ЛЕКЦИЯ 5
ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ БД И СУБД

Объектно-ориентированные базы данных (ОО-БД) и поддерживающие их объектно-ориентированные СУБД начали развиваться примерно в середине 80-х годов XX века. Их появление было вызвано сложностью моделирования данных в традиционных реляционных БД, необходимостью обработки больших объемов информации, и, конечно же, активной популяризацией и развитием концепции объектно-ориентированного программирования.

Появление объектно-ориентированных баз данных было вызвано потребностями программистов, работающих с ОО-языками. Для сохранения объектов разработчики нуждались в удобных инструментах, которые не укладывались в традиционную схему для хранения данных в реляционной БД. Первые ОО-СУБД облегчали разработчикам работу с БД в объектно-ориентированном стиле, позволяя представлять и манипулировать данными как объектами. Однако они не имели автоматического механизма для отображения объектов в базе данных на объекты в программном коде – задача, которую впоследствии успешно решили с помощью технологий ORM в 1990-2000-х годах.

До появления технологий ORM, которая позволяет автоматически отображать данные между СУБД и объектно-ориентированной моделью программы, разработчики должны были вручную реализовывать код для взаимодействия ОО-приложения с базой данных. Этот процесс включал преобразование между объектами и табличными структурами. Такие техники известны как *прописной код* или *ручное отображение*. Необходимо было не только определить табличную структуру БД, написать запросы для добавления, выборки и обновления данных, но и создать код для преобразований между SQL-результатами и объектами программы. Связи между объектами и связи между таблицами разработчик должен был обрабатывать вручную, путем написания дополнительных SQL-запросов с использованием многочисленных JOIN-ов. Также был необходим уникальный код для обработки ошибок базы данных и управления транзакциями.

Основные концепции объектно-ориентированного подхода

Становление и развитие ОО-БД обеспечивают как работы в области стандартизации баз данных, так и постоянно развивающиеся языки с абстрактными типами данных и объектно-ориентированные языки программирования.

Основы объектно-ориентированного программирования были заложены в начале 1960-х годов. Первым языком программирования для работы с объектами стал Simula 67. Идеи Simula оказали существенное влияние на такие языки, как Smalltalk, объектный вариант языка Lisp (Common Lisp Object System, CLOS), Object Pascal, C++. Доминирующей методологией объектно-ориентированное программирование стало в начале и середине 1990-х годов с появлением и широкой доступностью языков Visual FoxPro 3.0, C++, и Delphi.

В наиболее общей и классической постановке объектно-ориентированный подход базируется на концепциях:

– *классов* и *объектов*;
– *инкапсуляции* атрибутов и методов;
– *иерархии* и *наследования* классов.

Любая сущность реального мира в объектно-ориентированных языках и системах моделируется в виде объекта. Каждый объект принадлежит определенному классу подобно тому, как каждая переменная является переменной какого-либо типа. Любой объект при своем создании получает генерируемый системой уникальный идентификатор, который связан с объектом во все время его существования и не меняется при изменении состояния объекта. Каждый объект имеет *состояние* и *поведение*. Под состоянием объекта понимается набор значений его атрибутов. Поведение объекта и изменение его состояния определяет набор методов. Значение атрибута объекта – это либо значение некоторого типа данных, либо некоторый объект, либо множество объектов. Во втором и третьем случае имеют место так называемые *сложные объекты*.

Состояние и поведение объекта инкапсулированы в объекте. Взаимодействие между объектами производится на основе пере-

дачи сообщений, которая реализуется через вызовы методов: объекты-инициаторы вызывают методы объектов-получателей.

Множество объектов с одним и тем же набором атрибутов и методов образует *класс объектов*. Объект должен принадлежать только одному классу. Допускается наличие примитивных предопределенных классов, объекты-экземпляры которых не имеют атрибутов. Класс, объекты которого могут служить атрибутами объектов другого класса, называется доменом этого атрибута.

Механизм порождения нового класса на основе уже существующего класса называется *наследованием*. В этом случае новый класс, называемый *подклассом* существующего класса наследует все атрибуты и методы *суперкласса*. В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы, некоторые методы суперкласса могут быть переопределены. Различаются случаи *простого* и *множественного наследования*. В первом случае подкласс может определяться только на основе одного суперкласса, во втором случае суперклассов может быть несколько. Если в языке или системе поддерживается единичное наследование классов, набор классов образует древовидную иерархию. При поддержании множественного наследования классы связаны в ориентированный граф с корнем, называемый решеткой классов. Объект подкласса считается принадлежащим любому суперклассу этого класса.

Объектная модель данных и стандартизация ОО-БД

Для реализации объектно-ориентированных баз данных разработана своя стандартная *объектная модель*. Она описана в рекомендациях стандарта ODMG-93 (Object Database Management Group). Эта модель действует как аналог реляционной модели в реляционных базах данных.

Вот несколько ключевых аспектов объектной модели, определенной в стандарте ODMG.

Объекты. В объектной модели ODMG данные представляются в виде объектов. Объект представляет собой сущность в системе, которая содержит описание состояния (атрибуты) и поведение (методы).

Атрибуты. Атрибуты объекта представляют его свойства, которые описывают его состояние. Атрибуты могут быть примитивными типами данных (например, строки, целые числа) или ссылками на другие объекты.

Методы. Методы объекта представляют действия, которые объект может выполнять. Они определяют поведение объекта и могут изменять его состояние.

Классы. Классы представляют собой определения, описывающие общую структуру группы связанных объектов. Класс определяет атрибуты и методы, которые будут у объектов этого класса.

Наследование. Одна из ключевых концепций объектно-ориентированного программирования, наследование, также поддерживается в объектной модели ODMG. Оно позволяет создавать иерархии классов, где подклассы могут наследовать атрибуты и методы от родительских классов.

Полиморфизм. В объектной модели ODMG также поддерживается полиморфизм, что позволяет объектам разных классов реагировать на один и тот же запрос или сообщение по-разному в зависимости от их типа.

Язык запросов. ODMG определяет стандартный объектно-ориентированный язык запросов OQL (Object Query Language), который предоставляет возможность управлять объектами и их связями.

Один из способов графического изображения объектно-ориентированной модели – это использование UML (Unified Modeling Language) – унифицированного языка моделирования. С помощью UML диаграммы классов можно визуализировать структуру объектов в ОО-модели. Диаграмма классов показывает классы и их свойства, а также отношения между классами – наследование, ассоциации, агрегации и композиции. Это очень эффективный способ представления системы и помогает в понимании связей и зависимостей между различными объектами в системе. Также пользуясь UML диаграммами можно изображать более сложные структуры, такие как составные классы, объединения и косвенные связи.

На рисунке 30 представлена расширенная версия обобщенной диаграммы классов. Она включает различные типы связей и

структур, такие как наследование, агрегация, композиция и зависимости, что помогает понять, как эти элементы могут быть реализованы в объектно-ориентированной системе.

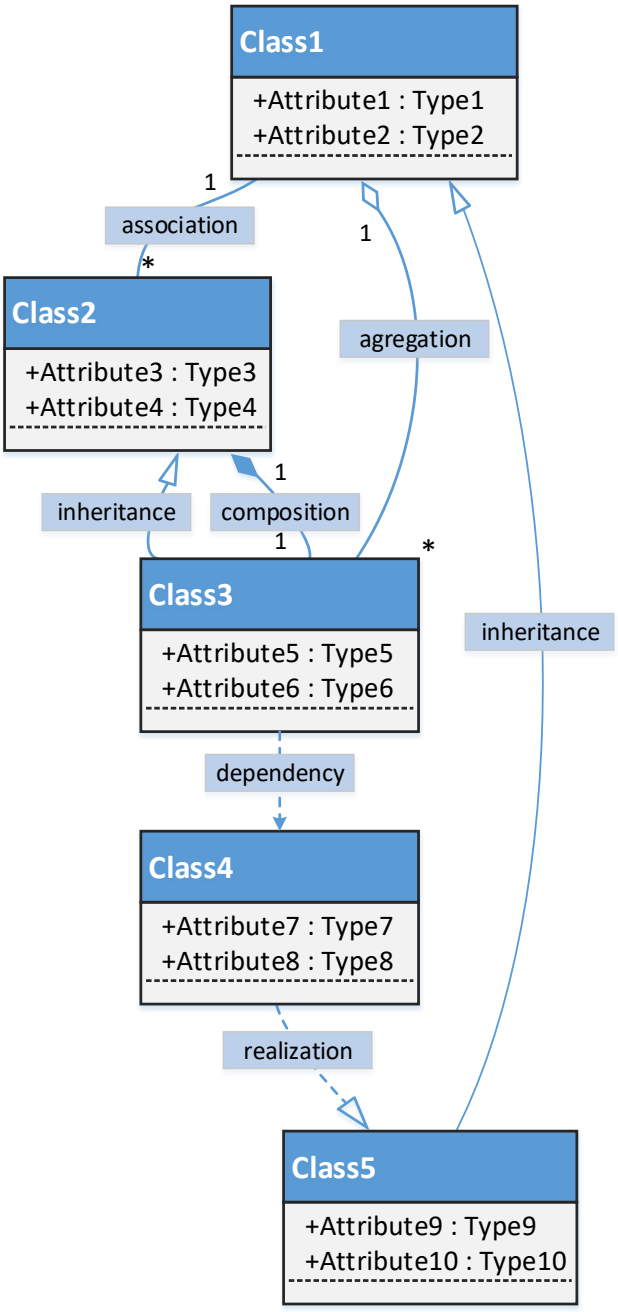

Рисунок 30 – Обобщенная диаграмма классов

Разработчики ОО-СУБД стараются следовать рекомендациям стандарта ODMG-93, который является своего рода набором правил и соглашений для разработки и поддержки функционально-

сти объектно-ориентированных СУБД. Однако, стоит отметить, что уровень соответствия стандарту может варьироваться в зависимости от конкретного продукта или провайдера.

В ряде случаев, ОО-СУБД могут предоставлять дополнительные функциональные возможности или поддержка некоторых особенностей будет реализована по-разному. Также не все СУБД могут выполнять полный набор функций, описанных в стандарте ODMG. Например, некоторые могут не поддерживать функционал OQL в полном объеме или иметь отличия в работе с объектами и их связями. В целом, поддержка этого стандарта в ОО-СУБД является достаточно важным аспектом, который может существенно определить удобство и эффективность работы с базой данных.

Последняя версия – ODMG 3.0, была выпущена в 2001 году. Причины, по которым ODMG не обновлялся так долго, являются сложными и многообразными, но вопрос в большей степени связан с изменениями в сфере технологий баз данных и программирования. Постоянно развивающиеся NoSQL технологии предлагают новые эффективные подходы к организации и обработке данных. Современные языки программирования обогащаются новыми возможностями и инструментами, что во многом обеспечивает поддержку объектно-ориентированного программирования и работу с базами данных без использования ODMG. Продолжающееся развитие и обновление технологий реляционных баз данных связано с появлением расширениий SQL. Это расширение функционала SQL заменяет многие особенности, которые были уникальными для ODMG. Таким образом, многие из функций, которые изначально поддерживались только ODMG, теперь доступны прямо в SQL. В результате этого, необходимость в использовании ODMG как отдельного стандарта сократилась.

Язык запросов объектно-ориентированных баз данных

С помощью языка запросов OQL можно формулировать запросы к ОО-базе данных, которые включают выборку объектов, фильтрацию, сортировку, группировку и агрегирование данных, а также выполнение различных операций над объектами. В целом,

OQL обеспечивает удобный и выразительный способ работы с объектно-ориентированными данными в базах данных, а также обеспечивает единый стандарт для разработчиков. Запросы OQL могут выполняться на любой объектно-ориентированной базе данных, поддерживающей стандарт ODMG.

Синтаксически OQL похож на SQL, они оба используют многие похожие ключевые слова и конструкции. Типичные команды OQL представлены в таблице 3.

Таблица 3

Типичные команды OQL

ВЫБОРКА (SELECT)	

`select x`
`from x in Class_Name`
`where x.attribute > value`
	Этот запрос выбирает объекты определенного класса, у которых атрибут соответствует заданному условию

ПРОЕКЦИЯ (PROJECTION)	

`select x.attribute1,`
`x.attribute2`
`from x in Class_Name`
	Запрос возвращает значения определенных атрибутов для всех объектов класса

ОБЪЕДИНЕНИЕ (JOIN)	

`select x, y`
`from x in Class_X, y in`
`Class_Y`
`where x.relation = y`
	Этот запрос соединяет объекты двух классов на основе определенного отношения или условия

ГРУППИРОВКА И АГРЕГАЦИЯ	

`select count(x), x.attribute`
`from x in Class_Name`
`group by x.attribute`
	Подсчитывает количество объектов с одинаковыми значениями определенного атрибута

ВЛОЖЕННЫЕ ЗАПРОСЫ	

`select x`
`from x in Class_Name`
`where x.attribute in (select`
`attribute from Class_Other)`
	Выборка объектов, где атрибут объекта соответствует результатам другого запроса.

Синтаксические различия между командами OQL и SQL проистекают из-за различий в подходах к представлению и обработке данным – объектно-ориентированного для OQL и реляционного для SQL.

1. Ориентация на объекты против ориентации на строки:

– OQL работает с объектами и их свойствами. Запросы в OQL могут прямо обращаться к атрибутам объектов, следовать по ссылкам между объектами и даже использовать методы объектов в выражениях;

– SQL оперирует таблицами и строками. Запросы в SQL фокусируются на выборке, обновлении, вставке и удалении строк в таблицах и не поддерживают прямое взаимодействие со сложными типами данных или методами.

2. Способ обращения к данным:

– в OQL запрос напрямую указывает на классы и их связи, позволяя выполнить выборку объектов с учетом их свойств и отношений в объектной модели;

– в SQL запросы формулируются через указание на конкретные таблицы и поля, возможно с использованием соединений (JOINs) для связывания данных из разных таблиц.

3. Использование методов и наследования:

– OQL поддерживает использование методов объектов в запросах, что позволяет, например, фильтровать объекты на основе результатов функций, определенных в классах;

– SQL не поддерживает методы или наследование напрямую, так как является более структурированным и ограниченным в плане обработки данных.

4. Обработка идентичности и экземпляров:

– OQL может учитывать идентичность объектов, так как каждый объект в базе является уникальным экземпляром класса;

– SQL управляет данными без учета уникальности объектов, так как каждая строка в таблице рассматривается как набор значений полей без внутренних связей между строками, кроме внешних ключей.

Рассмотрим несколько примеров таких различий.

```
# ВЫБОРКА ДАННЫХ
OQL:
select object.attribute
from object in Class_Name
where object.attribute > value

SQL:
SELECT column_name
FROM table_name
WHERE column_name > value
```

В OQL Class_Name обозначает класс, а не таблицу, и object.attribute обращается к атрибуту объекта. В SQL используются названия таблиц и столбцов.

```
# СВЯЗИ МЕЖДУ ОБЪЕКТАМИ
OQL:
select object.relatedObject.attribute
from object in Class_Name

SQL:
SELECT table1.column, table2.column
FROM table1
JOIN table2 ON table1.common_column = table2.common_column
```

OQL позволяет напрямую обращаться к связанным объектам через точечную нотацию, что отражает связи, определённые в объектной модели. SQL требует явного указания соединений (JOIN) между таблицами для доступа к связанным данным.

```
# ИСПОЛЬЗОВАНИЕ МЕТОДОВ
OQL:
SELECT object.computeValue()
FROM object in Class_Name

SQL:
EXECUTE stored_procedure_name @param1=value1, @param2=value2;
```

OQL допускает вызов методов внутри запроса. SQL не поддерживает методы объектов, так как данные в реляционных базах не имеют методов. Однако альтернативой может служить вызов хранимой процедуры. В этом случае stored_procedure_name – это имя процедуры, которую нужно вызвать, а value1 и value2 - это значения, передаваемые в качестве параметров.

```
# ОБРАБОТКА КОЛЛЕКЦИЙ:
OQL:
select object
from object in Class_Name
where someElement in object.collectionAttribute

SQL:
# нет
```

OQL умеет работать с коллекциями объектов, трактуя их как часть запроса. Это означает, что вместо работы с отдельными объектами, можно использовать группы объектов, объединенные в коллекции, что делает обработку данных более эффективной. SQL ориентирован на обработку каждой записи как отдельной строки в таблице, что ограничивает его способность к прямой работе с коллекциями на сколько-либо серьезном уровне..

```
# ПРИМЕР ЗАПРОСА НА ВЫБОРКУ:
OQL:
select book
from book in Books
where book.genre.name = "Фантастика"
and book.author.name = "Иванов"

SQL:
SELECT book.title
FROM books
JOIN genres ON books.genre_id = genres.id
JOIN authors ON books.author_id = authors.id
WHERE genres.name = 'Фантастика'
AND authors.name = 'Иванов'
```

Эти различия подчеркивают, что OQL удобен для работы в среде, где данные организованы в виде взаимосвязанных объектов, а SQL лучше подходит для работы с относительно простыми и структурированными реляционными данными.

OQL – не единственный язык запросов в ОО-БД. Одним из примеров может служить Hibernate в Java или Entity Framework в .NET, которые предоставляют ORM возможности для эффективного взаимодействия между объектно-ориентированным кодом и реляционными базами данных. Эти инструменты позволяют использовать SQL-подобные запросы прямо в коде на Java или C#, соответственно.

Помимо языка OQL модель ODMG поддерживает два других языка для определения структуры объектов и для манипуляции объектами:

ODL (Object Definition Language) – используется для определения структуры объектов, аналогично тому, как в SQL используется DDL (Data Definition Language) для определения структуры таблиц.

OML (Object Manipulation Language) – предоставляет средства для изменения, добавления и удаления объектов в базе данных, аналогично языкам манипулирования данными (DML) в SQL.

В объектно-ориентированных базах данных нет единого обобщающего языка, который бы объединял ODL, OQL, и OML, аналогично тому как SQL объединяет DDL, DML, и DQL в реляционных базах данных. Главное отличие между двумя подходами заключается в том, что SQL обеспечивает унифицированный синтаксис и единый стандарт для всех этих операций, тогда как ODL, OQL и OML могут значительно варьироваться в своем синтаксисе и поведении в зависимости от конкретной системы управления базами данных, которую они используют. Однако, все три языка призваны обеспечить функциональность, аналогичную SQL, в контексте объектно-ориентированных баз данных.

Сравнение реляционного и объектно-ориентированного подходов к проектированию БД

Сравнивая объектно-ориентированный и реляционный подходы к БД, можно отметить следующие особенности. В реляционных БД сущности предметной области представляются как структуры, состоящие из набора элементарных типов данных. Такое представление имеет понятную интерпретацию – строка в плоской таблице. В том случае, когда специфика приложения позволяет работать с таким представлением реальных объектов, реляционные БД отлично справляются со своей задачей. Однако, реляционная модель и ее способ описания предметной области в виде набора плоских таблиц не всегда способны отразить внутреннюю структуру для многих предметных областей, являются

искусственными и становятся совершенно непонятными при увеличении количества таблиц. Основная причина несостоятельности реляционного подхода заключается в слишком сильной абстракции реальных объектов, что ведет к усложнению семантики.

В отличие от реляционных баз данных объектно-ориентированные базы данных обладают простой и естественной связью с предметной областью, что облегчает проектирование и положительно сказывается на понимании принципов функционирования программ, обеспечивает более тесную интеграцию с объектно-ориентированными языками программирования. Все удобство применения соответствующей технологии – именно в простоте и естественном описании предметной области.

Однако нельзя не остановиться и на проблемах ОО-реализации баз данных и систем управления этими базами данных. Например, там отсутствует мощная математическая база, подобная реляционной алгебре, которая обеспечивает простые и надежные способы выполнения запросов и операций с данными. Объектно-ориентированной СУБД придется полагаться на методы и алгоритмы, которые могут быть гораздо менее оптимизированы и логически сложны. В реляционных СУБД, благодаря основам теории множеств и использованию стандарта SQL, существуют уже изученные методы оптимизации запросов. В ОО-СУБД же может потребоваться более глубокая и индивидуальная настройка для достижения максимальной эффективности. ОО-БД не всегда имеют четкое понимание, как использовать стандартные операторы сравнения, такие как равно, меньше или больше, что может создать проблемы при сортировке или поиске данных.

В приложениях с ОО-БД обычно вообще обходятся без языка запросов, аналогичного языку SQL в реляционных базах данных. Вопрос, является ли это преимуществом или недостатком, во многом зависит от специфики приложения и требований к нему. С одной стороны, преимуществом отсутствия языка запросов является упрощенное изучение и работа с базой данных. С другой стороны отсутствие языка запросов в БД может затруднять извлечение и манипулирование данными по сравнению с реляционными базами данных. Например, приложению необходимо выбрать из ОО-БД объекты некоторого класса, которые соответствуют определенным критериям. Разработчику вместо простой

команды SELECT-FROM-WHERE придется написать собственный код для перебора всех объектов этого класса, проверить условия и вернуть подходящие значения, что несет за собой дополнительные трудности и может быть менее эффективно.

Немалые сложности стоят на пути поддержки множественного наследования. Несмотря на то, что множественное наследование предлагает гибкие возможности для проектирования объектных моделей, оно также вносит дополнительные трудности в управление, проектирование и производительность объектно-ориентированных баз данных. Структура классов должна быть верно отображена в БД чтобы избежать потенциальных конфликтов в иерархии наследования. Извлечение и обновление данных в случае множественного наследования может стать намного сложнее, в особенности, если иерархия классов и/или свойства классов были изменены после сохранения объектов в базе данных. С точки зрения проектирования БД, важно продумывать, как организовать хранение данных, чтобы обеспечить их консистентность, устойчивость к изменениям и эффективность обработки запросов. Механизмы множественного наследования могут значительно отличаться от языка к языку, и программа с БД, написанная на одном языке и использующая множественное наследование, может не работать на другом языке без значительной переработки кода.

ОО-СУБД обычно проектируются с учетом принципов объектно-ориентированного программирования для упрощения работы разработчиков при создании и поддержке баз данных. Это означает, что они могут легко интегрироваться с различными объектно-ориентированными языками программирования.

С другой стороны, большинство ОО-СУБД не привязаны к одному конкретному языку и могут поддерживать несколько различных языков программирования. Это решение обычно принимается с целью увеличения универсальности и гибкости системы.

Текущее положение и проблемы ОО-БД на рынке IT

Хотя объектно-ориентированные базы данных предлагают уникальные преимущества для работы с комплексными и взаимо-

связанными объектами, их распространённость и популярность на рынке не сравнимы с реляционными базами данных или системами NoSQL. На заре своего развития ОО-БД казались перспективным новшеством, однако не смогли занять доминирующее положение. Среди основных причин относительно низкой популярности ОО-БД можно выделить следующие:

1. Часто ОО-БД требуют более тесной связи между базой данных и приложением, что может представлять проблемы с точки зрения масштабируемости и производительности.

2. Большинство ОО-БД также требуют использования специфических языков запросов, которые могут быть не так хорошо поддержаны или понятны разработчикам, как SQL.

3. SQL, язык запросов, который используется в основном для взаимодействия с реляционными БД, чрезвычайно популярен и мощен, а его знание – стандартное требование для многих позиций разработчиков.

Согласно данным IT-сервиса db-engines.com на март 2024 года, из 419 известных систем управления базами данных доля ОО-СУБД составляет всего 4,8%, или 20 систем (рисунок 31), при этом их рейтинг не превышает 0,2% (рисунок 32).

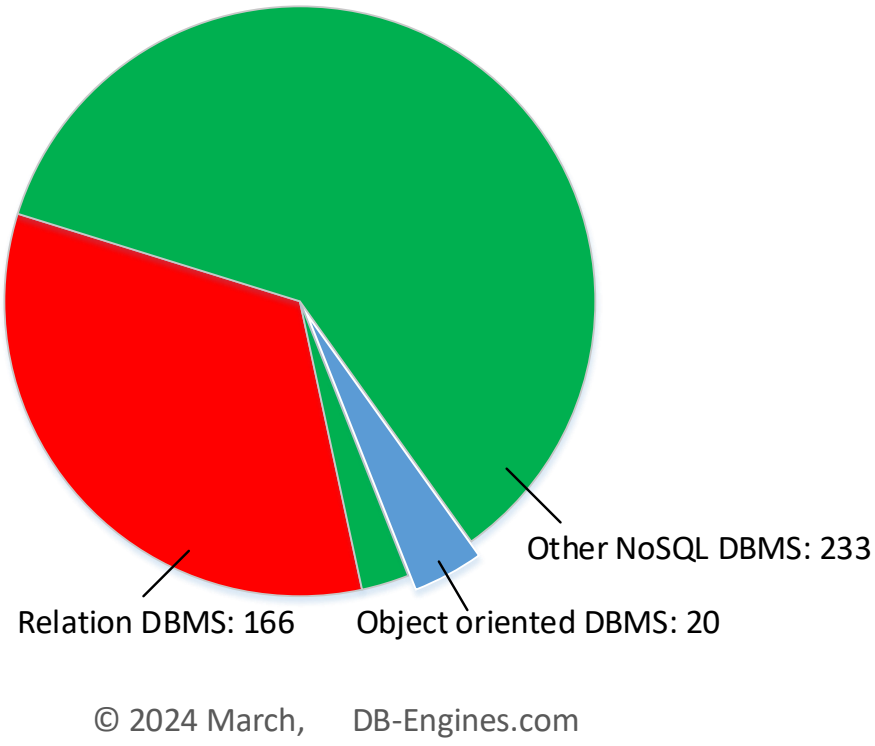

Рисунок 31 – Доля ОО-СУБД согласно данным на март 2024 г.

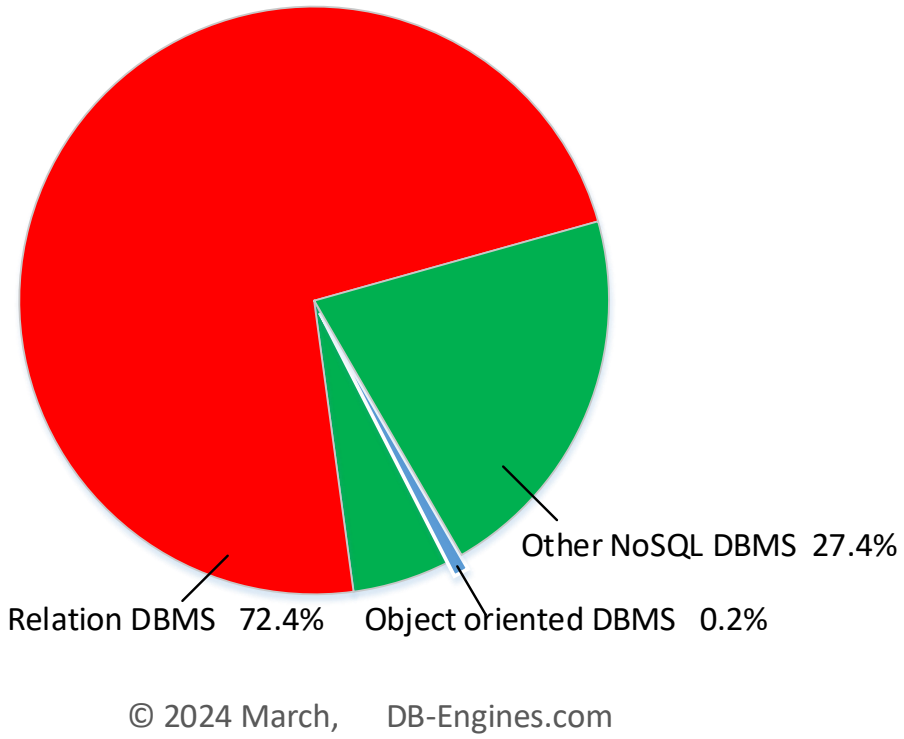

Рисунок 32 – Рейтинг ОО-СУБД согласно данным на март 2024 г.

Однако это не означает, что ОО-БД не развиваются или не используются вовсе. Справедливости ради следует отметить, что уже значительная часть современных СУБД принадлежат более чем к одной категории по реализуемой модели данных. Это значит, что объектно-ориентированная модель поддерживается не только в указанных 20 ОО-СУБД, но и во многих *мультимодельных базах данных*.

Среди ранних платформ для объектно-ориентированного хранения данных можно назвать: ObjectStore – одну из первых коммерчески успешных ОО-СУБД, выпущенную в начале 1990-х; Versant Object Database – другую раннюю систему, которая также была выпущена в начале 1990-х; GemStone – система, запущенную в 1986 году.

К популярным системам управления данными, имеющими поддержку для использования с Python, относятся:

– ZODB (Zope Object Database) – чистая объектно-ориентированная база данных, полностью написанная на Python и предназначенная для хранения объектов Python;

– DB4O (Database for Objects) – хотя она не так широко используется, как раньше, db4o поддерживает Python и позволяет хранить объекты напрямую.

Среди популярных мультимодельных систем, которые могут интегрировать объектно-ориентированные характеристики с другими моделями баз данных, можно выделить:

– MongoDB – это прежде всего документо-ориентированная база данных, но она поддерживает объектно-ориентированные концепции через JSON-подобные документы; для работы с MongoDB из Python используется библиотека PyMongo;

– CouchDB – также документо-ориентированная, использует JSON для хранения данных, что позволяет легко работать с объектами.

– Neo4j – это графовая база данных, но она поддерживает объектно-ориентированное представление данных через узлы и связи, которые могут содержать различные свойства.

Объектно-ориентированные базы данных по-прежнему находят свое применение в конкретных областях или сценариях, где их особенности могут быть особенно полезны. Например, они могут быть более подходящими для работы с большими и сложными структурами данных, которые нелегко отображаются на реляционные структуры и там, где требуется более тесная интеграция данных с объектно-ориентированными языками программирования.

Однако, реалии бизнес-приложений и требования к масштабируемости и гибкости вызвали потребность в более универсальных решениях. Это привело к разработке объектно-реляционных баз данных, которые комбинируют удобства объектно-ориентированного подхода с проверенной эффективностью и широкой поддержкой реляционных систем. Параллельно, технология ORM облегчает разработку приложений, автоматизируя преобразование данных между несовместимыми типами систем, тем самым создавая мост между объектно-ориентированным программированием и реляционными базами данных.

Объектно-реляционные базы данных

Объектно-реляционные базы данных (ОРБД) соединяют в себе лучшие качества реляционных и объектно-ориентированных баз данных. Они расширяют традиционные реляционные СУБД, вводя поддержку объектных типов, наследования, полиморфизма и других ОО-характеристик. Это позволяет разработчикам использовать более естественные и гибкие структуры данных при работе с реляционными базами данных.

ОО-СУБД играли ключевую роль в развитии ОРБД, так как именно они продемонстрировали ценность объектно-ориентированных методологий в управлении данными. Идеи, заложенные в ОО-СУБД, были частично адаптированы и интегрированы в ОРБД, обеспечивая разработчикам лучшие инструменты для работы со сложными системами. Например, Oracle и PostgreSQL предлагают объектно-реляционные возможности, которые позволяют встроить объектно-ориентированный код прямо в базу данных.

Перечислим основные элементы, осуществляющие объектные расширения в реляционных базах данных, используемые на сегодняшний день:

– *Хранение больших объемов данных*. Наряду с теми данными, которые хранились в БД традиционно, современные объектно-реляционные базы данных позволяют хранить в столбцах таблицы картинки, видеоролики и другие большие документы.

– *Списки в столбцах*. Объектно-реляционные базы данных позволяют хранить в столбцах целые списковые структуры.

– *Пользовательские расширения*. В объектно-реляционных базах данных пользователи имеют возможность вмешиваться в изначально предоставляемый СУБД инструментарий, создавая, в частности, новые пользовательские типы данных.

– *Хранимые процедуры*. В определенном смысле хранимые процедуры также являются объектным расширением, осуществляя необходимые пользователю воздействия на данные (поддержка процедурного подхода).

Компании, такие как IBM и Oracle, внесли значительные усилия в развитие этой технологии, включив в свои продукты объ-

ектные возможности для расширения функциональности традиционных реляционных СУБД.

IBM разработала версии своей СУБД DB2, которые включают объектно-реляционные характеристики. Это позволило разработчикам использовать сложные типы данных и структуры, которые традиционно ассоциируются с объектно-ориентированным программированием, такие как встроенные типы и массивы, а также поддержку для иерархий наследования и полиморфизма. В DB2 добавлены также возможности для определения пользовательских функций и процедур, что позволяет более тесно интегрировать бизнес-логику с управлением данными.

Oracle продвинула концепцию ОРБД с помощью своей системы Oracle Database, добавив поддержку объектных типов данных прямо в реляционную модель. Разработчики могут определять свои собственные типы – классы объектов, которые могут включать методы, а также поля данных. Эти типы могут быть использованы для создания столбцов в таблицах, позволяя тем самым объектам быть сохранёнными и извлекаемыми из базы данных как единые сущности. Oracle поддерживает также ссылки между объектами, что облегчает представление и обработку сложных связей.

PostgreSQL изначально разрабатывалась как реляционная СУБД, но в процессе развития приняла объектно-реляционные концепции, предоставляя поддержку для пользовательских типов данных, инкапсуляции данных и функций, а также сложных запросов, включающих наследование и полиморфизм. Это сделало PostgreSQL одной из самых гибких бесплатных систем управления базами данных на сегодняшний день.

Informix – ещё один пример объектно-реляционной СУБД от IBM, который предоставляет расширенные возможности для работы с пользовательскими типами данных. Помимо пользовательских типов данных, Informix поддерживает создание пользовательских функций и методов, которые могут быть привязаны к определённым типам данных. Эти функции могут быть вызваны для выполнения операций над объектами данных прямо внутри SQL-запросов, что увеличивает гибкость и мощность обработки запросов.

Эти объектные расширения позволяют объектно-реляционным системам управления данными эффективно справляться с задачами, которые традиционные реляционные СУБД могли бы найти сложными или невозможными для выполнения. ОРСУБД стали мощными инструментами для решения специфических задач в таких областях, как ГИС, комплексная аналитика и обработка сложных структурированных данных.

Однако, как и любая технология, ОРБД имеют ряд недостатков, которые могут влиять на их применение в зависимости от конкретных требований проекта. Вот некоторые из этих недостатков:

Проектирование схемы базы данных становится более сложным, поскольку необходимо учитывать не только структуру данных, но и поведение объектов. Интеграция объектно-ориентированной и реляционной модели данных может привести к увеличению сложности, как в реализации, так и в использовании программных систем.

Преобразование данных между объектами и таблицами может замедлить выполнение операций, особенно при неоптимальных запросах. Оптимизация запросов в ОРБД может быть более сложной по сравнению с чисто реляционными СУБД из-за дополнительной сложности объектных структур и взаимодействий.

Программы, разработанные для использования с определённой ОРБД, могут испытывать сложности с переносимостью на другие системы из-за уникальных объектно-реляционных расширений и возможностей. Миграция существующих данных из реляционных систем в ОРБД или наоборот может быть сложной и ресурсоёмкой из-за различий в структуре данных и поведении объектов.

Отмеченные недостатки не делают объектно-реляционные БД непригодными для использования, они подчеркивают важность тщательного анализа требований к проекту и квалификации команды перед их внедрением. К тому же, многие из этих проблем могут быть частично устранены благодаря применению универсальных стандартов и технологий, таких как объектно-реляционное отображение (ORM), которые облегчают интеграцию и управление объектно-реляционными данными, минимизируя сложность и улучшая производительность.

Технология объектно-реляционного отображения ORM

В условиях современной разработки программного обеспечения, где реляционные базы данных продолжают доминировать, а объектно-ориентированное программирование является стандартом, возникает необходимость в технологиях, которые могут эффективно связывать эти две парадигмы. Одним из таких решений является технология объектно-реляционного отображения - ORM, которая позволяет разработчикам работать с базами данных в терминах объектов и классов, как если бы это были обычные элементы программного кода, не заботясь о деталях SQL-запросов и структуры баз данных.

Рассмотрим простой пример – добавления нового пользователя в базу данных – иллюстрирующий различия между кодом, который использует ORM, и традиционным SQL-кодом для взаимодействия с базой данных. В начале рассмотрим традиционный подход с использованием чистого SQL для добавления атрибутов пользователя name, и email в таблицу users.

```
import sqlite3

# Подключение к базе данных
conn = sqlite3.connect('example.db')
cursor = conn.cursor()

# Создание таблицы users, если она еще не существует
cursor.execute("""
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    email TEXT NOT NULL
)
""")

# SQL-запрос для добавления пользователя
query = "INSERT INTO users (name, email) VALUES (?, ?)"
data = ("Иван Иванов", "ivanov@example.com")

# Выполнение запроса
cursor.execute(query, data)
conn.commit()

# Закрытие подключения
cursor.close()
conn.close()
```

В этом примере напрямую ведется работа с SQL-запросами, что требует от программиста точного знания структуры таблицы и синтаксиса SQL.

Теперь рассмотрим, как та же задача может быть выполнена с использованием ORM-фреймворка SQLAlchemy, который позволяет работать с базами данных на более высоком уровне абстракции.

```
from sqlalchemy import create_engine, Column, Integer, String,
Sequence, declarative_base, sessionmaker

Base = declarative_base()

# Определение модели пользователя
class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, Sequence('id_seq'), primary_key=True)
    name = Column(String(50))
    email = Column(String(50))

# Создание подключения к базе данных
engine = create_engine('sqlite:///example.db')
Base.metadata.create_all(engine)

Session = sessionmaker(bind=engine)
session = Session()

# Добавление нового пользователя
new_user = User(name='Иван Иванов',email='ivanov@example.com')
session.add(new_user)
session.commit()

# Закрытие сессии
session.close()
```

В этом коде определен класс User, который отражает структуру таблицы в базе данных. Для выполнения операций с базой данных используются объекты этого класса. ORM позволяет работать с данными как с объектами Python, что делает код более интуитивно понятным и уменьшает количество ошибок.

ORM автоматизирует процесс преобразования (отображения) данных между несовместимыми типами систем, позволяя объектам приложений свободно взаимодействовать с реляционными базами данных. К преимуществам использования ORM следует отнести следующее:

– предоставления высокоуровневого интерфейса для взаимодействия с базой данных, что упрощает разработку и поддержку;

– автоматическую защиту кода приложения от *SQL-инъекций*[1];

– упрощение выполнение сложных запросов и операций с данными, таких как наследование, ассоциации и транзакции;

– код, написанный с использованием ORM, легче адаптировать под различные СУБД.

Однако использование ORM также требует внимания к производительности запросов и правильной конфигурации, так как неоптимально настроенные ORM-системы могут привести к ухудшению производительности приложений. Понимание того, как ORM взаимодействует с базой данных, может помочь разработчикам оптимизировать производительность и обеспечить масштабируемость приложения.

Примерами популярных ORM-фреймворков могут служить упомянутые выше Hibernate для Java, Entity Framework для .NET, и Django ORM для Python. Эти инструменты предоставляют мощные возможности для управления реляционными данными в объектно-ориентированной манере, облегчая миграцию данных, управление схемами и выполнение сложных запросов с минимальными усилиями.

Таким образом, ORM обеспечивает необходимую связь между объектно-ориентированным программированием и реляционными базами данных. Использование этой технологии позволяет значительно упростить разработку, повысить качество приложений и сократить время на разработку, делая процесс разработки более интуитивно понятным и менее затратным по времени.

Введение. Понятие документа. Операции над документами. Гибкость схемы данных. Проблемы обновления хранимых документов. Примеры документо-ориентированных СУБД. CouchDB. Сближение реляционных и документо-ориентированных баз данных.

Документальные информационные системы, основанные на концепции БД. Основные понятия. Теоретико-множественная модель индексирования и поиска в документоориентированной БД.
Обзор документо-ориентрованных СУБД. Преимущества и недостатки документо-ориентрованных БД.

[1] SQL-инъекция — это вид атаки на приложения, при котором злоумышленник вводит или «инъецирует» вредоносный SQL-код в запросы к базе данных через пользовательский ввод.