Сравнительный анализ реляционной и объектной моделей данных

Проведем сопоставление реляционной и объектной моделей данных по структурной, манипуляционной и целостной составляющим.

На рис. 1 приведено структурное представление реляционной и объектной баз данных. На нем сплошная линия со стрелкой обозначает отношение «включает», а в случае, если такая линия исходит от множества, она обозначает подобное отношение для каждого члена множества. Штриховая линия обозначает семантически различные варианты использования.

Структурные модели реляционной и объектно-ориентированной БД

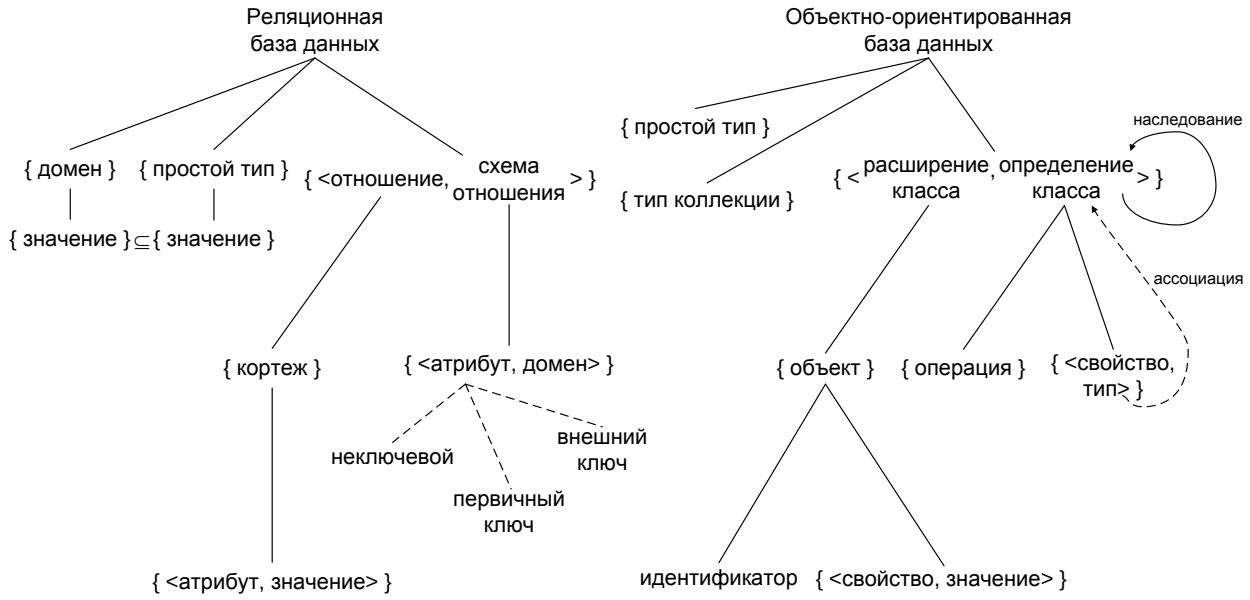

Рис. 1

И в той, и в другой модели представление данных основано на системе простых типов. Но в объектной модели помимо простых типов всегда есть предопределенные типы коллекций, значениями которых являются списки, множества, кортежи и т.д.

На базе простых типов в реляционной модели определяются домены. Для каждого атрибута в схеме отношений указывается домен значений, принимаемых этим атрибутом. Кортежи состоят из значений атрибутов, набор которых определяется схемой отношения.

Объект со структурной точки зрения представляется своим состоянием, то есть значениями своих атрибутов (свойств). Атрибуты являются именованными, а набор атрибутов является одинаковым для всех объектов определенного типа. Здесь наблюдается прямая аналогия с кортежем и схемой отношения в реляционной модели. Но в качестве атрибутов объекта могут выступать не только значения простых типов, но и другие объекты. Такой объект называется объектом со сложным состоянием, а тип объекта-атрибута называется доменом атрибута. Объекты-атрибуты, в свою очередь, тоже могут обладать сложным состоянием, что позволяет строить комплексные системы объектов. Сложное состояние объектов противоречит требованию атомарности значений атрибутов в реляционной модели.

В объекте в явном или неявном виде хранится информация о типе объекта, что необходимо для реализации наследования. При использовании наследования в качестве атрибута типа t может храниться значение типа t’, если тип t является предком типа t’ в иерархии классов. Можно сказать, что механизм наследования ортогонален относительно

структуры данных, то есть образует второе измерение объектной базы данных. В реляционной модели невозможно хранить в качестве значений одного атрибута значения разных типов данных.

В реляционной модели множество кортежей, отвечающих некоторой схеме отношения, образуют отношение. В объектной модели аналогом отношения является расширение класса. Расширение класса в объектной модели, в отличие от отношения, является мультимножеством, так как позволяет наличие равных (но не идентичных) объектов.

Сравниваемые модели данных относятся к логическому уровню проектирования баз данных. Начальной же стадией проектирования системы базы данных является разработка модели предметной области, или концептуальное проектирование системы. Инструментальные средства для спецификации концептуальной модели предметной области также принято называть моделями данных. Одной из наиболее популярных моделей данных подобного рода является модель сущностей-связей. В структурном аспекте объектно-ориентированный подход весьма близок к подходу семантического моделирования данных. Фундаментальные абстракции, лежащие в основе моделей сущностей-связей используются и в объектно-ориентированном подходе. На абстракции агрегации основывается построение сложных объектов, значениями атрибутов которых могут быть другие объекты. Абстракция группирования - основа формирования классов объектов. На абстракциях специализации/обобщения основано построение иерархии классов.

Таким образом, объектная модель данных способна более адекватно реализовывать модель предметной области на логическом уровне, чем реляционная.

Кроме того, объектная модель данных в большей степени соответствует моделям, используемым в объектно-ориентированных языках программирования, что облегчает разработку приложений соответствующих баз данных.

Из вышеизложенного следует, что на структурном уровне можно осуществлять взаимное преобразование объектных и реляционных моделей данных. Такое преобразование называется объектно-реляционным и является достаточно подробно разработанным вопросом, так как при построении объектно-ориентированных систем в подавляющем большинстве случаев используются РСУБД.

Теперь обратим внимание на средства манипулирования данными в этих моделях.

Достоинством реляционной модели является наличие двух эквивалентных (переводимых друг в друга) и математически строгих механизмов манипулирования базой данных – реляционной алгебры и реляционного исчисления. При этом реляционная алгебра является замкнутой относительно отношений, то есть любая операция над отношением в данной алгебре производит отношение. Реляционное исчисление может производиться над кортежами или доменами. Существующие языки РСУБД (например, SQL) реализуют, как правило, комбинацию этих двух механизмов.

Ситуацию, когда при создании объектно-ориентированного приложения приходится одновременно использовать объектно-ориентированный язык программирования и ориентированный на работу с множествами декларативный язык запросов, принято называть потерей соответствия (impedance mismatch).

Выделяется 5 стандартных операций над объектами:

- конструктор – операция создания объекта и/или его инициализации;
- деструктор – операция, освобождающая состояние объекта и/или разрушающая сам объект;
- модификатор – операция, изменяющая состояние объекта;
- селектор – операция, считывающая состояние объекта, но не меняющая состояния;
- итератор – операция, позволяющая организовать доступ ко всем частям объекта в строго определенной последовательности.

Отсутствие математического аппарата манипуляции объектами вкупе с принципом инкапсуляции и наличием операций итерации привело к превалированию процедурной (программной) навигации по объектам в объектной модели. Обращение по ссылке (traverse) является наиболее часто используемой операцией в объектно-ориентированном программировании. Применение процедурной навигации заставляет программиста вручную кодировать выполнение запросов любой сложности, что неизбежно приводит к уменьшению сопровождаемости программных систем, а в некоторых случаях и к спаду производительности.

Осталось рассмотреть целостную часть реляционной и объектной моделей данных.

В реляционной модели существует три уровня целостности: домены обеспечивают семантическую целостность значений атрибутов, первичные ключи – целостность сущностей, внешние ключи – ссылочную целостность.

В объектной модели не определено средств поддержания целостности за исключением объектных идентификаторов. Они обеспечивают аналог целостности сущностей, гарантируя отсутствие в объектной базе данных идентичных объектов.

Целостность сущностей в смысле, вкладываемом в это понятие в реляционной модели, а также целостность свойств в объектной модели можно поддерживать только за счет реализации соответствующего поведения. Так как, в общем случае, все свойства объекта закрыты для пользователя, в конструктор и модификаторы объекта можно добавить логику обеспечения целостности сущностей и свойств, соответственно. Ссылки в объектной модели являются обычными атрибутами, поэтому проверка их целостности может осуществляться также программно.

Таким образом, в объектной модели можно реализовать поддержку целостности на уровне реляционной модели, но это влечет за собой дополнительные накладные расходы.

Подводя итог, среди наиболее существенных ограничений реляционной модели данных можно отметить:

- невозможность использования одинакового объектно-ориентированного подхода при разработке баз данных и приложений, их использующих;
- невозможность определения новых типов данных с использованием системных или ранее определенных типов;
- отсутствие механизмов наследования и инкапсуляции;
- невозможность связывать с данными операции их обработки.

Среди недостатков объектной модели основными являются:

- отсутствие математически обоснованного механизма манипуляции объектами;
- отсутствие обеспечиваемой самой моделью поддержки целостности данных.

Устойчивость и модель персистентности[1]

Устойчивость – способность объекта существовать во времени, переживая породивший его процесс, и (или) в пространстве, перемещаясь из своего первоначального адресного пространства. По продолжительности существования можно выделить следующие группы объектов: промежуточные результаты вычисления выражений; локальные переменные; глобальные переменные и динамически создаваемые данные; данные, сохраняющиеся между сеансами выполнения программы; данные, сохраняемые при переходе на новую версию программы; данные, которые переживают программу. Как отмечалось ранее, СУБД существуют, чтобы обеспечить постоянное хранение данных, то есть для работы с тремя последними классами объектов.

В обычном языке программирования, чтобы сделать объект устойчивым, необходимо использовать явные команды для сохранения объекта во внешней памяти

[1] Персистентность – способность программного обеспечения создавать и поддерживать перманентные объекты (persistent object)

(при этом возможно использование СУБД как посредника). Затем, чтобы многократно использовать объект, который был сохранен ранее, программист должен включить явные команды, чтобы выбрать объект из постоянной памяти (например, из СУБД).

В идеальном случае, персистентность должна быть:

· ортогональной (к типам): любые данные любого типа могут сохраниться в любое время;

· прозрачной: программист может одинаково обрабатывать постоянные и временные объекты;

· независимой (от внешней памяти): не должно быть явных операций чтения и записи в базу данных.

Моделью персистентности называется совокупность методов, обеспечивающих манипуляции с хранимыми в СУБД объектами.

При бесшовной модели персистентности модель данных или система типов СУБД являются расширением таковой большого количества языков программирования. Такая модель персистентности обладает рядом преимуществ при разработке приложений, обслуживании, и многократном использовании. Однако, обеспечение бесшовности вызывает множество проблем. Во-первых, большинство языков программирования третьего поколения не имеют надлежащих концепций для работы с СУБД (например, версий, кластеров, индексов, транзакций, устойчивости). Во-вторых, в случае поддержки нескольких языков программирования, эти языки могут иметь различные системы типов, модели совместного использования данных, и т.д. Наконец, основные механизмы СУБД (например, перемещение данных между постоянной и оперативной памятью) должны быть эффективны, что может противоречить бесшовной модели персистентности.

Указание устойчивости объекта

Можно выделить три основных способа указания устойчивости объекта.

1. Объявление схемы базы данных. Именно этот способ используется обычными РСУБД, а также некоторыми ООСУБД. При этом способе для устойчивых объектов обеспечивается отдельное пространство (базу данных). Приложения работают с отдельным набором временных классов в оперативной памяти. Объекты становятся хранимыми путем явного создания их в базе данных, и становятся не хранимыми при удалении их из базы данных.

Перевод постоянных объектов СУБД во временные объекты языка программирования и наоборот осуществляется программистом или специальной библиотекой путем перемещения переменных состояния между постоянными и переходными объектами.

2. Особый механизм инициализации устойчивых объектов. Объекты становятся устойчивыми при явном указании их как устойчивых (и удаляются при явном удалении). Подобное указание может быть сделано для выбранных объектов любого класса, или для всего класса. Можно создавать как постоянные, так и временные объекты этих классов. Таким образом, пространства устойчивых и временных объектов перекрываются.

В данном способе возникают трудности с обеспечением ссылочной целостности: приложение может создать ассоциацию между устойчивым объектом и временным объектом, а впоследствии не сможет сделать временный объект устойчивым при записи первого в базу данных.

3. Достижимость по ссылке из другого устойчивого объекта.

Данный способ является наиболее удобным для программиста, так как позволяет делать постоянным объектный граф целиком, сохранив явно лишь одну из его вершин.

Доступ к устойчивым объектам

Этот аспект модели персистентности определяет, как устойчивые объекты загружаются в память, как приложения проходят по хранимым структурам, насколько это прозрачно и какие механизмы требуются для поддержки этого доступа.

В СУБД, поддерживающих полностью отдельное пространство сохраняемых объектов возможны два подхода. Первый подход - приложение получает доступ к сохраняемому объекту, явно считывая его состояние во временный объект, и затем выполняет операции с временным объектом. По завершении манипуляций с временным объектом его состояние используется для явного обновления соответствующего объекта в пространстве сохраняемых объектов. Второй подход - приложение получает ссылки на сохраняемые объекты и посылает сообщения этим объектам для вызова операций, выполняющихся целиком в пространстве сохраняемых объектов. Это может быть сделано на уровне отдельных объектов или путем посылки запросов к СУБД.

В идеальном интерфейсе программа следовала бы по ссылке, как будто бы это был указатель на другой объект в памяти, СУБД автоматически обнаруживала бы, что объект отсутствует в памяти, читала объект с диска в оперативную память приложения, и приложение продолжало бы свою работу. Степень прозрачности этот процесс фактически зависит от способа реализации указателей в приложениях и объектных идентификаторов в СУБД.

Как правило, в базе данных и приложениях используются различные форматы указателей, и при перемещении объекта из базы данных в оперативную память приложения вызывается некоторый механизм трансляции. Это является следствием различных требований к указателям в оперативной памяти и хранящимся на диске. В качестве указателей на объекты в языках программирования используются, как правило, адреса виртуальной памяти. Для сохраняемых объектов должны применяться указатели, не зависящие от физического положения объекта, так как это положение может в разных процессах. В качестве таких указателей обычно используется генерируемый СУБД идентификатор объекта. Разыменование этих указателей обычно требует поиска значения в таблице для нахождения объекта на диске. Как только объект найден и прочитан в виртуальную память, желательно устранить накладные расходы на поиск в таблице, заменяя в оперативной памяти любые ссылки на объект новым адресом виртуальным памяти объекта. Для этого и обратного преобразования (замены адреса виртуальной памяти на идентификатор объекта при возвращении объекта на диск) были предложены различные схемы.

Объекты перемещаются между памятью приложения и базой данных операциями, называемыми активацией и дезактивацией. Активация включает передачу состояния объекта из базы данных в память. Во время активации все ссылки активизируемого объекта на другие уже прочитанные объекты переводятся из формы объектных идентификаторов в адреса виртуальной памяти. Подобным же образом переводятся все ссылки на активизируемый объект из уже прочитанных объектов. Дезактивация – процесс, обратный активации: адреса памяти переводятся обратно в идентификаторы объектов, и объекты записываются в базу данных.

В приложениях могут использоваться абстрактные или прямые ссылки на объекты. При использовании абстрактных ссылок все ссылки проходят один уровень косвенности, и их безопасность гарантируются системой. Если указываемый абстрактной ссылкой объект неактивен (находится на диске), это обнаруживается при разыменовании абстрактной ссылки, и упомянутый объект автоматически активизируется системой, с использованием хранящейся в абстрактной ссылке информации для нахождения объекта на диске. Активация, таким образом, прозрачна за счет дополнительной косвенности. Абстрактные ссылки могут использоваться, чтобы гарантировать безопасность ссылки, при активизации отдельных объектов.

Если приложение может само определить действительные ссылки, могут использоваться прямые ссылки. В этом случае при активизации объекта ссылки на уже находящиеся в виртуальной памяти объекты заменяются указателями виртуальной памяти. Прямые ссылки используются без проверки и имеют такую же производительность, что и обычные указатели, но приложение должно избегать разыменования непреобразованных ссылок.

Средства объектно-реляционного преобразования

РСУБД по определению не имеют в своем составе средств обеспечения устойчивости объектов. Поэтому при разработке приложений с использованием объектно-ориентированной методологии, нуждающихся в обеспечении устойчивости объектов, встает вопрос о реализации данного свойства с помощью средств, предлагаемых РСУБД (то есть в рамках реляционной модели данных).

Основные сложности возникают при преобразовании отношений (ассоциация, наследование) между объектами в связи между таблицами, построенные на внешних ключах. Например, при преобразовании ассоциаций объектов вида «один-ко-многим» в реляционное представление необходимо учитывать, что именно «один» объект содержит коллекцию ссылок на «многие» объекты, а в РСУБД кортежи отношения «многие» включают в качестве внешнего ключа значение первичного ключа кортежа отношения «один». Таким образом, при сохранении графа объектов в РСУБД происходит преобразования прямой ассоциации в инверсную. Не меньшие сложности возникают и при переводе в реляционную модель и обратно отношений наследования.

Можно выделить две стратегии обеспечения устойчивости объектов в РСУБД. Во-первых, можно ввести в структуру приложения отдельный слой представления, который обеспечит персистентность объектов данного приложения в РСУБД. Во-вторых, можно использовать стороннее средство объектно-реляционного преобразования (СОРП), которых известно в настоящее время не менее 50.

Значительная часть объектно-ориентированных проектов так и не вышла из стадии разработки по причине невозможности обеспечить должным образом персистентность. В этих проектах было принято решение самостоятельно разрабатывать слой персистентности, в результате чего усилия разработчиков в основном тратились на поддержку кода устойчивости объектов, а не на логику функционирования приложений.

СОРП состоят, как правило, из двух компонентов: библиотеки времени выполнения, выполняющей прямое и обратное объектно-реляционное преобразование, и утилиты, позволяющей: а) смоделировать устойчивые объекты приложения и отношения между ними и б) выполнить объектно-реляционное преобразование и получить в результате схему данных для РСУБД. Результатом работы утилиты является карта преобразования, хранимая обычно в виде XML-файла.

Механизм действия большинства СОРП основан на создании средствами библиотеки времени выполнения прокси-объектов, то есть объектов-заместителей в оперативной памяти для объектов, находящихся в БД. При этом классы для этих объектов создаются «на лету» по результатам анализа карты преобразования.

Указание сохраняемости объектов достигается комбинацией объявления схемы данных (то есть указанием преобразования) и любого из двух оставшихся методов указания, рассмотренных выше.

Доступ к сохраняемым объектам в СОПР осуществляется только при помощи абстрактных ссылок, которые в некоторых случаях «маскируются» в исходном коде как прямые, за счет сокрытия деталей реализации доступа в библиотеке времени исполнения.