МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Магістерської та аспірантської підготовки

Кафедра інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка візуального редактору EAV структур даних»

Виконав студент 2 року групи МК- 2
спеціальності 122 Комп'ютерні науки
Щербина Михайло Богданович

Керівник к.т.н., доц.
Трегубова Ірина Анатоліївна

Консультант

Рецензент к.т.н., доцент
Гнатовська Ганна Арнольдівна

Одеса 2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Комп'ютерних наук
Кафедра Інформаційних технологій
Рівень вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри _____
_____
" 29 " жовтня 2018 р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Щербині Михайлу Богдановичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка візуального редактору EAV структур даних» _____
_____,
керівник роботи к.т.н., доцент Трегубова Ірина Анатоліївна,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 5 " жовтня 2018 р. №271-С
2. Строк подання студентом роботи 10.12.2018 р.
3. Вихідні дані до роботи мова програмування PHP, середа розробки Jetbrains PHPStorm Community Edition, фреймворки Yii2 та AngularJS, модулі JQuery, LoDash, фронтенд фреймворк Twitter UI Bootstrap, Socket.IO та поліфіли EcmaScript 2016 для Javascript
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1 Аналіз існуючих систем для конструювання контенту
1.1 Конструктор контенту Seblod
1.2 Конструктор контенту K2
1.3 Конструктор контенту Cobalt
1.4 Конструктор контенту Zoo
1.5 Порівняльний аналіз функціоналу систем конструювання контенту
2 Аналіз основних підходів до розробки систем з EAV структурою
2.1 Модель «Сутність-аттрибут-значення»
2.2 Огляд візуального редактору WYSIWYG
3 Опис розробленого програмного забезпечення та результатів
3.1 Опис використаних технологій
3.2 Структура розробленого програмного забезпечення
3.3 Концептуальне проектування бази даних інформаційної системи
3.4 Досдіження роботи створення, зберігання сутностей у EAV форматі в базі даних
3.5 Опис алгоритму роботи з EAV структурами за допомогою ActiveRecord
3.6 Реалізація фронтенду

5. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 29 жовтня 2018р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Термін виконання етапів роботи	Оцінка виконання етапу	
			у %	за 4-х бальною шкалою
1	Огляд літературних джерел за темою магістерської роботи			
2	Аналіз основних підходів при розробці EAV систем			
3	Класифікація основних підходів навчання			
4	Рубіжна атестація	19.11.18-24.11.18		
5	Опис алгоритму роботи з БД у режимі EAV			
6	Опис та аналіз структури розробленої бази даних			
7	Опис та аналіз виконаної роботи			
8	Оформлення висновків до роботи			
9	Здача роботи на кафедру	10.12.18		
10	Перевірка на плагіат	13.12.18-14.12.18		
11	Рецензування	20.12.18		
	Інтегральна оцінка виконання етапів календарного плану (як середня по етапам)			

Студент _____ Щербина М.Б.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Трегубова І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

на магістерську роботу

«Розробка візуального редактору EAV структур даних»

Об’єкт дослідження – системи конструювання та зберігання унікальних структур за допомогою методології EAV у SQL СКБД.

Предмет дослідження – методи зберігання, конструювання унікальних структур.

Методи дослідження: об’єктно-орієнтоване моделювання, концептуальне моделювання.

Метою магістерської роботи є дослідження та розробка методів оптимального зберігання EAV структур у SQL СКБД.

Актуальність дослідження обумовлена зберегти можливості і обмеження SQL СКБД і отримати додатково можливості NoSQL СКБД.

Результатом виконання магістерської роботи є розроблений EAV фреймворк для розробки, зберігання, керування унікальних структур, даний фреймворк являє собою модуль та встановлюється у групі з Yii2, ActiveRecord.

Структура магістерської роботи складається з анотації, вступу, трьох розділів, висновків, переліку посилань на 16 найменувань. Повний обсяг роботи становить 62 сторінок і містить 10 рисунків.

Ключові слова: ІНФОРМАЦІЙНА СИСТЕМА, EAV, NOSQL, SQL, PHP, FRAMEWORK, YII2.

SUMMARY

for master's work

"Development of a Visual Editor for EAV Data Structures"

The object of the study - systems for designing and storing unique structures using the EAV methodology in SQL DBMS.

Subject of research - methods of storage, designing of unique structures.

Research methods: object-oriented modeling, conceptual modeling.

The aim of the master's thesis is to research and develop methods for optimal storage of EAV structures in SQL DBMS.

Relevance of the research is due to preserve the capabilities and limitations of SQL DBMS and to get additional capabilities of NoSQL DBMS.

The result of the master's work is the EAV framework for the development, storage, management of unique structures, this framework is a module and is installed in the group with Yii2, ActiveRecord.

The structure of the master's thesis consists of abstract, introduction, three sections, conclusions, list of references for 16 titles. The full work volume is 62 pages and contains 10 drawings.

Keywords: Information System, EAV, NOSQL, SQL, PHP, FRAMEWORK, YII2.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМИНІВ

БД – База даних
ВНЗ – Вищий навчальний заклад
ІС – Інформаційна система
ПЗ – Програмне забезпечення
ПП – Програмний продукт
СУБД – Система управління базами даних
API – Application Programming Interface (інтерфейс прикладного програмування);
CCK – Content Creation Kit (Набір для створення контенту)
CMS – Content management system (Система управління контентом);
CSS – Cascading Style Sheets (Каскадні таблиці стилів);
ERD – Entity-Relationship Diagrams (Діаграми сутність-зв'язок);
HTML – HyperText Markup Language (Мова розмітки гіпертекстових документів);
ISO – International Standardization Organization (Міжнародна організаціяпо стандартизації);
MVC – Model-View-Controller (Модель-вид-контролер);
SQL – Structured Query Language (Мова структурованих запитів);
UCS – Universal Character Set (Універсальний Набір Символів);
UML – Unified Modeling Language (Універсальна мова моделювання);
URL – Uniform Resource Locator (Єдиний вказівник ресурсу);
WYSIWYG – What You See Is What You Get (те що бачишь те і получаєшь);

ВСТУП

Розробка програмного продукта (ПП) часто є складним процесом, який вимагає від розробника виконання поставленного завдання в стислі терміни, внаслідок чого зменшується час на проектування і програміст може випустити з поля зору деякі специфічні моменти. Доопрацювання цих неврахованих моментів призводить до часткової, а іноді і до повної реструктуризації програмного забезпечення (ПЗ). В якості вирішення проблеми можна розглянути розробку ПЗ, що дозволяє швидко створювати прототип майбутньої інформаційної системи із змінною структурою метаданих. Це дозволить поєднати проектування бази даних (БД) з конструюванням, а також знизити ризики від помилок при проектуванні БД.

Подібним альтернативним підходом до організації БД є адаптована вертікальна модель даних «Сутність-атрібут-значення» (Entity-attribute-value model, EAV). Модель EAV – це модель даних, що дозволяє описати сутності, в яких кількість атрибутів (властивостей, параметрів), що характеризують їх, потенційно величезна, але та кількість, яка реально буде використовуватися в конкретній сутності, відносно мала. На даний час існують лише наробки ПЗ для зберігання EAV структур для фреймворку Yii2, що не були доведені до готового ПП, тому актуальним завданням є створення подібного відкритого ПЗ.

Метою даної магістерської роботи є розробка універсальної системи створення та зберігання даних з гнучкою динамічно змінювальною структурою, призначеної для роботи з фреймворком Yii2. Конфігурація сутностей, що можуть зберігатися системою, може відрізнятися від схеми до схеми, від проекту до проекту, вони можуть легко конфігуруватися за допомогою WYSIWYG редактору.

Для досягнення поставленої мети в роботі необхідно вирішити наступні завдання:

− виконати порівняльний аналіз різних систем, які дозволяюсь створювати сутності у базі даних за допомогою WYSIWYG редактору

− обґрунтувати вибір методів, що будуть використовуватися для вирішення проблеми ефективного зберігання;

− надати методологічні основи вирішення поставленої проблеми;

− створити зручний інтерфейс для створення сутностей, які будут зберігатися за патерном EAV;

− виконати проектування та імплементацію програмного забезпечення;

− виконати аналіз отриманих результатів.

1 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ ДЛЯ КОНСТРУЮВАННЯ КОНТЕНТУ

1.1 Конструктор контенту Seblod

SEBLOD – це потужний набір будівельних матеріалів та розробник веб-додатків для системи управління контентом Joomla! Випущений в 2009 році, SEBLOD визнано одним з провідних та інноваційних компонентів такого роду. Завдяки своїй системі плагінів, додаткових застосувань, інтерфейсу перетягування та падіння, пакетів підтримки преміум-класу, великих посібників та інше, SEBLOD є найкращим способом повністю контролювати Joomla! і поширювати його відповідно до потреб користувача (рис.1.1).

Перші, хто застосовував SEBLOD, також використовували її маленького брата, так званого "Little CCK", який допомагав редактору Joomla! створювати та налаштовувати Joomla! статті за власними шаблонами HTML. З тих пір вона перетворилася на повнофункціональний набір Content Construction Kit і Web Application Builder для Joomla! CMS, який доступний для спільноти безкоштовно протягом всіх цих років [1].

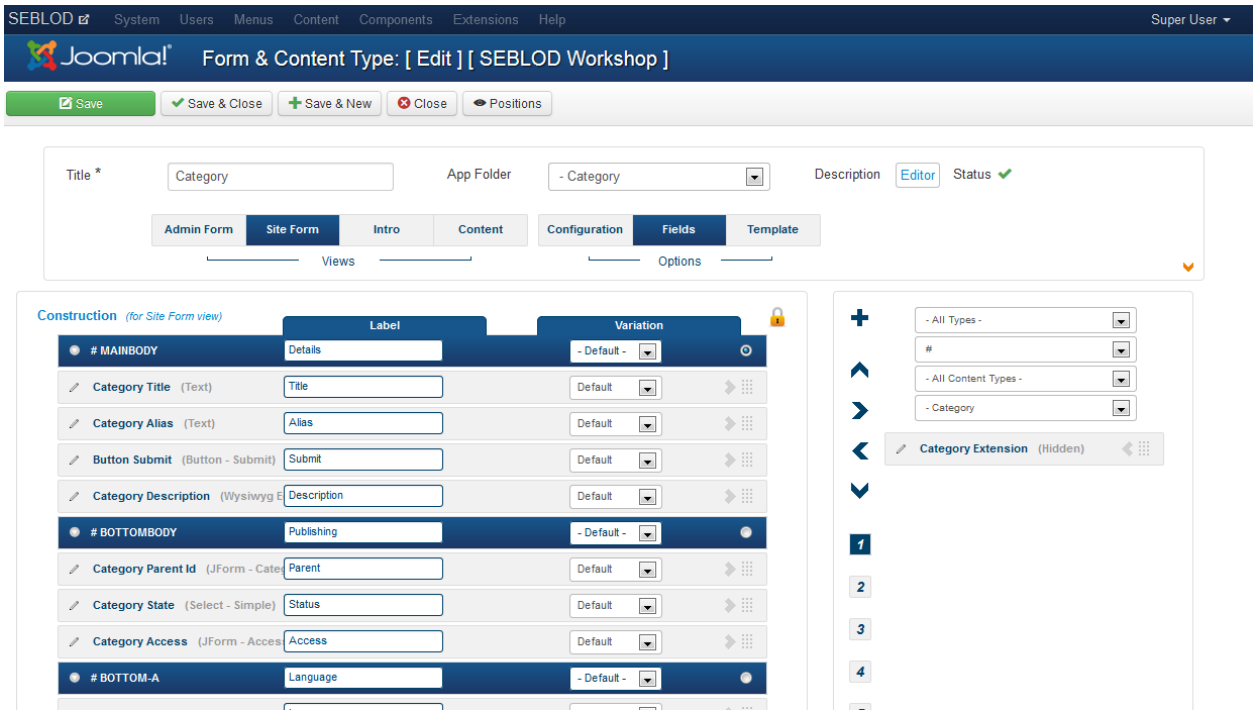

Рисунок 1.1 – Скріншот системи конструювання контенту Seblod

За замовчуванням система управління контентом Joomla! постачається з різними "об'єктами": статтями, категоріями, користувачами та групами користувачів. Кожен з них має єдиний набір полів (тип вмісту) та параметри, які до-

зволяють веб-адміністраторам легко додавати/керувати новими елементами вмісту, але майже все є заздалегідь визначеним. В якості будівельного комплекту (CCK), SEBLOD дозволяє змінювати ці типи вмісту за замовчуванням, додавши та видаляючи поля або навіть створюючи абсолютно нові спеціальні власні типи вмісту, такі як портфоліо, продукти, рецепти тощо.

SEBLOD поширюється за умовами загальної публічної ліцензії GNU версії 2 або GPLv2, що означає, що кожен може безкоштовно завантажити його та поділитися ним з іншими. Ця відкрита модель розробки означає, що програмисти постійно працюють над тим, щоб SEBLOD залишался передовим компонентом, який часто оновлюється новими функціями.

1.2 Конструктор контенту K2

K2 було побудовано як повна заміна стандартної системи статей у Joomla! Встановіть його можна як будь-яке Joomla! розширення (рис.1.2). Далі користувач може імпортувати свої статті зі стандартної Joomla! системи, і миттєво отримати безліч нових функцій для існуючого вмісту: багаті форми вмісту для предметів (статті Joomla! з додатковими полями для зображень, відео, підкасти та інші аудіофайли, галереї зображень та вкладень), безкоштовне керування зображеннями (зображення завантажених об'єктів автоматично змінюються до 6 налаштовуваних параметрів, глобально або за категорією і нема потреби змінювати їх розміри у Photoshop), коментарі, теги, вбудовані можливості для розширення форм вмісту (наприклад, для створення продукту каталогів), потужні модулі контенту, редагування інтерфейсу з легкими у використанні налаштуваннями керування доступом (для веб-сайтів із важким вмістом), потужний, але простий шаблон (і підшаблон) для переміщення над Joomla, розширені профілі користувачів, групи користувачів, блоґи, потужний API плагінів для розширення елементів/категорій/форм користувача, "перетягування " менеджера медіа та багато іншого [2].

K2 – це ідеальне рішення для керування вмістом, незалежно від розміру сайту. Його можна використовувати як для невеликого блогу, так і на складному корпоративному сайті або навіть в багатопрофільному середовищі (портали, журнали тощо). Використовуючи K2 можна перетворити свій Joomla! веб-сайт на сайт новин/журналів з блогами авторів, каталогами продуктів, робочим портфоліо, базою знань, менеджером завантаження/документообігу, списком каталогів, списком подій та інше, все це входить до складу одного пакету. А оскільки K2 розширюється з додатковими полями до форми базово-

го елемента, то є можливість легко створювати типи вмісту для певних категорій, наприклад, стаття, публікація в блозі, сторінка продукту, список каталогів.

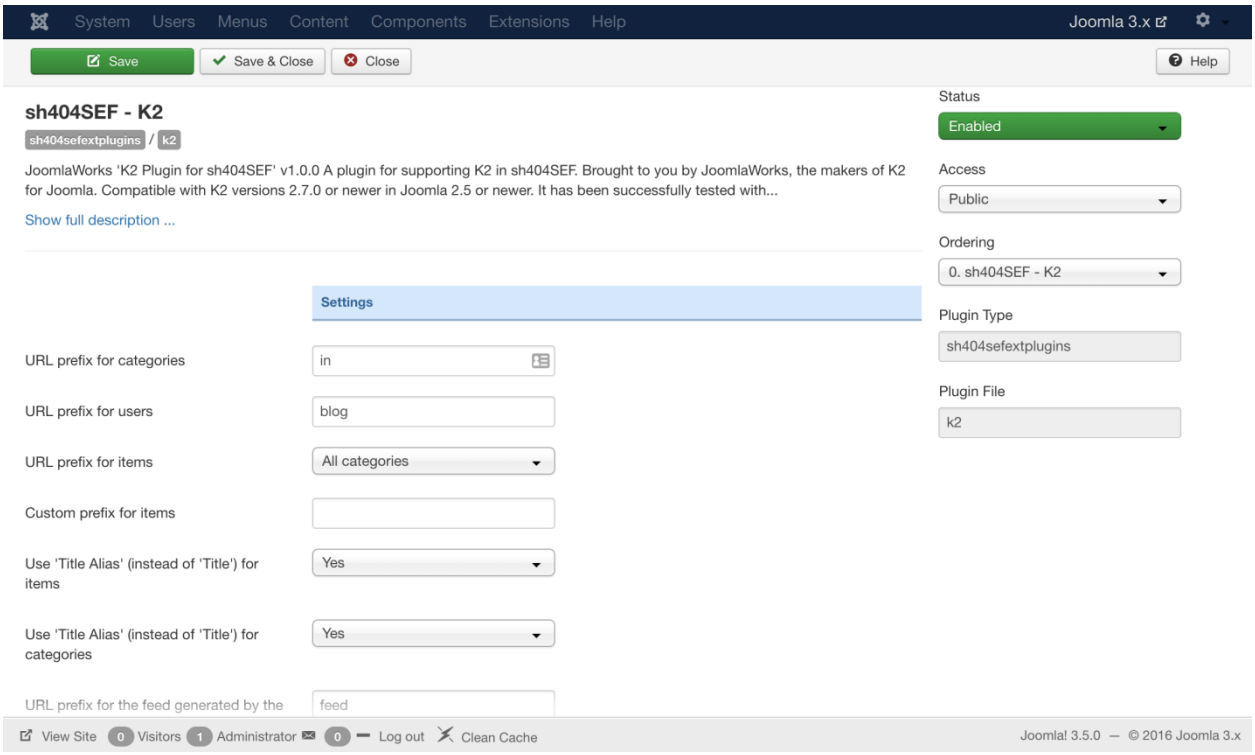

Рисунок 1.2 – Скріншот системи конструювання контенту K2

Не дивно, що K2 – це найбільша і найпопулярніша версія створення сайтів Joomla! по всьому світу. Ці інтегровані функції в K2 не тільки заощаджують адміністраторам веб-сайтів дорогоцінний час керування (від управління десятками розширень, які інакше були б потрібні), але вони також дозволяють підвищити продуктивність. Насправді, K2 був побудований за цими чотирма принципами: багатофункціональний контент, зручність використання, гнучкі шаблони, продуктивність.

1.3 Конструктор контенту Cobalt

Cobalt є наступником Mighty Resources, але більш новим, повністю переробленим (рис.1.3). Має гнучкий CCK: широкий вибір полів тексту, текстовій області, HTML, зображення, вибір, мульти-вибір тощо. Всі поля дуже легко піддаються редагуванню і надають користувачу можливостей більше, ніж він може очікувати. За допомогою Cobalt можна побудувати практично що за-

вгодно: блог, форум, службу підтримки або приватну систему квитків, нерухомість або автомобільний ринок, робочу дошку або каталог товарів [3].

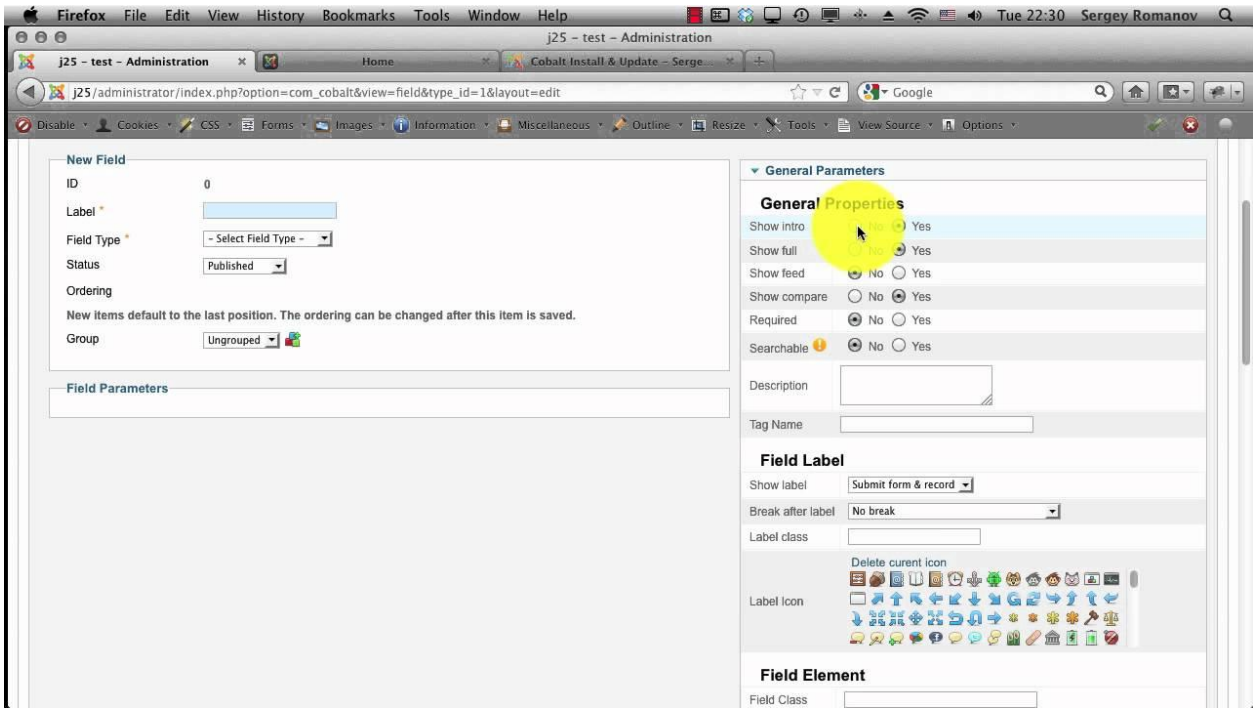

Рисунок 1.3 – Скріншот системи конструювання контенту Cobalt

Система сповіщень про стиль iPhone або Facebook дозволяє користувачам підписатися на статті або розділи/категорії та отримувати сповіщення про будь-які події.

Multi-type дозволяє надсилати, обробляти, шукати, відображати, замовляти та фільтрувати різні типи вмісту в межах одного розділу/категорії.

Система пошуку та фільтрації – інтелектуальна та налаштовувана система пошуку FULL TEXT. Розширений пошук за допомогою фільтрів за значеннями полів.

Multi-Vendor – користувачі подають з інтерфейсу і навіть можуть створювати свої власні категорії та мати свої власні домашні сторінки. Є вбудована система коментарів, а також можливість використання будь-якої іншої системи, як коментарі. Рейтингова система має 3 різни режими обчислення. Одиночні чи багаторазові оцінки майна.

Менеджер модераторів дозволяє встановити користувачів як модераторів у будь-якому розділі та налаштувати його права. Кожен список записів або запис показує шаблони, які можна керувати за допомогою диспетчера шаблонів.

1.4 Конструктор контенту Zoo

Конструктор контенту Zoo – повний аналог попередніх систем (рис.1.4). Він дозволяє створювати декілька екземплярів застосувань. Кожен керує власним вмістом, таким як елементи та категорії [4].

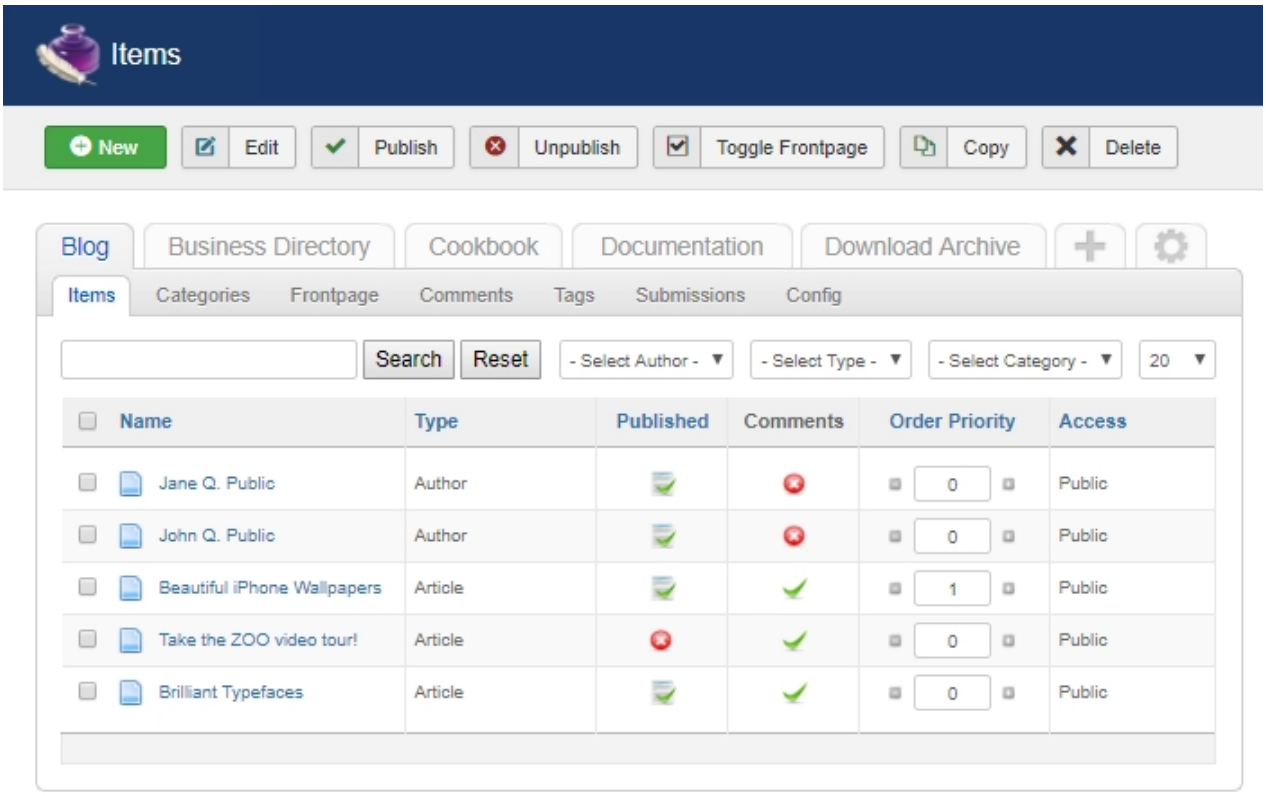

Рисунок 1.4 – Скріншот системи конструювання контенту Zoo

Zoo забезпечує чистий дизайн із підтримкою JavaScript на основі користувальницького інтерфейсу для керування застосуваннями. Дозволяє створювати власні типи вмісту відповідно до вимог програміста. Вибирати із насиченого набору елементів, таких як форми, зображення, відео, рейтинги тощо. Упорядковувати вміст за допомогою перетягування, призначаючи елементи для розташування макетів. Розширювати категорії за допомогою спеціальних полів та сортувати їх за допомогою перетягування. Система слайдів коментарів підтримує Akismet, Gravatar, Twitter і Facebook Connect. Додавання тегів має автоматичне завершення та різні алгоритми завершення. Довірені автори можуть переглядати та редагувати створюваний вміст безпосередньо з інтерфейсу.

1.5 Порівняльний аналіз функціоналу систем конструювання контенту

Результати порівняльного аналізу функціоналу сучасних систем конструювання контенту наведені в табл. 1.1.

Таблиця 1.1 – Порівняльний аналіз запросів, використаної пам’яті та часу виконання запросу

Система конструювання контенту	Cobalt	Seblod	K2	Zoo
Кількість запросів	30	23	26	23
Використана пам’ять	9.28Mb	12.04Mb	7.6Mb	8.11Mb
Час виконання	0.086s	0.108s	0.056s	0.073s

Як видно з таблиці 1.1 з точки зору часу виконання та з точки зору використонаї пам’яті на першому місті буде K2, на другому Zoo, на третьому Cobalt, а на останньому Seblod, але з точки зору кількості запросів перше місце буде поділене між Seblod та Zoo [5].

У своїй магістрській кваліфікаційній роботі я намагатимусь превзойти ці характеристики.

2 АНАЛІЗ ОСНОВНИХ ПІДХОДІВ ДО РОЗРОБКИ СИСТЕМ З EAV СТРУКТУРОЮ

2.1 Модель «Сутність-аттрибут-значення»

Модель «Сутність-атрибут-значення» (EAV) – це модель даних для кодування об'єктів у більш компактному вигляді, де кількість атрибутів (властивостей, параметрів), які можуть бути використані для їх опису, потенційно велика, але кількість, яка буде насправді застосовуватися до даного об'єкта відносно мала. Такі сутності відповідають математичному поняттю розрідженої матриці. EAV також відома як модель об'єкт-атрибут-значення, і є вертикальною моделлю бази даних і відкритою схемою.

Представлення даних в EAV аналогічно просторово-ефективним методам зберігання розрідженої матриці, де зберігаються лише непусті значення. У моделі даних EAV кожна пара атрибут-значення є фактом, який описує сутність, а рядок в таблиці EAV зберігає один факт. Таблиці EAV часто описуються як "довгі та тонкі": "довгий" відноситься до кількісті рядків, "тонкий", до кількості стовпців.

Класична EAV модель складається з трьох таблиць (рис.2.1):

Entity – таблиця відповідає за сутність в базі даних (аналог об’єктів в ООП).

Attributes – таблиця відповідає за атрибути в базі даних (аналог атрибтів в ООП). Таблиця атрибутів може містити такі стовпці: ідентифікатор атрибуту, назва атрибуту, опис, тип даних та стовпці, які допомагають перевірити вхідні дані, наприклад, максимальна довжина рядка та регулярний вираз, набір дозволених значень і т.п.

Values – таблиця відповідає за значення атрибутів.

Розглянемо, як можна спробувати представити клінічний запис загального призначення в реляційній базі даних. Очевидно, що створення таблиці (або набору таблиць) з тисячами стовпців неможливо, оскільки переважна більшість стовпців буде нульовою. Крім того, у медичному документі, що супроводжує пацієнта впродовж тривалого часу, може бути декілька значень одного параметра: висота та вага дитини, наприклад, змінюються по мірі її зростання. Нарешті, безліч клінічних даних продовжує зростати: наприклад, виникають захворювання та розробляються нові лабораторні тести; це потребує постійного додавання стовпчиків та постійного перегляду користувальницько-

го інтерфейсу (ситуація, коли список атрибутів часто змінюється в базі даних, називається " attribute volatility").

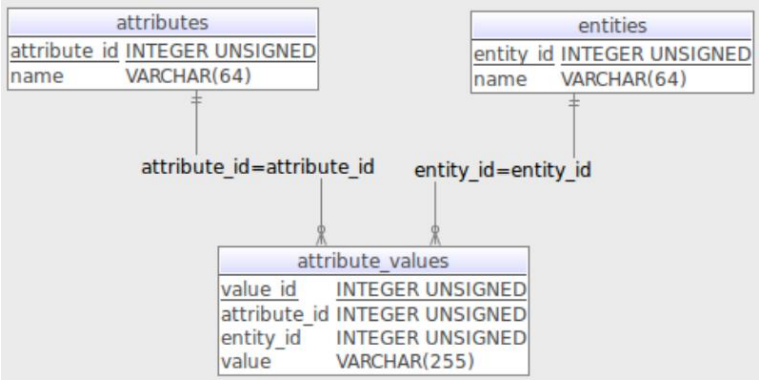

Рисунок 2.1 – Класична EAV модель

Нижче наведений знімок таблиці EAV для клінічних результатів відвідування лікаря у зв'язку з підвищенням температури вранці 1/5/98. Записи, що показані в кутових дужках, є посиланнями на записи в інших таблицях, показані тут як текст, а не як закодовані значення зовнішнього ключа для полегшення розуміння. У цьому прикладі всі значення є літеральними значеннями, але вони також можуть бути попередньо визначеними списками значень. Останні особливо корисні, коли відомо, що можливі значення обмежені.

Entity. Для клінічних результатів сутністю є подія пацієнту: зовнішній ключ в таблиці, який містить як мінімум ідентифікатор пацієнта та одну або декілька міток часу (наприклад, початок і кінець дати/часу огляду), які записуються, коли подія, що описується, сталася.

Attribute – зовнішній ключ у таблицю атрибутів. Таблиця атрибутів буде містити такі стовпці: ідентифікатор атрибуту, назва атрибуту, опис, тип даних, одиниці виміру та стовпчики, що допомагають перевірці введення, наприклад, максимальна довжина рядка та регулярний вираз, максимально та мінімально допустими значення, набір допустимих значень тощо.

Value. Буде залежати від типу даних.

Наведений нижче приклад ілюструє результати симптомів, які можуть спостерігатися у пацієнта з пневмонією.

(<patient XYZ, 1/5/98 9:30 AM>,<Temperature in Fahrenheit>, "102")

(<patient XYZ, 1/5/98 9:30 AM>,<Presence of Cough>,"True")

(<patient XYZ, 1/5/98 9:30 AM>,<Type of Cough>,"With phlegm, yellowish, streaks of blood")

(<patient XYZ, 1/5/98 9:30 AM>, <Heart Rate in beats per minute>, "98")

Термін "база даних EAV" відноситься до структури бази даних, в якій значна частина даних моделюється як EAV. Однак навіть в базі даних, що описується як "EAV-орієнтована", деякі таблиці в системі є традиційними реляційними таблицями.

Як зазначалося вище, моделювання EAV має сенс для категорій даних, таких як клінічні дані, де атрибути численні та рідкі. Якщо ці умови не виконуються, бажано використовувати стандартне реляційне моделювання (наприклад, один стовпець на атрибут); використання EAV не означає відмову від принципів реляційного дизайну. У системах клінічних записів схеми, що стосуються демографічних показників та білінгу пацієнтів, зазвичай моделюються умовно (хоча більшість схем бази даних постачальників є власністю, система VistA, яка використовується у медичній системі Департаменту ветеранів США, відома під назвою Адміністрація охорони здоров'я ветеранів, є відкритим джерелом, і її схема легко перевіряється, хоча вона використовує двигун бази даних MUMPS, а не реляційну базу даних).

Як буде показано нижче, база даних EAV, практично не підтримується без численних допоміжних таблиць, що містять допоміжні метадані. Таблиці метаданих, які зазвичай перевищують кількість таблиць EAV щонайменше в три або більше разів, зазвичай є стандартними реляційними таблицями. Прикладом таблиці метаданих є таблиця атрибутів, згадана вище.

Дані EAV, описані вище, можна порівняти з вмістом квитанції про продаж в супермаркеті. У квитанції перераховані лише дані про фактично придбані товари, а не лістинг кожного продукту в магазині, який замовник міг придбати, але не придбав. Як і клінічні дані для пацієнта, квитанція про продаж містить рідкі дані.

"Entity" – це ідентифікатор продажу/транзакції – зовнішній ключ у таблиці транзакцій продажи. Використовується для внутрішнього маркування кожної позиції, хоча в квитанції інформація про продаж відображається вгорі (місце розташування магазину, дата/час продажу) і внизу (загальна вартість продажу).

"Attribute" – це зовнішній ключ в таблиці продуктів, звідки можна знайти опис, ціну одиниці, знижки, акції тощо. Продукти є настільки ж непостійними, як і клінічні дані, можливо, навіть більше: нові продукти вводяться щомісяця. Компетентний розробник баз даних не буде жорстко кодувати окремі продукти, як стовпці в таблиці.

"Value" – це кількість придбаних товарів та загальна ціна позиції.

Факти (в нашому випадку транзакції продаж) записуються в вигляді декілька рядків, а не декількох стовпців. Це є стандартною методикою моделювання даних. Розглянемо в чому різниця між моделюванням рядків та EAV (яку можна вважати узагальненням рядкового моделювання):

1) Рядкова таблиця є однорідною за фактами, які вона описує: таблиця «Позиція» описує лише продані продукти. Навпаки, таблиця EAV містить практично будь-який тип факту.

2) Тип даних стовпца/стовпців у рядковій таблиці попередньо визначається сутністю фактів, які він містить. На відміну від цього, у таблиці EAV концептуальний тип даних в певному рядку залежить від атрибута в цьому рядку. Звідси випливає, що у виробничих системах пряме ввведення даних в таблицю EAV може стати причиною проблем, оскільки сам движок бази даних не зможе виконати надійну перевірку введенних даних. Пізніше буде розглянуто, як можна побудувати загальні структури, які виконують більшість завдань валідації вводу, без нескінченного кодування на основі атрибутів за атрибутами.

У сховищі клінічних даних моделювання рядків також знаходить використання: нприклад, так звичайно моделюється схема лабораторних випробувань, оскільки результати лабораторних випробувань, як правило, є числовими або можуть бути закодовані чисельно.

Нижче наведено обставини, за якими доведеться виходити за рамки стандартних моделей рядків до EAV:

1) Тип даних окремих атрибутів варіюється (як це видно з клінічних даних).

2) Категорії даних численні, зростають або коливаються, але кількість екземплярів (записів/рядків) в кожній категорії дуже мала. Тут при традиційному моделюванні діаграма відносин між сутностями та базою даних може мати сотні таблиць: таблиці, які містять тисячі / мільйони рядків / екземплярів, візуально виділяються в тій же мірі, що й ті, що мають дуже мало рядків. Останні є кандидатами для перетворення в подання EAV.

Ця ситуація виникає в середовищах моделювання онтологій, де категорії ("класи") часто повинні бути створені на льоту, і деякі класи часто виключаються в наступних циклах прототипів.

Деякі ("гібридні") класи мають деякі атрибути, які не є розрідженими (присутні у всіх або в більшості випадків), в той час як інші атрибути є сильно мінливими і розрідженими. Останні підходять для моделювання EAV. Наприклад, описи продуктів, виготовлених конгломератной корпорацією, залежать

від категорії продукту, наприклад, атрибути, необхідні для опису марки лампочки, відрізняються від тих, які потрібні для опису медичного пристрою візуалізації, але обидва мають загальні атрибути, такі як упаковка, ціна за одиницю товару.

Entity. У клінічних даних сутність, як правило, є клінічною подією, як описано вище. У більшості випадках сутність є зовнішнім ключом в таблиці «сутностей», в якій записана загальна інформація про кожну «сутність» в базі даних – як мінімум, ім'я та короткий опис, а також категорія / клас, до якої вона належить. Кожному запису (сутності) в цій таблиці присвоюється згенерований машиною ідентифікатор.

Підхід «таблиці сутностей» був вперше запропонований Томом Слезаком і його колегами з Лабораторії Лоуренса Лівермора для бази даних «Хромосома 19», і в даний час є стандартним в більшості великих баз даних з біоінформатики. Використання таблиці сутностей не вимагає одночасного використання дизайну EAV: звичайні таблиці можуть використовуватися для зберігання специфічних для категорії деталей кожної сутності.

Основна перевага центральної таблиці сутностей полягає в тому, що, маючи допоміжну таблицю синонімів і ключових слів сутностей, можна забезпечити стандартний механізм пошуку, подібний Google, по всій системі, де користувач може знайти інформацію про будь-яку сутність без необхідності спочатку зазначення категорії, до якої вона належить. Це важливо в біонаучних системах, де ключове слово, таке як «ацетилхолін», може ставитися або до самої молекулі, яка є нейромедіатором, або до біологічного рецептора, з яким вона зв'язується.

Attribute. У самій таблиці EAV це просто ідентифікатор атрибута, зовнішній ключ в таблиці визначень атрибутів, як зазначено вище. Однак зазвичай існує декілька таблиць метаданих, які містять інформацію, що відноситься до атрибутів.

Value. Приведення всіх значень в рядки, як у наведеному вище прикладі даних EAV, призводить до простої, але не масштабуємої структурі: взаємні перетворення типів даних з константою потрібні, якщо хтось хоче щось зробити зі значеннями, і індекс за значенням стовпець таблиці EAV по суті даремний. Крім того, незручно зберігати великі двійкові дані, такі як зображення, в закодованої формі Base64 в тій же таблиці, що і маленькі цілі числа або рядки. Тому у великих системах використовуються окремі таблиці EAV для кожного типу даних, включаючи великі двійкові об'єкти «BLOBS», з метаданими для даного атрибута, що ідентифікує таблицю EAV, в якій будуть зберігатися її

дані. Цей підхід насправді досить ефективний, тому що скромна кількість метаданих атрибутів для даного класу або форми, з якими користувач вибирає працювати, може бути легко кешувати в пам'яті. Однак, це вимагає переміщення даних з однієї таблиці в іншу, якщо тип даних атрибута змінюється. Це трапляється не часто, але помилки можуть бути зроблені при визначенні метаданих, як і при розробці схеми бази даних.

2.1.1 Моделі підструктур EAV/CR

У простому дизайні EAV, стосовно СКБД, значення атрибута – це простий примітивний тип даних. Проте в системі EAV, що використовується для представлення дуже різноманітних даних, можливо що даний об'єкт (екземпляр класу) може мати підструктуру, тобто деякі його атрибути можуть представляти інші типи об’єктів, які в свою чергу, можуть мати підструктуру будь-якого рівня складності. Наприклад, автомобілі мають двигуни та трансмісії, а двигун має такі компоненти, як циліндри.

Для придставлення підструктури в стовпці значення використовується спеціальна таблиця EAV, що містить посилання на інші об'єкти в системі (тобто зовнішні значення ключа до таблиці об'єктів). Для того, щоб отримати всю інформацію про конкретний об'єкт, потрібний рекурсивний обхід метаданих з наступним рекурсивним обходом даних, який зупиняється, коли кожний вилучений атрибут є простим (атомарним). Рекурсивний обхід необхідний незалежно від того, чи представлені деталі окремих класів у традиційному або EAV форматі. Такий обхід здійснюється, наприклад, у стандартних об'єктно-реляційних системах. Насправді, в більшості класів кількість рекурсійних рівнів, як правило, є відносно невеликою. Деградація продуктивності через рекурсію є скромною, особливо в індексації ідентифікаторів об'єктів.

EAV/CR (EAV з класами і відносинами) відноситься до каркасу, який підтримує складні базові структури. Його назва дещо неправильна: насправді більшість класів в такій системі можуть бути представлені в стандартній реляційній формі в залежності від того чи є атрибути редкими або щільними. EAV/CR має багаті, дуже детальні метадані, достатні для підтримки автоматичної генерації інтерфейсів веб-перегляду для окремих класів без не обхідності написання коду користувальницького інтерфейсу для кожного класу. Основою такого інтерфейсу браузера є те, що можна генерувати пакет динамічних SQL запитів, які не залежать від класу об'єкта, спочатку звірившись з його метаданими та використовуючи інформацію метаданих для генерації послідовності

запитів до таблиць даних, і деякі з цих запросів можуть бути довільно рекурсивними. Цей підхід добре працює з запитами об'єктів за об'єктами, як і веб-інтерфейс перегляду, де всі деталі об'єкта відображаються на різних сторінках при натисканні на назву об'єкта. Метадані пов'язані з класом цього об'єкта, тому що представлення деталей об'єкта включає в себе заголовок окремих атрибутів, порядок їх представлення та спосіб їх групування.

Один з підходів до EAV/CR надає необхідну структуру класу, дозволяючи стовпчику створити структуру JSON. Наприклад, PostgreSQL підтримує двійкові рядки JSON (JSONB) з версії 9.4, і може запитувати, індексувати та об'єднувати атрибути JSON.

2.1.2 Метадані в системах EAV

За словами професора д-ра Даніеля Масіса (який раніше очолював факультет медичної інформатики Університету Вандербільта), проблеми роботи з EAV пов'язані з тим, що в базі даних EAV «фізична схема» (спосіб зберігання даних) радикально відрізняється від «логічної схеми» – того, як користувачі і програмні застосування, такі як пакети статистики, розглядають її, тобто як звичайні рядки і стовпці для окремих класів.

Метадані допомагають користувачам взаємодіяти з системою з точки зору логічної схеми, а не фізичної: програмне забезпечення постійно консультується з метаданими для різних операцій, таких як подання даних, інтерактивна перевірка, витяг масових даних і спеціальний запит. Метадані можуть фактично використовуватися для настройки поведінки системи.

Системи EAV обмінюють простоту фізичної та логічної структури даних на складність їх метаданих, які, крім іншого, грають роль обмежень бази даних і посилальної целостності. Такий компроміс, як правило, має сенс, тому що в типовій змішаній схемі виробничих систем дані в звичайних реляційних таблицях також можуть отримати переваги від таких функцій, як автоматичне створення інтерфейсу. Структура метаданих досить складна, так що вона включає свою власну підсхему в базі даних: різні зовнішні ключі в таблицях даних посилаються на таблиці в цій підсхеме. Ця підсхема стандартно-реляційна, з такими ознаками, як обмеження і посилальна цілісність.

Коректність змісту метаданих з точки зору передбачуваної поведінки системи має вирішальне значення, і завдання забезпечення правильності означає, що при створенні системи EAV необхідно докласти значних зусиль для створення користувацьких інтерфейсів для редагування метаданих, які можуть

використовуватися користувачами, які знають проблемну область (наприклад, клінічна медицина), але не обов'язково є програмістами.

Коли система EAV реалізується через RDF, мова схеми RDF зручно використовується для представлення таких метаданих. Ця інформація про схему використовується двигуном бази даних EAV для динамічної реорганізаії його внутрішньої структури таблиц для збепеечення максимальної ефективності.

Деякі попереджень про метадані:

1) Оскільки бізнес-логіка міститься в метаданих, а не в явному вигляді в схемі бази даних (тобто видалена на один рівень у порівнянні з традиційно розробленими системами), вона менш очевидна для того, хто не знайомий з системою. Тому інструменти перегляду метаданих та звітності про метадані важливі для забезпечення зручності обслуговування системи EAV. У звичайному сценарії, коли метадані реалізуються у вигляді реляційної підсхеми, ці інструменти являють собою застосування, створені з використанням готових інструментів звітності або запитів, які працюють з таблицями метаданих.

2) Недостатньо обізнаному користувачу легко зіпсувати (тобто внести невідповідності і помилки в) метадані. Таким чином, доступ до метаданих повинен бути обмежений, і повинен бути введений контрольний журнал доступу і змін, особливо у випадку, коли кілька людей мають доступ до метаданих. Використання СКБД для метаданих спростить процес забезпечення узгодженості при створенні і редагуванні метаданих завдяки використанню функцій СКБД, таких як підтримка транзакцій. Крім того, якщо метадані є частиною тієї ж бази даних, що і самі дані, це забезпечує резервне копіювання принаймні так само часто, як і самих даних, щоб їх можна було відновити на певний момент часу.

3) Якість анотації та документації в метаданих (тобто описовий / пояснювальний текст в стовпцях підсхеми метаданих) має бути набагато вищою, щоб полегшити розуміння різними членами команди розробників. Забезпечення якості метаданих (і підтримання його актуальності по мірі розвитку системи) має дуже високий пріоритет в довгостроковому управлінні і обслуговуванні будь-якого проекту, в якому використовується компонент EAV. Погано документовані або застарілі метадані можуть поставити під загрозу довгострокову життєздатність системи.

Розглянемо докладніше типи атрибутів метаданих.

Метадані перевірки (Validation metadata) включають в себе типи даних, діапазон допустимих значень або членство в наборі значень, збіг з регулярним виразом, значення за замовчуванням і допустимість значення NULL. У систе-

мах EAV, що представляють класи з підструктурою, метадані перевірки також будуть записувати, до якого класу належить даний атрибут, якщо такий є.

Метадані представлення (Presentation metadata) – це як атрибут повинен відображатися користувачеві (наприклад, у вигляді текстового поля або зображення заданих розмірів, списку або набору перемикачів). Коли складений об'єкт складається з декількох атрибутів, як в проекті EAV/CR, існують додаткові метадані в порядку, в якому повинні представлятися атрибути, і як ці атрибути повинні бути необов'язково згруповані (під описовими заголовками).

Групування метаданих (Grouping metadata) – атрибути зазвичай представлені як частина групи вищого порядку, наприклад, спеціальної специфічної форми. Групування метаданих включає в себе таку інформацію, як порядок подання атрибутів. Певні метадані представлення, такі як шрифти/кольори і кількість атрибутів, що відображаються в рядку, застосовуються до групи в цілому.

Метадані розширеної перевірки.

Метадані залежності (Dependency metadata) – у багатьох інтерфейсах користувача, для введення конкретних значень для окремих полів / атрибутів потрібно вимкнути / приховати інші конкретні поля або активувати або відображати інші поля. Наприклад, якщо користувач вибирає відповідь "Ні" на логічне запитання "Чи є у вас діабет?", треба скасувати наступні питання щодо періоду діабету, діабету та ін. Загальна структура включає зберігання залежностей між керуючими і контрольованими атрибутами.

Обчислення і комплексна перевірка. Як і в електронній таблиці, значення певних атрибутів можуть бути обчислені і відображені на основі значень, введених в поля, які представлені раніше в послідовності. Наприклад, площа поверхні тіла залежить від висоти і ширини. Точно так само можуть існувати «обмеження», які повинні бути істинними, щоб дані були дійсними: наприклад, при диференціальному підрахунку білих комірок сума підрахунків окремих типів білих комірок завжди повинна дорівнювати 100, оскільки окремі підрахунки представляють відсотки. Обчислені формули і комплексна перевірка зазвичай здійснюються шляхом збереження виразів в метаданих, які замінюються макросами значеннями, які вводить користувач, і які можуть бути оцінені. У веб-браузерах JavaScript і VBScript є функція Eval(), яку можна використовувати для цієї мети.

Валідація, представлення і групування метаданих роблять можливим створення структур коду, які підтримують автоматичну генерацію призначеного для користувача інтерфейсу як для перегляду даних, так і для інтерактив-

ного редагування. У виробничій системі, що поставляється через Інтернет, завдання перевірки даних EAV по суті переміщається з рівня бек-енду бази даних на рівень середнього рівня веб-сервера. У той час як внутрішня перевірка завжди ідеальна, тому що неможливо підірвати спроби прямого введення даних в таблицю, перевірка середнього рівня через універсальну платформу цілком працездатна, хоча значна частина зусиль з проектування програмного забезпечення повинна йти в першу чергу при побудові цього середовища.

2.1.3 Сценарії для моделювання EAV

EAV-моделювання під альтернативними термінами «моделювання загальних даних» або «відкрита схема» довгий час було стандартним інструментом для досвідчених розробників моделей даних. Як і будь-яка просунута техніка, вона може бути обопільно гострою і повинна використовуватися розумно.

Крім того, використання EAV не перешкоджає використанню традиційних підходів моделювання реляційних баз даних в одній і тій же схемі бази даних. У EMR, які покладаються на RDBMS, такі як Cerner, які використовують підхід EAV для своєї підсхеми клінічних даних, переважна більшість таблиць в схемі фактично традиційно моделюється за атрибутами, представленими як окремі стовпці, а не як рядки.

Моделювання підсхеми метаданих системи EAV, насправді, дуже добре підходить для традиційного моделювання через взаємозв'язки між різними компонентами метаданих. Наприклад, в системі TrialDB кількість таблиць метаданих в схемі перевищує число таблиць даних приблизно в десять разів. Оскільки правильність і узгодженість метаданих має вирішальне значення для правильної роботи системи EAV, розробник системи хоче використовувати всі переваги всіх функцій, що надаються СКБД, таких як посилальна цілісність і програмовані обмеження. Отже, численні таблиці метаданих, які підтримують схеми EAV, зазвичай мають реляционную форму в третій нормальній формі.

Комерційні електронні системи медичних записів (EHR) використовують моделювання рядків для класів даних, таких як діагнози, виконані хірургічні процедури і результати лабораторних випробувань, які розділені в окремі таблиці. У кожній таблиці «сутність» являє собою сукупність ідентифікатора пацієнта і дати / часу, коли був поставлений діагноз (або проведена операція або лабораторний тест); атрибут є зовнішнім ключем в спеціально призначеній таблиці пошуку, яка містить контрольований словник – наприклад, ICD-10 для

діагностики, поточна процедурна термінологія для хірургічних процедур, з набором значень атрибутів. Наприклад, для результатів лабораторних випробувань можна записати виміряне значення, чи знаходиться воно в нормальному, низькому або високому діапазоні, ідентифікатор особи, відповідальної за проведення тесту, дата / час проведення тесту тощо. Як зазначалося раніше, це не повноцінний підхід EAV, оскільки домен атрибутів для даної таблиці обмежений, так само як домен ідентифікаторів продуктів в таблиці продажів супермаркету буде обмежений доменом продуктів в продуктах.

Однак для збору даних про параметри, які не завжди визначаються в стандартних словниках, EHR також надають «чистий» механізм EAV, в якому спеціально призначені досвідчені користувачі можуть визначати нові атрибути, їх тип даних, максимальні і мінімальні допустимі значення (або допустимий набір значень/кодів), а потім дозволити іншим збирати дані на основі цих атрибутів. У Епіс EHR цей механізм називається «технологічними схемами» і зазвичай використовується для збору даних спостережень в стаціонарі.

Типовий випадок використання моделі EAV – це дуже розріджені, неоднорідні ознаки, такі як клінічні параметри в електронній медичній карті (EMR), як зазначалося вище. Однак навіть тут можна з упевненістю стверджувати, що принцип моделювання EAV застосовується до підсхеми бази даних, а не до всього її вмісту. Наприклад, демографічні дані пацієнтів найприродніше моделюються в традиційній реляційній структурі «один стовпець на атрибут».

Отже, аргументи про EAV і «реляційному» дизайні відображають неповне розуміння проблеми: дизайн EAV повинен використовуватися тільки для тієї підсхеми бази даних, де необхідно моделювати розріджені атрибути: навіть тут вони повинні підтримуватися по третій нормальній формі таблиці метаданих. Існує досить мало проблем проектування баз даних, де зустрічаються розріджені атрибути: ось чому обставини, в яких застосовується дизайн EAV, зустрічаються відносно рідко. Навіть там, де вони зустрічаються, набір таблиць EAV не є єдиним способом вирішення розріджених даних: може бути застосовано рішення на основі XML, коли максимальна кількість атрибутів на об'єкт відносно мала, а загальний обсяг розріджених дані також мал. Прикладом такої ситуації є проблеми захоплення змінних атрибутів для різних типів продуктів.

Розріджені атрибути можуть також виникати в ситуаціях електронної комерції, коли організація купує або продає великий і дуже різноманітний набір товарів, причому деталізація окремих категорій товарів сильно варіюється.

Програмне забезпечення електронної комерції Magento використовує підхід EAV для вирішення цієї проблеми.

Інше застосування EAV – моделювання класів і атрибутів, які хоча і не розряджені, але динамічні, де число рядків даних на клас буде відносно скромним – максимум кілька сотень рядків. Розробник також зобов'язаний надати веб-інтерфейс для кінцевого користувача в дуже короткі терміни. «Динамічний» означає, що нові класи і атрибути повинні постійно визначатись і підлягати коригуванню, щоб представляти модель даних, що розвивається. Цей сценарій може відбуватися в наукових областях, що швидко розвиваються, а також в розробці онтологій, особливо на етапах прототипування і ітеративного уточнення.

Хоча створення нових таблиць і стовпців для представлення нової категорії даних не є особливо трудомістким, має місце програмування веб-інтерфейсів, які підтримують перегляд або базове редагування з перевіркою на основі типів і діапазонів. В такому випадку більш прийнятним довгостроковим рішенням є створення інфраструктури, в якій визначення класів і атрибутів зберігаються в метаданих, а програмне забезпечення динамічно генерує базовий призначений для користувача інтерфейс з цих метаданих.

Платформа EAV/CR була створена для вирішення цієї самої ситуації. Модель даних EAV тут не обов'язкова, але розробник системи може вважати її прийнятною альтернативою створенню, скажімо, шістдесяти або більше таблиць, що містять в цілому не більше двох тисяч рядків. Тут, оскільки кількість рядків в класі дуже мала, міркування ефективності менш важливі; при стандартній індексації за ідентифікатором класу / ідентифікатором атрибута оптимізатори СКБД можуть легко кешувати дані для невеликого класу в пам'яті при виконанні запиту, що включає цей клас або атрибут.

У сценарії з динамічними атрибутами варто відзначити, що структура опису ресурсів (RDF) використовується як основа роботи над онтологією, пов'язаної з семантичною павутиною. RDF, призначений для загального методу подання інформації, є формою EAV: трійка RDF містить об'єкт, властивість і значення.

Моделювання EAV застосовується тільки до підсистеми, де традиційне реляційне моделювання відомо апріорі буде громіздким (як в області клінічних даних) або виявленим в ході еволюції системи ставити серйозні завдання по обслуговуванню. Гуру бази даних (і в даний час віце-президент по основним технологіям в корпорації Oracle) Том Кайт, наприклад, правильно вказує на недоліки використання EAV в традиційних бізнес-сценаріях і підкреслює,

що однією лише «гнучкості» недостатньо критерій використання EAV. Однак він заявляє, що EAV слід уникати за будь-яких обставин, навіть незважаючи на те, що саме підрозділ Oracle Health Sciences використовує EAV для моделювання атрибутів клінічних даних в своїх комерційних системах ClinTrial і Oracle Clinical.

2.1.4 Робота з даними EAV

Проблемою EAV є складність обробки великих обсягів даних EAV. Часто необхідне тимчасове або постійне взаємне перетворення між стовпцовими та рядковими або EAV-моделюємими представленнями одних і тих самих даних; це може привести до помилок, якщо робиться вручну, так і при інтенсивном використанні процесору. Універсальні структури, які використовують атрибути та метадані групування атрибутів, звертаються до першого, але не до другого обмеження; їх використання є більш або менш обов'язковим у випадку сумісних схем, які містять сумі звичайних реляційних даних та даних EAV, де коефіцієнт помилок може бути дуже значним.

Операція перетворення називається зведенням. Зведення потрібний не тільки для даних EAV, але також для будь-яких даних у формі рядків. Наприклад, реалізація алгоритму Apriori для аналізу асоціацій, широко використовуваного для обробки даних про продажі в супермаркетах для визначення інших продуктів, які покупці даного продукту також можуть купувати, в якості першого шага об'єднують дані, що моделюються за рядками. Багато механізмів бази даних мають власні розширения SQL для облегчения зведення, і такі пакети, як Microsoft Excel, також підтримують його. Нижче розглянемо обставини, коли потрібно виконувати зведення.

Перегляд малих обсягів даних для окремої сутності, за якими, можливо, йде редагування даних на основі залежностей між атрибутами. Ця операція полегшується за рахунок кешування невеликої кількості необхідних метаданих підтримки. Деякі програми, такі як TrialDB, отримують доступ до метаданих для створення напівстатичних веб-сторінок, які містять вбудований програмний код, а також структури даних, що містять метадані.

Масове вилучання перетворює великі (але передбачувані) обсяги даних (наприклад, повні дані клінічного дослідження) в набір реляційних таблиць. Незважаючи на інтенсивне завантаження процесора, це завдання нечасто і не вимагає виконання в режимі реального часу; тобто користувач може чекати завершення пакетного процесу. Важливість масового вилучення неможливо переоцінити, особливо коли дані повинні оброблятися або аналізуватися за

допомогою стандартних сторонніх інструментів, які повністю не знають про структуру EAV. Тут краще за все просто витягувати дані EAV в реляційні таблиці, а потім працювати з ними за допомогою стандартних інструментів.

Спеціальні інтерфейси запитів до даних, отриманих по рядках або EAV, при запиті з точки зору окремих атрибутів (наприклад, «отримати всіх пацієнтів з наявністю захворювання печінки, з ознаками печінкової недостатності і відсутністю зловживання алкоголем в анамнезі») зазвичай показують результати запиту з окремими атрибутами у вигляді окремих стовпців. Для більшості сценаріїв бази даних EAV продуктивність спеціальних запитів повинна бути терпимою, але відповіді в частки секунди не потрібні, оскільки запити, як правило, носять дослідницький характер.

Однією з можливих оптимізацій є використання окремої «складської» або запитуваної схеми, вміст якої оновлюється в пакетному режимі зі схеми виробництва (транзакції). Таблиці в сховищі інтенсивно індексуються і оптимізуються з використанням денормалізації, яка об'єднує кілька таблиць в одну, щоб мінімізувати зниження продуктивності через об'єднання таблиць. Саме цей підхід використовує Kalido для перетворення сильно нормалізованих таблиць EAV в стандартні схеми звітності.

Деякі дані EAV в сховище можуть бути перетворені в стандартні таблиці з використанням «матеріалізованих уявлень», але, як правило, це останній засіб, який слід використовувати обережно, оскільки число представлень такого типу має тенденцію до нелінійного зростання з кількістю атрибутів в системі.

Структури даних в пам'яті: можна використовувати хеш-таблиці і двовимірні масиви в пам'яті разом з метаданими групування атрибутів для зведення даних, по одній групі за раз. Ці дані записуються на диск у вигляді плоского файлу з роздільниками, з внутрішніми іменами для кожного атрибута в першому рядку: цей формат може бути легко імпортовано в реляційну таблицю. Цей метод «в пам'яті» значно перевершує альтернативні підходи, зберігаючи запити до таблиць EAV настільки простими, наскільки це можливо, і зводячи до мінімуму кількість операцій введення-виведення. Кожен оператор витягує великий обсяг даних, а хеш-таблиці допомагають виконати операцію зведення, яка включає в себе розміщення значення для даного екземпляра атрибута до відповідного рядка і стовпеця. Оперативна пам'ять (ОЗУ) є досить великою і доступною на сучасному обладнанні, тому повний набір даних для однієї групи атрибутів навіть у великих наборах даних зазвичай повністю поміщається в пам'ять, хоча алгоритм можна зробити розумніше, працюючи над шматками даних.

Очевидно, що незалежно від того, які підходи використовувати, запит EAV не буде таким же швидким, як запит стандартних реляційних даних по стовпцях для певних типів запитів, багато в чому так само, як і доступ до елементів в розріджених матрицях не такий швидкий, як для розрідженої матриці, якщо останні цілком вписуються в основну пам'ять. Розріджені матриці, що будуються на зв'язаних списках, вимагають обходу списку для доступу до елемента в даній позиції XY, в той час як доступ до елементів в матрицях, представлених у вигляді двовимірних масивів, може бути виконаний з використанням швидких операцій регістра CPU.

2.1.5 EAV та Universal Data Model

Спочатку «Універсальна модель даних» (UDM), яка постульована Майєром, Уллманом і Варди, прагне спростити запит складної реляційної схеми простим користувачам, створюючи ілюзію, що все зберігається в єдиній гігантської «універсальній таблиці». Це досягається за рахунок використання взаємозв'язків між таблицями, тому користувачеві не потрібно турбуватися про те, яка таблиця містить який атрибут. CJ Date, однак, зазначив, що в обставинах, коли таблиця багаторазово пов'язана з іншою (наприклад, в генеалогічних базах даних, де батько і мати особи також є окремими особами, або в деяких бізнес-базах даних, де всі адреси зберігаються централізовано, і організація може мати різні службові адреси та адреси доставки), в схемі бази даних недостатньо метаданих, щоб вказати однозначні об'єднання. Коли UDM комерціалізується, як в SAP BusinessObjects, це обмеження обходить шляхом створення «Юніверсом», що представляють собою реляційні подання з зумовленими об'єднаннями між наборами таблиць: розробник «Юніверс» усуває неоднозначні об'єднання, включаючи багаторазово пов'язані таблиці в поданні кілька разів, використовуючи різні псевдоніми.

Крім способу, яким дані моделюються явно (UDM просто використовує реляційні подання для взаємодії між користувачем і схемою бази даних), EAV відрізняється від універсальних моделей даних тим, що він також застосовується до транзакційних систем, не тільки орієнтованим на запити (тільки для читання).) Системи як в UDM. Крім того, при використанні в якості основи для систем запитів клінічних даних реалізації EAV не обов'язково захищають користувача від необхідності вказувати клас об'єкта, що представляє інтерес. У кіоску клінічних даних i2b2 на основі EAV, наприклад, коли користувач шукає термін, у нього є можливість вказати категорію даних, в якій він зацікa-

влений. Наприклад, фраза «літій» може зверніться або до ліків (яке використовується для лікування біполярного розладу) або лабораторний аналіз рівня літію в крові пацієнта.

2.1.6 XML і JSON

Для реалізації відкритих схем можна використовувати стовпчики XML у таблиці, щоб отримати змінну / розріджену інформацію. Подібна ідея може бути застосована до баз даних, які підтримують стовпці значень JSON. Легкі ієрархічні дані можуть бути представлені як JSON. Якщо БД має підтримку JSON, таку як PostgreSQL та (частково) SQL Server 2016 або пізнішої версії, є сможливість запитувати, індексувати та об'єднувати атрибути. Хоча це може призвести до поліпшення продуктивності в більш ніж 1000 разів за допомогою простої реалізації EAV, це не обов'язково робить всю програму баз даних більш надійною.

Існує два способи зберігання даних XML або JSON. Один із способів полягає в тому, щоб зберегти його як непрозорий простий рядок на сервері баз даних. Інший спосіб полягає у використанні сервера баз даних, які можуть "бачити" структуру. Зрозуміло, що зберігання непрозорих рядків має деякі серйозні недоліки. Вони не можуть бути запитані безпосередньо, вони не можуть створювати індекс на основі їх вмісту, і не можна виконувати з'єднання на основі їх вмісту.

Створення програми, яка потребує керування даними за допомогою моделі EAV, дуже складна через інфраструктуру, яка повинна бути розроблена з точки зору таблиці метаданих та коду основної програми. Використання XML дозволить вирішити проблему перевірки даних на серверах (яка повинна бути зроблена з середнього та браузерного коду на основі EAV), але має ряд недоліків.

Хоча схему XML важко писати вручну, рекомендується створити XML-схему, визначивши реляційні таблиці, створивши схему XML-схеми та видаляючи ці таблиці. Це проблема в багатьох виробничих операціях, включаючи динамічні схеми, які потребують визначення нових атрибутів користувачами енергії, які розуміють конкретні області застосування (управління запасами, біомедичні тощо), але не обов'язково програмісти. На відміну від цього, у виробничих системах, що використовують EAV, такі користувачі визначають нові атрибути (і пов'язані з ними типи даних та перевірки) за допомогою програми GUI. Оскільки метадані, пов'язані з валідацією, повинні зберігатися в

декількох реляційних таблицях в нормованому дизайні, програма GUI, яка зв'язує ці таблиці разом і забезпечує відповідні перевірки метаданих, є єдиним практичним способом, що дозволяє вводити інформацію атрибута, навіть для розвинених розробників – навіть якщо кінцевий результат використовує XML або JSON замість окремих реляційних таблиць.

Серверна діагностика, яка виникає при використанні рішення XML/ JSON, якщо намагатися вставити неправильні дані (наприклад, перевірка діапазону або порушення правил регулярного вираження), є прихованим для кінцевого користувача: для точної передачі помилки, принаймні, потрібно пов'язувати детальну діагностику помилок з кожним атрибутом.

Рішення не розглядає проблему генерації інтерфейсу користувача.

Всі вищезгадані недоліки можуть бути виправлені шляхом створення шару метаданих та коду додатків, але при створенні цього, оригінальна "перевага" відсутності створення інфраструктури втрачається. Справа в тому, що ефективне моделювання розріджених атрибутів даних представляє складну проблему проектування застосувань БД незалежно від того, який підхід до сховища використовується. Робота Сарки, однак, доводить доцільність використання поля XML замість реляційних таблиць EAV для сховища даних, а в ситуаціях, коли кількість атрибутів на об'єкт є смалою (наприклад, змінні атрибути продукту для різних типів продукту) рішення на основі XML є більш компактним, ніж EAV-таблиця. Сам XML може розглядатися як засіб представлення даних атрибутивного значення, хоча він базується на структурованому тексті, а не на реляційних таблицях.

2.1.7 EAV та хмарні обчислення

Багато постачальників хмарних обчислень пропонують сховища даних на основі моделі EAV, де з даним об'єктом може бути пов'язано довільна кількість атрибутів. Роджер Дженнінгс надає їх докладний порівняння. У пропозиції Amazon, SimpleDB, тип даних обмежений рядками і дані, які по своїй суті не є рядковими, повинні бути приведені до рядка (наприклад, числа повинні бути доповнені початковими нулями), якщо треба виконувати такі операції, як сортування. Пропозиція Microsoft, Windows Azure Table Storage, пропонує обмежений набір типів даних: byte [], bool, DateTime, double, Guid, int, long і string. Google App Engine пропонує найбільшу різноманітність типів даних: крім поділу числових даних на int, long або float, він також визначає призначені для користувача типи даних, такі як номер телефону, адреса електронної

пошти, ГеоКод і гіперпосилання. Google, але не Amazon або Microsoft, дозволяє визначати метадані, які перешкоджатимуть зв'язуванню неприпустимих атрибутів з певним класом сутностей, дозволяючи створити модель метаданих.

Google дозволяє також працювати з даними, використовуючи підмножину SQL; Microsoft пропонує синтаксис запитів на основі URL, який абстрагується через постачальника LINQ; Amazon пропонує більш обмежений синтаксис. Викликає занепокоєння те, що вбудована підтримка об'єднання різних об'єктів за допомогою об'єднань в даний час (квітень '10) відсутня у всіх трьох двигунах. Такі операції повинні виконуватися кодом програми. Це може не турбувати, якщо сервери додатків розташовані спільно з серверами даних в центрі обробки даних постачальника, але якщо вони будуть географічно розділені, то буде генеруватися великий обсяг мережевого трафіку.

Підхід EAV виправданий тільки в тому випадку, якщо модельовані атрибути численні і рідкісні: якщо зібрані дані не відповідають цій вимозі, підхід EAV за замовчуванням, пропонований хмарними постачальниками, часто не відповідає програмам, яким потрібно справжня внутрішня база даних. Модернізація переважної більшості існуючих додатків баз даних, що використовують традиційний підхід до моделювання даних, до хмарної архітектурі типу EAV, потребує серйозної операції. Наприклад, Microsoft знайшла, що її база розробників додатків баз даних значною мірою неохоче вкладає такі зусилля. Тому недавно Microsoft представила преміум-пропозицію – повнофункціональний реляційний движок SQL Server Azure, доступний в хмарі, який дозволяє переносити існуючі програми баз даних з малими змінами.

Одним з обмежень SQL Azure є те, що за станом на січень 2015 року розмір фізичних баз даних обмежений 500 ГБ. Microsoft рекомендує розбивати набори даних більшого розміру на кілька фізичних баз даних та отримувати до них доступ за допомогою паралельних запитів.

EAV, як засіб представлення знань загального призначення, виникла з поняттям «списки асоціацій» (пари атрибут-значення). Вони були вперше представлені на мові LISP. Пари атрибут-значення широко використовуються для різних додатків, таких як файли конфігурації (з використанням простого синтаксису, наприклад атрибут = значення). Прикладом використання EAV без бази даних є UIMA (Uniform Information Management Architecture), стандарт, яким в даний час управляє Apache Foundation і який використовується в таких областях, як обробка природної мови., Програмне забезпечення, яке аналізує текст, зазвичай позначає («анотує») сегмент: приклад, наведений в

керівництві UIMA, – це програма, яка виконує розпізнавання іменованих сутностей (NER) для документа, позначаючи текстовий сегмент «Президент Буш» аннотацією – трійка атрибута-значення (Person, Full_Name, "Джордж Буш"). Такі анотації можуть зберігатися в таблиці бази даних.

Хоча EAV не має прямого з'єднання з AV-парами, Stead і Hammond, мабуть, першими задумалися про їх використання для постійного зберігання довільно складних даних. Першими системами медичної документації, які використовували EAV, були електронна медична карта Regenstrief (робота Клемента Макдональда), система TMR (The Medical Record) Вільяма Стед і Еда Хаммонда і HELP Clinical Data Repository (CDR), створений групою Гомера Уорнера в лікарні СПД, Солт-Лейк-Сіті, штат Юта. [32] [33] Система Regenstrief фактично використовувала схему Patient-Attribute-Timestamp-Value: використання тимчасової мітки підтримувало отримання значень для даного пацієнта / атрибута в хронологічному порядку. Всі ці системи, розроблені в 1970-х роках, були випущені до появи комерційних систем на основі EF реляційної бази даних моделі Кодда були доступні, хоча HELP був набагато пізніше перенесений на реляційну архітектуру і комерційною корпорацією 3M.

Група в Колумбійському-пресвітеріанської Медичному Центрі була першою, хто використовував механізм реляційної бази даних в якості основи системи EAV.

Система управління даними клінічних досліджень TrialDB з відкритим вихідним кодом Nadkarni et al. був першим, хто використав кілька таблиць EAV, по одній для кожного типу даних СКБД.

Каркас EAV / CR, розроблений в основному Луїсом Маренко і Пракашем Надкарні, наклав принципи орієнтації об'єкта на EAV; він побудований на підході таблиці об'єктів Тома Слезака. SenseLab, загальнодоступна база даних по нейронауці, побудована на основі EAV / CR. Крім того, існує безліч комерційних додатків, які використовують аспекти EAV всередині, включаючи Oracle Designer (застосовується для моделювання ER), Kalido (застосовується для зберігання даних і управління даними майстра) і Lazysoft Вироки (стосовно розробки призначеного для користувача програмного забезпечення). Система EAV, яка знаходиться не над табличною структурою, а безпосередньо над B-деревомето InfinityDB, що усуває необхідність в одній таблиці для кожного типу даних значення.

2.2 Огляд візуального редактору WYSIWYG

WYSIWYG – акронім від What You See Is What You Get (англ. що бачиш, те й отримуєш). Застосовується до комп'ютерних програм, які надають можливість користувачу бачити щось дуже подібне до кінцевого результату під час створення документів або зображень. Наприклад, користувач може бачити на екрані, як виглядатиме документ, видрукований на папері або відображений у веб-оглядачі [6].

У презентаційних програмах, складних документах та веб-сторінках WYSIWYG означає, що дисплей точно відображає зовнішній вигляд сторінки, відображеної кінцевому користувачеві, але не обов'язково відображає, як буде надруковано сторінку, якщо тільки принтер не буде спеціально підігнаний до програми редагування , як це було з Xerox Star і ранні версії Apple Macintosh.

У програмах для обробки текстів та настільних публікацій WYSIWYG означає, що на екрані імітується зовнішній вигляд і відображається ефект шрифтів та розривів рядків у кінцевих сторінках, використовуючи певну конфігурацію принтера.

WYSIWYG також описує способи маніпулювати 3D-моделями у стереохімії, комп'ютерному проекті та 3D комп'ютерній графіці.

Сучасне програмне забезпечення робить хорошу роботу щодо оптимізації екрана для певного виду продукції. Наприклад, текстовий процесор оптимізований для виведення на типовий принтер. Програмне забезпечення часто імітує роздільну здатність принтера, щоб максимально наблизитись до WYSIWYG. Проте це не головна привабливість WYSIWYG, це здатність користувача візуалізувати те, що вони виробляють.

У багатьох ситуаціях незначні відмінності між тим, що бачить користувач і що отримує користувач, є неважливими. Фактично, додатки можуть пропонувати кілька режимів WYSIWYG з різними рівнями "реалізму", в тому числі.

Режим композиції, в якому користувач бачить дещо збігається з кінцевим результатом, але має додаткову корисну інформацію під час створення, таку як переривання розділів та символи, що не друкують, і використовує макет, що більш сприятливий для складання, ніж для макета.

Режим макета, в якому користувач бачить щось дуже схоже на кінцевий результат, але з деякою додатковою інформацією, корисною для забезпечення того, щоб елементи були правильно вирівняні та розташовані на відстані, наприклад, поля маржи.

Режим попереднього перегляду, в якому додаток намагається представити представлення, яке є максимально наближеним до кінцевого результату.

Перед використанням функціоналу WYSIWYG текст з'являвся в редакторі, використовуючи системний стандартний шрифт і стиль з невеликим вказівкою макета (поля, відстань тощо). Користувачам потрібно було вводити спеціальні коди, які не друкують (тепер називають марками коду розмітки), щоб вказати, що текст повинен бути накреслений жирним шрифтом, курсивом або іншим шрифтом або розміром. У цьому середовищі дуже мало відмінностей між текстовими редакторами та текстовими редакторами.

Ці програми зазвичай використовують довільну мову розмітки для визначення кодів / тегів. Кожна програма мала свій особливий спосіб форматування документа, і це було складним і трудомістким процесом для переходу від одного текстового процесора до іншого.

Використання тегів та кодів розмітки залишається популярним сьогодні в деяких програмах завдяки їхній здатності зберігати складну інформацію форматування. Однак, коли теги стають видимими в редакторі, вони займають простір у форматі неформатованого тексту і, таким чином, порушують бажаний макет і потік.

Bravo, програма підготовки документів для Alto, вироблена компанією Xerox PARC Батлером Ламспоном, Чарльз Сімоні та її колегами в 1974 році, як правило, вважається першою програмою для інтеграції технології WYSIWYG , яка відображає текст з форматуванням (наприклад, з обґрунтуванням, шрифтами та пропорційний інтервал символів) Монітор Alto (72 пікселів, заснований на типографічній одиниці) був розроблений таким чином, щоб можна було побачити одну повну сторінку тексту, а потім роздрукувати на перших лазерних принтерах. Коли текст було викладено на екрані, було використано 72 метричних файлів PPI шрифту, але при друкуванні 300 файлів PPI – таким чином іноді можна було б знайти символи та слова трохи, проблема, яка триває і донині.

Bravo був випущений комерційно, і програмне забезпечення, яке в кінцевому підсумку було включено в Xerox Star, можна розглядати як безпосередній нащадок.

Паралельно, але незалежно від роботи в Xerox PARC, Hewlett Packard розробляв і випустив в кінці 1978 року першу комерційну програму WYSIWYG для створення накладних слайдів або те, що сьогодні називають презентаційною графікою. Перший випуск, названий BRUNO (після кулькової манекені з продажів HP), запустив міні-комп'ютер HP 1000, скориставшись першим растровим комп'ютерним терміналом HP HP 2640. BRUNO потім був перенесений на HP-3000 і знову вийшов як "HP Draw" ".

До 1981 року MicroPro оголосив про те, що його текстовий редактор WordStar мав WYSIWYG , але його екран обмежувався показом стилізованого тексту в WYSIWYG-моді; Жирний і курсивний текст буде представлений на екрані, а не оточений тегами або спеціальними контрольними символами . У 1983 році Weekly Reader оголосила про своє освітнє програмне забезпечення Stickybear з гаслом "що ви бачите, це те, що ви отримуєте", з фотографіями своєї графіки Apple II , але домашні комп'ютери 1970-х – початку 1980-х років не мали складних графічних можливостей, необхідних для відображати документи WYSIWYG, що означає, що такі програми, як правило, обмежуються обмеженими цільовими, високотехнологічними робочими станціями (такими як IBM Displaywriter System), які були занадто дорогими для широкої громадськості. Однак до середини 1980-х років речі почали змінюватися. Удосконалення технологій дозволило виробляти дешеві дисплеї з растровими зображеннями, а програма WYSIWYG почала з'являтися на більш популярних комп'ютерах, включаючи LisaWrite для Apple Lisa, випущеної в 1983 році, і MacWrite для Apple Macintosh, випущеної в 1984 році.

Система Apple Macintosh була спочатку розроблена таким чином, щоб роздільна здатність екрана та роздільна здатність матричних принтерів ImageWriter, що продаються Apple, були легко масштабовані: 72 пікселів для екрану та 144 дюймів для принтерів. Таким чином, масштаб і розміри екранного відображення в програмах, таких як MacWrite та MacPaint, були легко переведені на друковане видання - якщо папір був затриманий до екрана, друковане зображення буде таким же, як і зображення на екрані , але в два рази дозвіл. Оскільки ImageWriter була єдиною моделлю принтерів, фізично сумісною з портом принтера Macintosh, це створило ефективну закриту систему. Пізніше, коли Macs, що використовують зовнішні дисплеї стали доступними, роздільна здатність була встановлена на розмір екрану, щоб досягти 72 DPI. Ці резолюції часто відрізнялися від стандартів VGA, прийнятих в той час у загальному світі. Таким чином, хоча монітор з 15-дюймовим (38 см) монітором Macintosh мав такий же роздільна здатність 640 × 480 як ПК, 16-дюймовий (41 см) екран був би фіксований на 832 × 624, а не на дозволі 800 × 600, що використовується комп'ютерами. Завдяки впровадженню сторонніх матричних принтерів, лазерних принтерів та мультисинхронних моніторів, зображення відрізняються навіть від кратних роздільних здатностей екрана, що робить справжню WYSIWYG важливішою.

3 ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА РЕЗУЛЬТАТІВ

3.1 Опис використаних технологій

3.1.1 Мова розмітки гіпертекстових документів HTML

HTML (HyperText Markup Language – мова розмітки гіпертекстових документів) – стандартна мова розмітки веб-сторінок в Інтернеті. Більшість веб-сторінок створюються за допомогою мови HTML (або XHTML) . Документ HTML оброблюється браузером та відтворюється на екрані у звичному для людини вигляді [7].

HTML є похідною мовою від SGML, успадкувавши від неї визначення типу документу та ідеологію структурної розмітки тексту. HTML разом із каскадними таблицями стилів та вбудованими скриптами – це три основні технології побудови веб-сторінок. HTML впроваджує засоби для:

− створення структурованого документу шляхом позначення структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати та інше;

− отримання інформації із Всесвітньої мережі через гіперпосилання;

− створення інтерактивних форм;

− включення зображень, звуку, відео та інших об'єктів до тексту.

Документ HTML. Для поліпшення взаємодії, SGML вимагає аби кожна похідна мова (HTML у тому числі) визначала свою кодову таблицю для кожного документа, яка складається з репертуара (перелік різноманітних символів) та позиції символа (перелік цифрових посилань на символи з репертуара). Кожен документ HTML – це послідовність символів з репертуара.

HTML використовує найповнішу кодову таблицю UCS (Universal Character Set – Універсальний Набір Символів).

Проте, однієї кодової таблиці недостатньо для того, щоб браузери могли правильно відтворювати документи HTML. Для цього браузерам потрібно «знати» специфічну кодову таблицю документа, яку автор має зазначати завжди в елементі meta із параметром charset. За замовчуванням використовується кодова таблиця ISO-8859-1, відома також як Latin-1.

Розмітка в HTML складається з чотирьох основних компонентів: елементів (та їхніх атрибутів), базових типів даних, символьних мнемонік та декларації типу документа.

Загальна структура. Документ HTML 4.01 складається з трьох частин:

1) декларація типу документу (Document type declaration, Doctype), на початку документа, в якій визначається тип документа (DTD);

2) шапка документу (знаходиться в межах елементу head), в якій записані загальні технічні відомості або додаткова інформація про документ, яка не відтворюється безпосередньо в браузері;

3) тіло документу (може знаходитися в елементах body або frameset), в якому міститься основна інформація документа.

Елементи. Елементи являють собою базові компоненти розмітки HTML. Кожен елемент має дві основні властивості: атрибути та зміст (контент). Існують певні настанови щодо кожного атрибута та контента елемента, які треба виконувати задля того, щоб HTML-документ був визнаний валідним.

У елемента є початковий тег, який має вигляд <element-name>, та кінцевий тег, який має вигляд </element-name>. Атрибути елемента записуються в початковому тегу одразу після назви елемента, контент елемента записується між його двома тегами. Наприклад: <element-name element-attribute=”attribute-value”>контент елемента</element-name>.

Деякі елементи, наприклад br, не містять контенту, тож і не мають кінцевого тега. Елемент може не мати початкового та кінцевого тега (наприклад, елемент head), проте він завжди буде представлений в документі. Нижче зазначені деякі типи елементів розмітки HTML.

Елементи структурної розмітки застосовуються задля опису семантики тексту, іншими словами ці елементи описують призначення тексту свого контенту. Вони не зазначають ніякого спеціального (візуального) відтворення тексту, проте більшість браузерів мають стандартні стилі форматування для кожного елемента. Для подальшого стилізування тексту рекомендується використовувати каскадні таблиці стилів (CSS).

Елементи розмітки гіпертексту застосовуються задля з'єднання частин документу з іншими документами.

Атрибути. Більшість з атрибутів елементу являє собою пару «назва-значення», розділених між собою знаком рівняння, та записаних у початковому тегу одразу після назви елемента. Значення атрибуту може бути окреслено лапками (подвійними або одиничними), також, якщо значення атрибуту складається з певних символів, його можна не виділяти лапками зліва. Проте, не виділення значення атрибутів в лапки вважається небезпечним кодом. На відміну від атрибутів виду «назва-значення», є певні атрибути, що впливають на елемент, назва яких лише з'явилась в початковому тегу (наприклад, атрибут

ismap елементу img). Більшість елементів можуть мати будь-який з загальних атрибутів:

Атрибут id впроваджує унікальний ідентифікатор елементу по всьому документу. Доданий до URL документу, він впроваджує глобальний унікальний ідентифікатор елементу.Це може використовуватися:

- таблицями стилів для впровадження презентаційних властивостей;
- браузерами для фокусування уваги на певному елементі;
- скриптами для виконання дій над елементом.

Атрибут title використовується для додавання пояснювального тексту для елементу. В більшості браузерів значення цього атрибуту можна побачити як виникаючу підказку, при наведені курсора на елемент.

Атрибут class впроваджує засіб об'єднання схожих елементів у класи.

Це може використовуватися для відтворення візуальних ефектів. Для презентаційної розмітки, наприклад, документ може містити class=»notation», який визначає всі елементи, у яких клас визначений як «notation», підпорядкованими головному тексту документу. Такі елементи можна зібрати докупи і показати як виноски внизу сторінки, замість того, щоб показувати їх на тому місці, де вони з'являються в самому HTML коді документу;

Структурного поділу тексту. Для семантичної розмітки, наприклад, класи використовуються у створенні мікроформатів.

Базові типи даних. Оскільки HTML є похідною мовою від SGML, усі типи даних HTML ґрунтуються на базових типах даних SGML (наприклад, PCDATA, CDATA, NAME, ID, NUMBER).

Кожен елемент має дві властивості – атрибути і вміст, які мають певні значення. Всі можливі значення цих двох властивостей прописуються відповідно до визначених у DTD типів даних. Нижче наведено кілька типів даних HTML:

- % Color – колір sRGB, записаний у шістнадцятковому вигляді;
- % ContentType – тип умісту/носія;
- % Charset – таблиця кодування символів;
- % Character – мнемоніка або окремий символ із UCS;
- % Length – nn розмір в пікселях, nn% – у відсотках;
- % URI – Універсальний ідентифікатор ресурсу;
- % Datetime – дата та час;
- % Script – скрипт;
- % StyleSheet – дані таблиць стилів;
- % Text – текстові рядки.

Мнемоніки. Існують такі випадки, коли в документі потрібно використати якийсь символ, якого немає в обраній для документу кодовій таблиці. Для таких випадків можливо замінити символ на еквівалентне йому SGML-посилання на символ (мнемоніку).

Перегляд. Для перегляду HTML-розмітки документа можна використовувати будь-який текстовий редактор. Для перегляду документу, відтвореного за правилами HTML-розмітки, використовується браузер.

Транспортування в мережі. HTML документи можуть бути транспортовані так само як і будь-які інші файли (наприклад, за допомогою протоколів FTP, TCP), проте, зазвичай вони транспортуються із сервера за допомогою протоколу HTTP, або по електронній пошті.

3.1.2 Формальна мова опису зовнішнього вигляду документу CSS

CSS використовується авторами та відвідувачами веб-сторінок, щоб визначити кольори, шрифти, верстку та інші аспекти вигляду сторінки. Одна з головних переваг – можливість розділити зміст сторінки (або контент, наповнення, зазвичай HTML, XML або подібна мова розмітки) від вигляду документу (що описується в CSS) [8].

Таке розділення може покращити сприйняття та доступність контенту, забезпечити більшу гнучкість та контроль за відображенням контенту в різних умовах, зробити контент більш структурованим та простим, прибрати повтори тощо. CSS також дозволя адаптувати контент до різних умов відображення (на екрані монітора, мобільного пристрою (КПК), у роздрукованому вигляді, на екрані телевізора, пристроях з підтримкою шрифту Брайля або голосових браузерах та ін.)

Один і той самий HTML або XML документ може бути відображений по-різному залежно від використаного CSS.

Стандарт CSS визначає порядок та діапазон застосування стилів, те, в якій послідовності і для яких елементів застосовуються стилі. Таким чином, використовується принцип каскадності, коли для елементів вказується лише та інформація про стилі, що змінилася або не визначена загальнішими стилями.

Переваги:

− інформація про стиль для усього сайту або його частин може міститися в одному .css-файлі;

− різна інформація про стилі для різних типів користувачів: наприклад, великий розмір шрифту для користувачів з послабленим зором, стилі для виводу сторінки на принтер, стиль для мобільних пристроїв;

− сторінки зменшуються в об'ємі та стають більш структурованими, оскільки інформація про стилі відділена від тексту та має певні правила застосування і сторінка побудована з урахуванням їх;

− прискорення завантаження сторінок і зменшення обсягів інформації, що передається, навантаження на сервер та канал передачі. Досягається за рахунок того, що сучасні браузери здатні кешувати (запам'ятовувати) інформацію про стилі і використовувати для всіх сторінок, а не завантажувати для кожної.

− CSS має порівняно простий синтаксис і використовує небагато англійських слів для найменування різних складових стилю.

Стилі складаються зі списку правил. Кожне правило має один або більше селектор (Selector) та блок визначення (Declaration block). Блок визначення складається із оточеного фігурними дужками списку властивостей.

Властивості в списку оформлюються у вигляді назва властивості, двокрапка (значення, крапка з комою (;)).

3.1.3 Прототипно-орієнтована сценарна мова програмування JavaScript

JavaScript (JS) – динамічна, об’єктно-орієнтована мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується як частина браузера, що надає можливість коду на стороні клієнта (такому, що виконується на пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки. Мова JavaScript також використовується для програмування на стороні серверу (подібно до таких мов програмування, як Java і C#), розробки ігор, стаціонарних та мобільних додатків, сценаріїв в прикладному ПЗ (наприклад, в програмах зі складу Adobe Creative Suite), всередині PDF-документів тощо [9].

JavaScript класифікують як прототипну (підмножина об’єктно-орієнтованої), мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам’яттю, наслідування, функції як об’єкти першого класу.

JavaScript є однією з найпопулярніших мов програмування в інтернеті. Але спочатку багато професіональних програмістів скептично ставилися до мови, цільова аудиторія якої складалася з програмістів-любителів. Поява AJAX змінила ситуацію та повернула увагу професійної спільноти до мови. В результаті, були розроблені та покращені багато практик використання JavaScript (зокрема, тестування та налагодження), створені бібліотеки та фреймворки, поширилося використання JavaScript поза браузером.

JavaScript має низку властивостей об'єктно-орієнтованої мови, але завдяки концепції прототипів підтримка об'єктів в ній відрізняється від традиційних мов ООП. Крім того, JavaScript має ряд властивостей, притаманних функціональним мовам,– функції як об'єкти першого класу, об'єкти як списки, каррінг (currying), анонімні функції, замикання (closures) – що додає мові додаткову гнучкість.

JavaScript має С-подібний синтаксис, але в порівнянні з мовою Сі має такі корінні відмінності:

- об'єкти, з можливістю інтроспекції і динамічної зміни типу через механізм прототипів;
- функції як об'єкти першого классу;
- обробка винятків;
- автоматичне приведення типів;
- автоматичнее прибирання сміття;
- анонімні функції.

JavaScript містить декілька вбудованих об'єктів: Global, Object, Error, Function, Array, String, Boolean, Number, Math, Date, RegExp. Крім того, JavaScript містить набір вбудованих операцій, які, строго кажучи, не обов'язково є функціями або методами, а також набір вбудованих операторів, що управляють логікою виконання програм. Синтаксис JavaScript в основному відповідає синтаксису мови Java (тобто, зрештою, успадкований від С), але спрощений порівняно з ним, щоб зробити мову сценаріїв легкою для вивчення. Так, приміром, декларація змінної не містить її типу, властивості також не мають типів, а декларація функції може стояти в тексті програми після неї.

При використанні в рамках технології DHTML JavaScript код включається в HTML-код сторінки і виконується інтерпретатором, вбудованим в браузер. Код JavaScript вставляється в теги <script></script> з обов'язковим по специфікації HTML 4.01 атрибутом type=»text/javascript», хоча в більшості браузерів мова сценаріїв за умовчанням саме JavaScript.

Слідуючи концепції інтеграції JavaScript в існуючі системи, браузери підтримують включення скрипта, наприклад, в значення атрибуту події:

<a href=”delete.php”

onclick=”return confirm(‘Ви впевнені?’);”>Видалити

Тут при натисненні на посилання функція confirm(‘Ви впевнені?’); викликає модальне вікно з написом «Ви впевнені?», a return false; блокує перехід за посиланням. Зрозуміло, цей код працюватиме тільки якщо в браузері є і включена підтримка JavaScript, інакше перехід за посиланням відбудеться без попередження.

Є і третя можливість підключення JavaScript – написати скрипт в окремому файлі, а по тому підключити його за допомогою конструкції:

<scripttype=”text/javascript”

src=”http://Шлях_до_файла_зі_скриптом”></script>

3.1.4 Бібліотека jQuery

jQuery – популярна JavaScript-бібліотека з відкритим сирцевим кодом. Вона була представлена у січні 2006 року у BarCamp NYC Джоном Ресігом (John Resig). Згідно з дослідженнями організації W3Techs, Jquery використовується понад половиною від мільйона найвідвідуваніших сайтів. jQuery є найпопулярнішою бібліотекою JavaScript, яка посилено використовується на сьогоднішній день [10].

jQuery є вільним програмним забезпеченням під ліцензією MIT (до вересня 2012 було подвійне ліцензування під MIT та GNU General Public License другої версії).

Синтаксис jQuery розроблений, щоб зробити орієнтування у навігації зручнішим завдяки вибору елементів DOM, створенню анімації, обробки подій, і розробки AJAX-застосунків. jQuery також надає можливості для розробників, для створення плагінів у верхній частині бібліотеки JavaScript. Використовуючи ці об’єкти, розробники можуть створювати абстракції для низькорівневої взаємодії та створювати анімацію для ефектів високого рівня. Це сприяє створенню потужних і динамічних веб-сторінок.

Основне завдання jQuery – це надавати розробнику легкий та гнучкий інструментарій кросбраузерної адресації DOM об’єктів за допомогою CSS та Xpath селекторів. Також даний фреймворк надає інтерфейси для Ajax-застосунків, обробників подій і простої анімації.

Принцип роботи jQuery полягає в використанні класу (функції), який при звертанні до нього повертає сам себе. Таким чином, це дозволяє будувати послідовний ланцюг методів.

```
$('#test') //знаходимо елемент з id=»test»
.text('Клікни по мені')  //встановлюємо текст елемента рівним «Клікни по мені»
.addClass(‘myAlert’)     //додаємо клас “myAlert”
.css('color','red')      //встановлюємо колір тексту червоним
.attr('alert','Привіт, світе!') // додаємо атрибут «alert» із значенням «Привіт, світе!»
.bind(    // додаємо в обробник події click функцію, яка відкриє модальне
 'click', // вікно із текстом, що вказаний в атрибуті «alert» («Привіт, світе!»)
function(){alert($(this).attr(‘alert’))}
   );
```

Бібліотека jQuery є JavaScript файлом, яка включає всю його DOM, події(events), ефекти (effects), і Ajax функції. Вона може бути додана до web-сторінки посиланням на локальну копію, або на одну з копій доступних на публічному сервері (наприклад Google або Microsoft CDN).

```
<script type=”text/javascript” src=”jquery.js”></script>
```

3.1.5 Скриптова мова програмування PHP

PHP (Hypertext Preprocessor, гіпертекстовий препроцесор), попередня назва: Personal Home Page Tools –мова програмування, була створена для генерації HTML-сторінок на стороні веб-сервера. PHP є однією з найпоширеніших мов, що використовуються у сфері веб-розробок (разом із Java, .NET, Perl, Python, Ruby). PHP підтримується переважною більшістю хостинг-провайдерів. PHP – проект відкритого програмного забезпечення [11].

PHP інтерпретується веб-сервером у HTML-код, який передається на сторону клієнта. На відміну від мови JavaScript, користувач не бачить PHP-коду, бо браузер отримує готовий html-код. Це є перевага з точки зору безпеки, але погіршує інтерактивність сторінок. Але ніщо не забороняє використовувати PHP для генерування і JavaScript-кодів які виконуються вже на стороні клієнта.

PHP – мова, код якої можна вбудовувати безпосередньо в html-код сторінок, які, у свою чергу, будуть коректно оброблені PHP-інтерпретатором. Обробник PHP просто починає виконувати код після відкриваючого тегу

(<?php) і продовжує виконання до того моменту, поки не зустріне закриваючий тег (?>).

Велика різноманітність функцій PHP дає можливість уникати написання багаторядкових функцій, призначених для користувача, як це відбувається в C або Pascal.

В PHP вбудовані бібліотеки для роботи з MySQL, PostgreSQL, mSQL, Oracle, dbm, Hyperware, Informix, InterBase, Sybase. Через стандарт відкритого інтерфейсу зв'язку з базами даних (Open Database Connectivity Standard – ODBC) можна підключатися до всіх баз даних, до яких існує драйвер.

Мова PHP здаватиметься знайомою програмістам, що працюють в різних областях. Багато конструкцій мови запозичені з C, Perl. Код PHP дуже схожий на той, який зустрічається в типових програмах на C або Pascal. Це помітно знижує початкові зусилля при вивченні PHP. PHP – мова, що поєднує переваги Perl і C і спеціально спрямована на роботу в Інтернеті, мова з універсальним і зрозумілим синтаксисом. І хоча PHP є досить молодою мовою, вона здобула таку популярність серед web-програмістів, що в наш час є мало не найпопулярнішою мовою для створення веб-застосувань (скриптів).

Стратегія Open Source, і розповсюдження початкових текстів програм в масах, безсумнівно справили благотворний вплив на багато проектів, в першу чергу – Linux хоч і успіх проекту Apache сильно підкріпив позиції прихильників Open Source. Сказане відноситься і до історії створення PHP, оскільки підтримка користувачів зі всього світу виявилася дуже важливим чинником в розвитку проекту PHP.

Ухвалення стратегії Open Source і безплатне розповсюдження початкових текстів PHP надало неоціненну послугу користувачам. Окрім цього, користувачі PHP в усьому світі є свого роду колективною службою підтримки, і в популярних електронних конференціях можна знайти відповіді навіть на найскладніші питання.

Ефективність є дуже важливим чинником у програмуванні для середовищ розрахованих на багато користувачів, до яких належить і web. Важливою перевагою PHP є те, що ця мова належить до інтерпретованих. Це дозволяє обробляти сценарії з достатньо високою швидкістю. За деякими оцінками, більшість PHP-сценаріїв (особливо не дуже великих розмірів) обробляються швидше за аналогічні їм програми, написані на Perl. Проте хоч би що робили розробники PHP, виконавчі файли, отримані за допомогою компіляції, працюватимуть значно швидше – в десятки, а іноді і в сотні разів. Але продуктивність PHP достатня для створення цілком серйозних веб-застосувань.

PHP 5 володіє прекрасним потенціалом реалізації об'єктного програмування. Окрім цього, PHP збагатився рядом цінних розширень для роботи з XML, різними джерелами даних, генерації графіки і інше. Серед інших украй корисних доповнень в PHP 5 слід зазначити нову схему обробки виключень. Конструкція try/catch/throw дозволяє весь код обробки помилок локалізувати в одному місці сценарію.

У PHP 5 також включені два нових модулі для роботи з протоколами SimpleXML і SOAP. SimpleXML дозволяє значно спростити роботу з XML-даними, представляючи вміст XML-документа у вигляді PHP-об'єкта. Розширення SOAP дозволяє будувати на PHP сценарії, що обмінюються інформацією з іншими застосуваннями за допомогою XML-повідомлень поверх існуючих веб-протоколів, наприклад, HTTP. Модуль для роботи з SOAP для PHP 5 надає розробникам засіб для достатньо швидкого створення ефективних SOAP-клієнтів і SOAP-серверів.

Новий модуль PHP 5 MySQLi (MySQL Improved) призначений для роботи з MySQL-сервером версій 4.1.2 і вище, реалізовуючи не тільки процедурний, але і об'єктно-орієнтований інтерфейс до MySQL. Додаткові можливості цього модуля включають – SSL,контроль транзакцій, підтримка реплікації та ін. Очевидно, що, на цьому історія PHP не закінчується. Слід очікувати наступних версій мови із розширеними можливостями.

Мова явно підтримує http cookies відповідно до специфікацій Netscape. Це дозволяє проводити встановлення та читання невеликих сегментів даних на стороні клієнта.

PHP надає можливість організації роботи з користувачем протягом сеансів (сесій). В сесії можна зберігати різні дані, включаючи об'єкти.

3.1.6 Мова структурованих запитів SQL

SQL (Structured query language – мова структурованих запитів) – декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних і її модифікації, системи контролю за доступом до бази даних. Сам по собі SQL не є ні системою керування базами даних, ні окремим програмним продуктом [12].

SQL – це діалогова мова програмування для здійснення запиту і внесення змін до бази даних, а також управління базами даних. Багато баз даних підтримує SQL з розширеннями до стандартної мови. Ядро SQL формує командна мова, яка дозволяє здійснювати пошук, вставку, оновлення, і вилучення

даних, використовуючи систему управління і адміністративні функції. SQL також включає CLI (Call Level Interface) для доступу і управління базами даних дистанційно.

SQL складається з наступних елементів мови:

− положення, які є складовими компонентами заяв і запитів;

− предикати для тризначної логіки (3VL) (так / ні / невідомо);

− запити для видачі скалярних величин або таблиць, що складаються із стовпців і рядків даних;

− запити вилучення даних на основі певних критеріїв;

− заяви впливу на схеми і даних, зв'язків;

− заяви управлінням транзакціями, ходом виконання програми.

Структура SQL складається з:

− DDL (Data Definition Language) – робота із структурою бази;

− DML (Data Manipulation Language) – робота із стрічками;

− DCL (Data Control Language) – робота з правами;

− TCL (Transaction Control Language) – робота з транзакціями.

Data Definition Language:

− CREATE – створення об'єкта (наприклад, таблиці);

− ALTER – зміна об'єкта (наприклад, додавання/зміна полів таблиці);

− DROP – видалення об'єкта.

Data Manipulation Language:

− INSERT – вставка стрічки;

− SELECT – вибірка;

− UPDATE – зміна;

− DELETE – видалення.

Data Control Language:

− GRANT – надання прав користувачу;

− DENY – явна заборона для користувача;

− REVOKE – відміна заборони/дозволу користувачу.

Transaction Control Language:

− BEGIN TRANSACTION – почати транзакцію;

− COMMIT – прийняти зміни прийняті в транзакції;

− ROLLBACK – відкат.

Оператор Опис Приклад

− = Рівно Author = 'Alcott'

− <> or != Не рівно Dept <> 'Sales'

− > Більше Hire_Date > '2012-01-31'

- < Менше Bonus < 50000.00
- >= Більше рівно Dependents >= 2
- <= Менше рівно Rate <= 0.05

Незалежність від конкретної СКБД. Незважаючи на наявність діалектів і відмінностей в синтаксисі, в більшості своїй тексти SQL-запитів, що містять, DDL і DML, можуть бути досить легко перенесені з однієї СКБД в іншу. Існують системи, розробники яких спочатку закладалися на застосування щонайменше кількох СКБД (наприклад, система електронного документообігу Documentum може працювати як з Oracle, так і з Microsoft SQL Server та IBM DB2). Природно, що при застосуванні деяких специфічних для реалізації можливостей такої переносимості добитися вже дуже важко.

Наявність стандартів. Наявність стандартів і набору тестів для виявлення сумісності і відповідності конкретній реалізації SQL загальноприйнятому стандарту тільки сприяє «стабілізації» мови. Правда, варто звернути увагу, що сам по собі стандарт місцями занадто формалізований і роздутий в розмірах, наприклад, Core-частину стандарту SQL:2003 включає понад 1300 сторінок тексту.

Декларативність. За допомогою SQL програміст описує тільки те, які дані потрібно витягнути або модифікувати. Те, яким чином це зробити, вирішує СКБД безпосередньо при обробці SQL-запиту. Проте не варто думати, що це повністю універсальний принцип – програміст описує набір даних для вибірки або модифікації, проте йому при цьому корисно уявляти, як СКБД розбиратиме текст його запиту. Особливо критичні такі моменти стають при роботі з великими базами даних і зі складними запитами – чим складніше сконструйований запит, тим більше він допускає варіантів написання, різних за швидкістю виконання, але тих самих за набором даних.

До недоліків SQL слід віднести наступне:

Невідповідність реляційної моделі даних. Творці реляційної моделі даних Едгар Кодд, Крістофер Дейт та їхні прихильники указують на те, що SQL не є істинно реляційною мовою. Зокрема вони указують на такі проблеми SQL:

- рядки, що повторюються;
- невизначені значення (null);
- явна вказівка порядку стовпчиків зліва направо;
- стовпці без імені та імена стовпчиків, що дублюються;
- відсутність підтримки властивості «=»;
- віикористання покажчиків.

Складність. Хоча SQL і замислювався, як засіб роботи кінцевого користувача, врешті-решт він став настільки складним, що перетворився на інструмент програміста.

Відступи від стандартів. Незважаючи на наявність міжнародного стандарту ANSI SQL-92, багато компаній, СКБД (наприклад, Oracle, Sybase, Microsoft, MySQL), що займаються розробкою, вносять зміни до мови SQL, вживаної в розроблених ними СКБД, тим самим відступаючи від стандарту. Таким чином з'являються специфічні для кожної конкретної СКБД діалекти мови SQL.

Складність роботи з ієрархічними структурами. Раніше SQL не пропонував стандартного способу маніпуляції деревовидними структурами. Деякі постачальники СКБД пропонували свої рішення. Наприклад, Oracle використовує вираз CONNECT BY. В наш час як стандарт прийнята рекурсивна конструкція WITH.

3.1.7 Система керування базами даних MySQL

MySQL – вільна система керування реляційними базами даних. MySQL була розроблена компанією «TcX» для підвищення швидкодії обробки великих баз даних . Ця система керування базами даних (СКБД) з відкритим кодом була створена як альтернатива комерційним системам. MySQL з самого початку була дуже схожою на mSQL, проте з часом вона все розширювалася і зараз MySQL – одна з найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування [13].

MySQL має подвійне ліцензування. MySQL може розповсюджуватися відповідно до умов ліцензії GPL. Але за умовами GPL, якщо якась програма використовує бібліотеки MySQL, то вона теж повинна розповсюджуватися за ліцензією GPL. Проте це може розходитися з планами розробників, які не бажають відкривати сирцеві тексти своїх програм. Для таких випадків передбачена комерційна ліцензія компанії Oracle, яка також забезпечує якісну сервісну підтримку. В разі використання та розповсюдження програмного забезпечення з іншими вільними ліцензіями, такими як BSD, Apache, MIT та інші, MySQL дозволяє використання бібліотек MySQL за ліцензією GPL.

Гілка MySQL 5.5 містить ряд значних поліпшень, пов'язаних підвищенням масштабованості та швидкодії, серед яких :

− використання за замовчуванням рушія InnoDB;

− підтримка напівсинхронного (semi-synchronous) механізму реплікації, заснованого на патчах до InnoDB від компанії Google;

− поліпшення функцій з партіціювання даних. Розширений синтаксис для розбиття великих таблиць на кілька частин, розміщених в різних файлових системах (partitioning). Додані операції RANGE, LIST і метод оптимізації «partition pruning»;

− новий механізм оптимізації вкладених запитів та операцій JOIN;

− перероблена система внутрішніх блокувань;

− інтегровані патчі Google з оптимізацією роботи InnoDB на CPU з великою кількістю ядер.

MySQL – компактний багатонитевий сервер баз даних.

Характеризується високою швидкістю, стійкістю і простотою використання. MySQL вважається гарним рішенням для малих і середніх застосувань. Сирцеві коди сервера компілюються на багатьох платформах. Найповніше можливості сервера виявляються в UNIX-системах, де є підтримка багатонитковості, що підвищує продуктивність системи в цілому.

Можливості сервера MySQL:

− простота у встановленні та використанні;

− підтримується необмежена кількість користувачів, що одночасно працюють із БД;

− кількість рядків у таблицях може досягати 50 млн;

− висока швидкість виконання команд;

− наявність простої і ефективної системи безпеки.

3.1.8 Фронт-енд фреймворк AngularJS

AngularJS (також написаний як Angular.js) – це інтерфейсні веб-додатки з відкритим вихідним кодом на базі JavaScript, головним чином підтримуваних Google та спільнотою окремих осіб та корпорацій для вирішення багатьох проблем, що виникають при розробці односторінкових додатків. Компоненти JavaScript доповнюють Apache Cordova, основу, що використовується для розробки крос-платформних мобільних додатків. Вона спрямована на спрощення як розробки, так і тестування таких додатків, шляхом створення структури для архітектури моделі-перегляду-контролера на стороні клієнта (MVC) та моделі модельно-перегляду моделі (MVVM) разом із компонентами, що часто використовуються у багатих Інтернет-програмах. (Ця гнучкість призвела до абревіатуру MVW, що означає "типовий вигляд-незалежно", а також може

охоплювати адаптер моделі-перегляду-презентатора та модель-перегляд). У 2014 році оригінальна команда AngularJS почала працювати над застосуванням кутового платформа [14].

Система AngularJS працює під час першого читання сторінки HTML, в якій вбудовані додаткові атрибути тегів. Кутовий інтерпретує ці атрибути як директиви для зв'язування вхідних або вихідних частин сторінки до моделі, яка представлена стандартними змінами JavaScript. Значення цих змінних JavaScript можна вручну встановити в коді або отримати з статичних або динамічних ресурсів JSON.

За даними служби аналізу JavaScript для Libscore, AngularJS використовується на веб-сайтах Wolfram Alpha, NBC, Walgreens, Intel, Sprint, ABC News та приблизно 12000 інших сайтів із 1 мільйона випробуваних в жовтні 2016 року. AngularJS наразі входить до трійки найбільш відібрані проекти на GitHub.

3.2 Структура розробленого програмного забезпечення

Для реалізації програмного забезпечення для прототипування та створення структур в БД за допомогою WYSIWYG редактору, а також автоматизації завдань була обрана мова програмування PHP з базовим фреймворком Yii2. Для програмування фронт-енду була обрана мова JS та фреймворк AngularJS. Середа розробки була вибрана за умовою підтримки обох фреймворков та буда обрана JetBrains PHP Storm, вона позволяє працювати також з Grunt та Node.js, що використовуються у цьому проекті для автоматизації.

Для створення структур був розроблений WYSIWYG редактор, котрий буквально дозволяє у два кліки мишею створювати сутності та поля для них. Для того, щоб сутності можна було використовувати повторно були введені схеми, поля можно додавати як до самої сутності, так і до комбінації сутності-схеми. Це дозволяє у подальшому розробляти різноманітні інтерфейси для кожного типу користувача на одній системі керуванням контенту.

Для того, щоб можно було інтегрувати ситему до інших систем були розроблени класи, котрі позволяють використовувати сутності у інших модулях фреймворку, наприклад інших програмістів; котрі створені у WYSIWYG редакторі за допомогою драйверу підключення до БД ActiveRecord.

3.3 Концептуальне проектування бази даних інформаційної системи

База даних є найважливішим елементом будьякої інформаційної системи. Обрана система управління контентом дозволяє плагінам виробляти всілякі операції з базою даних. Для цього в YII2 є свій БД фреймворк ActiveRecord. Сервером бази даних для веб-додатку обрана система керування базами даних – MySQL.

Почнемо з етапу розробки концептуальної бази даних інформаційно-пошукової системи для пошуку вакансій для людей з обмеженими можливостями. Цей етап включає в себе аналіз об'єктів реального світу, які необхідно змоделювати в базі даних і складається з етапів:

− ідентифікація функціональної діяльності предметної області. В даному випадку мова йде про діяльність організації і в якості функціональної діяльності можна розглядати ведення реєстру вакансій в компаніях, котрі мають робочі місця для людей з обмеженими можливостями;

− ідентифікацію об'єктів, які здійснюють цю функціональну діяльність і ідентифікувати всі сутності і взаємозв'язку між ними. Процес "ведення реєстру вакансій" ідентифікує такі сутності: OBJECT, ATTRIBUTE, ATTRIBUTE_TYPE, OBJECT_TYPE;

− ідентифікацію характеристик цих сутностей;

− ідентифікацію взаємозв'язків між сутностями.

Перерахуємо сутності:

− сутність OBJECT_TYPE може включати такі характеристики як: ідентифікатор, ідентифікатор батька, закодоване ім’я, ім’я, та опис об’єктного типу;

− сутність ATTRIBUTE_TYPE може включати такі характеристики як: ідентифікатор, ідентифікатор об’єктного типу, закодоване ім’я, ім’я, тип поля, ім’я вкладки, особа конфігурація поля;

− сутність OBJECT може включати такі характеристики як: ідентифікатор, ідентифікатор батька, ідентифікатор об’єктного типу, закодоване ім’я, ім’я, та опис об’єкту;

− сутність ATTRIBUTE може включати такі характеристики як: ідентифікатор, ідентифікатор об’єкту, ідентифікатор атрибутного типу, значення, та датове значення;

Наступний етап проектування БД полягає у встановленні відповідності між сутностями і характеристиками предметної області і відносинами і атрибутами в нотації обраної СКБД. Оскільки кожна сутність реального світу володіє якимись характеристиками, в сукупності утворюють повну картину її

прояви, можна поставити їм у відповідність набір відносин (таблиць) і їх атрибутів (полів).

Перерахувавши всі відносини і їх атрибути, вже на цьому етапі можна почати усувати зайві позиції. Кожен атрибут повинен з'являтися тільки один раз і треба вирішити, яке відношення буде власником якого набору атрибутів.

На наступному етапі визначаються атрибути, які унікальним чином ідентифікують кожен об'єкт. Це необхідно для того, щоб система могла отримати будь-який одиничний рядок таблиці.

Наступний етап передбачає вироблення правил, які будуть встановлювати і підтримувати цілісність даних. Будучи певними, такі правила в клієнт-серверних СКБД підтримуються автоматично – сервером баз даних; в локальних же СКБД їх підтримку доводиться покладати на користувальницький додаток.

Кожен з різних типів зв'язків повинен бути змодельований в базі даних. Існує кілька типів зв'язків: зв'язок "один-до-одного", зв'язок "один-до-багатьох", зв'язок "багато-до-багатьох".

Зв'язок "один-до-одного" представляє собою найпростіший вид зв'язку даних, коли первинний ключ таблиці є в той же час зовнішнім ключем, що посилаються на первинний ключ іншої таблиці.

Зв'язок "один-до-багатьох" в більшості випадків відображає реальну взаємозв'язок сутностей в предметної області. Вона реалізується вже описаною парою "зовнішній ключ-первинний ключ", тобто коли визначено зовнішній ключ, що посилається на первинний ключ іншої таблиці.

Зв'язок "багато-до-багатьох" в явному вигляді в реляційних базах даних не підтримується. Однак є ряд способів непрямої реалізації такого зв'язку, які з успіхом відшкодовують її відсутність. Важливість нормалізації полягає в тому, що вона дозволяє розбити великі відносини, як правило, містять велику надмірність інформації, на більш дрібні логічні одиниці, що групують тільки дані, об'єднані "по природі".

Кожна таблиця в реляційній базі даних задовольняє умові, відповідно до якого в позиції на перетині кожного рядка і стовпчика таблиці завжди знаходиться єдине значення, і ніколи не може бути безлічі таких значень.

Вигляд проектованої бази даних після нормалізації демонструють табл. 3.1 – 3.4.

Таблиця 3.1 – Структура таблиці “OBJECT_TYPE”

Ім’я поля	Тип

ID	INT(11)
PARENT_ID	INT(11)
CODE	VARCHAR(255)
NAME	VARCHAR(255)
DESCRIPTION	TEXT

Таблиця "OBJECT_TYPE" містить в собі інформацію про типи об’єктів.

Таблиця 3.2 – Структура таблиці “ATTRIBUTE_TYPE”

Ім’я поля	Тип
ID	INT(11)
OBJECT_TYPE_ID	INT(11)
CODE	VARCHAR(32)
NAME	VARCHAR(255)
FIELD_TYPE	VARCHAR(255)
FIELD_TAB	VARCHAR(255)
CONFIG	TEXT

Таблиця "ATTRIBUTE_TYPE" містить в собі інформацію про типи полів котрі відносяться до об’єктів.

Таблиця 3.3 – Структура таблиці “OBJECT”

Ім’я поля	Тип
ID	INT(11)
PARENT_ID	TEXT
OBJECT_TYPE_ID	DECIMAL(10,2)
NAME	TINYINT(1)
DESCRIPTION	TEXT

Таблиця "OBJECT" містить у собі всі екземпляри об’єктів.

Таблиця 3.4 – Структура таблиці “ATTRIBUTE”

Ім’я поля	Тип
ID	INT(11)
OBJECT_ID	INT(11)
ATTRIBUTE_TYPE_ID	INT(11)
VALUE	TEXT

DATE_VALUE	TIMESTAMP

Таблиця "ATTRIBUTE" містить в собі інформацію про всі екземпляри полів що відносяться до екземплярів об’єктів.

Заключним етапом є вирішення питань надійності даних і, при необхідності, збереження секретності інформації. Для цього необхідно відповісти на наступні питання:

- хто буде мати права (і які) на використання бази даних;
- хто буде мати права на модифікацію, вставку і видалення даних;
- чи потрібно робити різницю в правах доступу;

яким чином забезпечити загальний режим захисту інформації.

3.4 Досдіження роботи створення, зберігання сутностей у EAV форматі в базі даних

Для зберігання сутностей була створена струкрута у базі даних MySQL, наведена на рисунку 3.1, також вона може бути портована на будь яку СУБД [15].

Такий формат зберігання дає змогу міняти структуру сутності в залежності від користувача а також в залежності від проекту до проекту

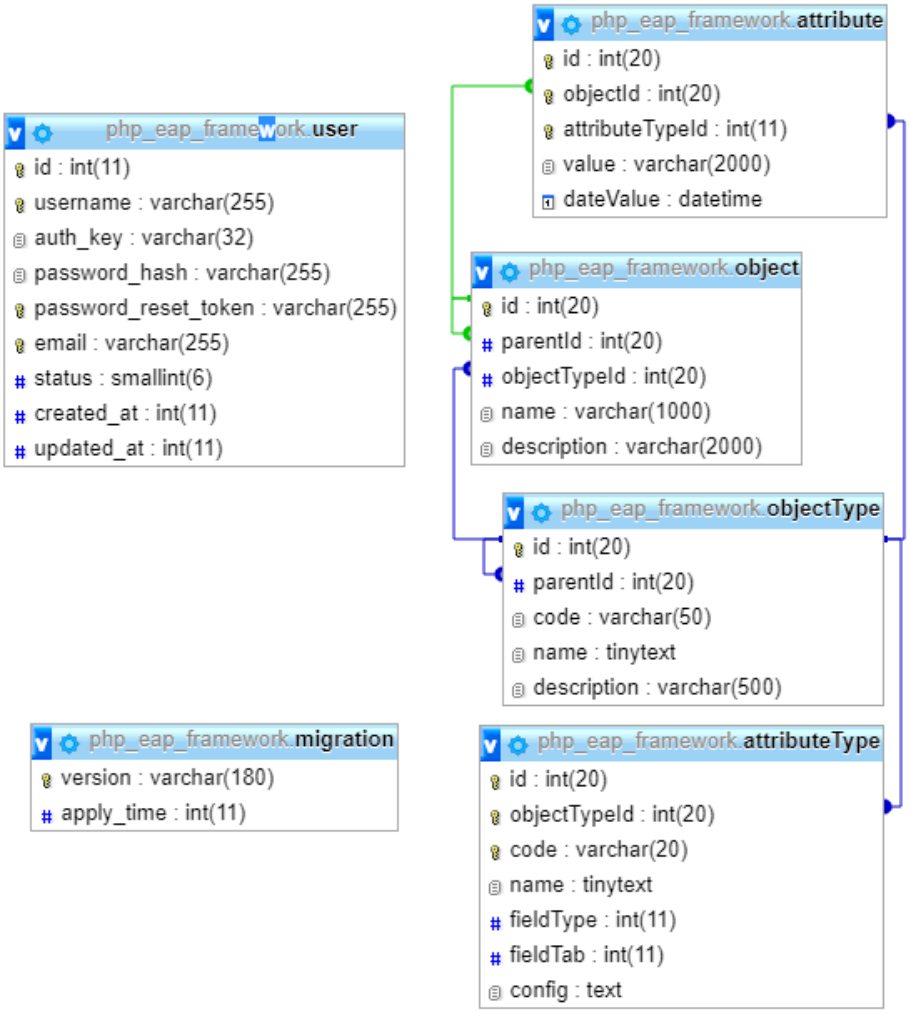

Рисунок 3.1 – Структура зв’язків між таблицями, що використовуються для зберігання EAV сутностей

3.5 Опис алгоритму роботи з EAV структурами за допомогою ActiveRecord

На початку система опрацьовує POST запит до /eav-service/get-object де параметрами objectIds та withAttributes є ідентифікатори об’єктів та потрібність повної вигризки об’єктів відповідно. Після цього виконується вилучення об’єктів з БД та наповнення структур сутностей атрибутами. Якщо атрибуту не має але є атрибутний тип – то значення виставляється у NULL, якщо об’єктів не знайдено, то відбувається відстріл виключної ситуації та виводиться повідомлення про помилку [16]. Алгоритм представлений у данному абзаці можно у графічній формі переглянути на рис 3.2-3.3

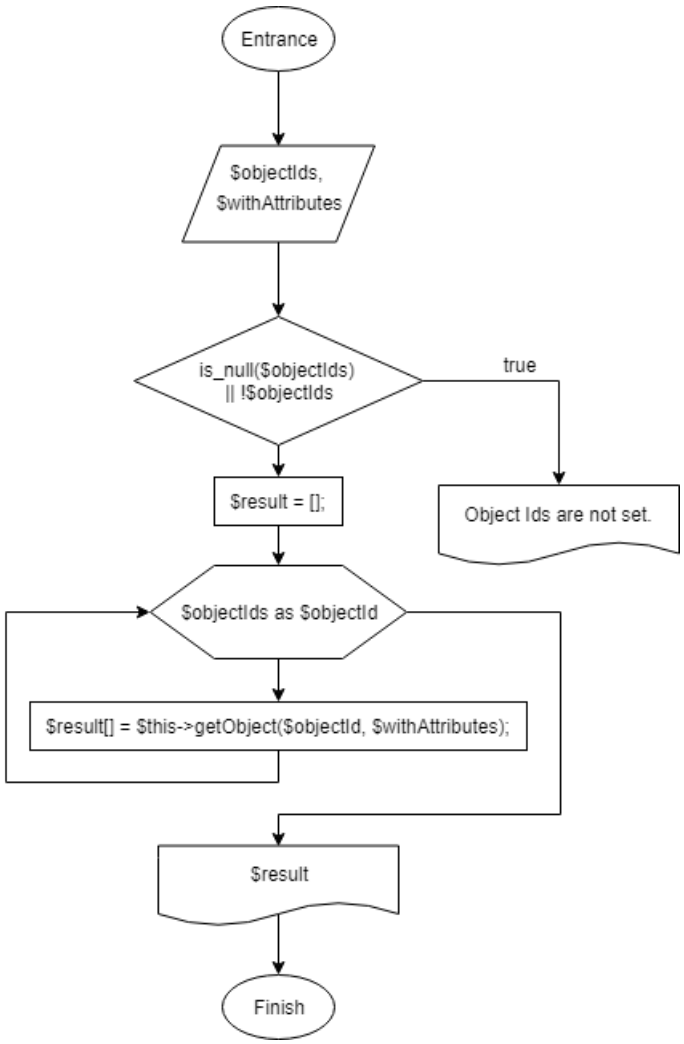

Рисунок 3.2 – Алгоритм роботи RESTfull API для отримання EAV сутностей

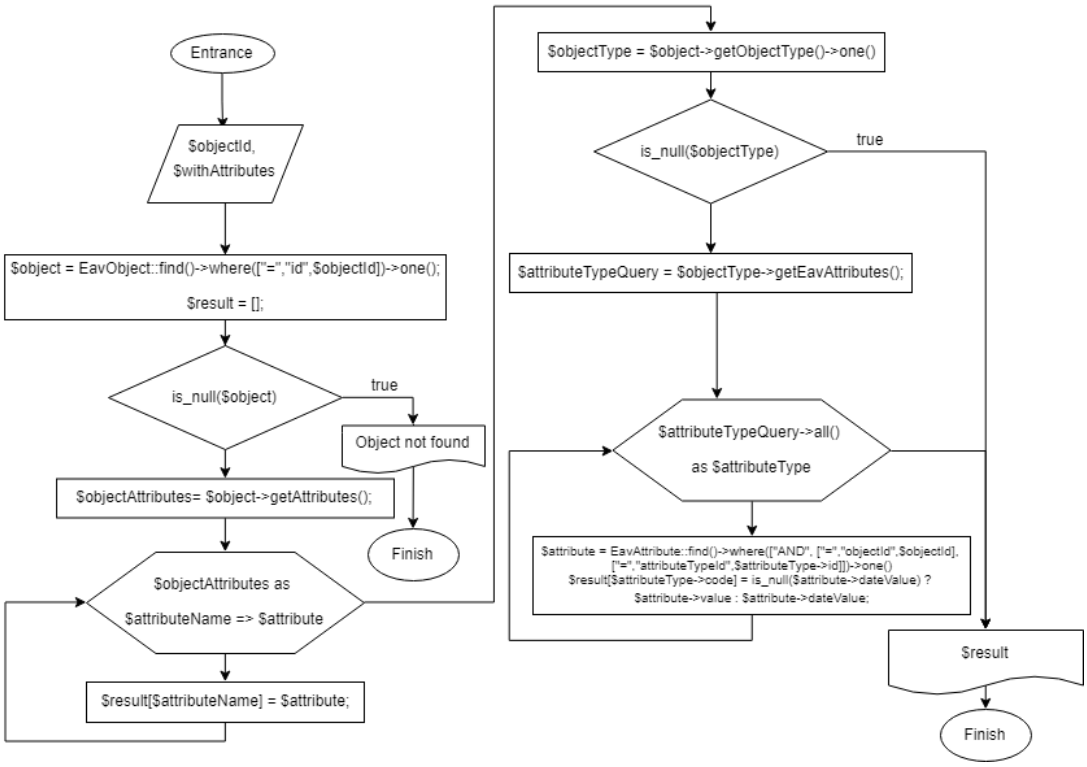

Рисунок 3.3 – Розгорнутий алгоритм роботи RESTfull API для отримання EAV сутностей

3.6 Реалізація фронтенду

За допомогою WYSIWYG-EAV редактору структур даних (рис. 3.4) – користувач створює свою структуру даних, котру потім буде наповнювати. Тут реалізовані наступні дії: виділення, редагування, клонування, переміщення, додавання та видалення сторінок, структур, вкладок, автоматизованих завдань та схем у системі.

За допомогою WYSIWYG-EAV редактору сутностей – користувач наповнює свою систему за тими чи іншими сутностями, котрим відповідає своя EAV-структура. Як і на редактору структур тут реалізовані наступні дії: виділення, редагування, клонування, переміщення, додавання та видалення сутностей з системи. Якщо потрібно, то розробник може наслідуватися від класу EavPage та сворювати свої обробники та інтерфейси у цій системі (рис. 3.5).

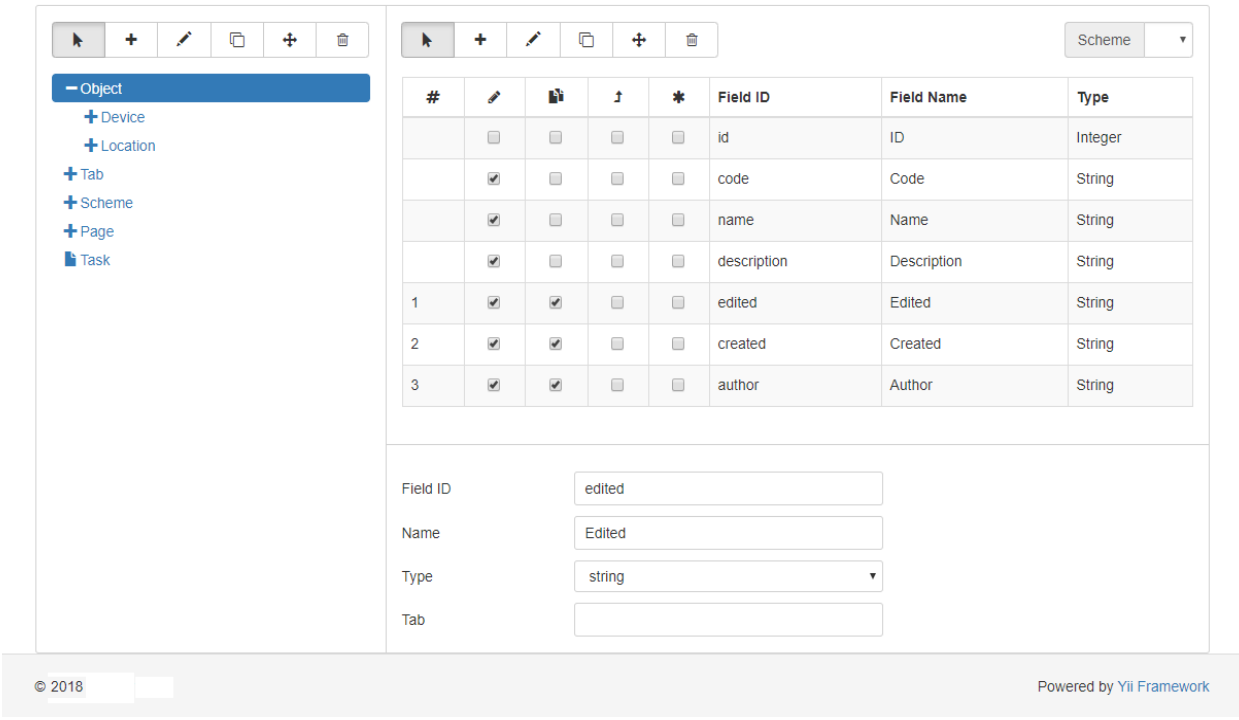

Рисунок 3.4 – Скріншот створеного редактору EAV структур

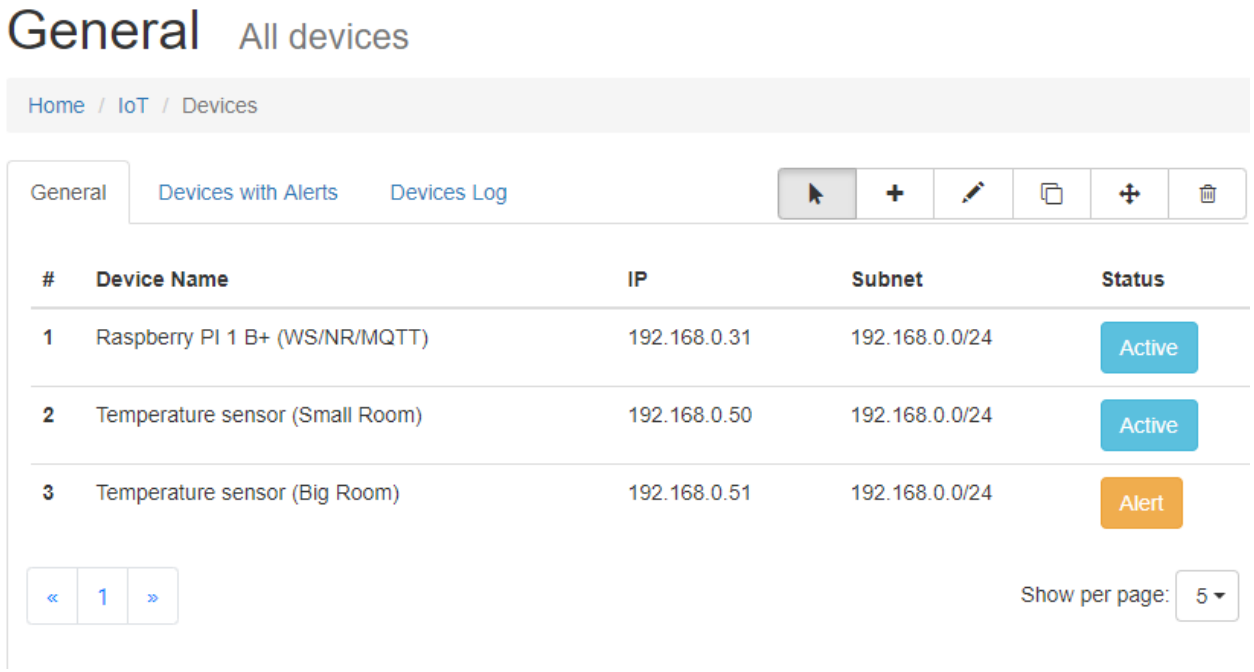

Рисунок 3.5 – Скріншот створеного модулю відображення EAV сутностей

ВИСНОВКИ

У магістерській роботі був виконаний аналіз та програмна реалізація модуля для фреймворку Yii2, котрий позволяє створювати та зберігати EAV структури у базі даних.

Для реалізації була використана СКБД MySQL та фреймворки AngularJS та Yii2.

Даний модуль дозволяє створювати та зберігати любі сутності у БД MySQL без втручання у саму базу даних та таблиці проекту, за допомогою WYSIWYG редактору.

Для взаємодії фронт-енду та бек-єнду був використований RESTful підхід. Основою є структуризація запросів та відповідей за допомогою JSON формату.

Дана система являє собою аналог програмного продукту для реалізації у OSS та BSS сфері, тому є можливість інтеграції її у інші системи за допомогою RESTfull API та розширення функціоналу за допомогою модулів інших розробників.

Магістерська робота складається з трьох розділів.

У першому розділі наведений порівняльний аналіз різних систем, які дозволяюсь створювати сутності у базі даних за допомогою WYSIWYG редактору. Більшість з них мають схожий функціонал, але алгоритми роботи в них закриті і невідомі, так як вони постачаються у скомпільованому виді.

У другому розділі виконаний аналіз існуючих систем конструювання контенту. У таблиці надан порівняльний аналіз характеристик роботи представлених систем.

У третьому розділі роботи виконано проектування та імплементацію програмного забезпечення. Для розробки програмного забезпечення обрано мову програмування – PHP та JS. Середа розробки – PHP Storm Community Edition. Фронтенд фреймворк – AngularJS. Розроблений додаток може створювати, налаштовувати структури даних, за допомогою WYSIWYG редактору а також API для зберігання, модифікування та видалення універсальних сутностей.

Як результат був створений веб-додаток, що використовує для отримання сутності з БД лише 7 запросів і 1.5Mb пам’яті та затрачує лише 76 мс, що перевершує всі аналоги, які були приведені у першому розділу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Seblod – конструктор контенту [Електронний ресурс] – Режим доступу: https://wedal.ru/seblod.html
2. Introducing K2 [Електронний ресурс] – Режим доступу: https://getk2.org/.
3. Cobalt [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Open_Cobalt
4. ZOO – це конструктор будь-якого контенту [Електронний ресурс] – Режим доступу: http://zoo.ru/
5. Обзор конструкторов контента (Content Construction Kit) для популярных CMS [Електронний ресурс] – Режим доступу: http://ajc.su/web-razrabotka/obzor-konstruktorov-kontenta-content-construction-kit-dlya-populyarnyx-cms/
6. What You See Is What You Get редактор [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/WYSIWYG
7. HTML та його особливості [Електронний ресурс] – Режим доступу: http://htmlbook.ru/html
8. Вивчення CSS у легшому виді [Електронний ресурс] – Режим доступу: https://htmlacademy.ru/courses/4/run/2
9. Інтерпретована мова програмування JavaScript. Новий спосіб навчання [Електронний ресурс] – Режим доступу: https://learn.javascript.ru/
10. Фронтенд фреймворк jQuery [Електронний ресурс] – Режим доступу: https://jquery.com/
11. Интерпретована мова програмування PHP [Електронний ресурс] – Режим доступу: http://www.php.net/
12. Структурна мова запросів SQL [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/SQL
13. Офіційний сайт MySQL [Електронний ресурс] – Режим доступу: https://www.mysql.com/
14. AngularJS — Superheroic JavaScript MVW Framework [Електронний ресурс] – Режим доступу: https://angularjs.org/
15. Peter Pin-Shan Chen. «The Entity-Relationship Model — Toward a Unified View of Data» (англ.) // ACM Transactions on Database Systems (TODS) : Сб. — Нью-Йорк: ACM, 1976. — Vol. 1. — P. 9-36. — ISSN 0362-5915. — DOI:10.1145/320434.320440
16. REST-API та REST запроси [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/REST