ER: диаграммы сущность – связь

Автор: Михаил Кривошеин (mikkri@mail.ru)
Статья была впервые опубликована в журнале «Программист» (www.programme.ru), март 2002 г.

В настоящее время большинство проектов информационных систем (ИС) разрабатывается в соответствии с какой-либо методологией разработки ПО. Как следствие, разработчикам требуется инструмент для моделирования данных на этапах анализа и проектирования. Таким инструментом являются ER-диаграммы (Entity-Relationship, «Сущность-Связь»). Фактически их использование является обязательным при разработке ИС, систем принятия решений, систем электронной торговли и B2B – большинства бизнес ориентированных систем.
ER-диаграммы позволяют строить модели логической структуры данных предметной области, а так же производить моделирование физической структуры систем хранения данных.

Основы моделирования схем данных

Логическая модель данных
Задача логической модели данных заключается в описании объектов данных предметной области и взаимосвязей между ними.
Рассмотрим пример. Пусть наша компания занимается продажей CASE-средств, и мы разрабатываем ИС для отдела продаж. При анализе требований была выявлена потребность в следующих функциях:

- Работа со справочником клиентов;
- Ведение каталога товаров с возможностью задания связей типа «Дополняет» между различными товарами. Например, BPWin дополняет ERWin;
- Регистрация счетов, ведение архива выставленных счетов;
- Регистрация заказов (накладных на продажу), ведение архива заказов;
- Помощник работы с клиентом (форма, отображающая список купленных клиентом программ, а так же список дополняющих программ, см. рис.№1);
- Помощник обновления программ (форма, отображающая список клиентов, купивших определенную программу, т.е. тех, кого следует уведомить о выходе нового дополнения или очередного service pack'а);

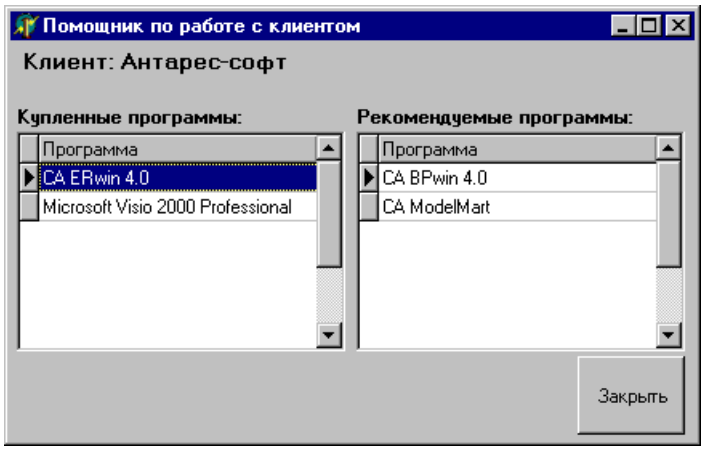

Рисунок №1. Помощник по работе с клиентом

На рисунке №2 отображена логическая модель данных описываемой системы. «Покупатель», «Заказ» и «Программа» – объекты данных. Линии между ними определяют наличие связей, а значки на концах – вид связи. Рассморим связь «Покупатель-Заказ». На рисунке она определена как «один ко многим», то есть каждый покупатель может сделать произвольное количество заказов, но любой заказ сделан в точности одним покупателем. Программа может быть куплена несколькими покупателями, но и покупатель может купить несколько программ, поэтому связь «Покупатель-Программа» обозначена как «многие ко многим».

Еще бывают связи «один к одному». С их помощью можно разделить сущность на несколько частей. Такая возможность может пригодиться для более точного задания прав доступа к данным или для повышения скорости работы СУБД. Частным случаем связи «один к одному» является связь «один к нулю или одному». Ее можно использовать при разделении сущности на две части – первая содержит атрибуты, обязательные к заполнению, вторая – необязательные, но занимающие значительное место в БД.

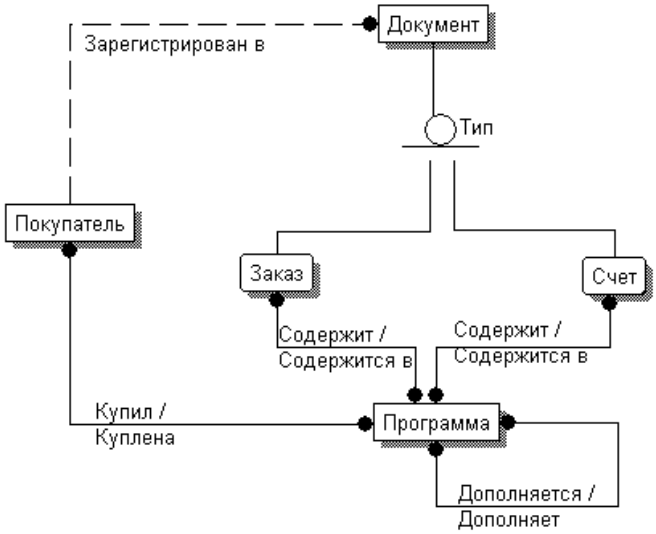

Рисунок №2. Логическая модель уровня сущностей

На рисунке есть интересная связь «Программа-Программа». Она означает, что программа может быть дополнена набором других программ, а так же то, что программа сама может функционально дополнять набор некоторых других программ. Эта связь типа «многие ко многим», помеченная ролями «Дополняется» и «Дополняет». Примером ее использования является существенная для ИС информация, что ERWin дополняется BPWin. Тогда программа сможет подсказать менеджеру по продажам, чтобы он напомнил клиенту, купившему ERWin, о возможностях по ее интеграции с BPWin (пока еще не купленной). Связи такого рода называются рекурсивными связями[1]. В нашем случае мы имеем сетевую рекурсию, так как объекты данных связаны отношением «многие ко многим». Рекурсивную связь «один ко многим» называют иерархической рекурсией. Для передачи смысла рекурсивных связей их рекомендуется преобразовывать к виду, изображенному на рисунке №3.

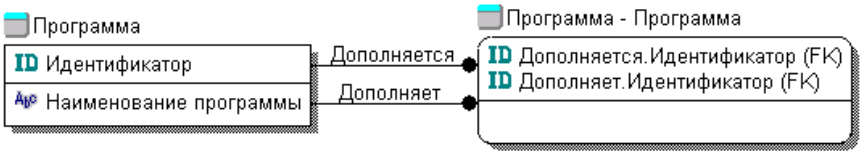

Рисунок №3. Преобразование рекурсивной связи

На рисунке №2 представлена ER-диаграмма логической модели данных с детализацией до уровня сущностей. Она несет информацию о составе объектов данных и существующих между ними связей. Но для полного описания модели требуется также описание атрибутов сущностей. При отображении логической модели данных в физическую модель атрибуты преобразовываются в поля таблиц. Атрибут задается наименованием, типом данных и, возможно, выполняемой ролью (см. рис. №3). Так на рисунке №4 отображена та же модель, но с указанием атрибутов сущностей. При этом тип атрибута отображен иконкой слева от наименования. Справа от атрибута могут отображаться суффиксы:

(FK) – атрибут является внешним ключом (Foreign Key).

(AKn.m) – атрибут входит в состав альтернативного ключа n в позиции m (Alternate Key). Альтернативные ключи являются основой для создания уникальных индексов.

(IEn.m) – атрибут входит в состав инверсионного входа n в позиции m (Inversion Entry). Фактически это означает наличие неуникального индекса по этому полю.

(O) – такой суффикс означает, что атрибут может не иметь значения (Optional).

[1] Также используется термин fish hook – рыбный крючок.

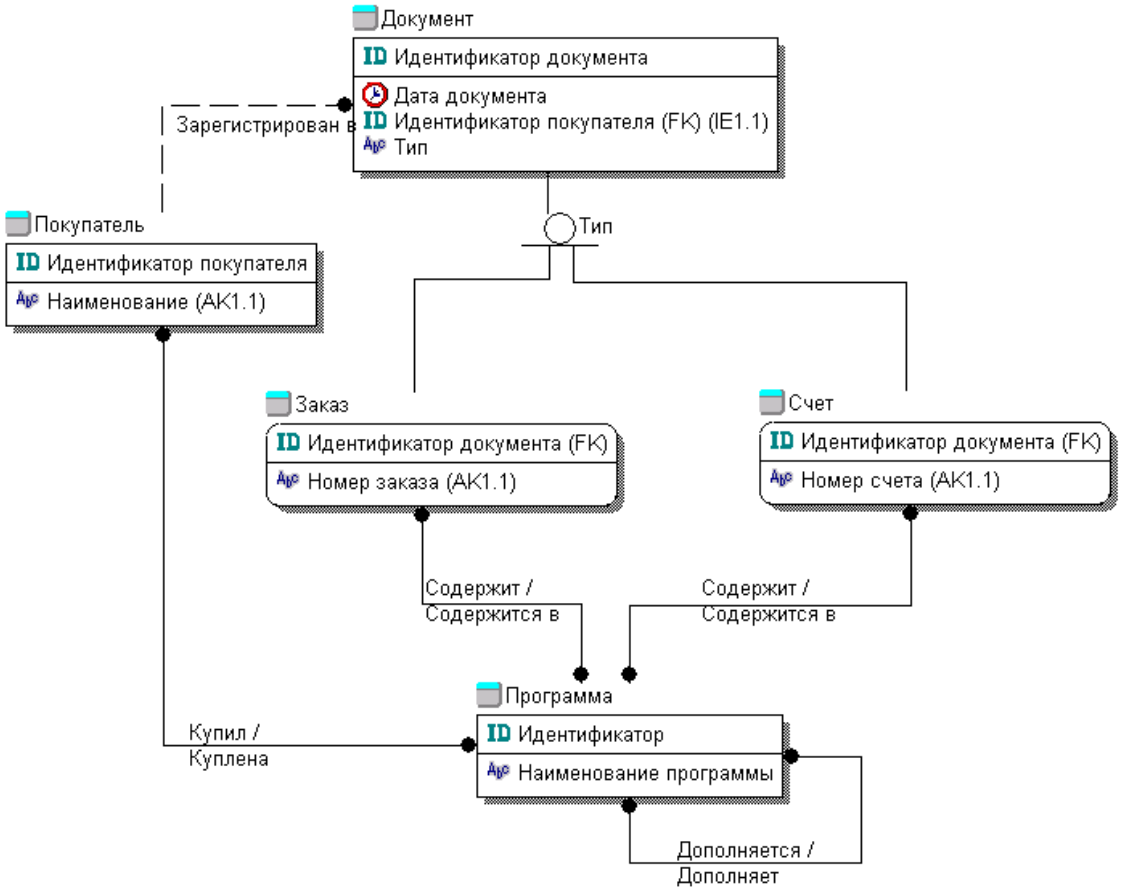

Рисунок №4. Логическая модель данных

Физическая модель данных

Логическая модель данных предметной области обеспечивает разработчикам понимание структур данных. После ее разработки следует приступать к моделированию физической структуры систем хранинения выявленных объектов данных, то есть к разработке физической модели данных.

По одной логической модели может быть построено несколько физических моделей – по одной для каждой поддерживаемой СУБД. Построение физической модели данных состоит из двух этапов:

- Нормализация модели данных.
- Денормализация модели данных.

Существует ряд правил организации структур данных, называемых нормальными формами. Нормализация – процесс приведения модели структуры данных к некоторой нормальной форме. Как правило, используется третья нормальная форма. Она обеспечивает эффективное и неизбыточное хранение данных. В таблице приведены требования, которым должна удовлетворять структура БД для того, чтобы быть в одной из первых трех нормальных форм. В процессе нормализации анализируется структура БД и выявляются элементы, противоречащие определенной нормальной форме. После этого разработчик меняет структуру так, чтобы устранить выявленные несоответствия.

1НФ	БД находится в 1НФ тогда и только тогда, когда поля всех таблиц содержат только атомарные значения и в таблицах нет полностью повторяющихся строк.
2НФ	БД находится в 2НФ тогда и только тогда, когда значение любого неключевого поля зависит от всего первичного ключа. Так же она должна находиться в 1НФ.
3НФ	БД находится в 3НФ тогда и только тогда, когда значение любого неключевого поля зависит только от значения первичного ключа, но не от значения другого неключевого поля. Так же она должна находиться во 2НФ.

В нашем случае в процессе нормализации логическая модель (рис. №4) преобразовалась в физическую модель данных (рис. №5). При этом были добавлены три таблицы-связки для связей «многие ко многим». Дополнительно в таблицы, хранящие информацию о составе заказов и счетов, было добавлено поле для информации о количестве приобретаемых лицензий.

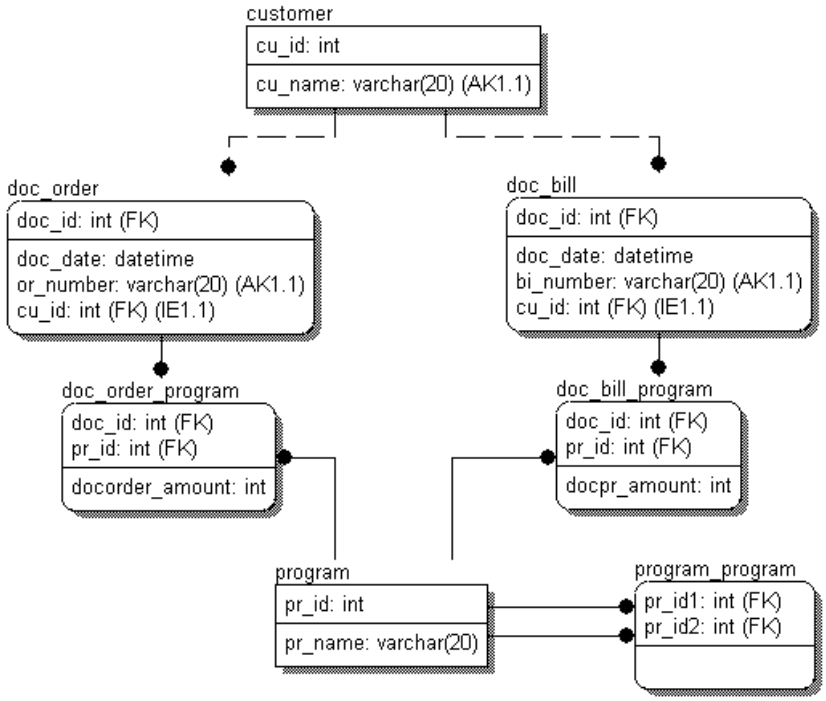

Рисунок №5. Физическая модель данных

Замечание: имена объектов данных принято называть существительными в единственном числе. При этом в физических моделях данных следует использовать только маленькие буквы и вместо пробела ставить нижнее подчеркивание.

Денормализация – процесс, обратный нормализации. Он заключается во внесении в структуру БД изменений, нарушающих требования нормальных форм. Основной целью денормализации является повышение производительности БД. Примером денормализации может служить модификация таблицы doc_order_program, показанная на рисунке №6. Так как связь «Покупатель-Программа» для нашей системы особенно важна, разумно ожидать большое количество запросов, связанных с ней. Но по нормализованной модели данных формирование списка программ, купленных заданным клиентом, возможно только при использовании оператора JOIN в SQL-запросе, а это пагубно сказывается на производительности, особенно если учесть, что объединяются две самые большие в БД таблицы. Но если мы в процессе денормализации сделаем копию поля cu_id из таблицы doc_order в таблице doc_order_program, такую информацию можно будет получить запросом всего лишь к одной таблице!

Как мы видим, роль ER-диаграмм столь велика, что без их использования не сможет обойтись большинство программных проектов. И, как следовало ожидать, в обоих наиболее популярных стандартах UML и IDEF* включены элементы, эквивалентные ER-диаграммам по назначению и выразительным средствам.

Нотации моделирования

Нотация IDEF1x

Семейство стандартов IDEF* (Integration Definition for Information Modeling) включает стандарт IDEF1x для описания объектов данных и их взаимосвязей. Первоначальная версия стандарта IDEF1 была опубликована в 1981 году и базировалась на работах Кодда (реляционная теория) и Чена (модели «Сущность-связь»). В последствии он был переработан и принят в разряд Федеральных стандартов обработки информации (США) 21 декабря 1993 года. Описание стандарта можно получить на www.idef.com.

В IDEF1x объекты данных подразделяются на зависимые и независимые. В соответствии с этим для обозначения зависимых сущностей используются прямоугольники со скругленными углами, а для независимых – обычные прямоугольники. Сущность называется зависимой, если ее экземпляр не может существовать вне рамок родительской сущности. Примером зависимой сущности является строка заказа. Объект данных «Покупатель» - независимая сущность. Связь от родительской сущности к зависимой называют идентифицирующей. Связь между двумя независимыми сущностями называют неидентифицирующей. Стандарт определяет понятие зависимого объекта данных несколько шире, чем следовало бы ожидать. Так, на рисунке №6 «Заказ» отмечен как зависимая сущность. В то же время он весьма даже самостоятелен. Дело в том, что «Заказ» является экземпляром абстрактной сущности «Документ», а значит, является зависимым от нее. Наименование сущности пишется над прямоугольником. Атрибуты перечисляются внутри прямоугольника. Атрибуты, образующие первичный ключ, отделяются от остальных атрибутов горизонтальной чертой.

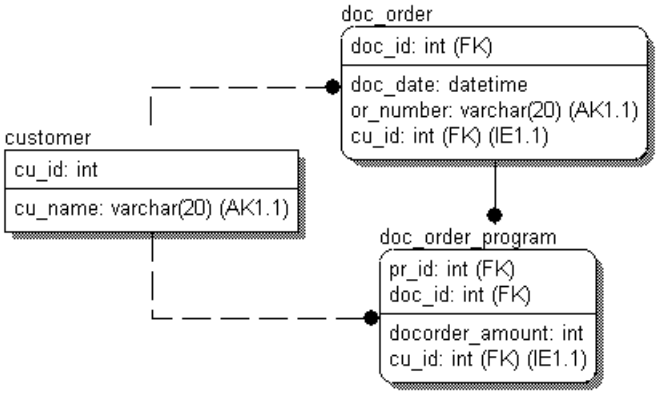

Рисунок №6. Пример диаграммы в IDEF1x

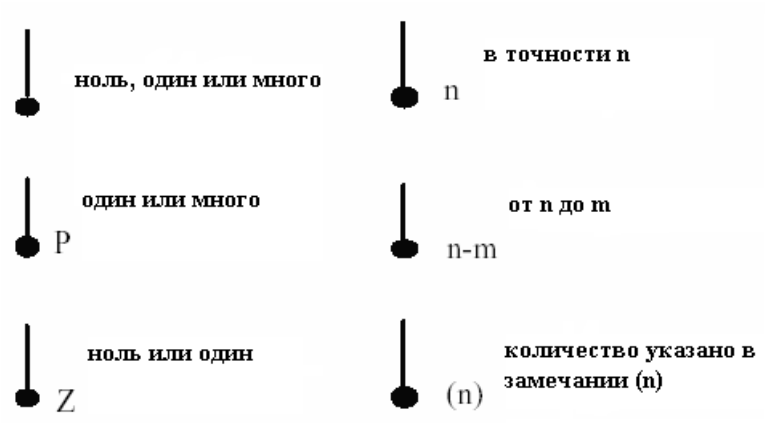

Рисунок №7. Обозначения, используемые в IDEF1x

В IDEF1x для обозначения связей между объектами данных используются линии. Отношение между сущностями определяется значками на концах линии (рис. №7). Так, на рисунке №6 отношение «Покупатель» - «Строка заказа» имеет тип «один к ноль, один или много». Как уже было сказано, связи бывают идентифицирующие и неидентифицирующие. Идентифицирующие связи обозначаются сплошной линией, неидентифицирующие – пунктирной. В связях «многие ко многим» кружок рисуется на обоих концах линии. В случае необязательной неидентифицирующей связи на конце стрелки рисуется незакрашенный ромб. Пусть мы дополнительно храним в БД информацию о всех полученных факсах. При этом факс может быть получен как от покупателя, так и от фирмы, не зарегистрированной в нашей БД. Связь между покупателем и факсом как раз и будет необязательной неидентифицирующей связью. Любая связь может помечаться ролями участвующих

Рисунок №8. Необязательная неидентифицирующая связь

сущностей. Причем связи «многие ко многим» должны помечаться описаниями отношения первой сущности ко второй и второй к первой, разделенных косой чертой, аналогично тому, как описана связь на рисунке №8.

Помимо отношения между объектами данных, нотация IDEF1x позволяет на уровне логической модели данных описывать иерархии наследования объектов данных (категориальные связи). Такая возможность используется для вынесения общих атрибутов в сущность родового предка, а также для указания на общность ряда сущностей, образующих категорию. При этом в общую сущность могут быть вынесены связи с другими объектами данных. В нашем примере мы имеем сущность «Документ» - родовой предок, категорию которого образуют сущности «Заказ» и «Счет». А связь между «Покупателем» и «Документом» означает наличие аналогичных связей для сущностей «Заказ» и «Счет». IDEF1x делит иерархии наследования на два типа – полные и неполные. Полная иерархия наследования обозначается значком  . Она означает, что все элементы категории определены на диаграмме. На рисунке №9 иерархия наследования неполная, а значит, к ней могут быть добавлены неопределенные в модели объекты данных, но обязательно удовлетворяющие всем свойствам родительской сущности (состав атрибутов, их связи с другими объектами данных).

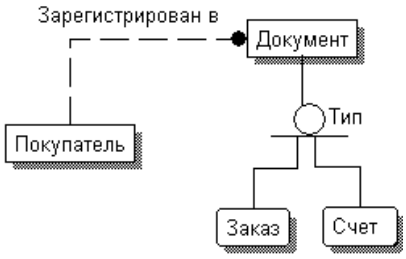

Рисунок №9. Иерархия наследования

Нотация UML

Unified Modeling Language (UML) – самая распространенная нотация, используемая в объектно-ориентированном анализе и проектировании (ООАП). Язык был создан как развитие ряда наиболее популярных методов ООАП. В данный момент его стандартизацией и развитием занимается Object Management Group (www.omg.org). Последняя версия UML 1.4 была опубликована осенью 2001 года.

Разумеется, UML предусматривает средства для моделирования объектов данных и связей между ними. Для моделирования структур данных используются диаграммы классов. Для обозначения объектов данных – классы с атрибутами, но без операций. Такой подход позволяет интегрировать диаграммы моделей данных с диаграммами классов служб данных. А для моделирования физической структуры данных можно использовать те же диаграммы классов и набор специализированных стереотипов.

Как уже было сказано, для описания объектов данных используются классы UML. При этом не делается различий между зависимыми и независимыми сущностями. Вся информация о связях между объектами данных описывается с помощью стрелок и прикрепленных к ним пояснений и комментариев. Так, кратность связи обозначается на конце стрелки интуитивно понятным образом («1» - в точности один, «0..n» - любое количество, «2..5» - от 2 до 5). Также можно указать роль, в которой выступает объект, участвующий в связи (термин роль в UML отличается от аналогичного в IDEF1x, где роль означает не роль сущности в связи, а роль связи). При этом линия означает возможность навигации по связи в обе стороны, а стрелка – только в одну. На рисунке №10 обозначена возможность получить информацию о покупателе, располагая информацией о документе. Это значит, что основное назначение связи – получать информацию о покупателе, зная данные документа. А отбор документов по заданному покупателю не является основным назначением рассматриваемой связи. Внимания заслуживает обозначение необязательной связи в UML. На конце линии от факса к покупателю указывается «0..1», что означает, что объект данных «Факс» может, но не обязан ссылаться на объект данных «Покупатель».

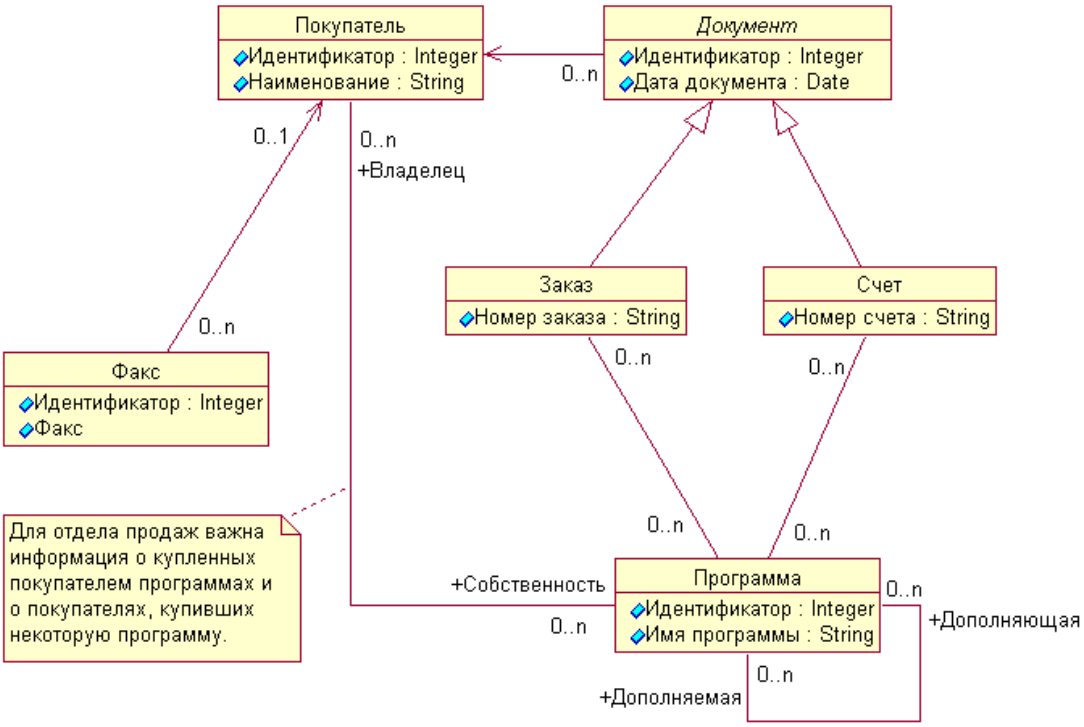

Рисунок №10. Пример модели данных в UML

При необходимости обозначить идентифицирующую связь двух сущностей используют связь агрегации (см. пример на рис. №18). В остальных случаях используют обычные связи ассоциации. UML предоставляет возможность определить класс ассоциации. В результате можно яснее описать связь между объектами данных. Еще одна возможность уточнить связь – указать атрибут, участвующий в связи.

Атрибуты сущностей описываются в целом аналогично IDEF1x. Но при этом есть возможность указать значение по умолчанию для атрибута прямо в его описании, как это сделано для количества лицензий в строке заказа на рисунке №11.

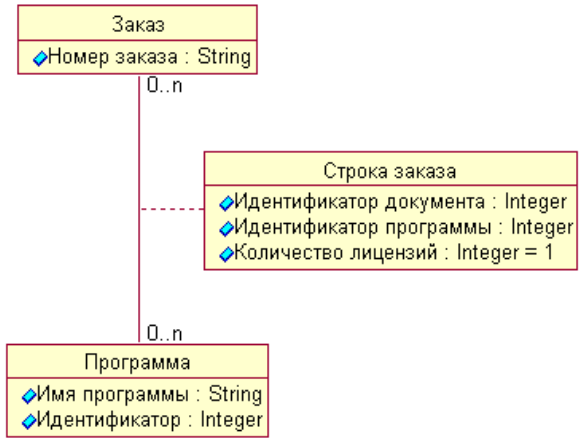

Рисунок №11. Спецификация связи сущностей

При описании иерархий наследования сказывается ориентация UML на объектный подход. На диаграмме указывается связь наследования, но при этом не указывается дескриминатор (идентификатор типа потомка) и в ней не участвуют прочие потомки родительской сущности. В то время как в IDEF1x иерархии наследования используются для задания категорий сущностей, в UML они используются только для обозначения наследования атрибутов. Разумеется, связи родительской сущности наследуются потомками. В описываемом примере используется реляционная модель данных, а поэтому в ней фактически отсутствуют сущности «Документ». Для обозначения этого факта можно указать, что сущность «Документ» - абстрактная (название сущности нарисовано курсивом).

Приятной и полезной возможностью являются текстовые комментарии прямо на диаграммах.
Для описания физической модели данных UML предоставляет механизм стереотипов. Например, для описания таблиц можно использовать классы со стереотипом <<Table>>, а для обозначения первичных и внешних ключей – стереотипы <<PK>>, <<PK, FK>> и <<FK>>. При этом можно

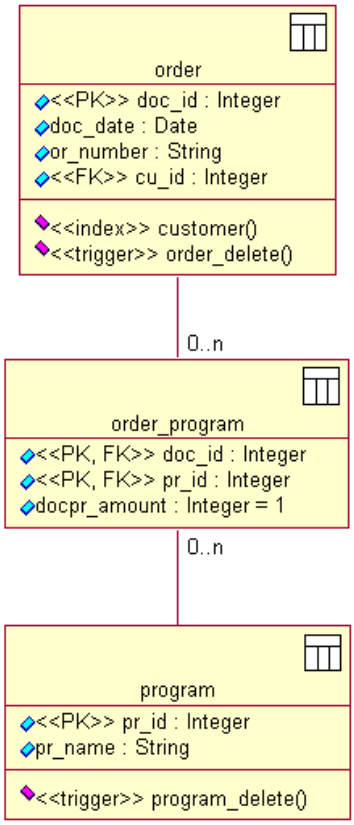

Рисунок №12. Использование стереотипов UML

использовать специфику UML для определения на той же диаграмме дополнительных индексов и триггеров. Для этого их следует описать как «методы классов» со стереотипами <<index>> и <<trigger>> соответственно (см. рис. №12).

Итак, мы описали назначение ER-диаграмм, их разработку с помощью двух наиболее часто используемых стандартов. Теперь речь пойдет об инструментальных средствах, их поддерживающих.

Инструментальные средства моделирования

Microsoft Visio

Профессиональная редакция Microsoft Visio предоставляет богатые возможности по разработке различных схем. В частности она поддерживает IDEF1x и UML. Но при этом следует учитывать, что Visio ориентирована на бизнес-пользователя и не является CASE-средством. Фактически это удобный и эффективный инструмент для подготовки моделей в презентационных целях, либо для использования в проектах с простой структурой данных.

Режим работы с ER-диаграммами наиболее полно реализован при использовании трафарета Entity Relationship, поддерживающего нотации Relational и IDEF1x. Содержательно они ничем не отличаются, но при использовании первой из них диаграммы выглядят гораздо симпатичнее (см. рис. №13).

Использование Visio существенно облегчается функцией обратной инженерии. В частности поддерживается генерация модели по БД (для большинства современных БД, включая DB2, Access, SQL Server и Oracle) и синхронизация модели с БД. Правда, попытка совместить прямую инженерию БД ERWin с функцией синхронизации Visio на рассматриваемом примере успехом не увенчалась. Функции прямой инженерии (генерация структуры БД или ее обновление) не поддерживаются.

В целом программу отличает очень удобный пользовательский интерфейс и возможность подготовки «интуитивно понятных» моделей структуры данных, в том числе с использованием обратной инженерии структуры БД.

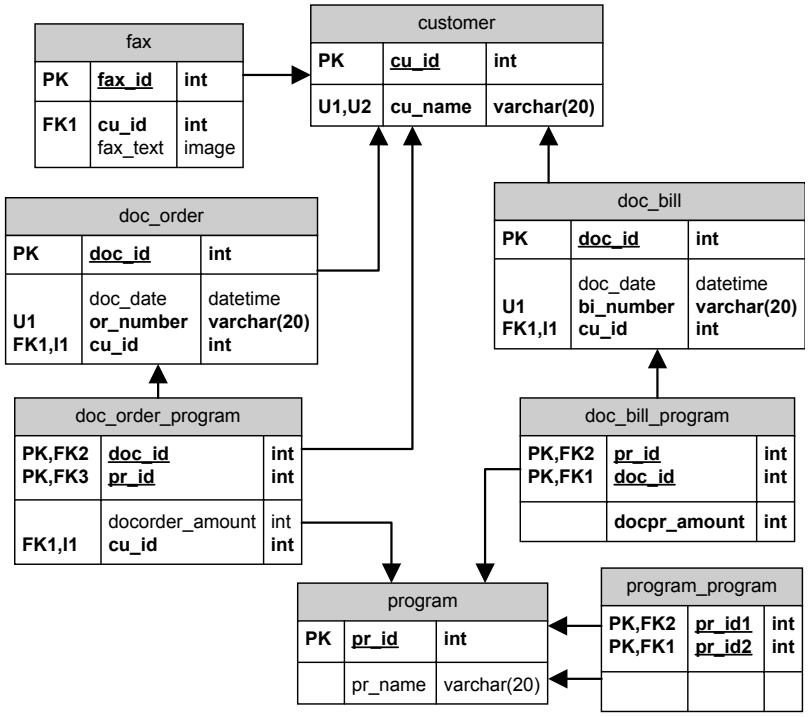

Рисунок №13. Модель данных в Visio

Computer Associates ERWin
Для нотации IDEF1x наиболее популярным CASE-средством является программа ERWin компании Computer Associates.

Проект в ERWin состоит из логической модели данных и физической модели данных для выбранной СУБД. При редактировании моделей происходит автоматическая синхронизация, хотя есть возможность указать, что элемент модели относится только к логическому или физическому уровню.

К достоинствам программы относится поддержка большого количества СУБД (более 20, включая все основные), работа с хранимыми отображениями (отображение подмножества модели на отдельной диаграмме), ориентация на моделирование структур БД. В частности, ERWin поддерживает задание новых типов данных, работу с текстом триггеров (включая автоматическую генерацию триггеров ссылочной целостности) и сохраненных процедур, ввод ограничений на содержимое полей.

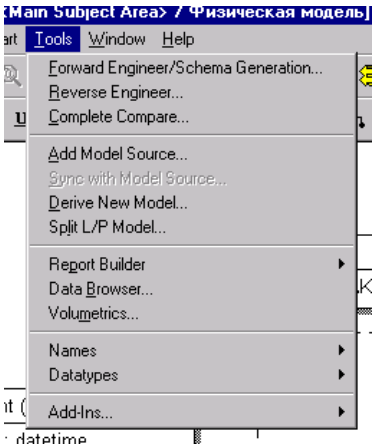

Рисунок №14. Меню ERWin

Но главным достоинством, бесспорно, являются богатые возможности прямой и обратной инженерии структуры БД. Так, ERWin может сгенерировать структуру БД по физической модели, создать физическую модель по источнику данных, поддерживается синхронизация модели и структуры БД и генерация модели физической структуры данных для другой БД.

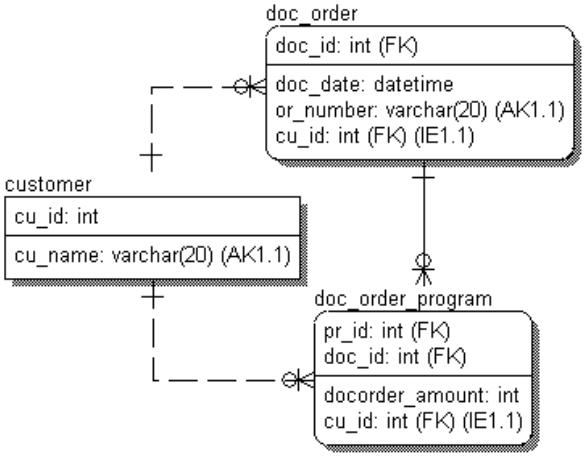

Рисунок №15. Модель в нотации ER

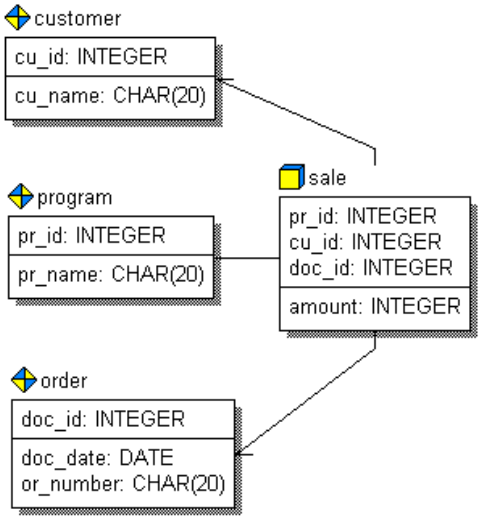

Рисунок №16. Модель хранилища данных

Помимо IDEF1x ERWin поддерживает нотации Information Engineering (см.рис.№15) и, для физической структуры данных, нотацию Dimensional Modeling. Последняя предоставляет разработчику возможность создавать модели хранилищ данных. На основе описываемого в статье примера может быть сформировано следующее хранилище данных: одна таблица факта sale и три таблицы размерности: customer, program и order. На рисунке №16 изображена модель хранилища данных, подготовленная в ERWin.

При выборе CASE-системы для моделирования структуры данных важно учесть тесную интеграция ERWin с BPWin – лучшим CASE-средством, поддерживающим нотации, используемые при структурном анализе (IDEF0, DFD, IDEF3).

Rational Rose

Rose - это CASE-система лидера технологий ООАП Rational. Как уже было сказано при описании UML, для работы с моделями данных можно использовать стандартные средства языка – диаграммы классов, классы, атрибуты, методы и расширяющие их стереотипы. Такой подход можно успешно применять при разработке логической структуры данных, которая получается тесно интегрированной в объектную модель разрабатываемого ПО. Но для моделирования физической структуры данных указанный подход крайне неэффективен. В первую очередь из-за того, что не предусматривает средств прямой и обратной инженерии, а также учета специфики конкретной СУБД и технологий БД в целом.

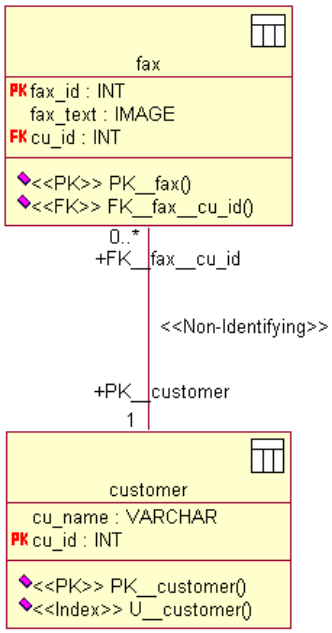

Рисунок №17. Неидентифицирующая связь в Rational Rose

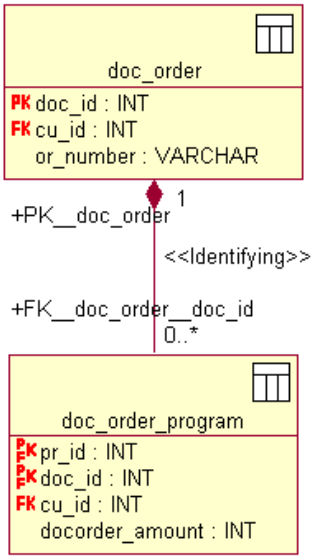

Рисунок №18. Идентифицирующая связь в Rational Rose

Для устранения перечисленных недостатков Rational выпустила расширение Data Modeler, дополняющее Rational Rose средствами для прямой и обратной инженерии структуры БД. К сожалению, синхронизация в Data Modeler не предусмотрена, хотя есть возможность сравнить модель со структурой БД. Результаты сравнения записываются в текстовый файл. Крайне интересна возможность генерации объектной модели (логической модели данных) по модели физической структуры данных.

Для моделирования структуры БД используются пакеты со стереотипом <<Schema>> (см. рис. №19). Объекты таких пакетов обслуживаются Data Modeler'ом, поэтому содержат нестандартные диаграммы Data Model (аналог диаграмм классов) и таблицы (аналог классов). Внимание заслуживает то, что индексы и триггеры отображаются прямо на диаграмме в позиции операций класса. Например, на рисунке №17 U__customer – уникальных индекс, FK__fax__cu_id – триггер ссылочной целостности для внешнего ключа cu_id.

Data Modeler предусматривает два стереотипа для связей между объектами данных. Для неидентифицирующих связей используются связи ассоциации со стереотипом <<Non-Identifying>>. Для идентифицирующих – связи агрегации со стереотипом <<Identifying>>, как показано на рисунке №18.

Ключевым достоинством Rational Rose является поддержка UML, обеспечивающая разработчика всеми необходимыми инструментами для анализа и проектирования. Расширение Data Modeler позволяет автоматизировать генерацию объектной модели разрабатываемого ПО за счет обратной инженерии структуры БД, а также дополнить модель в Rational Rose информацией о физической структуре используемой БД. Более того, благодаря Data Modeler разработчики могут ограничить набор используемых CASE-средств продуктами Rational.

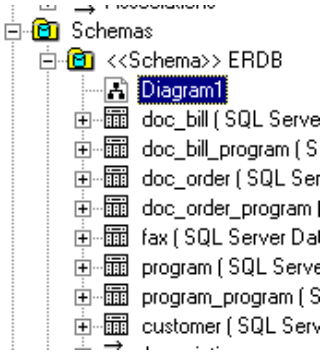

Рисунок №19. Навигация по дереву объектов в Rational Rose

Заключение

В заключение можно сказать, что у обоих рассмотренных подходов есть свои сильные и слабые стороны. Так, IDEF1x является общепринятой нотацией для моделирования схем данных, а ERWin предоставляет богатые возможности по ее использованию. В свою очередь UML располагает богатым набором выразительных средств, а Rational Rose позволяет работать с целостной моделью разрабатываемого ПО, что существенно облегчает использование CASE-средства.

Важным критерием для выбора является используемая методология. При использовании структурного анализа лучшим решением будет комплекс из ERWin и BPWin, а при объектно-ориентированном подходе – пакет Rational Rose с Data Modeler. В то же время достаточно эффективной может быть комбинация ERWin для проектирования БД и Rational Rose для работы над клиентским ПО.

Я надеюсь, статья помогла вам сориентироваться в мире современных средств моделирования структур данных. Вопросы и комментарии прошу направлять мне на mikkri@mail.ru.