збереження значень різних спеціалізованих полів, що вимагає або використовувати різні колонки для збереження значень різних типів, або накладати обмеження на тип даних спеціалізованих полів. Іншим варіантом є застосування окремих таблиць для кожного з типів даних.

2. *Об'єктно-реляційна надбудова «Нове поле – нова таблиця»* (рис.17., модель 2). Потребує застосування DDL з метою створення окремих таблиць для зберігання «значень полів». Дана модель є природним розвитком попередньої – тут вирішується згадана проблема подання полів різних типів. Відбувається це за рахунок створення спеціалізованих таблиць для кожного з полів. Недоліком даної моделі є складність «збирання» об'єкта, так як об'єкт виявляється розподіленим між різними таблицями, і його композиція вимагає значних SQL-витрат.

3. *Природна реляційна модель «Новий тип об'єктів – нова таблиця»* (рис.17., модель 3). Потребує застосування DDL для створення таблиць, що представляють об'єкти стандартним для реляційних БД чином – кожен запис таблиці представляє об'єкт, кожне поле – значення поля об'єкта. Додаткова мета-інформація про тип об'єктів та налаштування полів можуть зберігатися у відповідних таблицях. Якщо застосовується приєднання об'єктів до контенту, тоді між таблицею сторінок та таблицею певного типу об'єктів існуватиме зв'язок «один-до-одного» (1–0..1). З іншого боку дана модель є найбільш природним рішенням у випадку розмежування контенту і інформаційних об'єктів. При цьому для реалізації спеціалізованої бізнес-логіки для маніпуляції об'єктами доцільним є наявність в CMS відповідного API для роботи із спеціалізованими об'єктами.

4. *Модель тегів* (рис.17., модель 4). Без використання DDL. Застосовується принцип тегування: сторінці, що є об'єктом, призначається набір міток, кожна з яких представляє деяку характеристику об'єкта, тобто значення деякого поля. Тут кожна мітка в таблиці тегів відповідає деякому одному значенню поля. Значення полів (тобто теги) можуть групуватися в словники, тоді словник представлятиме деяке поле, а теги – область його значень. Даний підхід має значні обмеження у виразності опису об'єктів, проте він є дуже зручним у випадку, коли спеціалізовані об'єкти використовуються для фільтрації та групування контенту за деякими ознаками. Кожне значення поля отримує свій власний ідентифікатор, тому формування URL та відповідні SQL-запити по відобуванню об'єктів виконуються технічно просто.

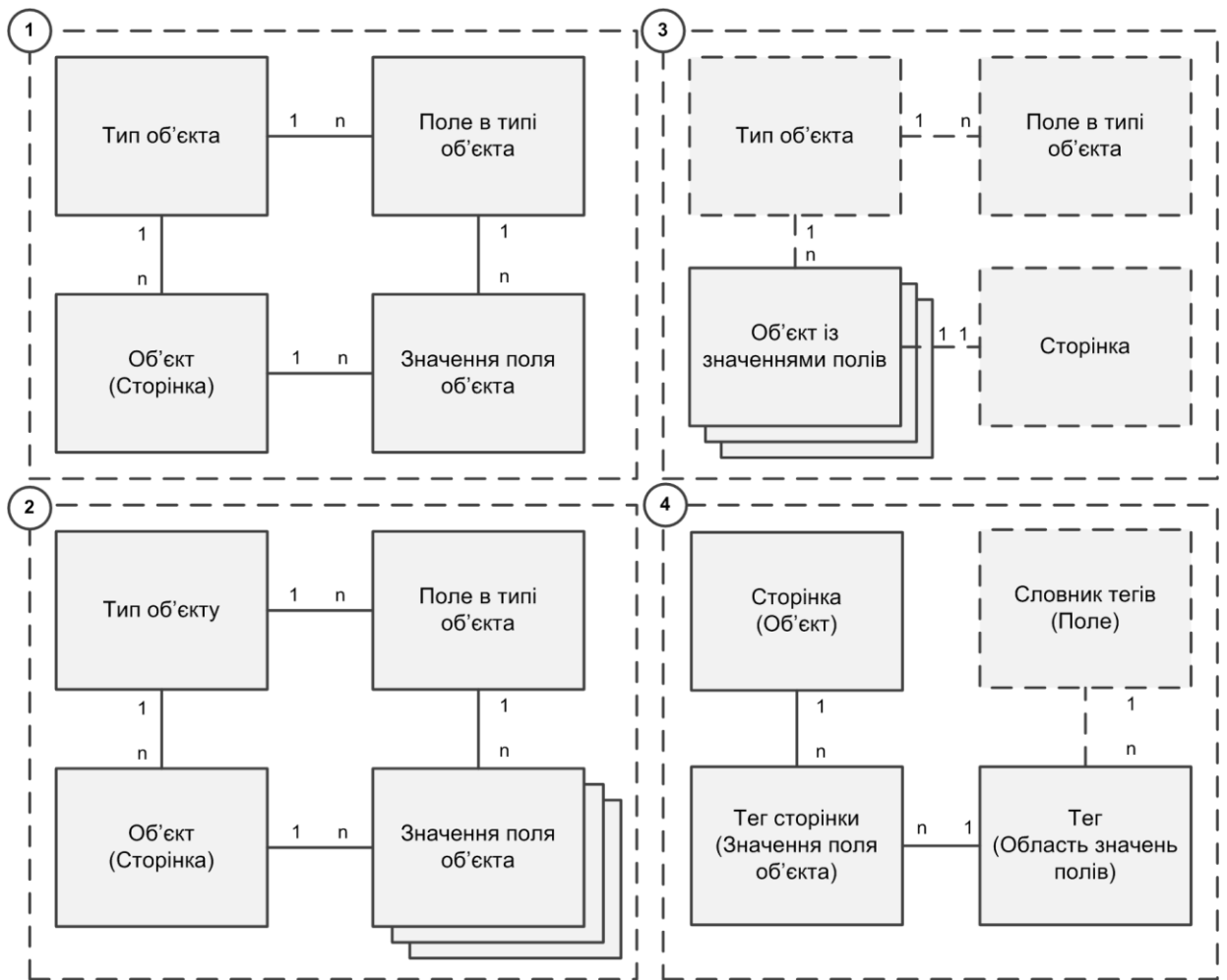


Рис.17. Способи реляційної реалізації спеціалізованих інформаційних об'єктів CMS

10. Адміністративна частина сайту та автоматизація побудови інтерфейсу редагування БД

Структура класів бібліотеки автоматизованої побудови інтерфейсу редагування БД

Клас поля – Field

Клас *Field* інкапсулює усю логіку роботи з полем таблиці, що включає:

- відображення поля в режимі перегляду;
- створення елемента керування для редагування поля в режимі редагування;
- отримання значення поля із http-параметрів запиту після редагування запису;
- подання значення поля у коректному форматі для *SQL*-запиту.

Ключові властивості класу:

- ім'я поля;
- тип поля;
- підпис поля;
- значення;
- значення по замовчуванню;
- автоінкрементне поле – прапор;
- налаштування видимості поля в різних режимах відображення таблиці;
- префікс – для іменування поля в html-формі;
- налаштування відображення – *CSS*-стилі, тощо.

Ключові методи класу:

- конструктор;
- публікація елемента керування для режиму редагування;
- отримати коротке значення – для режиму перегляду;
- отримати значення для *SQL*;
- отримати значення для елемента керування (поля *html*-форми);
- записати значення поля, отримане після редагування запису – видобувається із масиву *POST*.

Клас таблиці – **Table**

Клас таблиці інкапсулює логіку роботи з таблицею:

- публікація таблиці в режимі перегляду із навігаційними посиланнями для редагування записів;
- формування html-коду форми редагування обраного запису;
- завантаження результатів редагування у внутрішні об'єкти;
- генерація SQL-запитів для перегляду та редагування таблиці.

Ключові описові властивості класу таблиці:

- ім'я таблиці;
- масив об'єктів-полів таблиці;
- підпис;
- ім'я ключового поля;
- спосіб сортування – для перегляду таблиці.

Властивості класу для організації різних режимів відображення та обробки дій:

- ім'я скрипта, через який відбуватиметься уся робота з таблицею – для коректного формування URL;
- дія, яка виконується в даний момент:
 - перегляд;
 - публікація форми редагування;
 - публікація форми додавання запису;
 - виконання дії: редагування, додавання або видалення.

Ключові методи класу:

- конструктор;

- додавання нового поля в масив полів та інші, пов'язані з мета-описом та налаштуванням інтерфейсу, методи;
- головний метод публікації і обробки дій;
- публікація таблиці в режимі перегляду;
- публікація форми редагування/додавання запису;
- отримати SQL-запит *SELECT* для отримання даних для перегляду таблиці;
- перенесення даних із *HTTP*-параметрів відправленої форми до внутрішніх об'єктів – полів таблиці;
- генерація SQL-запиту для редагування запису – *UPDATE*;
- генерація SQL-запиту для додавання запису – *INSERT*;
- генерація SQL-запиту для видалення запису – *DELETE*;

Організація роботи застосунку з точки зору HTTP-параметрів

Ключовим параметром HTTP, який сповіщає про наміри користувача, є параметр *action* – у ньому передається на сервер дія, яку виконує користувач (рис.18). Якщо параметр дії не встановлено, застосовується дія по замовчуванню – перегляд таблиці.

Коли параметр *action* містить деяку дію, відмінну від *browse*, також передається ідентифікатор запису, для якого дану дію потрібно застосувати.

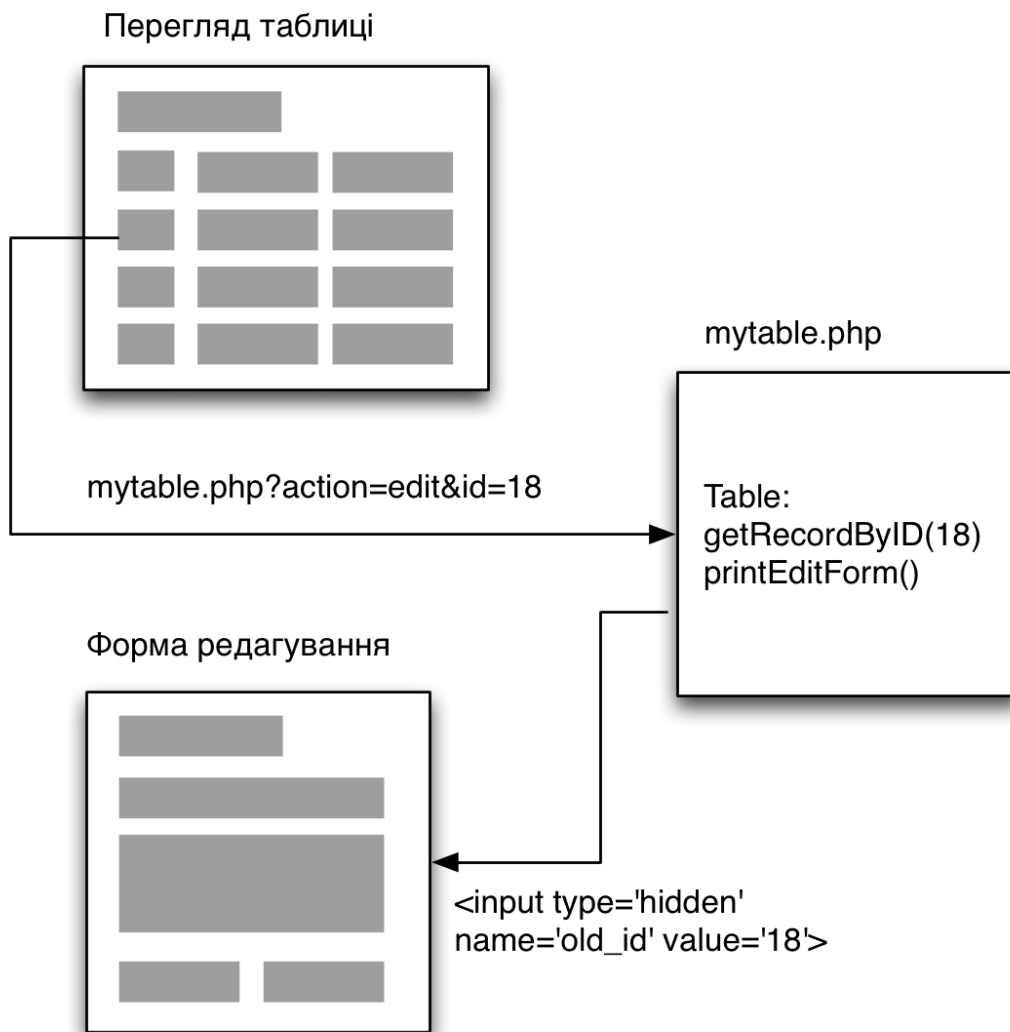


Рис.18. Схема роботи скрипта редагування таблиці БД

Особливості форми редагування запису

Для успішної реалізації *редагування запису* разом із новим значенням ключового поля потрібно також обов'язково передавати старе значення. Старе значення ключового поля попередньо публікується у формі редагування у *прихованому вигляді* (*hidden*). Таким чином, після надсилання форми на сервер об'єкт таблиці має можливість коректно побудувати SQL-запит на зміну запису, вказавши в секції *WHERE* старе значення ключового поля.

Підготовка скрипта та мета-опис таблиці

На етапі мета-опису таблиці необхідно здійснити наступні кроки:

1. Створити PHP-файл і включити в нього бібліотеку з класами таблиці і поля.
2. Створити об'єкт таблиці і задати ім'я, підпис та ключове поле.
3. Послідовно додати і описати поля таблиці (метод *addField*).

4. Задати необхідні налаштування, такі як спосіб сортування таблиці тощо.

Режим перегляду таблиці

Для відображення таблиці в режимі перегляду виконуються наступні дії:

1. Формується і виконується SQL-запит на отримання даних таблиці з урахуванням сортування;
2. Друкується описовий заголовок – підпис таблиці;
3. Початок виведення html-таблиці;
4. Цикл по *масиву полів* – виведення підписів колонок таблиці;
5. Цикл по *записам* таблиці:
 - Виведення першої комірки з *діями* – редагувати запис:

```
<a href='script.php?action=edit&id=<?=$id?>'>  
Редагувати запис</a>
```
 - Завантаження значень полів запису до об'єктів полів;
 - Цикл по масиву полів – виведення комірки із значенням даного поля;
6. Завершення виведення html-таблиці.

Обробка даних html-форми та генерація SQL-запитів

Після приходу даних форми на сервер у вигляді POST-запиту клас таблиці запускає необхідний метод.

Якщо, форма надіслана з метою *редагування* існуючого запису, метод-обробник редагування здійснює такі кроки:

- завантаження даних форми в об'єкти полів;
- ініціалізація змінної, що зберігає старе значення *ідентифікатора* запису, який редагується (дане значення разом з усіма полями також повинно надсилатися через форму – через GET-параметр або через приховане поле);
- генерація події *onBeforeUpdate* класу таблиці (опціонально); обробник події повинен мати можливість заборонити виконання *UPDATE* за необхідності;
- генерація SQL-запиту *UPDATE* з використанням мета-інформації з об'єктів-полів та із значеннями, що отримуються за допомогою геттерів `getSqlValue()` кожного із об'єктів-полів;
- виконання запиту; обробка помилки, якщо вона виникає;
- генерація події *onUpdate* класу таблиці (опціонально).

Якщо, форма надіслана з метою *додавання* запису, метод-обробник додавання здійснює такі кроки:

- завантаження даних форми в об'єкти полів;

- генерація події *onBeforeInsert* класу таблиці (опціонально) з можливістю зупинки подальшого виконання запиту;
- генерація SQL-запита *INSERT* з використанням мета-інформації з об'єктів-полів та із значеннями, що отримуються за допомогою геттерів *getSqlValue()* кожного із об'єктів-полів;
- виконання запиту; обробка помилки, якщо вона виникає;
- генерація події *onInsert* класу таблиці (опціонально).

Якщо, форма надіслана з метою **видалення** запису, метод-обробник видалення здійснює такі кроки:

- отримання значення ідентифікатора запису для видалення;
- генерація події *onBeforeDelete* класу таблиці (опціонально) з можливістю зупинки подальшого виконання запиту;
- генерація SQL-запита *DELETE* з використанням мета-інформації про ідентифікатор запису та його значення;
- виконання запиту; обробка помилки, якщо вона виникає;
- генерація події *onDelete* класу таблиці (опціонально).

Lookup-поля вибору як засіб реалізації зв'язку один-до-багатьох в інтерфейсі редагування

Lookup-поля та реляційна модель

Під *lookup-полем* або *полем підстановки* розуміють такий елемент введення даних в поле запису таблиці, який дозволяє обрати значення із списку, а сам список формується на основі даних іншої таблиці (рис.19).

products, Edit record [ProductID: 1]

[Back to list](#)

CategoryID	Beverages	
Discontinued	Please select	<input type="checkbox"/>
ProductName	Beverages	<input type="checkbox"/>
QuantityPerUnit	Condiments	<input type="checkbox"/>
ReorderLevel	Confections	<input type="checkbox"/>
SupplierID	Dairy Products	<input type="checkbox"/>
UnitPrice	Grains/Cereals	<input type="checkbox"/>
UnitsInStock	Meat/Poultry	<input type="checkbox"/>
	Produce	<input type="checkbox"/>
	Seafood	<input type="checkbox"/>
		<input type="checkbox"/>
	18	<input type="checkbox"/>
	39	<input type="checkbox"/>
		<input type="checkbox"/>

Рис. 19. Приклад lookup-поля.

Lookup-поле відповідає реляційному відношенню один-до-багатьох. Lookup-поле належить таблиці, що представляє сторону «багато» та фактично служить для редагування значень зовнішнього ключа (рис.20). Щоб надати користувачу зручний інтерфейс у вигляді випадаючого списку підписів та позбавити його необхідності оперувати із значеннями зовнішнього ключа створюється даний елемент інтерфейсу. У формуванні lookup-поля беруть участь три поля:

- 1) поле зовнішнього ключа таблиці, що редагується;
- 2) поле первинного ключа таблиці-джерела значень;
- 3) поле, що є підписом із таблиці-джерела значень.

Таблиця зі сторони «один» є джерелом значень для зовнішнього ключа таблиці «багато». Відповідно значення отримуються із колонки первинного ключа. Значення для відображення в інтерфейсі редагування отримуються із колонки підписів. Значення первинних ключів не відображаються в інтерфейсі користувача-редактора таблиці, натомість редактор оперує підписами, замість яких на рівні html-форми та SQL-запиту підставляються значення ключів.

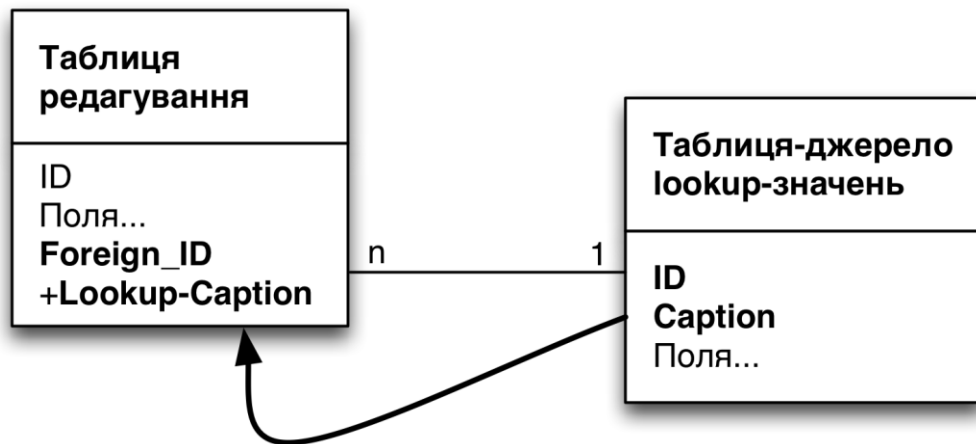


Рис. 20. Реляційна схема побудови lookup-поля

Реалізація lookup-поля в бібліотеці автоматизованої побудови інтерфейсу редагування БД

Для реалізації lookup-поля в бібліотеці достатньо внести необхідні доопрацювання в клас поля *Field*, який буде інкапсулювати в собі усю логіку роботи з даним елементом керування. На рівні класу таблиці робота з полем підстановки відбуватиметься прозоро та не потребуватиме додаткових змін у коді класу.

На рівні класу *Field* слід передбачити:

- 1) ідентифікацію даного поля як lookup-поля – певний булевий прапорець (*isLookup*);
- 2) мета-опис lookup-поля (у методі *setLookup*), що містить:
 - назву таблиці-джерела (*lookupTableName*);
 - назву поля-джерела значень (*lookupKeyName*);
 - назву поля-джерела підписів (*lookupCaptionName*);
- 3) підготовку елемента редагування lookup-поля для html-форми:
 - виконати SQL-запит на отримання даних для поля;
 - побудувати html-елемент для редагування (випадаючий список) за принципом ключ-значення;
- 4) для режиму перегляду *browse* передбачити виведення відповідного підпису для поля підстановки:
 - на етапі виведення звернутись до методу *getLookupCaption()*, який в свою чергу здійснить необхідний SQL-запит до таблиці-джерела.

На рівні обробки форм редагування та додавання запису для роботи з lookup не потрібно здійснювати додаткових доопрацювань, так як в *POST*-запит на етапі відправки форми попадатиме коректне значення ключа lookup-поля, що обумовлюється використанням html-елементів керування з принципом ключ-значення.

Автоматизована побудова інтерфейсу головний-детальний

Користувацький інтерфейс головний-детальний служить для редагування даних, що реалізують відношення один-до-багатьох (рис. 21). Даний інтерфейс є одним із найпоширеніших в застосунках баз даних.

Customers:

Company Name	Contact Name
Alfreds Futterkiste	Maria Anders
Ana Trujillo Emparedados y helados	Ana Trujillo
Antonio Moreno Taquería	Antonio Moreno
Around the Horn	Thomas Hardy
Berglunds snabbköp	Christina Berglund

Change page: < 1 2 3 4 5 6 7 8 9 10 ... > Displaying page 1 of 19, items 1 to 5 of 91.

Orders:

OrderID	OrderDate	ShipCountry
10308	18/09/1996	Mexico
10625	8/08/1997	Mexico
10759	28/11/1997	Mexico
10926	4/03/1998	Mexico

Change page: < 1 > Displaying page 1 of 1, items 1 to 4 of 4.
Ready

Рис. 21. Приклад інтерфейсу головний-детальний

Реалізація даної функціональності передбачає створення класу-нащадка *Table* – *SubTable*. Екземпляр *SubTable* відповідає за відображення та редагування таблиці «детальний». *SubTable* міститиме мета-інформацію про зв'язок з «головною» таблицею (ім'я поля *SubTable*, що є зовнішнім ключем на «головну»).

У клас таблиці *Table* необхідно внести зміни, що дозволять зберігати інформацію про наявність детальної таблиці (посилання на екземпляр *SubTable*).

При відображенні *Table* серед колонок також з'явиться стовпець, що містить посилання-перехід до «детальної», відповідно до поточного запису «головної» (рис.22).

У свою чергу стандартний інтерфейс *SubTable* буде доповнено навігаційними можливостями повернення до відповідної «головної» таблиці (рис.23).

← Сутності

Все записи + Новая запись

Видалити вибрані Увага! Усі дочірні елементи будуть також видалені!









	№	Код сутності	Підпис Ru	Підпис Ua	Номер для сортування
<input type="checkbox"/>  Значення фільтра	56	armchair	Избранные	Вибрані	0
<input type="checkbox"/>  Значення фільтра	55	armchair	Назначение	Призначення	0
<input type="checkbox"/>  Значення фільтра	52	armchair	Производитель	Виробник	1
<input type="checkbox"/>  Значення фільтра	58	armchair	Страна-производитель	Країна-виробник	2
<input type="checkbox"/>  Значення фільтра	53	armchair	Функции	Функції	2
<input type="checkbox"/>  Значення фільтра	54	armchair	Цвет	Колір	3
<input type="checkbox"/>  Значення фільтра	57	armchair	Музыка и подключение устройств	Музика і підключення пристроїв	4
<input type="checkbox"/>  Значення фільтра	59	armchair	Цена, грн.	Ціна, грн.	10

Рис. 22. Приклад таблиці «головний» із колонкою для посилань на «детальну». Тут, «Значення фільтра» - посилання на «детальну» таблицю

Фільтри : [Избранные \[56\]](#) → Значення фільтра

← Фільтри

Все записи + Новая запись

Видалити вибрані Увага! Усі дочірні елементи будуть також видалені!

	Код пошуку	Підпис Ru	Підпис Ua	Номер для сортування
<input type="checkbox"/>  Сторінки фільтра	192	Акции	Акції	0
<input type="checkbox"/>  Сторінки фільтра	191	Новинки	Новинки	1
<input type="checkbox"/>  Сторінки фільтра	190	Хиты продаж	Хіти продажів	5

Рис. 23. Приклад таблиці «детальний» із навігаційними елементами повернення до «ГОЛОВНОЇ»

11. Лабораторні роботи

Загальні вимоги до виконання лабораторних робіт

На початку курсу кожен студент повинен обрати та узгодити з викладачем індивідуальну тему розроблюваного веб-сайту. Подальші лабораторні роботи виконуються в контексті обраної теми індивідуально. Таким чином комплекс лабораторних робіт, що виконується протягом курсу, має на меті поступове створення складної системи керування сайтом в обраній тематиці. Сукупний результат лабораторних робіт подається у вигляді працюючого програмного забезпечення веб-сайту, що має унікальне оформлення та верстку, а також сукупність програмних компонентів, які розроблені студентом відповідно до особливостей індивідуального завдання.

Лабораторна робота №1

Розробка базової архітектури CMS

Мета – здобути вміння конструювати базову архітектуру системи керування контентом.

Завдання

Створити базову версію системи керування контентом, що міститиме клас ядра системи та клас сторінки. Реалізувати процес публікації сторінки. Виконати роботу відповідно до індивідуальної теми розроблюваного веб-сайту.

Лабораторна робота №2

Розробка бази даних системи керування контентом

Мета – здобути вміння створювати та використовувати базу даних для збереження контенту CMS.

Завдання

Створити базу даних для збереження сторінок CMS-системи. Створити таблицю, що відповідає за збереження сторінок. Доопрацювати CMS для роботи з БД. Реалізувати методи класу сторінки, що відповідають за ініціалізацію об'єкта сторінки на основі даних БД.

Виконати роботу відповідно до індивідуальної теми розроблюваного веб-сайту.

Лабораторна робота №3

Ієрархія контенту в CMS

Мета – здобути вміння моделювати ієрархічну структуру контенту за допомогою реляційної БД.

Завдання

Доповнити таблицю контенту в БД CMS-системи ієрархічним відношенням на основі зворотного зв'язку. Створити відповідні індекси та зовнішні ключі в БД.

Доповнити програмну частину системи засобами обробки ієрархії контенту:

- 1) доопрацювати клас сторінки, доповнивши відповідним полем батьківської сторінки;
- 2) в класі ядра системи реалізувати метод побудови ієрархічного меню сайту;
- 3) внести зміни в метод публікації з урахуванням функції навігації по ієрархії контенту.

Виконати роботу відповідно до індивідуальної теми розроблюваного веб-сайту.

Лабораторна робота №4

Сторінки-контейнери в CMS

Мета – здобути вміння програмно реалізувати сторінки контейнери системи керування контентом.

Завдання

Доопрацювати систему керування сайтом, додавши функціональність сторінок-контейнерів. Внести зміни в наступні компоненти системи:

- 1) БД – змінити таблицю сторінок, доповнивши її полями, що відповідають за функціонування контейнерів;
 - 2) клас сторінки – додати методи роботи з контейнерами.
- Забезпечити два типи контейнера (плитка, список).

Реалізувати можливість керування способом сортування елементів в контейнері.

Виконати роботу відповідно до індивідуальної теми розроблюваного веб-сайту.

Лабораторна робота №5

Ациклічний оргграф контенту та сторінки-псевдоніми в CMS

Мета – здобути вміння конструювати структуру контенту CMS у вигляді ациклічного оргграфа засобами реляційної БД.

Завдання

Розвинути структуру контенту CMS таким чином, щоб вона являла собою ациклічний оргграф. Доопрацювати систему керування сайтом, додавши функціональність сторінок-псевдонімів. Внести зміни в наступні компоненти системи:

3) БД – змінити таблицю сторінок, доповнивши її полем, що відповідає за функціонування сторінок-псевдонімів;

4) клас сторінки – додати методи роботи з псевдонімами.

Застосувати сторінки-псевдоніми на сайті відповідно до теми індивідуального завдання.

Лабораторна робота №6

Макрокоманди в CMS

Мета – здобути вміння розробляти програмні компоненти, що відповідають за синтаксичний аналіз та обробку макрокоманд CMS.

Завдання

Доопрацювати CMS, додавши функцію макрокоманд контенту:

1) реалізувати модуль парсингу макрокоманд;

2) реалізувати механізм обробки макрокоманд.

Реалізувати декілька прикладних варіантів використання макрокоманд відповідно до індивідуальної теми веб-сайту.

Лабораторна робота №7

Фільтри в CMS інтернет-магазину – базова функціональність

Мета – здобути вміння реалізовувати функції підвищеної складності у веб-системах на прикладі фільтрів контенту в CMS.

Завдання

Доопрацювати CMS, додавши базові функції інтернет-магазину. Доопрацювати концептуальну модель БД системи, додавши сутності, що відповідають за реалізацію фільтрів. Реалізувати програмний модуль та його методи, що відповідають за аналіз HTTP-запиту на фільтрацію контенту, формування SQL-запиту до БД та публікацію результатів фільтрації.

Забезпечити можливість застосування логіки «Та»/«Або» в алгоритмі фільтрації контенту.

Реалізувати завдання в контексті індивідуального оформлення веб-сайту відповідно до персональної теми.

Лабораторна робота №8

Фільтри в CMS інтернет-магазину – панель вибору фільтрів

Мета – здобути вміння реалізовувати функції підвищеної складності у веб-системах на прикладі фільтрів контенту в CMS.

Завдання

Доопрацювати CMS, додавши функцію побудови елементів інтерфейсу користувача, що відповідають за вибір параметрів фільтрації контенту.

Передбачити сигналізацію про поточні параметри фільтрації.

Реалізувати завдання в контексті індивідуального оформлення веб-сайту відповідно до персональної теми.

Лабораторна робота №9

Конструювання нових типів контенту в CMS

Мета – здобути вміння програмно реалізовувати складні функції моделювання та обробки контенту в CMS на прикладі конструювання спеціалізованих типів контенту.

Завдання

Реалізувати функцію конструювання спеціалізованих типів контенту за допомогою обраного метода. Внести необхідні зміни в БД системи та реалізувати відповідні програмні компоненти, класи та модулі. Продемонструвати функціональність на прикладі відповідно до індивідуальної теми сайту.

Лабораторна робота №10

Розробка засобів автоматизованого редагування БД для CMS – базова функціональність

Мета – здобути вміння програмно реалізовувати адміністративну частину веб-системи із використанням професійних методів високорівневого програмування та повторного використання коду.

Завдання

Створити базову програмну архітектуру бібліотеки автоматизованої побудови інтерфейсу редагування БД. Створити клас таблиці та клас поля. Реалізувати функцію перегляду даних.

Продемонструвати функціональність на прикладі таблиці сторінок відповідно до індивідуальної теми сайту.

Лабораторна робота №11

Розробка засобів автоматизованого редагування БД для CMS

Мета – здобути вміння програмно реалізовувати адміністративну частину веб-системи із використанням професійних методів високорівневого програмування та повторного використання коду.

Завдання

Доопрацювати бібліотеку автоматизованої побудови інтерфейсу редагування БД. Реалізувати функції редагування, вставки та видалення перегляду даних.

Продемонструвати функціональність на прикладі таблиці сторінок відповідно до індивідуальної теми сайту.

Лабораторна робота №12

Lookup-поля як засіб реалізації зв'язку один-до-багатьох в інтерфейсі редагування

Мета – здобути вміння програмно реалізовувати адміністративну частину веб-системи із використанням професійних методів високорівневого програмування та повторного використання коду на прикладі функції полів вибору.

Завдання

Доопрацювати бібліотеку автоматизованої побудови інтерфейсу редагування БД. Реалізувати функцію застосування lookip-поля. Внести необхідні зміни в клас поля.

Продемонструвати функціональність на прикладі відповідно до індивідуальної теми сайту.

Лабораторна робота №13

Автоматизована побудова інтерфейсу головний-детальний

Мета – здобути вміння програмно реалізовувати адміністративну частину веб-системи із використанням професійних методів високорівневого програмування та повторного використання коду на прикладі реалізації інтерфейсу головний-детальний.

Завдання

Доопрацювати бібліотеку автоматизованої побудови інтерфейсу редагування БД. Реалізувати інтерфейс «головний-детальний». Для цього внести необхідні зміни в клас таблиці, а також створити клас-нащадок таблиці, відповідальний за «детальну» частину інтерфейсу.

Продемонструвати функціональність на прикладі відповідно до індивідуальної теми сайту.

Рекомендована література

1. Коггзолл Дж. PHP5 Полное руководство.: - пер.с англ. – М.: Издательский дом «Вильямс», 2006 – 752 с.
2. Титенко С. В. Структурные основы онтологически-ориентированной системы управления информационно-учебным Web-контентом / С. В. Титенко // Управляющие системы и машины: информационные технологии : междунар. науч. журн. - 2012. - № 2. - С. 35-42. ISSN 0130-5395
3. Титенко С. В. Інформаційно-логічні та архітектурні засади універсальних систем керування web-контентом/ С. В. Титенко // XII международная научная конференция имени Т. А. Таран «Интеллектуальный анализ информации ИАИ-2014», Киев, 14-16 мая 2014 г. : сб. тр./ гл. ред. С.В.Сирота. – К. : Просвіта, 2014. – С. 214-220.
4. Титенко, С. В. Моделювання спеціалізованих інформаційних об'єктів в універсальних системах керування Web-контентом / С. В. Титенко //

- Вісник Східноукраїнського національного університету імені Володимира Даля – 2012. – №8 (179). Ч.2. — С. 235-239.
5. Марти Холл, Лэрри Браун. Программирование для Web. М.: Вильямс, 2002. – 1264 с.
 6. Колисниченко Д. Н. Самоучитель PHP 5 – СПб.: Наука и Техника, 2007. – 640 с.
 7. Д. Гудман. JavaScript и DHTML: сборник рецептов. Для профессионалов. СПб.: Питер, 2004. – 528 с.
 8. Глушаков С.В., Жакин И.А., Хачиров Т.С. Программирование Web-страниц. М.: Фолио, 2005. – 390 с.
 9. Муссиано Ч., Кеннеди Б. Изучаем HTML, XHTML и CSS., 2011. – 752 с.
 10. Изучаем HTML, XHTML и CSS. - Эрик Фримен, Элизабет Фримен.,2010 – 656 с.
 11. Дари К., Бринзаре Б., Черchez-Тоза Ф., Бусика М. AJAX и PHP. Разработка динамических веб-приложений. – СПб.: Символ-плюс, 2006. –336 с.