AllFusion ERwin Data Modeler

проектирование баз данных,
хранилищ данных, витрин данных

AllFusion ERwin Data Modeler (ранее: ERwin) позволяет проектировать, документировать и сопровождать базы данных, хранилища данных и витрины данных (data marts). Создав наглядную модель базы данных, вы сможете оптимизировать структуру БД и добиться её полного соответствия требованиям и задачам организации. Визуальное моделирование повышает качество создаваемой базы данных, продуктивность и скорость её разработки. Подробнее >>

Кому нужен AllFusion ERwin Data Modeler 4.1: всем компаниям, разрабатывающим и использующим базы данных. Администраторам баз данных, системным аналитикам, проектировщикам БД, разработчикам, руководителям проектов.

Аргументы и факты:

- поддерживается прямое (создание БД на основе модели) и обратное (генерация модели по имеющейся базе данных) проектирование для 20 типов СУБД.
- увеличивает производительность труда благодаря удобному интерфейсу и автоматизации рутинных процедур.
- поддерживает методологию структурного моделирования SADT и следующие нотации: IDEF1x, IE, Dimensional (последняя - для проектирования хранилищ данных).
- ERwin является стандартом де-факто
- поддерживает 20 различных СУБД: настольные, реляционные и специализированные СУБД, предназначенные для создания хранилищ данных.
- интегрирован линейкой продуктов Computer Associates для поддержки всех стадий разработки ИС, CASE-средствами Oracle Designer, Rational Rose, средствами разработки и др.
- позволяет повторно использовать компоненты созданных ранее моделей, а также использовать наработки других разработчиков. Повышается эффективность!
- возможна совместная работа группы проектировщиков с одними и теми же моделями (с помощью AllFusion Model Manager 4.1).
- позволяет переносить структуру БД (не сами данные!) из СУБД одного типа СУБД в другой
- позволяет документировать структуру БД.
- продукт легко освоить. Есть курсы по ERwin.
- в России тысячи пользователей ERwin. Так, книга по ERwin и BPwin Сергея Маклакова, руководителя УЦ Interface Ltd., разошлась тиражом 15 тыс.
- продукт можно использовать на всех стадиях жизненного цикла баз данных: при проектировании, разработке, тестировании и поддержке.

Дополнительные аргументы для разработчиков ПО:

- позволяет получить точную и наглядную информацию, где хранятся данные и как получить к ним доступ.
- позволяет, используя визуальные средства, описать структуру БД, а затем автоматически сгенерировать файлы данных для любого типа СУБД.

Дополнительные аргументы для администраторов баз данных:

- позволяет максимально повысить производительность информационной системы благодаря поддержке работы с БД на физическом уровне, учитывая особенности каждой конкретной СУБД.
- уменьшит число рутинных операций, облегчит и сократит работу.

Дополнительные аргументы для руководителей:

- повышает гибкость и эффективность организации за счёт возможности быстрой адаптации базы данных к меняющимся потребностям рынка.
- использование профессиональных средств - фактор конкурентной борьбы.

Дополнительные аргументы для руководителей проектов:

- ERwin поможет тщательно задокументировать структуру БД
- позволит получить отчеты презентационного качества
- через AllFusion Model Manager возможно эффективное управление ходом проектирования.

Дополнительные аргументы для крупных компаний:

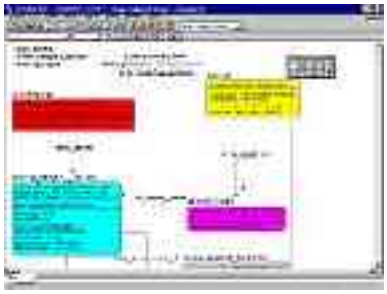
- документирование собственной ИС. Если этого не делать, может возникнуть путаница.
- при переходе на другие СУБД - нет проблем.

Дополнительные аргументы для системных интеграторов:

- Поддержка различных типов СУБД (ERwin Data Modeler поддерживает СУБД более 20 производителей).
- поможет подстроиться под изменяющиеся требования заказчика.

Материалы по продукту:

Вводные материалы, знакомящие с продуктом:



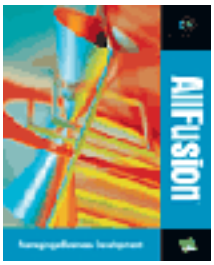
- Описание последней версии продукта
- ERwin на службе у банков: проектирование хранилищ данных
- ERwin - современное средство проектирования баз данных
- Моделирование баз данных при помощи ERwin
- Объединение структурного и объектного подхода в новом поколении CASE-средств Computer Associates
- По разным банкам я бродил, и мой ERwin со мною
- Использование ERwin при создании приложений для Oracle

Технические материалы: Использование расширенных функциональных возможностей AllFusion ERwin Data Modeler - продукта компании Computer Associates

- Интеграция AllFusion ERwin Data Modeler с AllFusion Component Modeler
- Использование языка макрокоманд в AllFusion ERwin Data Modeler
- AllFusion ERwin Data Modeler API и ERwin Spy Utility
- AllFusion ERwin Data Modeler API – это проще, чем кажется
- Генерация отчетов по вашим моделям ERwin
- Понятие отношения
- Основные компоненты диаграммы ERwin - сущности, атрибуты, связи. Часть 2. Понятие атрибута
- Основные компоненты диаграммы ERwin - сущности, атрибуты, связи. Часть 1. Понятие сущности
- Обновленные возможности AllFusion ERwin Data Modeler 4.1
- Базовые концепции моделирования данных

Оглавление

IlFusion ERwin Data Modeler 4.1



Компания **Computer Associates (CA)** представляет новую версию **AllFusion ERwin Data Modeler 4.1**. Этот продукт предназначен для проектирования, внедрения и поддержки баз данных, хранилищ данных и моделей данных масштаба предприятия.

Основным требованием, предъявляемым проектировщиками баз данных к решениям для моделирования данных является поддержка полного жизненного цикла разработки приложения. AllFusion ERwin Data Modeler - это мощное и удобное в использовании средство моделирования данных и проектирования баз данных, которое эффективно работает на любой стадии жизненного цикла разработки, включая проектирование, генерацию кода и поддержку приложений базы данных.

Поддерживаемые среды

Операционная система:

- Windows 95/98/2000/NT 4.0/XP

Базы данных:

- Advantage Ingres Enterprise Relational Database
- Advantage CA-Clipper
- DB2, dBASE
- FoxPro
- HiRDB
- Informix
- InterBase
- Microsoft Access
- Teradata
- Microsoft SQL Server
- ODBC 2.0, 3.0
- Oracle
- Paradox
- Rdb
- Red Brick Warehouse
- SAS
- SQL Anywhere
- SQL Base
- Sybase

Преимущества для пользователей AllFusion ERwin Data Modeler:

- Увеличенная продуктивность благодаря удобной в использовании графической среде, которая упрощает проектирование баз данных и автоматизирует многие трудоемкие задачи. AllFusion ERwin Data Modeler ускоряет процесс создания высококачественных и высокопроизводительных баз данных и хранилищ данных.
- Эффективное общение между администраторами баз данных и разработчиками благодаря совместному и повторному использованию моделей, а также графическому отображению громоздких и сложных массивов корпоративных данных в удобном для понимания и сопровождения формате.
- Быстрое реагирование на меняющиеся нужды бизнеса благодаря улучшенному пониманию влияния изменения свойств информации в масштабе всей организации и облегченному быстрому внедрению этих изменений.

Возможности AllFusion ERwin Data Modeler:

- Многоуровневая архитектура проектирования. Наряду с традиционной комбинированной логической и физической моделью, AllFusion ERwin Data Modeler поддерживает и раздельные логическую и физическую модели. AllFusion ERwin Data Modeler поддерживает сведения об отношениях и полной истории проектирования, а также позволяет

пользователю быстро оценить влияние одного уровня на другой.

- Технология трансформации. Физический дизайн базы данных редко совпадает с ее изначальным логическим дизайном. Ограничения, накладываемые бизнесом, диктуют необходимость денормализации таблиц для соблюдения требований по производительности. Технология трансформации, примененная в AllFusion ERwin Data Modeler, позволяет вносить такие изменения, как денормализация, одновременно с поддержкой целостности изначального дизайна.
- Определяющие стандарты. AllFusion ERwin Data Modeler поддерживает определение и поддержку стандартов через доменный словарь (Domain Dictionary), редактор стандартов именования (Naming Standards Editor) и редактор стандартов типов данных (Datatype Standards Editor). Доменный словарь содержит многократно используемые атрибуты и обеспечивает целостность имен и определений в процессе проектирования базы данных. Редактор стандартов наименований позволяет пользователю создавать словарь допустимых слов, аббревиатур и правил наименования, которые можно многократно использовать в модели данных масштаба предприятия. Редактор стандартов типов данных позволяет пользователю определять стандарты соответствия определенным типам данных СУБД как для определяемых пользователем типов данных, так и для типов данных по умолчанию.
- Управление большими моделями. AllFusion ERwin Data Modeler облегчает управление большими моделями благодаря использованию предметных областей (Subject Areas) и хранимых изображений (Stored Displays). Предметные области предоставляют отдельным проектировщикам подробное представление модели, что достигается разбиением модели на небольшие управляемые подмножества. Хранимые изображения предлагают множество графических представлений модели и ее предметных областей, облегчая специализированным группам пользователей обмен информацией.
- Полное сравнение (Complete Compare). Эта передовая технология автоматизирует синхронизацию модели и базы данных. Она сравнивает модель с базой данных, показывает отличия между ними, и позволяет пользователю выбрать, какое отличие отразить в модели, а какое - внести в базу данных. Если изменения в модели переносятся в базу данных, то AllFusion ERwin Data Modeler автоматически сгенерирует сценарий изменения базы.
- Генерация дизайна базы данных. AllFusion ERwin Data Modeler располагает оптимизированными шаблонами триггеров ссылочной целостности и богатым макроязыком, пригодным для различных баз данных, для обеспечения настройки триггеров и хранимых процедур. AllFusion ERwin Data Modeler генерирует из физического дизайна модели полные определения следующих объектов (с поправками на конкретную целевую базу данных): базы данных, табличные пространства, таблицы, представления, столбцы со значениями по умолчанию, доменные ограничения, первичные и внешние ключи, индексы, хранимые процедуры, триггеры, размеры объектов и другие физические параметры.
- Проектирование хранилищ и витрин данных. Производительность, удобство, простота использования и полезность хранилища данных определяются лежащим в его основе дизайном. Для оптимизации соответствия характеристик хранилищ данных требованиям производительности и задачам пользователей AllFusion ERwin Data Modeler предоставляет специфичные для хранилищ данных техники моделирования, например, схему звезды (Star Schema) и моделирование измерений в виде снежинки (Snowflake dimensional modeling). Также AllFusion ERwin Data Modeler собирает и документирует широкий спектр информации о хранилище данных, включая источники данных, логику трансформации данных и правила управления данными.

"ERwin настолько прост, что нового специалиста можно обучить ему за день-другой, и настолько многофункционален, что соответствует нашим самым взыскательным потребностям".
Л. Хенден, Pricewaterhouse Coopers

ERwin на службе у банков, проектирование хранилищ данных

В Древней Греции тогдашние банкиры давали ответ на площади в присутствии всех граждан, а затем цифры выбивались на камне. Представьте себе, сколько камней потребовалось бы для ведения записей современным банкам! Сегодня для хранения и обработки информации в банковском бизнесе применяются передовые достижения в области IT-технологий - хранилища данных, OLAP, B2B. В этой статье мы рассмотрим процесс создания и использования хранилищ данных с помощью средства ERwin компании Computer Associates.

Что такое Data Warehouse?

Хранилище данных (Data Warehouse) - это отдельная база данных, в которой аккумулируется вся самая разнообразная информация, необходимая менеджерам банка для подготовки управленческих решений: о клиентах банка, операционных днях филиалов, кредитах, процентных ставках, курсах валют и т. д. При этом хранилище оснащено инструментами для быстрой и несложной настройки на новые виды данных, то есть оно может непрерывно развиваться.

В целях экономии времени руководителей любая запрошенная информация предоставляется очень быстро. Для этого в хранилище содержатся заранее вычисленные показатели, например обороты балансовых счетов за день, квартал, год.

Огромные объемы данных хранилища легко использовать за счет того, что в хранилище изначально встроены удобные инструменты поиска информации, средства оперативного анализа (OLAP) и генераторы отчетов. Хранилище снабжено мощной системой загрузки данных из разных источников, при этом в процессе загрузки происходит автоматическое согласование и очистка данных от ошибок.

Из опыта создания хранилищ данных

К необходимости создания хранилищ данных российские банки пришли уже давно. Как правило, хранилища данных оперируют с огромными объемами информации, что предъявляет к их проектированию и реализации повышенные требования. Выбор в качестве платформы хранилища данных такой высокопроизводительной РСУБД позволяет существенно повысить общую эффективность создаваемой информационной системы. Для этих целей используются мощные инструменты графического проектирования информационных систем - так называемые CASEсредства (CASE расшифровывается как Computer Aided System Engineering), например ERwin компании Computer Associates.

В создании хранилищ данных ERwin становится незаменимым инструментом, поскольку, с одной стороны, эффективно поддерживает на физическом уровне проектирование объектов РСУБД, с другой стороны, имеет специализированные средства моделирования хранилищ данных. Ниже рассматриваются основные возможности ERwin по проектированию хранилищ

данных.

К проектированию хранилищ данных обычно предъявляются следующие требования:

- Структура данных хранилища должна быть понятна пользователям.
- Должны быть выделены статистические данные, которые регулярно модифицируются: ежедневно, еженедельно, ежеквартально.
- Требования к запросам должны быть упрощены с целью исключения запросов, которые могли бы требовать множественных утверждений SQL в традиционных реляционных СУБД.
- Должна быть обеспечена поддержка сложных запросов SQL, которые требуют последовательной обработки тысяч или миллионов записей.

Именно выполнение этих требований отличает структуру хранилищ данных от структуры реляционных СУБД и хранилищ данных. Нормализация данных в реляционных СУБД приводит к созданию множества связанных между собой таблиц. В результате выполнение сложных запросов неизбежно влечет за собой объединение многих таблиц, что существенно увеличивает время отклика. Проектирование хранилища данных подразумевает создание денормализованной структуры данных (допускается избыточность данных и возможность возникновения аномалий при манипулировании данными), ориентированной в первую очередь на высокую производительность при выполнении аналитических запросов. Нормализация делает модель хранилища слишком сложной, затрудняет ее понимание и ухудшает эффективность выполнения запроса.

Как работает ERwin?

Размерная (Dimensional) модель. Для эффективного проектирования хранилищ данных ERwin использует размерную модель. Размерная модель - это методология проектирования, специально предназначенная для разработки хранилищ данных. Наиболее простой способ перейти к нотации размерной модели при создании новой модели (меню File/New) в диалоге ERwin Teamplate Selection - выбрать из списка предлагаемых шаблонов DIMENSION. В шаблоне DIMENSION сделаны все необходимые для поддержки нотации размерного моделирования настройки, которые, впрочем, можно установить вручную.

Моделирование Dimensional сходно с моделированием связей и сущностей для реляционной модели, но отличается целями. Реляционная модель акцентируется на целостности и эффективности ввода данных. Размерная модель ориентирована в первую очередь на выполнение сложных запросов к БД.

Роль таблицы в схеме (Dimensional Modeling Role). По умолчанию ERwin автомати чески определяет роль таблицы на основании созданных связей. Таблица без связей определяется как таблица размерности, таблица факта не может быть родительской в связи, таблица размерности может быть родительской по отношению к таблице факта, консольная таблица может быть родительской по отношению к таблице размерности.

Правила хранения данных (Data Warehouse Rules). Для каждой таблицы можно задать шесть типов правил работы с данными: обновление (Refresh), дополнение (Append), резервное копирование (Backup), восстановление (Recovery), архивирование (Archiving) и очистка (Purge). Для задания правила следует выбрать имя правила из соответствующего списка выбора. Каждое правило должно быть предварительно описано в диалоге Data Warehouse Rule Editor. Для каждого правила должно быть задано имя, тип, определение. Например,

определение правила дополнения данных может включать частоту и время дополнения (ежедневно, в конце рабочего дня), продолжительность операции и т. д. Связать правила с определенной таблицей можно с помощью диалога Table Editor.

При проектировании хранилища данных важно определить источник данных (для каждой колонки), метод, которым исходные данные извлекаются, преобразовываются и фильтруются, прежде чем они импортируются в хранилище данных. Хранилище данных может объединять информацию из текстовых файлов и многих баз данных, как реляционных (в том числе других БД на платформе Informix), так и нереляционных, в единую систему поддержки принятия решений. Чтобы поддерживать регулярные обновления и проверки качества данных, необходимо знать источник для каждой колонки в хранилище данных. Для документирования информации об источниках данных используется редактор Data Warehouse Source Editor.

ERwin — современное средство проектирования баз данных

Обработка счетов. Электронная торговля. Анализ данных. Управление знаниями. Все это невозможно без использования баз данных. Системы с архитектурой клиент/сервер строятся на основе реляционных серверных СУБД. Приложения для Internet и интрасетей осуществляют доступ и динамическое обновление данных. Пакеты программ необходимо адаптировать и интегрировать с существующими системами. Хранилища данных объединяют и интегрируют множество баз данных, обеспечивая необходимые бизнесу гибкость и интеллектуальность. Успех применения всех этих приложений зависит от того, насколько хорошо спроектирована база данных.

PLATINUM ERwin - мощное и простое в использовании средство конструирования баз данных завоевавшее широкое признание и популярность. Оно обеспечивает высочайшую продуктивность труда при разработке и сопровождении приложений с использованием баз данных.

На протяжении всего процесса - от логического моделирования требований к информации и бизнес-правил, которые определяют базу данных, до оптимизации физической модели в соответствии с заданными характеристиками - ERwin позволяет наглядно отобразить структуру и основные элементы вашей БД.

ERwin - это не просто мощное средство проектирования, но и инструмент разработки, способный автоматически создавать таблицы и генерировать тысячи строк текста хранимых процедур и триггеров для всех популярных СУБД. Революционная технология Complete-Compare (Завершить-Сравнить) позволяет организовать итеративную разработку, поддерживая постоянную согласованность модели и базы данных. Благодаря интеграции с популярными средами разработки программ, ERwin позволяет ускорить создание приложений для обработки данных.

ERwin может масштабироваться путем интеграции с продуктом PLATINUM ModelMart. Эта мощная система управления моделями позволяет проектировщикам баз данных разработчикам приложений и пользователям коллективно работать с информацией о моделях ERwin. Благодаря возможностям разбиения на фрагменты, а также совместного и многократного использования моделей, может быть повышена эффективность моделирования и обеспечено соблюдение корпоративных стандартов.

Стандартизация моделирования и проектирования

ERwin облегчает проектирование баз данных. Для этого достаточно создать графическую E-R модель (объект-отношение), удовлетворяющую всем требованиям к данным и ввести бизнес-правила для создания логической модели, которая отображает все элементы, атрибуты, отношения и группировки. Вы можете расширить возможности Erwin воспользовавшись уникальной поддержкой пользовательских свойств для ввода в модель любой дополнительной информации, значимой для вашей деятельности.

Развитые средства моделирования помогают лучше спроектировать базу данных. Предусмотрены возможности манипулирования атрибутами путем их буксировки, внесения изменений и нормализации "на лету". Средства редактирования непосредственно на диаграммах позволяют вносить в модель изменения, не открывая специальных диалоговых

окон. Навигация по отношениям обеспечивает быстрое перемещение в больших моделях для перехода к родительским или дочерним объектам. Формируемые системой отчеты позволяют быстро проверить корректность спроектированной базы данных.
ERwin - это не что гораздо большее, чем просто инструмент для "рисования"; он автоматизирует процесс проектирования. Например, ERwin предусматривает возможность создания каталога наиболее часто используемых атрибутов, что обеспечивает согласованность имен и описаний по всему проекту. Представления БД поддерживаются как интегрированные компоненты модели, что позволяет автоматически отображать в их описаниях изменения, внесенные в базовые таблицы. Автоматический перенос ключей обеспечивает ссылочную целостность базы данных.
Кроме того, ERwin позволяет работать с большими моделями общекорпоративного масштаба, разбивая их на фрагменты и легко управляемые подмножества, предоставляя отдельным специалистам возможность сосредоточить свои усилия в определенной области. Возможность сохранения отображений позволяет хранить множество представлений одной предметной области, ориентированных на различную целевую аудиторию.
Созданные с помощью ERwin модели данных можно редактировать, просматривать и распечатывать различными способами. В состав ERwin входит RPTwin - простая в использовании, оснащенная графическим интерфейсом утилита для формирования отчетов и встроенное средство для просмотра с настраиваемыми режимами ,которые обеспечивают полный контроль над отображением содержимого отчетов. Кроме этого, уникальный интерфейс, построенный на использовании шаблонов, позволяет реализовать единые стандарты проектирования и отображать настройки для всех моделей.

Автоматическая генерация БД

ERwin - не только лучший инструмент для проектирования баз данных, но и средство для их быстрого создания. ERwin оптимизирует модель в соответствии с физическими характеристиками целевой базы данных. В отличие от других инструментальных средств ERwin автоматически поддерживает согласованность логической и физической схем и осуществляет преобразование логических конструкций, таких как отношения многие-ко-многим, в их реализацию на физическом уровне.
ERwin устанавливает естественную динамическую связь между моделью и базой данных, что позволяет реализовать как прямой, так и обратный инжиниринг. Используя эту связь, ERwin автоматически генерирует таблицы, представления, индексы, правила поддержания целостности ссылок (первичных и внешних ключей), устанавливает значения по умолчанию и ограничения для доменов/столбцов. В состав ERwin включен целый ряд оптимизированных шаблонов триггеров, обеспечивающих целостность ссылок, и мощный макроязык, который позволяет создавать собственные триггеры и хранимые процедуры. Таким образом могут быть автоматически сформированы тысячи строк кода, что обеспечивает непревзойденную продуктивность разработки на основе моделей.
Средства расчета объема позволяют точно оценить первоначальный размер и характер роста базы данных или хранилища, облегчая эффективное распределение ресурсов системы и планирование мощности.
База данных может быть спроектирована и создана без написания отдельных SQL-предложений типа CREATE TABLE или INDEX. Поскольку физическая схема формируется на основе описательной логической модели, ваше приложение будет сразу же полностью документировано. ERwin позволяет также проводить обратный инжиниринг существующих баз данных путем построения модели непосредственно на основе ее таблиц. Таким образом можно получить четкое представление о структуре и содержании существующего

приложения.
ERwin поддерживает все наиболее популярные реляционные СУБД, включая Oracle, Microsoft SQL Server, Sybase, DB2 и Informix. Одна и та же модель может быть использована для создания нескольких баз данных или для переноса приложения с платформы одной СУБД на другую.

Быстрая разработка приложений

ERwin интегрирует проектирование базы данных в процесс разработки приложения. Благодаря возможностям взаимодействия с популярными средствами разработки для архитектуры клиент/сервер и Web, ERwin поддерживает соответствие между серверной базой данных и формами в клиентской части, позволяя ускорить создание высококачественных приложений.

Моделирование баз данных при помощи ERwin

Аржан Кинжалин
Опубликовано в журнале "Компьютер Информ"

В предыдущей статье, мы рассказали об инструменте системного анализа и моделирования бизнес-процессов - BPwin. Следующим логическим этапом при создании программной системы, автоматизирующей эти бизнес-процессы, будет разработка соответствующей структуры данных. Представляем сегодня последнюю версию инструмента моделирования баз данных – CA/Logic Works ERwin 3.5.

Структура данных и будущее приложение

Ни одну область деятельности человека, поддерживаемую информационными технологиями, невозможно представить себе без использования баз данных, помогающих получить быстрый доступ к информации, увеличивая тем самым продуктивность работы. Клиент-серверные приложения, получившие в последнее время широкое распространение, построены на основе баз данных; приложения Internet и intranet могут получать доступ к базам данных, открывая широкие возможности для публикации информации, необходимой широкому кругу пользователей. Большинство клиент-серверных систем в данный момент представляют собой приложения по оперативной обработке транзакций (On-Line Transaction Processing, OLTP), которые служат для быстрой обработки и сохранения данных. Примерами таких приложений могут служить системы выписки счетов, регистрации и учета продукции и т.п. В то же время в последние годы значительное внимание уделяется построению хранилищ данных (data warehousing) - это базы данных специального назначения, складирующие всю информацию предприятия. Хранилища данных лежат в основе так называемых систем оперативного анализа данных (On-Line Analisys Processing, OLAP), которые позволяют принимать решения и помогают планировать стратегию развития предприятия. Успех любого приложения зависит от того, насколько хорошо смоделирована и разработана база данных приложения, поэтому разработке базы данных необходимо уделить много внимания.

База данных создается в несколько этапов, на каждом из которых необходимо согласовывать структуру данных с заказчиком и, что самое важное, подвергать созданную структуру данных экспертизе внутри команды, которая создает систему. Поэтому представление данных должно быть простым и понятным всем заинтересованным лицам. Именно по этой причине, наибольшее распространение получило представление базы данных под названием "сущность-отношение" (entity-relationship), которое также известно как ER-диаграмма. Модели, представленные в виде ER-диаграмм, крайне просты и удобны для понимания. Фрагмент такой модели изображен на рис. 1.

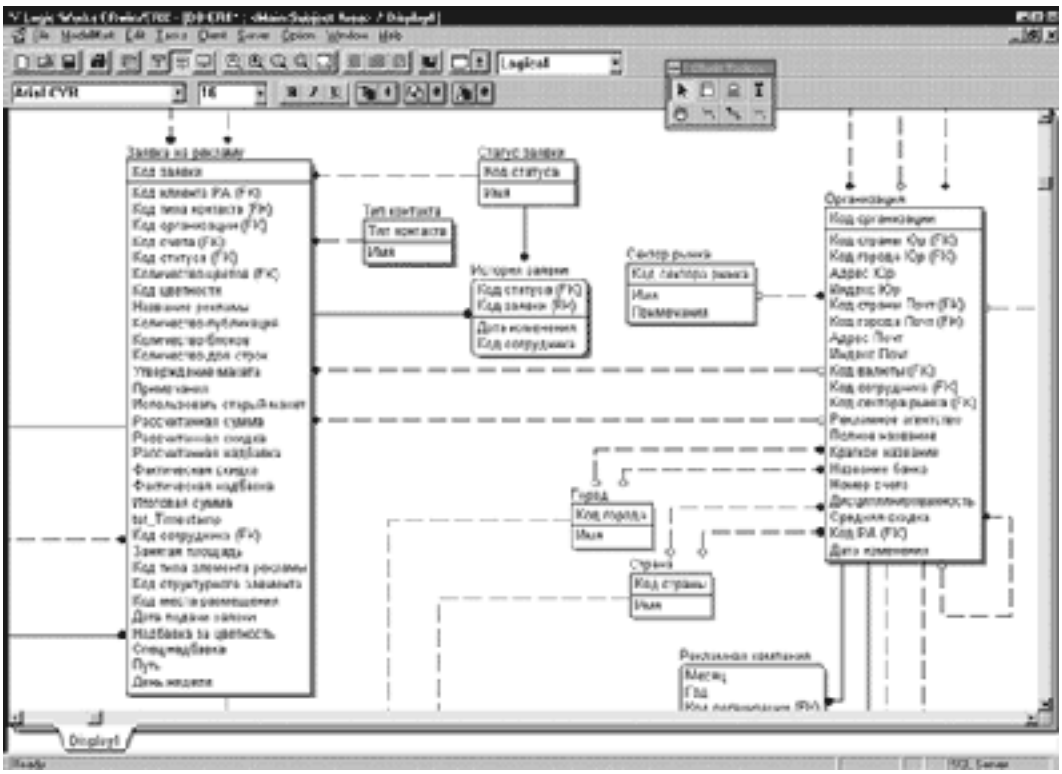


РИСУНОК astr1_1.gif. Модель базы данных разрабатываемого приложения, представленная в виде ER-диаграммы, построенной в среде ERwin

ER-диаграммы были приняты в качестве основы для создания стандарта IDEF1X. Предварительный вариант этого стандарта был разработан в военно-воздушных силах США и предназначался для увеличения производительности при разработке компьютерных систем. В 1981 году этот стандарт был формализован и опубликован организацией ICAM (Integrated Computed Aided Manufacturing), и с тех пор является наиболее распространенным стандартом для создания моделей баз данных по всему миру.

С развитием компьютерных технологий и появлением CASE-моделирования (Computer Aided Software Engineering) возникла потребность в инструментах, которые бы поддерживали стандарты моделирования. Современный инструмент моделирования баз данных должен удовлетворять ряду требований.

- Позволять разработчику сконцентрироваться на самом моделировании, а не на проблемах с графическим отображением диаграммы. Инструмент должен автоматически размещать сущности на диаграмме, иметь развитые и простые в управлении средства визуализации и создания представлений модели.
- Инструмент должен проверять диаграмму на согласованность, автоматически определяя и разрешая несоответствия. Однако инструмент должен быть настраиваемым и при желании предоставлять разработчику некоторую свободу в действиях и право самому разрешать несоответствия или отступления от методологии.
- Инструмент моделирования должен поддерживать как логическое, так и физическое моделирование.
- Современный инструмент должен автоматически генерировать базу данных на СУБД назначения.

Все современные инструменты моделирования в той или иной степени удовлетворяют перечисленным выше общим требованиям, однако в этой статье речь пойдет об инструменте моделирования баз данных ERwin версии 3.5, продукте компании CA/Logic Works. Выбор инструмента не случаен, т.к. на нынешний момент ERwin является наиболее мощным средством для разработки структуры данных как на логическом, так и на физическом уровне. Следует отметить, что существует несколько модификаций ERwin, каждая из которых, помимо моделирования, предназначена для выполнения специфических целей. Здесь мы рассмотрим ERwin3.5/ERX, который предназначен для работы именно с системами управления базами данных. Остальные члены семейства ERwin предназначены для использования с инструментами разработки клиентской части приложения, такими, как Power Builder, Visual Basic и прочими. Продукт CA/Logic Works ERwin 3.5 был выпущен в феврале этого года и сразу же получил признание широкого круга пользователей за многие усовершенствования по сравнению с предыдущими версиями, которые в него были внесены. Этот инструмент моделирования полностью поддерживает стандарт IDEF1X и является лидером на рынке инструментов разработки баз данных.

Разработка в среде ERwin

Обычно разработка модели базы данных состоит из двух этапов: составление логической модели и создание на ее основе физической модели. ERwin полностью поддерживает такой процесс, он имеет два представления модели: логическое (logical) и физическое (physical). Таким образом, разработчик может строить логическую модель базы данных, не задумываясь над деталями физической реализации, т.е. уделяя основное внимание требованиям к информации и бизнес-процессам, которые будет поддерживать будущая база данных. ERwin имеет очень удобный пользовательский интерфейс, позволяющий представить базу данных в самых различных аспектах. Например, ERwin имеет такие средства визуализации как "хранимое представление" (stored display) и "предметная область" (subject area). Хранимые представления позволяют иметь несколько вариантов представления модели, в каждом из которых могут быть подчеркнуты определенные детали, которые вызвали бы перенасыщение модели, если бы они были помещены на одном представлении. Предметные области помогают вычленить из сложной и трудной для восприятия модели отдельные фрагменты, которые относятся лишь к определенной области, из числа тех, что охватывает информационная модель. Интерфейс среды разработки ERwin представлен на рисунке.

Возможности редактирования и визуализации в среде ERwin весьма широки, так, например, создание отношений возможно при помощи перетаскивания атрибута из одной сущности в другую. Такое редактирование модели позволяет вносить изменения и проводить нормализацию быстрее и эффективнее, чем с использованием других инструментов. Для того, чтобы добавить новый элемент на диаграмму, его просто нужно выбрать на панели инструментов (Toolbox) и перенести в нужное место диаграммы. Добавив новую сущность на диаграмму, в нее можно добавить атрибуты, не открывая никаких редакторов, а просто ввести их названия прямо на диаграмме. Таким образом, ERwin позволяет значительно снизить время на создание самой диаграммы и сконцентрироваться на самих задачах, стоящих перед разработчиком.

ERwin имеет мощные средства визуализации модели, такие, как использование различных шрифтов, цветов и отображение модели на различных уровнях, например, на уровне описания сущности, на уровне первичных ключей сущности и т.д. Эти средства ERwin значительно помогают при презентации модели в кругу разработчиков системы или сторонним лицам.

Возможность использования модели ERwin одновременно для логического и физического представления данных позволяет по окончании работы получить полностью документированную модель.ERwin, как и инструмент моделирования бизнес-процессов BPwin, интегрирован с генератором отчетов фирмы CA/Logic Works - RPTwin. Это средство позволяет получать подробные отчеты по модели, освещая самые различные ракурсы и аспекты. Инструмент RPTwin поставляется вместе с ERwin и имеет богатый набор встроенных отчетов, позволяющих получать многогранную информацию по модели. Документирование структуры данных является очень важной частью моделирования, т.к. это позволяет другим разработчикам или лицам, которые будут сопровождать систему, быстрее начать ориентироваться во внутренней структуре и понимать назначение компонентов.

Как уже говорилось, ERwin является не только инструментом для дизайна баз данных, он также поддерживает автоматическую генерацию спроектированной и определенной на физическом уровне структуры данных. ERwin 3.5 поддерживает широчайший спектр серверных и настольных СУБД. В этот список входят такие продукты, как Microsoft SQL Server, Oracle, Sybase, DB2, INFORMIX, Red Brick, Teradata, PROGRESS, Microsoft Access, FoxPro, Clipper и многие другие. Для каждой из перечисленных СУБД в ERwin предусмотрено присоединение по "родному" для этой СУБД протоколу и поддержка всех средств управления данными, присущих этой СУБД. Инструмент имеет богатый и гибкий макроязык, позволяющий создавать сценарии (pre- и postscripts), которые будут выполняться до и после генерации определенного объекта на СУБД назначения. С помощью этого макроязыка можно также сгенерировать на СУБД назначения тысячи строк шаблонов, хранимых процедур и триггеров. ERwin не поддерживает моделирования механизмов защиты базы данных, однако при помощи макроязыка можно автоматически выдать права на объект, пользуясь языком определения прав, который используется в конкретной СУБД.

ERwin имеет средство, выполняющее задачу, обратную генерации, что называется "обратная разработка" (reverse engineering). Т.е. ERwin может присоединиться к СУБД, получить всю информацию о структуре базы данных и отобразить ее в графическом интерфейсе, сохранив все сущности, связи, атрибуты и прочие свойства. Таким образом, можно переносить существующую структуру данных с одной платформы на другую, а также исследовать структуру существующих баз данных.

Но и это еще не все. ERwin имеет средство Complete-Compare, которое является единственным на данный момент средством интерактивной разработки. ERwin демонстрирует разногласия между моделью и базой данных, эти несоответствия можно переносить или оставлять без изменений. При помощи этого средства можно все изменения модели вносить в базу данных автоматически без необходимости контроля за соответствием модели и базы данных "вручную", при этом существующие данные не будут затронуты.

Начиная с версии 3.5 ERwin, поддерживает многомерное моделирование, которое используется при построении хранилищ данных. Производительность OLAP-приложений определяется, в основном, качеством дизайна хранилища данных, поэтому критически важно при разработке хранилища иметь инструмент, который бы поддерживал распространенные технологии. ERwin поддерживает две технологии моделирования хранилищ данных: звезда (star) и снежинка (snowflake).

ERwin тесно интегрирован с другими продуктами CA/Logic Works. Словарь данных, созданный при анализе бизнес-процессов при помощи инструмента BPwin, может быть использован как основа для построения модели базы данных. Однако взаимосвязь между

этими двумя инструментами двусторонняя, модели BPwin и ERwin можно постоянно поддерживать в согласованном состоянии. Интеграция этих двух продуктов очень важна с точки зрения их совместного использования при разработке программного обеспечения, т.к. отпадает необходимость в повторном выполнении действий и процесс создания словаря данных становится практически автоматическим.

Мы рассмотрели лишь незначительную часть средств, которые облегчают и помогают разработку в среде ERwin; остальные средства выходят за рамки данной статьи и являются предметом более специального и детального обсуждения.

Коллективная разработка

При коллективной разработке базы данных возникает проблема синхронизации моделей, созданных различными разработчиками. Такая проблема особенно актуальна, когда над одной моделью работает большое количество разработчиков. Для эффективного решения этой проблемы CA/Logic Works предлагает продукт ModelMart, который представляет собой репозитарий для хранения моделей. Этот репозитарий имеет средства защиты модели, позволяет гибко управлять различными моделями или версиями модели, предоставляет единый активный словарь данных, которым можно централизованно управлять и использовать его в нескольких моделях сразу.

ModelMart не зависит от используемой платформы и прост в установке и администрировании. Использование такого коллективного репозитария позволяет масштабировать разработку моделей с помощью ERwin до масштабов предприятия. ModelMart также может предоставлять репозитарий для хранения и управления моделями BPwin, начиная с версии 2.02. Этот факт также говорит в пользу разработки информационных систем с использованием BPwin и ERwin.

Заключение

Моделирование играет большую роль в разработке успешных информационных систем. Вашему вниманию в этой и предыдущей статьях были предложены два инструмента компании CA/Logic Works: BPwin и ERwin. Использование этих продуктов совместно поможет правильно оценить стоящие задачи, предложить адекватное решение (анализ бизнес-процессов, BPwin) и разработать центральную часть любой информационной системы - базы данных - с использованием информации, полученной во время обследования предприятия (моделирование базы данных, ERwin). Эти инструменты сами по себе не являются решением проблемы, но их грамотное и своевременное использование поможет свести рутинный труд разработчика к минимуму, позволит ему сконцентрироваться на собственно разработке системы и снизит потери времени, которые обычно происходят при согласовании моделей со специалистами предметной области. Кроме того, использование этих инструментов дает возможность получить набор полностью документированных и согласованных моделей, что в значительной степени облегчит поддержку созданных систем в будущем, а также может быть повторно использовано при разработке других систем.

Объединение структурного и объектного подхода в новом поколении CASE-средств Computer Associates

Сергей Маклаков,

Руководитель Учебно-консалтингового центра

В статье комментируются различные положения, высказанные авторами статей "Rational Rose, BPwin и другие — аспект анализа бизнес-процессов" и "Пример описания предметной области с использованием Unified Modeling Language (UML) при разработке программных систем"

Создание систем автоматизации предприятий является очень сложной задачей. В технологическом цикле создания программного обеспечения принято выделять следующие этапы [1]:

- анализ - определение того, что система будет делать,
- проектирование - определение подсистем и их взаимодействие,
- реализация - разработка подсистем по отдельности, объединение - соединение подсистем в единое целое,
- тестирование - проверка работы системы,
- установка - введение системы в действие,
- функционирование - использование системы.

В [1] показано, что наиболее критичными являются ранние этапы создания информационных систем – этап анализа и этап проектирования, поскольку именно на этих этапах могут быть допущены наиболее опасные и дорогостоящие ошибки. Существуют различные методологии и CASE-средства, обеспечивающие автоматизацию этих этапов. Такие CASE-средства должны выполнять следующие задачи:

1. Построение модели бизнес-процессов предприятия и анализ этой модели, в том числе стоимостной анализ (ABC) и анализ эффективности бизнес-процессов с помощью имитационного моделирования.
2. Создание структурной модели предприятия и связывание структуры с функциональной моделью. Результатом такого связывания должно быть распределение ролей и ответственности участников бизнес-процессов.
3. Описание документооборота предприятия.
4. Создание сценариев выполнения бизнес-функций, подлежащих автоматизации и полного описание последовательности действий (включающее все возможные сценарии и логику развития).
5. Создание сущностей и атрибутов и построение на этой основе модели данных.
6. Определение требований к информационной системе и связь функциональности информационной системы с бизнес-процессами.
7. Создание объектной модели, на которой в дальнейшем может быть автоматически сгенерирован программный код.
8. Интеграция с инструментальными средствами, обеспечивающими поддержку групповой разработки, системами быстрой разработки, средствами управления проектом, средствами управления требованиями, средствами тестирования,

средствами управления конфигурациями, средствами распространения и средствами документирования.

Практика показывает, что одна отдельно взятая нотация или инструмент не могут в полной мере удовлетворить всем перечисленным требованиям. Новое поколение CASE-средств фирмы Computer Associates (CA) представляет собой набор связанных между собой инструментальных средств, в полной мере обеспечивающих решение всех задач анализа, проектирования, генерации, тестирования и сопровождения информационных систем.

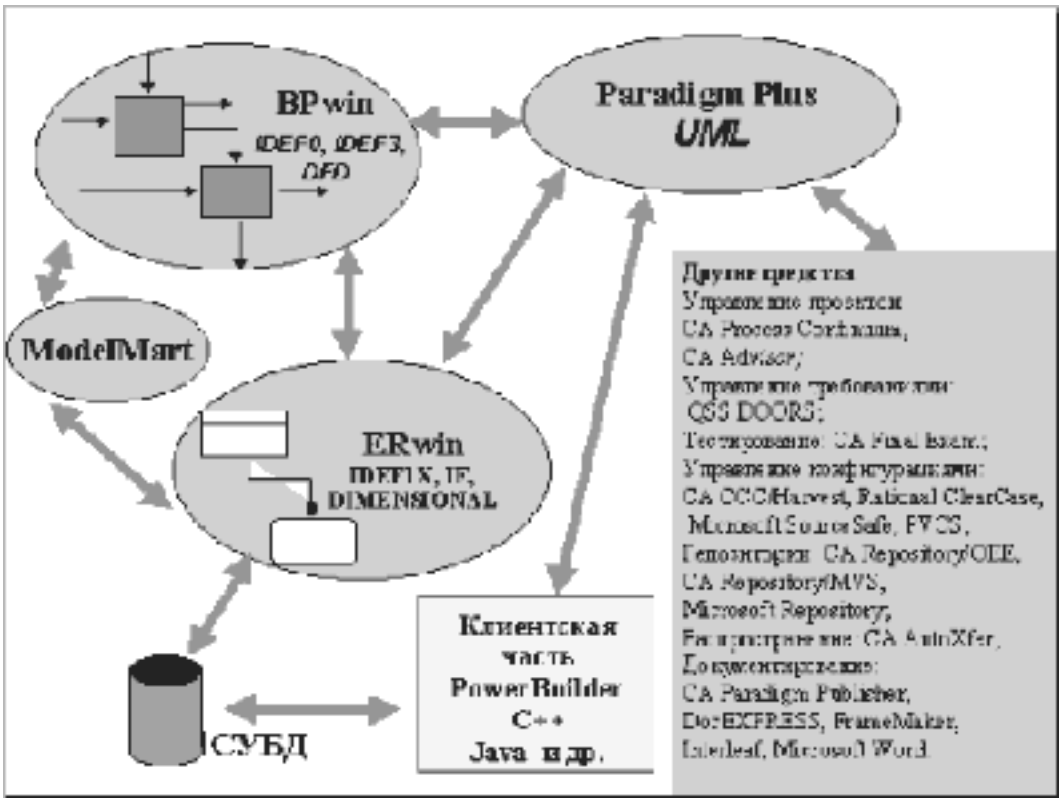


Рис.1. Схема взаимодействия CASE-средств Computer Associates.

Так, новая версия инструмента построения функциональных моделей BPwin 4.0 [2, 3] позволяет решить задачи, перечисленные в пунктах 1-4 и, частично,5. BPwin позволяет создавать модели процессов и поддерживает три стандарта (нотации) моделирования - IDEF0, DFD и IDEF3. Каждая из трех нотаций, поддерживаемых в BPwin, позволяет рассмотреть различные стороны деятельности предприятия.

Модель IDEF0 предназначена для описания бизнес-процессов на предприятии, она позволяет понять, какие объекты или информация служат сырьем для процессов, какие результаты производят работы, что является управляющими факторами и какие ресурсы для этого необходимы. Методология структурного моделирования предполагает построение модели AS-IS (как есть), анализ и выявление недостатков существующих бизнес-процессов и построение модели TO-BE (как должно быть), то есть модели, которая должна использоваться при построении автоматизированной системы управлением предприятия.

Нотация IDEF0 позволяет наглядно представить бизнес-процессы и легко выявить такие недостатки как недостаточно эффективное управление, ненужные, дублирующие, избыточные или неэффективные работы, неправильно использующиеся ресурсы и т.д. При этом часто выясняется, что обработка информации и использование ресурсов неэффективны,

важная информация не доходит до соответствующего рабочего места и т.д. Признаком неэффективной организации работ является, например, отсутствие обратных связей по входу и управлению для многих критически важных работ. Встроенная система стоимостного анализа (ABC) позволяет количественно оценить стоимость каждой работы и эффективность реализации той или иной технологии.

Диаграммы потоков данных (Data flow diagramming, DFD) используются для описания документооборота и обработки информации. DFD описывают функции обработки информации, документы, объекты, а также сотрудников или отделы, которые участвуют в обработке информации. Наличие в диаграммах DFD элементов для описания источников, приемников и хранилищ данных позволяет более эффективно и наглядно описать процесс документооборота.

Для описания логики взаимодействия информационных потоков более подходит IDEF3, называемая также workflow diagramming, - нотация моделирования, использующая графическое описание информационных потоков, взаимоотношений между процессами обработки информации и объектов, являющихся частью этих процессов. Диаграммы IDEF3 позволяют описать как отдельные сценарии реализации бизнес-процессов, так и полное описание последовательности действий. Диаграммы нового типа - Swim Lane, использующие методологию Process Flow Network и могут быть добавлены в модель, содержащую диаграммы IDEF3. Диаграммы Swim Lane иллюстрируют несколько параллельных потоков, что позволяет отобразить процесс вместе с зависящими от него процессами как параллельные потоки на одной диаграмме (рис.2). Кроме того, на диаграммах Swim Lane можно указать роли исполнителей работ, тем самым более качественно задокументировать роли и ответственности.

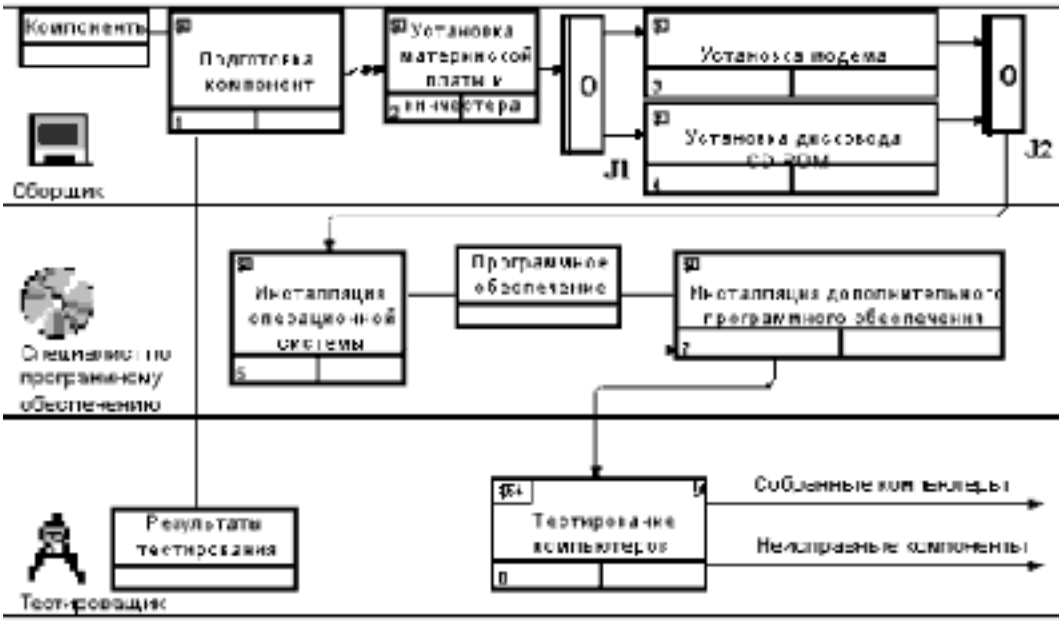


Рис.2. Распределение ролей при выполнении работ на диаграмме Swim Lane.

Организационные диаграммы (organization charts) позволяют описать структуру предприятия и создаются на основе предварительно созданных ролей. Благодаря организационным диаграммам можно отобразить как структуру организации, так и любую другую иерархическую структуру (рис.3).

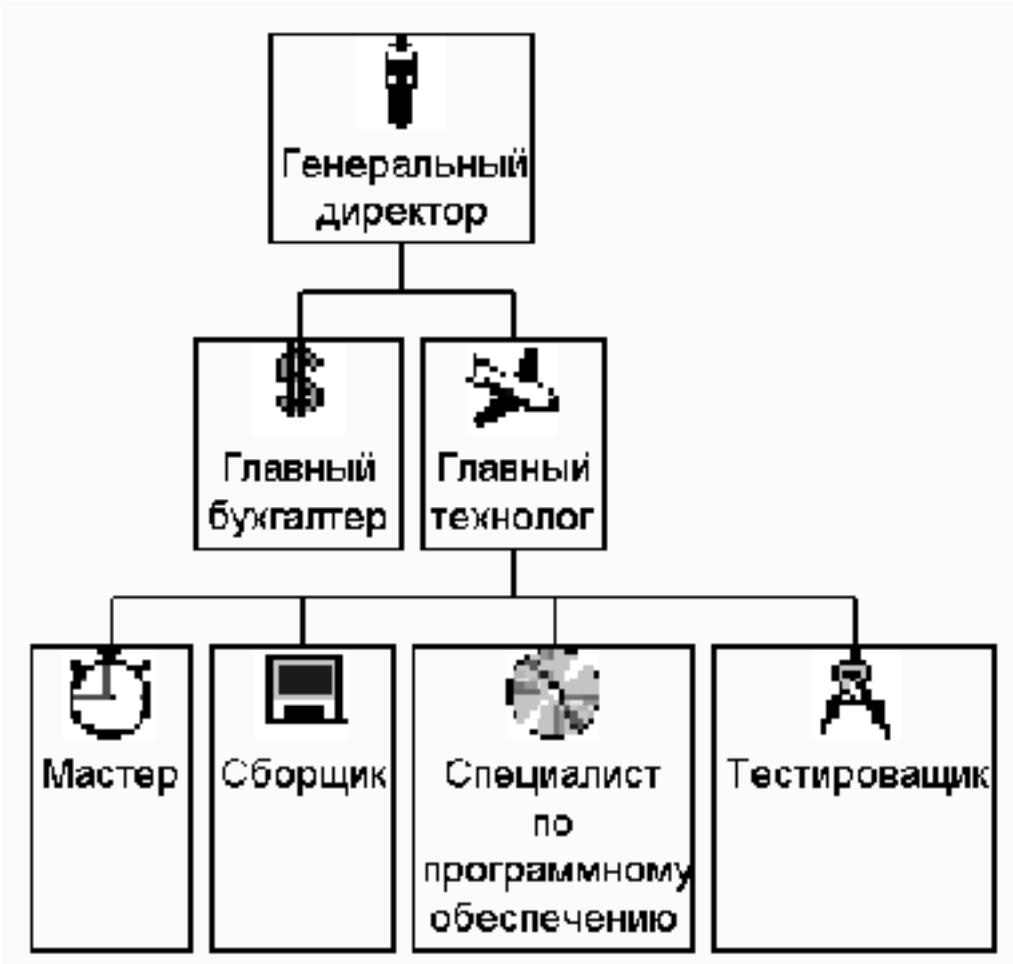


Рис.3. Организационная диаграмма.

В BPwin 4.0 стал возможен экспорт модели в систему имитационного моделирования Arena (Systems Modeling Corp.). Имитационное моделирование – это метод, позволяющий строить модели, учитывающие время выполнения функций. Полученную модель можно “проиграть” во времени и получить статистику происходящих процессов так, как это было бы в реальности. В имитационной модели изменения процессов и данных ассоциируются с событиями. “Проигрывание” модели заключается в последовательном переходе от одного события к другому. Обычно имитационные модели строятся для поиска оптимального решения в условиях ограничения по ресурсам, когда другие математические модели оказываются слишком сложными. Экспорт модели процессов в Arena позволит аналитикам более качественно производить реорганизацию деятельности предприятий и оптимизировать производственные процессы.

BPwin 4.0 поддерживает словари сущностей и атрибутов, что позволяет создавать объекты модели данных непосредственно в среде BPwin, связывать их с объектами модели процессов и экспортировать в систему моделирования данных ERwin. Такая связь гарантирует завершенность анализа, гарантирует, что есть источник данных (Сущность) для всех потребностей данных (Работа) и позволяет делить данные между единицами и функциями бизнес-процессов. Каждая стрелка в модели процессов может быть связана с несколькими атрибутами различных сущностей. Связи объектов способствуют согласованности, корректности и завершенности анализа.

Для построения модели данных Computer Associates предлагает мощный и удобный

инструмент - ERwin. ERwin имеет два уровня представления модели - логический и физический. На логическом уровне данные представляются безотносительно конкретной СУБД, поэтому могут быть наглядно представлены даже для неспециалистов. Физический уровень данных - это, по - существу, отображение системного каталога, который зависит от конкретной реализации СУБД. ERwin позволяет проводить процессы прямого и обратного проектирования для СУБД более 20 типов. Это означает, что по модели данных можно сгенерировать схему БД или автоматически создать модель данных на основе информации системного каталога с учетом реализации конкретной СУБД. Кроме того, ERwin позволяет выравнивать модель и содержимое системного каталога после редактирования того, либо другого. ERwin подерживает три нотации (IDEF1X, IE и DIMENSIONAL), что делает его незаменимым как для проектирования оперативных баз данных, так и для создания хранилищ данных.

Создание современных информационных систем, основанных на широком использовании распределенных вычислений, объединении традиционных и новейших информационных технологий, требует тесного взаимодействия всех участников проекта: менеджеров, бизнес- и системных аналитиков, администраторов баз данных, разработчиков. Для этого использующиеся на разных этапах и разными специалистами средства моделирования и разработки должны быть объединены общей системой организации совместной работы. Фирма Computer Associates разработала систему ERwin - хранилище моделей BPwin и ERwin, к которому открыт доступ для участников проекта создания информационной системы.

Хотя перечисленные выше задачи 1-5 достаточно эффективно решаются с помощью структурных средств BPwin и ERwin, современные объектно-ориентированные CASE – методологии и CASE – средства позволяют более эффективно решать задачи проектирования и кодогенерации клиентских приложений. Одним из таких средств является Paradigm Plus фирмы Computer Associates [4]. Paradigm Plus является мощным объектно-ориентированным инструментальным средством, позволяющим эффективно генерировать код приложений. Этот продукт интегрирован с целой линейкой инструментальных средств Computer Associates, что позволяет реализовать коллективную разработку крупных информационных проектов. Последняя версия Paradigm Plus поддерживает широкий набор нотаций, используемых для объектного моделирования, в том числе UML 1.1, CLIPP, TeamFusion, OMT, Booch, OOCL, Martin/Odell, Shlaer/Mellor, Coad/Yourdon. Каждая нотация может быть дополнена диаграммами Use Case (Jacobson), и моделями БД. Paradigm Plus имеет специализированные средства для разработки приложений в многоуровневой архитектуре клиент-сервер (middleware). В частности, поддерживается интеграция с технологиями COM/DCOM, CORBAPlus, IBM Component Broker, Objectbroker, Orbix и VisiBroker.

Paradigm Plus призван обеспечить полный технологический цикл разработки крупных информационных систем. С этой целью он интегрирован с целым рядом инструментальных средств CA и других фирм:

- Средства управления проектом: CA Process Continuum, CA Ad*visor;*
- Средства управления требованиями: QSS DOORS;
- Средства тестирования: CA Final Exam;
- Средства управленияконфигурациями: CA CCC/Harvest, Rational ClearCase (http://www.interface.ru/fset.asp?Url=/rational/cc/caseh.htm), Microsoft SourceSafe, PVCS;
- Репозитории: CA Repository/OEE, CA Repository/MVS, Microsoft Repository;
- Средства распространения: CA AutoXfer;
- Средства документирования: CA Paradigm Publisher, DocEXPRESS, FrameMaker,

Interleaf, Microsoft Word.

Кроме того, Paradigm Plus интегрирован со следующими средствами разработки: CA Aion, CA RuleServer, CA SQL-Station, Ada, ANSI C/C++, CORBA IDL, Delphi, Forte, GDMO/ASN.1, IBM VisualAge, Java, ParcPlace/Digitalk, PowerBuilder, Microfocus Object COBOL, Microsoft Visual Basic, Microsoft Visual C++, Microsoft Visual J++, Symantec Visual Cafer.

Разработчики крупных информационных систем в процессе создания программного обеспечения сталкиваются с целым рядом трудновыполнимых задач. Работая с объектно-ориентированными технологиями создания приложений, они создают клиент – серверные приложения, которые должны удовлетворять требованиям надежности, управляемости и высокой производительности. Решение этих задач возможно только в условиях высокоэффективного анализа и проектирования. С одной стороны, BPwin позволит построить адекватную модель (модель работ) существующих на предприятии процессов (AS-IS), проанализировать эту модель и построить модель будущих процессов (TO-BE). С другой стороны, разработчики, использующие такие средства объектно – ориентированного анализа и проектирования как Paradigm Plus могут описать требования к информационной системе при помощи диаграмм Use Cases. Бизнес-процессы современных предприятий и организаций весьма сложны. В результате анализа могут быть описаны работы (activity) и функции (use case), информация о которых получена из самых разных источников, поэтому необходима синхронизация работ и функций.

Для связи модели процессов BPwin и объектной модели Paradigm Plus используется утилита BpLink, которая вызывается как отдельная программа из среды Paradigm Plus (рис. 4).

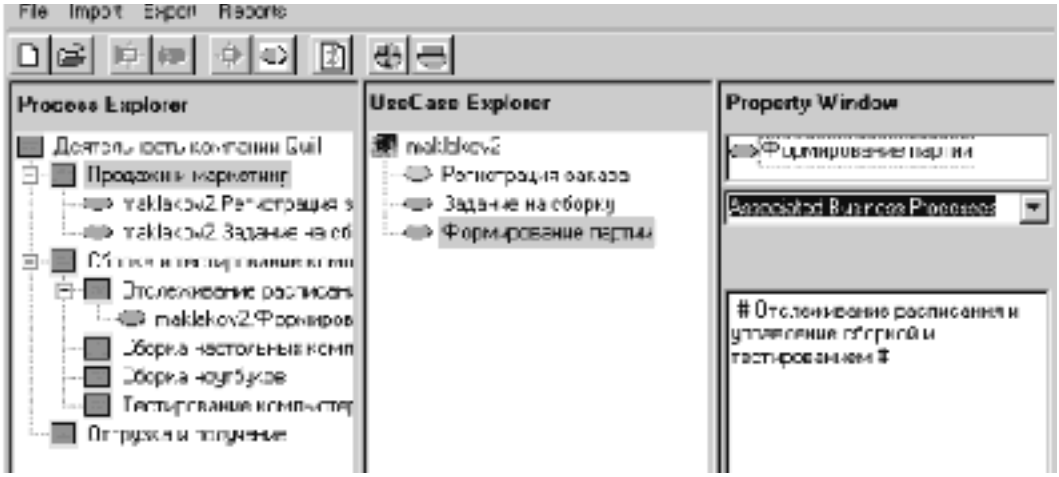


Рис.4. Связывание работ (activity) и функций (use case) с помощью BpLink.

В последних версиях Paradigm Plus 3.6 и 3.7 реализована взаимная интеграция с ERwin 3.5.2. Комбинация физического моделирования баз данных в Paradigm Plus и углубленных возможностей моделирования данных в ERwin предоставляет комплексное решение для моделирования данных. Такая интеграция повышает производительность и сокращает время разработки информационных систем.

В Paradigm Plus обеспечена двунаправленная связь Paradigm Plus и между объектной моделью и моделью данных. Реляционная модель ERwin может быть конвертирована в объектную модель Paradigm Plus и наоборот, объектная модель Paradigm Plus может быть конвертирована в реляционную модель данных. С помощью ERwin можно оптимизировать физическую модель данных с учетом особенностей конкретного сервера БД для обеспечения

наивысшей производительности.

Интеграция ERwin и Paradigm Plus обеспечивает:

- Возможность импорта из физической модели ERwin в физическую модель Paradigm Plus;
- Документирование определенных пользователем типов для проверки типов БД;
- Разработку приложений с использованием хранилищ моделей для обеспечения коллективной разработки;
- Объектно-ориентированный анализ и разработку с использованием объектно-ориентированных методов с тесной интеграцией с физической разработкой БД;
- Перенос информации, внесенной на этапе анализа и проектирования в модель данных и использование ее для кодогенерации;
- Автоматическую генерацию отчетов по проекту на основе информации, находящейся в хранилище проектов;
- Использование типов данных ERwin'а на этапе анализа и разработки;
- Моделирование систем с многоуровневой архитектурой в Paradigm Plus, что может быть использовано для разработки корпоративных систем;
- Поддержку компонентной разработки в сочетании с режимом многопользовательской работы с моделями;
- Размещение существующих моделей БД посредством обратного проектирования в хранилище Paradigm Plus и отображение их в нотации Martin&Odell;
- Переход от объектной модели к реляционной модели.

Итак, взаимная интеграция CASE- средств CA позволяет наиболее эффективно использовать преимущества как объектного, так и структурного подхода к созданию информационных систем.

Литература.

1. Дэвид А. Марка и Клемент МакГоуэн "МЕТОДОЛОГИЯ СТРУКТУРНОГО АНАЛИЗА И ПРОЕКТИРОВАНИЯ SADT"
2. С. В. Маклаков "Новые возможности CA BPwin 4.0."Компьютер Пресс, в печати.
3. С. В. Маклаков "ERwin и BPwin. CASE-средства разработки информационных систем." – М.: ДИАЛОГ-МИФИ, 2000 – 256 с.
4. С. В. Маклаков " Интеграция объектной модели Paradigm Plus 3.7 с моделями процессов BPwin и моделями данных ERwin. "Компьютер Пресс, в печати.

По разным банкам я бродил, и мой ERwin со мною

ERwin – кроссплатформенное средство моделирования данных

Наталия Елманова, Interface Ltd.

В течение последних нескольких лет информационные системы, основанные на архитектуре "клиент/сервер", приобрели заметную популярность. Причиной этого является давно назревшая необходимость автоматизации самых разнообразных сфер человеческой деятельности и, следовательно, значительное увеличение объемов и типов хранимых и обрабатываемых данных.

В отличие от информационной системы (ИС) эпохи FoxPro и dBase, обычно имеющих дело с несколькими таблицами, современная ИС оперирует десятками и сотнями связанных между собой таблиц, и, соответственно, не меньшим количеством индексов, триггеров и хранимых процедур. Наличие последних есть жизненная необходимость -- в системах с большим количеством таблиц очень сложно возложить на клиентскую часть контроль за бизнес-правилами, используемыми в ИС. При этом такая система должна быть спроектирована безошибочно, иначе в дальнейшем возникнет множество проблем с ее эксплуатацией и модернизацией.

Конечно, можно создать все таблицы и индексы вручную, а триггеры и хранимые процедуры написать на процедурном расширении SQL, характерном для данного сервера (скрипт, содержащий процедуры их создания, называется DDL-сценарием -- сокращение от Data Definition Language). Но, как показывает опыт автора и его коллег, для скромной ИС с десятком таблиц требуется почти 60 килобайт кода (например на PL/SQL). Еще одна проблема заключается в том, что нередко разработчик должен писать клиентскую часть в условиях, когда заранее неизвестно, с какой именно СУБД она будет иметь дело. Это обычно для групп программистов, создающих либо ИС для заказчиков, уже имеющих какой-нибудь сервер БД, либо ИС, которая в перспективе может быть перенесена на другую платформу. В этом случае переписывание серверных частей ИС может оказаться неизбежным.

Как избежать всех этих трудностей? К счастью, существуют мощные инструменты для графического проектирования информационных систем -- так называемые CASE-средства (CASE расшифровывается как Computer Aided System Engineering), например ERwin фирмы Logic Works. Работа с ERwin представляет собой рисование схемы базы данных (она называется ER-диаграммой или диаграммой "сущность-связь"). В ER-диаграмме таблицы изображаются в виде прямоугольников, в которых могут быть перечислены поля таблиц, а связи -- в виде линий, соединяющих прямоугольники между собой. При этом в процессе создания диаграммы можно не думать о ее конкретной реализации, а сосредоточиться на бизнес-логике работы ИС.

После создания диаграммы можно выбрать целевую платформу и сгенерировать либо саму базу данных, либо вручить DDL-сценарий администратору БД. При этом в сценарии будут содержаться все процедуры создания таблиц, индексов, триггеров, необходимых для успешного функционирования ИС, и сам сценарий будет создан автоматически из ER-диаграммы на процедурном расширении языка SQL, характерном для выбранного сервера. Таким образом, написание кода заменяется его автоматической генерацией на основе нарисованной схемы.

Наличие такого инструмента, как ERwin, существенно облегчает разработку ИС в условиях, когда сервер БД не определен или может быть заменен. Так как первоосновой серверной части ИС в этом случае является ER-диаграмма, при замене сервера требуется просто выбрать другую целевую платформу и сгенерировать для нее новый сценарий.

В последнее время значительную актуальность приобрели задачи, связанные с переносом уже имеющихся настольных ИС (обычно основанных на наборе таблиц dBase или Paradox) в архитектуру "клиент/сервер". В этом случае удобно воспользоваться предусмотренной в ERwin возможностью так называемого обратного проектирования, т. е. восстановления ER-диаграммы по имеющейся базе данных. При этом, в отличие от других CASE-средств, ERwin позволяет восстанавливать некоторые связи даже в наборах плоских таблиц, основываясь на имеющихся индексах.

Что еще полезного предлагает ERwin профессиональным разработчикам? Во-первых, возможность редактирования шаблонов триггеров. Во-вторых, разнообразные средства документирования, позволяющие включить в ER-диаграмму описание таблиц, образцы запросов, примеры данных и в дальнейшем генерировать разнообразные отчеты, которые могут послужить основой для документации созданной ИС.

Нельзя не отметить, что для ряда средств разработки приложений (Visual Basic, Power Builder, SQLWindows) созданы версии ERwin, позволяющие генерировать формы и прототипы приложений. Существуют также поставляющиеся с ERwin библиотеки (например MetaBASE) для создания форм Delphi, содержащих интерфейсные элементы, чувствительные к изменениям в модели данных.

Таким образом, ERwin позволяет с минимальными трудозатратами спроектировать структуру будущей базы данных с учетом бизнес-правил функционирования информационной системы, сгенерировать ее на разных целевых платформах без написания кода, перенести базу данных на другую платформу, в том числе из набора плоских таблиц на какой-либо сервер баз данных и в ряде случаев автоматически сгенерировать клиентское приложение.

Если у Вас возникли вопросы, ждем Вас в фирме Interface Ltd. -- крупнейшем в России дистрибьюторе продуктов Logic Works и авторизованном учебном центре Logic Works, Centura и Borland. У нас Вы можете получить ознакомительную версию ERwin, обучиться работе с этим средством, а также получить необходимые консультации по созданию проектов ИС, разработке приложений, генерации отчетов.

Использование ERwin при создании приложений для Oracle

Наталия Елманова, Interface Ltd.
elmanova@interface.msk.su

Анализ современных тенденций развития информационных систем однозначно свидетельствует о все большем усложнении используемых в них структур данных при одновременном увеличении количества хранимой и обрабатываемой информации. Современные ИС требуют принципиально иного качественного подхода к процессам их проектирования и разработки, отличного от подходов, доминировавших в предыдущие несколько лет, когда широко использовались программные продукты, созданные с помощью dBase, FoxPro или Clipper, и оперирующие относительно небольшим набором таблиц.

Что обычно происходило при развитии такой системы? Типичный сценарий жизни подобной ИС таков. После ввода в эксплуатацию выявляются недочеты (нередко связанные не с ошибками программистов, а с неверно сформулированным техническим заданием), у пользователей возникают новые потребности (создать тот или иной дополнительный сервис, модернизировать структуру таблиц, добавить новые таблицы, запросы или отчеты, изменить выходные формы или предусмотреть возможность их редактирования). Начинается переработка исходных текстов, структур таблиц, добавление "заплат". В конечном итоге получается программный продукт, в исходном тексте которого никто, кроме автора, разобраться не в состоянии. Примерно так же выглядит и используемая структура данных. При этом связи между таблицами могут быть не просто неочевидными, порой они оказываются невероятно причудливыми. О подчинении данных реляционной модели, как правило, не может быть и речи. Чем дальше, тем сложнее становится модернизировать подобную ИС, и в конечном итоге приходится ее перепроектировать и переписывать заново, портировать старые данные в новую структуру, что нередко приводит к немалым трудовым и материальным затратам. Автору приходилось наблюдать случай, когда из-за накопившегося количества переделок действующей ИС, неудачно спроектированной структуры данных и некорректной обработки транзакций при завершении работы приложения переписывались заново несколько десятков dBase-таблиц, что занимало несколько минут. При сбоях происходило нарушение ссылочной целостности БД, и система становилась неработоспособной. Из-за этого пришлось оснастить рабочие станции источниками бесперебойного питания (а пользователей снабдить валокордином и валерьянкой). Проблема генерации отчетов, не предусмотренных приложением, на основе полученной БД не решалась в принципе -- никто, кроме автора, не мог понять, как между собой связаны таблицы. Самое удивительное, что существуют ИС, основанные на архитектуре "клиент/сервер" (например, использующие сервер Oracle 7), созданные и развивающиеся подобным образом. Опыт работы многих программистов показывает, что, как правило, редко удается создать идеальную ИС. Следовательно, необходимо изначально предусматривать возможность ее безболезненной модернизации. Поэтому нужны иные подходы к проектированию ИС, нежели те, что применялись до недавнего времени. Одним из таких подходов и является применение специализированных средств проектирования структур данных -- так называемых CASE-средств (CASE расшифровывается как Compu-ter Aided Software Engineering или Computer Aided System Engineering, что, по существу, отражает многообразие решаемых этими средствами за-дач -- от создания структур данных до генерации приложений). CASE-средства позволяют автоматизировать процесс создания структур данных, создавать серверные части приложений, вносить в них бизнес-логику приложения, правила контроля целостности

данных, а также приписывать полям в таблицах расширенные атрибуты (тип интерфейсного элемента, максимальные и минимальные значения, значения по умолчанию и т.д.). Немаловажно, что в процессе проектирования и перепроектирования структур данных можно документировать создаваемые таблицы, их поля и связи между ними и отображать графически полученную структуру.

К сожалению, CASE-технология на сегодняшний день используется недостаточно широко. В лучшем случае разработчики считают CASE-средства просто инструментом для рисования, чем-то вроде CorelFlow. В большинстве случаев об этих средствах разработчики не слышали вообще или имеют смутное представление. Как показало проведенное автором небольшое статистическое исследование, примерно треть (!) выставляющих свои программные продукты на стендах выставки SofTool'96 специалистов-разработчиков, не говоря уже о менеджерах, никогда о CASE не слышали, еще треть слышали, но не видели, и буквально единицы видели и используют. Примерно такая же картина наблюдалась среди посетителей стенда фирмы Borland на выставке WindowsExpo' 96И, видимо, придется приложить немало усилий, чтобы эти замечательные средства завоевали умы и сердца разработчиков. Одним из CASE-средств, наиболее удачных с точки зрения соотношения цены, простоты использования и возможностей, является ERwin фирмы Logic Works. Данное средство обладает всеми перечисленными возможностями и при этом имеет удобный интерфейс, интуитивно понятные инструменты, разнообразные возможности графического представления структуры данных. Это средство поддерживает много разных форматов плоских таблиц и серверных БД и умеет создавать триггеры и хранимые процедуры на соответствующих процедурных расширениях языка запросов SQL, поддерживаемых обслуживаемыми серверами (в случае Oracle это PL/SQL). При этом, проектируя стуктуру данных, не обязаны знать ни SQL, ни его расширения.

Следует отметить, что большинство CASE-средств позволяет восстанавливать схему БД по имеющейся физической структуре (такая процедура называется обратным проектированием, или реинжинирингом). Что касается ERwin, то на сегодняшний день это единственное средство, позволяющее не только восстанавливать сведения о плоских таблицах типа dBase или Paradox, но и сведения о некоторых связях, основываясь на индексах, имеющихся в такой БД.

Как осуществляется проектирование БД для сервера Oracle с использованием ERwin? Все очень просто. При разработке новой ИС вы можете использовать полностью или частично структуру ее прежней версии или иного прототипа. В этом случае сначала вы делаете реинжиниринг этого прототипа и получаете некий полуфабрикат вашей схемы, требующий, как правило, модернизации (изменения имен полей, восстановления связей и др.). Если же прототипа нет, можно создать схему с самого начала. В этом случае вы создаете на экране прямоугольники, соответствующие вашим таблицам, описываете поля этих таблиц, а затем рисуете связи между таблицами и описываете свойства этих связей. То, что получится в результате, называется ER-диаграммой (или диаграммой "сущность-связь"). Далее выбирается соответствующий сервер вашей БД (в нашем случае Oracle), и после этого можно либо соединиться с сервером и создать на нем пустую структуру из таблиц, индексов и триггеров, либо оставить это увлекательное занятие на откуп администратору БД и SQL*Plus, сохранив вашу диаграмму в виде программы на языке PL/SQL. Такая программа называется DDL-сценарием приложения (DDL расшифровывается как Data Definition Language) и делает то же самое -- создает таблицы, индексы и триггеры в соответствии с вашим описанием.

Что можно делать дальше? А дальше можно писать приложение, используя имеющуюся структуру БД. При этом, если вы правильно выбрали средство разработки (автор очень рекомендует Borland Delphi), в случае нарушения ссылочной целостности или иных установленных вами правил, ваше приложение будет цитировать сообщения созданных ERwin триггеров (что такое триггер, вы, вообще говоря, тоже знать не обязаны, хотя в условиях российской действительности их сообщения не вредно перевести на русский язык или, слегка поредактировав шаблоны в ERwin, заменить чем-нибудь более понятным для пользователя, нежели фразы типа "Foreing key not found").

Следует отметить, что для некоторых средств разработки приложений (SQL Windows, Power Builder, VB) созданы специальные версии ERwin, позволяющие создавать формы конечных приложений (например, для ввода данных). Президентом фирмы Logic Works Беном Коганом обещан ERwin для Delphi. То, что известны форматы словарей данных как самого ERwin, так и Delphi, дает повод для создания специализированных программ-экспертов для переноса данных между этими словарями, встраиваемых в среду разработки Delphi (один из таких экспертов входит в комплект поставки Delphi 2.01).

Итак, структура базы данных готова, приложение написано и отлажено...

Вы создаете документацию к ваше-му приложению. Вспомните о ER-диаграмме! Если вы не забыли внести в нее все описания, комментарии к полям, таблицам и связям -- это замечательная основа для вашей документации. Да и сама схема БД послужит не один год важным источником информации для ваших пользователей, когда они будут создавать отчеты, и для вас и ваших коллег, когда вы в очередной раз будете модернизировать созданную вами информационную систему.

И еще одна приятная деталь. Если у вас возникла необходимость сменить один сервер на другой, вы можете использовать все ту же ER-диаграмму для создания схемы новой базы данных на этом сервере. При этом вы получите триггеры уже на другом диалекте SQL, поддерживаемом новым сервером -- ERwin сделает это за вас автоматически. А если вдруг вы решите создать набор dBase-таблиц, то вместе с их заголовками получите и текст программы для создания индексов на нужном диалекте xBase.

Использование расширенных функциональных возможностей AllFusion ERwin Data Modeler
Работа с уровнями проектирования

Зайцев С.Л.

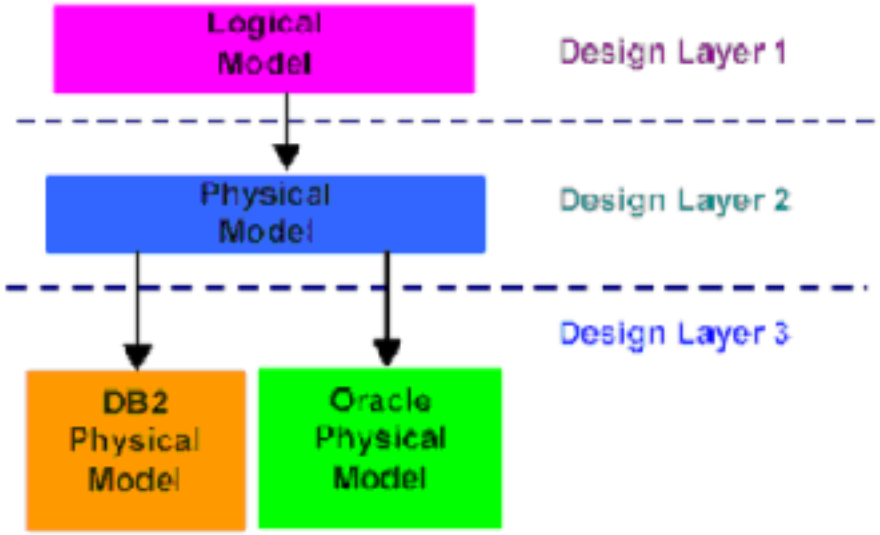
В целом в иерархии уровней проектирования различные типы модели используются в жизненном цикле разработки приложений с целью различения типов модели между собой. В частности с помощью логической модели можно представить бизнес-требования и правила. Из этой модели можно получить универсальную физическую модель, в которой физические конструкции спроектированы для универсальной базы данных. Если универсальная физическая модель является устойчивой, из нее можно получить несколько моделей, специфичных для базы данных. Таким способом универсальная физическая модель становится моделью стандартов.

Различные функциональные возможности AllFusion ERwin Data Modeler (ранее: ERwin), (далее по тексту - AEDM) позволяют поддерживать разделение типов модели, а также соединение и синхронизацию связанных моделей. Эти функциональные возможности будут кратко рассмотрены в этой статье.

Что такое уровень проектирования?

Уровень проектирования – это отдельная модель данных или набор моделей данных, которые используются в процессе разработки приложений для определенной цели. Каждый уровень проектирования является составной частью иерархии двух или более уровней проектирования.

В самом простом представлении иерархии уровней проектирования первый уровень проектирования (Design Layer 1) представляет собой логическую модель данных (Logical Data Model), которая определяет бизнес-требования к приложению. Затем на втором уровне проектирования (Design Layer 2) эти бизнес-требования преобразуются в правила внедрения базы данных в физической модели данных (Physical Data Model). Универсальная физическая модель может быть создана с помощью универсального ODBC в качестве целевой базы данных. Третий уровень проектирования (Design Layer 3) может представлять различные физические внедрения одной модели данных - например, физическая модель DB2 (Physical Model DB2) и Физическая модель Oracle (Physical Model Oracle), но на различных целевых серверных платформах:



Иерархия уровней проектирования.

В целом использовать одну модель для всех фаз процесса проектирования невозможно. Вместо этого необходимо разработать и соединить связанные модели на различных уровнях проектирования. На каждом уровне проектирования необходимо принимать и регистрировать проектные решения, которые преобразуют структуру от уровня к уровню. В конечном счете, необходимо поддерживать связи между моделями на различных уровнях проектирования и синхронизировать изменения, внесенные на различных уровнях, одновременно поддерживая соответствующие структуры на каждом из них. Эта возможность обеспечивается в AEDM комбинацией связи моделей на различных уровнях проектирования посредством источников моделей и применения трансформаций в рамках модели.

Первый уровень проектирования: концептуальная логическая модель данных

AEDM предлагает превосходный подход к визуализации структур базы данных и облегчению проектирования логических и физических моделей данных. Этот структурный, систематический подход к управлению информацией и разработке приложений начинается с концептуальной логической модели, первого из нескольких уровней проектирования, позволяющего определить конкретные бизнес-требования (включая универсальные сущности и структуры супертипа/подтипа).

Второй уровень проектирования: универсальная физическая модель данных

На этом уровне проектирования определяют структуру таблиц и столбцов и универсальное именование, необходимые для представления бизнес-приложения. Однако в универсальной физической модели данных объекты и свойства не зависят от базы данных. Другие модели данных, специфичные для базы данных, могут быть получены из универсальной физической модели данных.

Третий уровень проектирования: физические модели, специфичные для базы данных

Теперь с помощью AEDM можно воплотить свой опыт во внедрение базы данных, создав

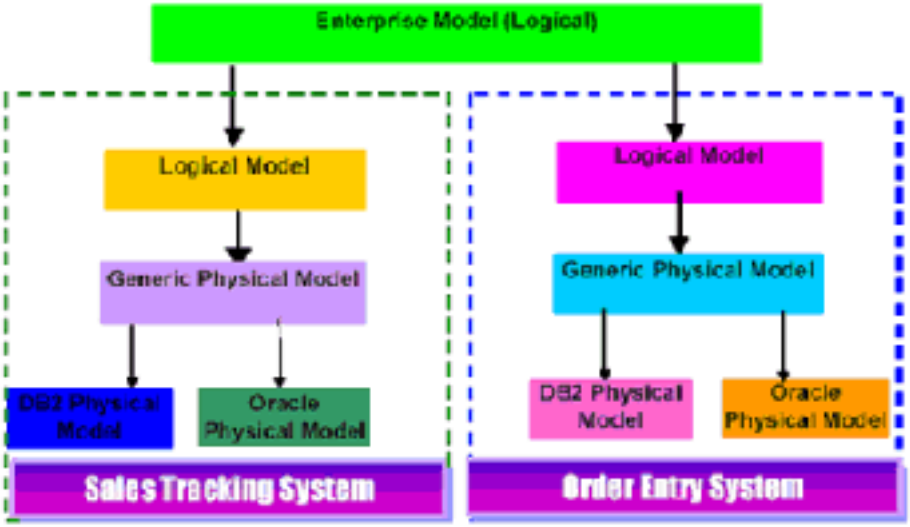
физический уровень проектирования, специфичный для базы данных. Каждое приложение может работать на нескольких платформах базы данных; последний уровень проектирования необходим для моделей данных, специфичных для базы данных.

Другие иерархии уровней проектирования

Иерархия модели масштаба предприятия

В подобном примере концептуальная модель данных может быть вместо этого моделью данных масштаба предприятия, которая определяет стандарты для всех приложений организации. Модель данных масштаба предприятия может быть логической и включать в себя все утвержденные стандарты сущностей и атрибутов, поддерживаемых организацией. На следующем уровне проектирования могут быть представлены несколько логических моделей данных для ряда бизнес-приложений, таких, как ввод заказа и комиссионный сбор за продажу.

Несмотря на большие различия этих приложений, в них вероятно совместно используются некоторые общие сущности, например, СОТРУДНИК и КЛИЕНТ. Модель масштаба предприятия (Enterprise Model) может включать в себя обе этих сущности (Logical Model) наряду с другими сущностями, которые ни для одной из этих специфических для приложения моделей выбрать невозможно. В этой иерархии следующим уровнем проектирования может быть универсальная физическая модель (Generic Physical Model) для каждого приложения. Как и в предыдущем примере, если каждое приложение работает на нескольких платформах базы данных (Physical Model DB2 и Oracle Physical Model), последний уровень проектирования необходим для моделей данных, специфичных для базы данных.



Логическая модель масштаба предприятия. Слева - Система отслеживания продаж, справа - система ввода заказа.

Иерархия хранилища данных

Для хранилища данных требуются дополнительные уровни проектирования для моделей всего хранилища и витрин данных. В физической модели AEDM предлагает опции для

нотации Dimensional и для функциональных возможностей, связанных с движением данных (информация об источнике данных, правила трансформации данных), эти опции позволяют оптимизировать модели хранилищ данных Warehouse Model.

Создание новых уровней проектирования

Разделение модели

Если использовалась модель данных, которая была сохранена в предыдущей версии AEDM, это - логическая/физическая модель. Если эту модель требуется разделить на две модели – логическую и физическую, можно использовать опцию Split Model (Разделить модель) в меню Tools (Инструменты).

При разделении модели AEDM предлагает сохранить отдельные логическую и физическую модели данных с различными именами. При сохранении новых моделей логическая модель становится источником физической модели, который необходим для синхронизации изменений в этих двух типах модели. Исходная логическая/физическая модель сохраняется со своим первоначальным именем.

Получение модели

Независимо от того, существует логическая/физическая модель или только логическая или только физическая модели, AEDM позволяет легко создать новую модель. Вместо того, чтобы копировать объекты из одной модели в другую или начинать все с самого начала, мастер Derive New Model (Получение новой модели) позволяет за несколько шагов получить новую модель из источника модели. AEDM связывает исходную модель в качестве источника новой модели. Затем можно приступать к работе над отдельными моделями, поскольку изменения, внесенные в каждую модель, можно синхронизировать в любое время.

Для получения новой модели выберите пункт Derive New Model (Получить новую модель) в меню Tools (Инструменты). Мастер Derive New Model (Получение новой модели) поможет выполнить все шаги процесса и позволяет определить объекты, которые требуется перенести из исходной модели в новую.

Добавление источника модели

Источник модели – это родительская модель, которая связана с другой моделью для синхронизации изменений. AEDM автоматически присваивает объектам в источнике модели и в связанной модели скрытые идентификаторы. С помощью этих идентификаторов объектов AEDM отслеживает изменения, вносимые в обе модели. Затем изменения можно синхронизировать, даже если имя объекта изменилось.

Иногда при построении иерархии уровней проектирования целесообразнее связать две существующие модели, чем получить новую модель из существующей. Например, имеется универсальная модель, которую требуется определить в качестве источника модели для других моделей, специфичных для базы данных. В этом случае можно добавить универсальную модель в качестве источника к модели, специфичной для базы данных. При добавлении источника модели необходимо определить объекты и свойства, которые требуется перенести из источника модели в целевую модель.

Для добавления источника модели выберите пункт Add Model Source (Добавить источник модели) в меню Tools (Инструменты). Мастер Add Model Source (Добавление источника модели) позволяет за несколько шагов определить объекты, которые требуется добавить в целевую модель. Объекты добавляются в целевую модель и связываются, тем самым обеспечивается синхронизация любых дальнейших изменений.

Если необходимо перенести объекты из различных моделей, можно добавить несколько источников модели. После добавления источника модели диалог Model Sources Properties (Свойства источников модели) может использоваться для просмотра и редактирования информации об источнике модели.

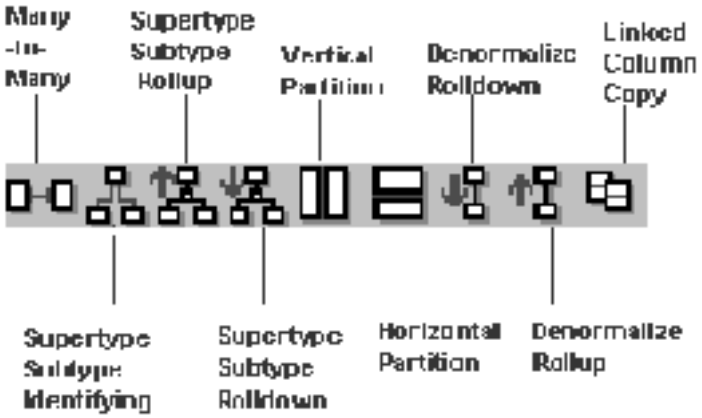
Что такое трансформация?

Трансформация – это метод, позволяющий применить и зарегистрировать проектное решение, т. е. решение о внесении изменений в объекты или свойства на определенном уровне проектирования. При применении трансформации в состояние ряда объектов вносятся изменения с целью усовершенствования, нормализации или денормализации модели. При использовании трансформаций можно выделить следующие основные преимущества:

- *Автоматизация.* AEDM упрощает совершенствование логической и физической моделей. Вместо применения изменений вручную можно использовать мастеры для автоматического применения изменений уровня проектирования.
- *Трассировка.* Для каждого объекта модели, создаваемого при трансформации, в AEDM ведется историческая информация. Историю трансформированных объектов можно проследить.
- *Сохранение свойств объекта.* Свойства трансформированных объектов сохраняются. (Повторного ввода информации вручную не требуется.)

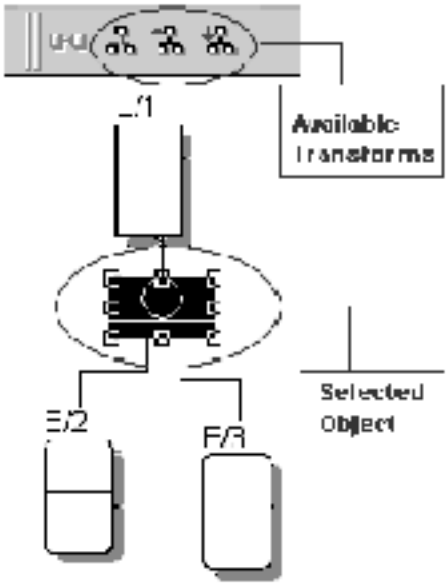
Панель инструментов Transform (Трансформация)

Панель инструментов Transform (Трансформация) предлагает набор инструментов для применения трансформации. Доступность инструментов на панели инструментов Transform (Трансформация) определяется типом модели и объектами, участвующими в трансформации.



Применение трансформаций

В большинстве случаев при выборе объектов, которые требуется трансформировать, соответствующие инструменты становятся доступными на панели инструментов Transform (Трансформация). Например, для применения трансформации свертывания супертипа/подтипа сначала необходимо выбрать символ супертипа/подтипа.

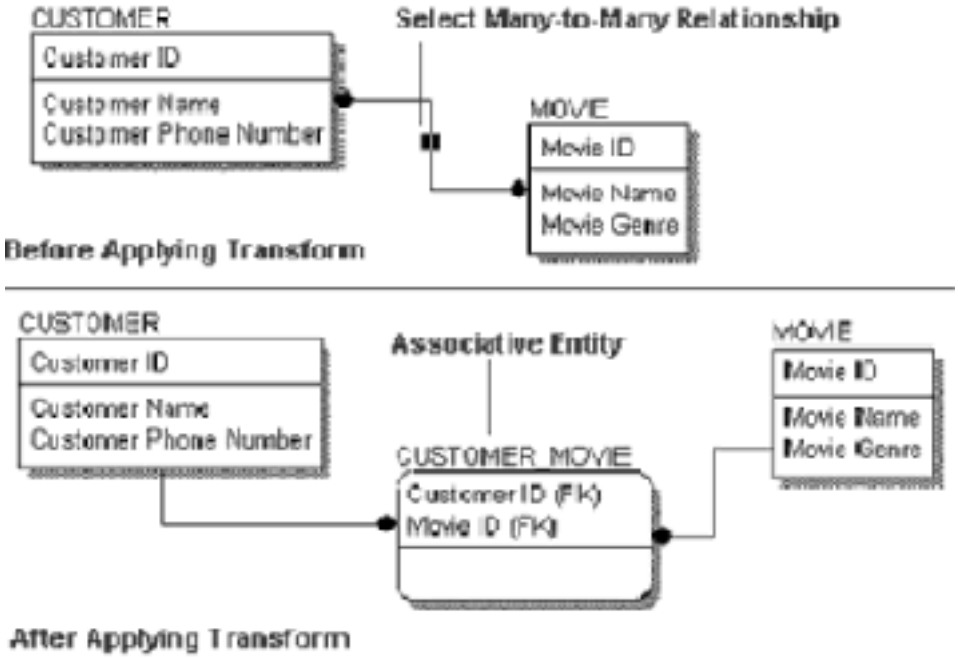


Как работает трансформация?

Трансформация "многие ко многим" позволяет, вероятно, лучше всего проиллюстрировать процесс трансформации AEDM. Для использования этой трансформации необходимы две сущности, имеющие связь "многие ко многим". Для установления этого вида связи часто между двумя исходными сущностями добавляется ассоциативная сущность и соединяется с каждой сущностью идентифицирующей связью. С помощью трансформации "многие ко многим" связь "многие ко многим" разрывается автоматически и заменяется ассоциативной сущностью и двумя идентифицирующими связями.

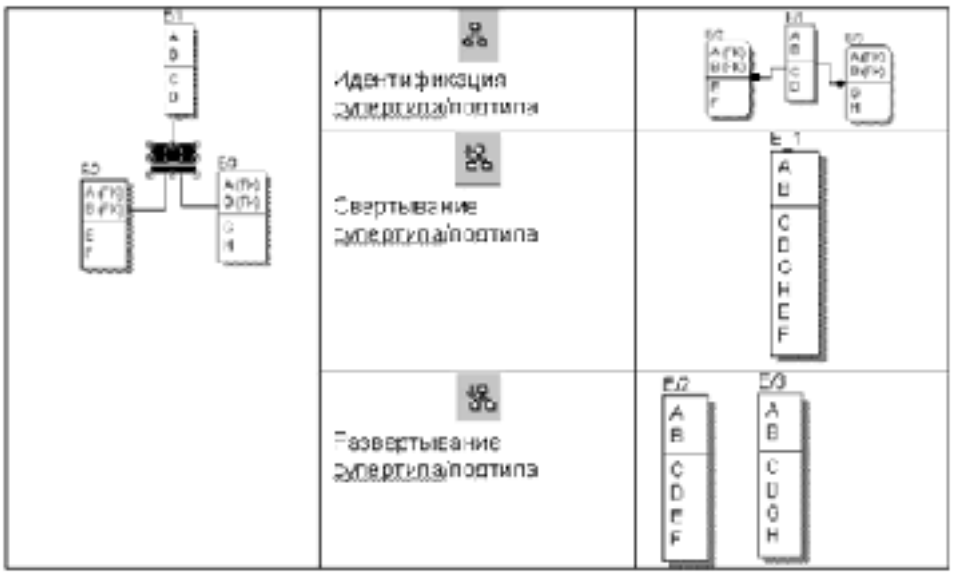
После выбора объектов, участвующих в трансформации, необходимо щелкнуть на кнопке панели, которая открывает мастер. Затем необходимо ответить на ряд вопросов, определяющих применение трансформации. Мастер четко располагает результаты после

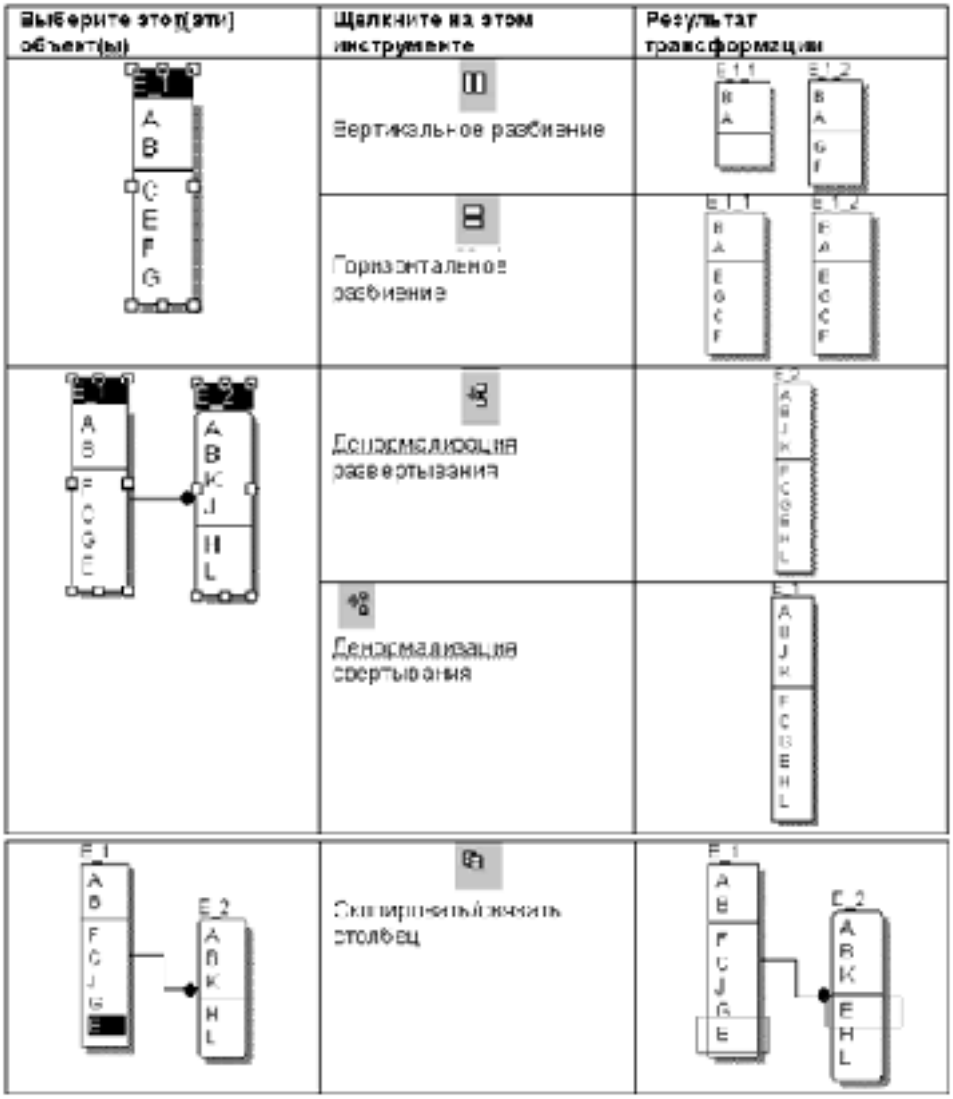
применения трансформации. Участвующие объекты преобразуются после завершения трансформации.



Ожидаемые результаты трансформации

На следующей диаграмме показано состояние объектов, участвующих в трансформации, до и после трансформации.





Трансформации и проводник моделей

В проводнике моделей при каждом применении трансформации важная информация в папке

Transforms (Трансформации) обновляется. В эту информацию включается имя трансформации, исходный и целевой объекты, участвующие в трансформации.

Создание вложенных трансформаций

Когда трансформация применяется к существующей трансформации, создается вложенная трансформация. Вложенные трансформации могут обрабатываться так же, как и отдельные трансформации.

Разрыв связей трансформации и отмена трансформаций

AEDM предлагает два метода "отмены" трансформации. Связи трансформации можно разорвать, либо трансформацию можно отменить.

Когда связи трансформации разрываются, объекты модели, созданные при трансформации, сохраняются, однако исходные объекты удаляются.

Чтобы разорвать связи трансформации, перейдите в проводник моделей и щелкните правой кнопкой мыши на трансформации. В контекстном меню выберите пункт Delete (Удалить) и затем выберите Resolve (Разорвать связи).

ПРИМЕЧАНИЕ: Если источник модели связан с моделью, в которой была применена трансформация, при разрыве связей трансформации связь между трансформированными и исходными объектами в источнике модели будет разорвана.

Когда трансформация отменяется, исходные объекты сохраняются, и трансформация и объекты модели, созданные при трансформации, удаляются.

Чтобы отменить трансформацию, перейдите в проводник моделей и щелкните правой кнопкой мыши на трансформации. В контекстном меню выберите пункт Delete (Удалить) и затем выберите Reverse (Отменить).

Синхронизация изменений между уровнями проектирования

Как отмечалось выше, модель может быть связана с источником модели в результате разделения логической/физической модели, получения модели или добавления источника модели. После того как модели присвоен источник модели, изменения в объектах, которые были перенесены из источника модели, отслеживаются автоматически. В любое время можно использовать мастер Sync with Model Source (Синхронизация с источником модели) для импорта и экспорта изменений между моделью и ее источником.

Мастер Sync with Model Source (Синхронизация с источником модели) позволяет за несколько шагов выполнить процесс выбора типов объектов и изменений, которые требуется сравнить и синхронизировать. Также можно определить правила замены регистра и максимальной длины для объектов логической и физической моделей и указать файл, позволяющий соблюдать стандарты именования в целевой модели. В результате выводится параллельный список изменений, внесенных в исходную и/или целевую модель.

При импорте или экспорте изменений AEDM обновляет дату синхронизации в моделях и

диалоге Model Sources Properties (Свойства источников модели).

Управление стандартами

Когда за проектирование ряда моделей данных отвечают несколько сотрудников или групп, соблюдение стандартов является важным аспектом проектирования. Как правило, основными сферами, в которых может иметь место противоречивость, являются именование объектов и отображение типов данных. AEDM предлагает встроенные инструменты, которые облегчают управление правилами именования и стандартами типов данных, применяемыми во всех моделях масштаба предприятия.

Стандарты именования

AEDM имеет широкий спектр возможностей стандартов именования, которые позволяют разрабатывать новые стандарты или внедрять существующие стандарты. Можно использовать все возможности стандартов именования или только те возможности, которые поддерживаются в конкретной организации.

В меню Tools (Инструменты) выберите пункт Names (Имена), затем выберите диалог Model Naming Options (Опции именования модели). Этот диалог позволяет определить стандарты именования для текущей модели данных. Можно определить регистр (нижний или верхний) и максимальную длину имени физических объектов. Для всех типов модели можно определить реагирование AEDM на одинаковые имена (спросить, разрешить или запретить).

Чтобы открыть редактор стандартов именования (Naming Standards Editor), в меню Tools (Инструменты) выберите пункт Names (Имена) и затем пункт Edit Naming Standards (Редактировать стандарты именования). Этот редактор позволяет определить отдельные стандарты именования для логических и физических объектов.

В редакторе стандартов именования на вкладке Glossary (Глоссарий) можно импортировать существующий глоссарий имен или создать новый. Можно определить способ внедрения этих стандартов и создать записи, включая условия деловой деятельности и сокращения, релевантные для данного бизнеса. В AEDM данные стандартов именования хранятся в файле стандартов именования (*.nsm). К каждой модели AEDM, в которой используются стандарты именования, необходимо присоединить файл стандартов именования. Один файл стандартов именования можно соотнести с несколькими моделями.

При присоединении файла стандартов именования к модели данных стандарты и правила, определенные в файле, применяются автоматически. Кроме того, файл может использоваться для проверки совместимости имен объектов модели, которая похожа на программу проверки орфографии в текстовом редакторе. Файл стандартов именования используется в качестве словаря и позволяет сравнить имена в модели данных с именами в словаре. Если найдено несоответствие, AEDM останавливается и предлагает игнорировать или заменить несовместимое имя.

Совет: *для присвоения имен объектам модели в AEDM можно использовать инструментарий для макросов AEDM. В AEDM имеются макросы, которые помогают различать имена, присваиваемые в логических и физических моделях.*

Отображение типов данных

Тип данных – это предварительно определенный набор признаков для атрибута или столбца, который определяет длину поля, допустимые символы и опциональные и обязательные параметры. Например, тип данных char(18) определяет, что в столбце может сохраниться до 18 буквенно-цифровых символов.

По умолчанию тип данных применяется к каждому атрибуту логической модели и к каждому столбцу физической модели. В логической модели тип данных определяется доменом, из которого атрибут унаследовал свои свойства, или присвоенным типом данных. В физической модели тип данных определяется значением по умолчанию, указанным целевым сервером, или присвоенным типом данных. Поскольку в модели данных обычно имеется большое число атрибутов или столбцов, присвоение типов данных и обеспечение их непротиворечивости вручную могут быть весьма утомительными. Для этого в AEDM существует несколько инструментов, облегчающих решение этой задачи.

Для редактирования стандартного отображения типов данных для логических и физических моделей может использоваться редактор стандартов типов данных. В физических моделях можно редактировать стандартный тип данных, присоединяемый к каждому столбцу автоматически. В логических моделях можно добавлять логические типы данных и присваивать типы данных атрибутам.

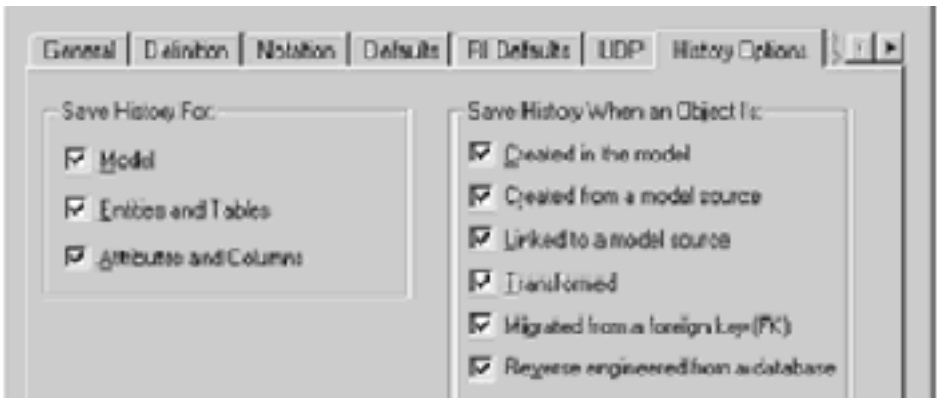
Для определения стандартов отображения типов данных необходимо указать, как логические типы данных отображаются на доступных типах данных для целевого сервера. Если приложения базы данных работают на нескольких серверных платформах, можно отобразить типы данных для всех целевых серверов. В AEDM данные отображения типов данных хранятся в файле стандартов типов данных (*.dsm). К каждой модели AEDM, в которой используются стандарты типов данных, необходимо присоединить файл стандартов типов данных. Если при открытии модели AEDM файл стандартов типов данных не присоединен, используется стандартное отображение типов данных.

Конечно, AEDM позволяет определить стандарты отображения типов данных один раз и применять их к нескольким моделям данных. Чтобы определить файл отображения типов данных, который необходимо применить к текущей модели, выберите пункт Datatypes (Типы данных) и затем пункт Model Datatype Options (Опции типа данных модели) в меню Tools (Инструменты).

Сохранение истории модели

Для модели, сущностей, атрибутов, таблиц и столбцов можно сохранить исторические данные. С помощью функциональной возможности History (История) можно отследить важные изменения, внесенные в полученные и трансформированные модели, а также стандартную информацию о датах создания и пересмотра моделей.

Для настройки опций истории выберите пункт Model Properties (Свойства модели) в меню Model (Модель). В диалоге Model Properties (Свойства модели) щелкните на вкладке History Options (Опции истории) и поставьте галочку или снимите ее у опций истории для объектов модели, историю которых требуется сохранить.



Исторические данные можно сохранить для следующих событий:

- Когда создается логическая модель, и добавляются сущности и атрибуты.
- Когда создается физическая модель, и добавляются таблицы и столбцы.
- Когда физическая модель получается из логической модели, и в полученной физической модели создаются таблицы и столбцы.
- Когда логическая/физическая модель разделяется на независимые логический и физический компоненты.
- Когда трансформация применяется к двум таблицам, и создается отдельная денормализованная таблица.
- Когда декомпилируются база данных или файл сценария, и в новой модели создаются таблицы и столбцы.
- Когда добавляется источник модели, из которого в модель переносятся таблицы и столбцы.

Интеграция AllFusion ERwin Data Modeler с AllFusion Component Modeler

Козодаев А.А.
технический специалист компании Interface Ltd.

В 2001 году компанией Computer Associates была выпущена линейка AllFusion Modeling Suite, которая состоит из пяти продуктов, а именно:

- AllFusion ERwin Data Modeler (ранее: ERwin)
- AllFusion Process Modeler (ранее BPwin)
- AllFusion Model Manager (ранее Model Mart)
- AllFusion Component Modeler (программный продукт пришедший на смену Paradigm Plus)
- AllFusion Data Model Validator (ранее ERwin Examiner)

В линейку AllFusion входят продукты, обеспечивающие поддержку полного цикла разработки программного обеспечения. В данной статье будет рассмотрена интеграция AllFusion ERwin Data Modeler (далее по тексту - AEDM) с AllFusion Component Modeler (далее по тексту - ACM).

AEDM является CASE-продуктом, который позволяет эффективным образом проектировать, документировать и сопровождать базы данных. К особенностям данного продукта можно отнести поддержку нескольких методик проектирования баз данных, а именно: IDEF1X, IE, а также методика моделирования хранилищ и витрин данных – DM, поддержку множества (более 20) серверов баз данных и некоторые другие функциональные возможности.

ACM является инструментом для построения информационных систем с использованием унифицированного языка моделирования – UML. В последней версии продукта полноценно поддерживается UML версии 1.4. ACM включает в себя ряд функциональных возможностей, которые успешно выделяют его из аналогичных программных продуктов.

К таким можно отнести:

- *Model Xpert Engine* – функция, позволяющая проверять модели, создаваемые в ACM на предмет соответствия правилам графического языка UML
- *Model Xfer* – функция, позволяющая переносить модели между различными репозиториями
- поддержка обратного генерирования кода для платформы Microsoft .Net
- поддержка прямого генерирования для следующих языков программирования: Java, CORBA, Visual C++, и Visual Basic

Для осуществления процесса интеграции в дистрибутив ACM включена утилита *AllFusion Component Modeler-ERwin Data Modeler Add-In.* Для того, чтобы воспользоваться возможностями данной утилиты, не обязательно устанавливать оба продукта (AEDM и ACM), однако в последнем случае предоставляются наиболее полные возможности для интеграции. Процесс интеграции можно инициировать как из ACM, так и из AEDM. В качестве примера в статье будет рассмотрена следующая ситуация. В ACM построена диаграмма классов, затем некоторые классы из этой диаграммы импортируются в AEDM. Следующим этапом является изменение полученной на основании диаграммы классов модели

данных ERwin и дальнейшая синхронизация изменений модели данных с моделью UML, построенной в ACM.

Диаграмма классов в ACM выглядит следующим образом:

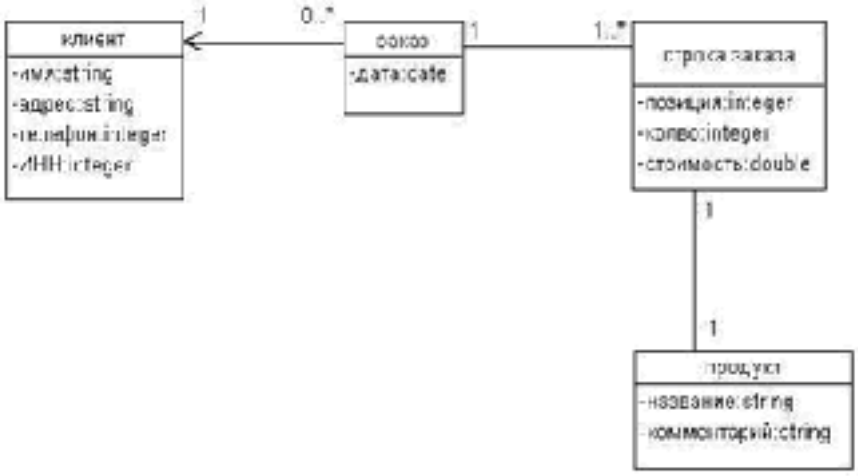


Рисунок 1. Исходная диаграмма классов.

Далее эта диаграмма импортируются в AEDM, это можно сделать, выбрав в AEDM меню Tools - Add-Ins… - Import from AllFusion Component Modeler. В возникшем окне нужно выбрать, из какой рабочей области (workspace) ACM необходимо загрузить элементы диаграммы классов. В следующем окне выбираются классы, которые будут представлены в модели данных AEDM.

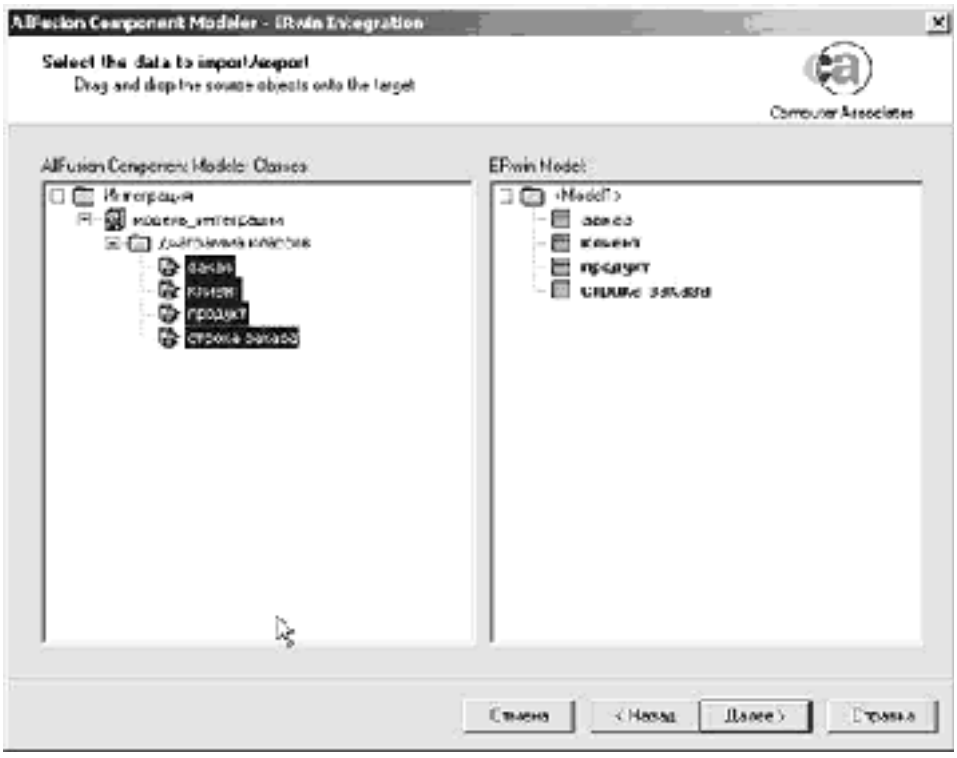


Рисунок 2. Импортирование необходимых классов.

В следующих двух окнах мастера интеграции необходимо присвоить импортируемым

классам стереотип ERwin и в случае, если мастер не смог конвертировать связи ACM в связи AEDM, нужно также уточнить типы связей. После того, как мастер отработает, будет создана соответствующая ER-диаграмма в AEDM. При этом необходимо учитывать, что в исходной диаграмме классов импортируемые классы и их атрибуты приобретут следующие стереотипы: ERWIN_ENTITY для классов и ERWIN_ATTRIBUTE - для атрибутов.

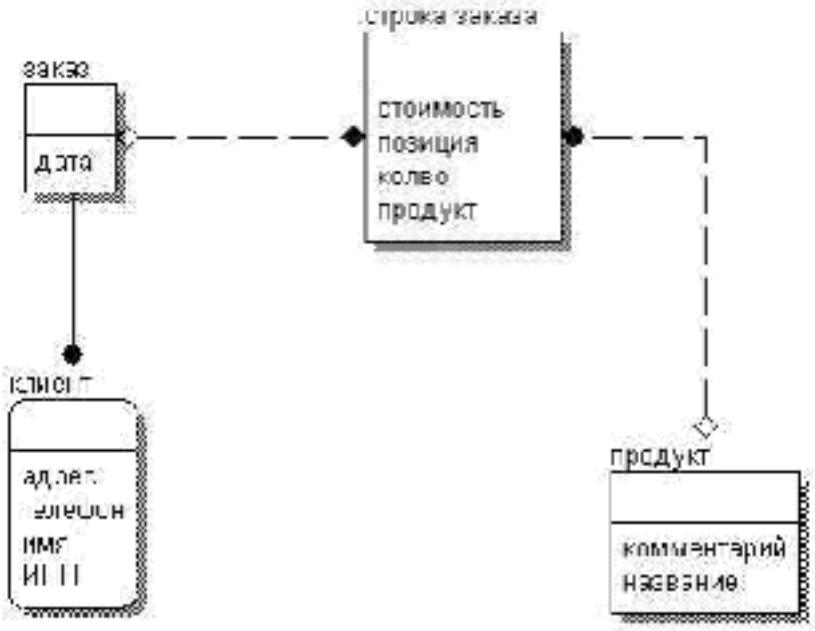


Рисунок 3. Полученная ER-диаграмма.

В результате анализа полученная ER-диаграмма была изменена, в нее были добавлены две новые сущности:

1) «**способ доставки** » с неключевыми атрибутами: **название, срок**
2) «**фирма производитель** » с неключевым атрибутом: **название фирмы**
а также в сущность «**продукт** » добавлен неключевой атрибут: **единица измерения.**

На этом этапе принимается решение импортировать произведенные в AEDM изменения в ACM, для этого в ACM выбирается пункт меню Tools – ERwin – Import. В первом окне мастера необходимо выбрать, из какого источника (xml или er1 файла) будет производиться импорт. Во втором окне необходимо указать элементы модели данных, построенной в AEDM, которые будут импортированы в модель ACM (окно, аналогичное указанному в рисунке 2). В окне под названием XMI Difference будут указаны все различия между файлом AEDM и моделью ACM, в которую будут импортироваться элементы модели данных. На этом этапе у пользователя есть возможность подтвердить либо отклонить те или иные элементы, которые будут импортированы.

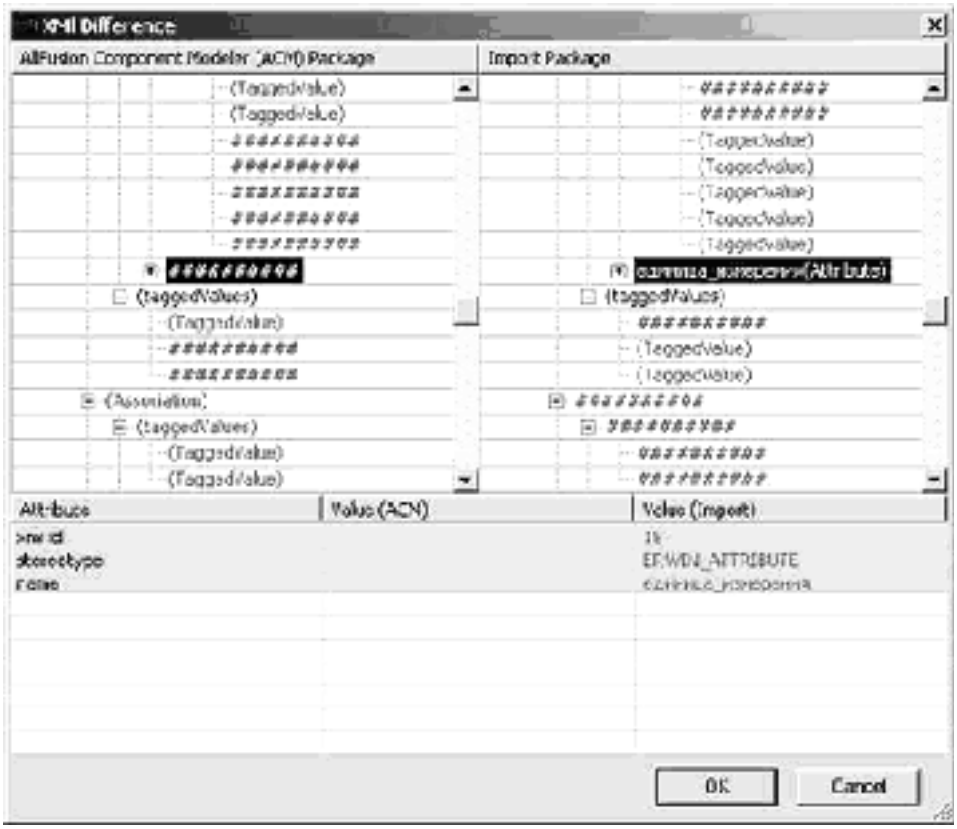


Рисунок 4. Просмотр различий между файлом AEDM и моделью данных ACM.

После завершения работы мастера выбранные элементы импортируются в модель ACM, что можно продемонстрировать на следующем рисунке. В дереве модели появились два новых класса: «**способ доставки**» и «**фирма производитель**» и у класса «**продукт**» появился атрибут **единица измерения.**

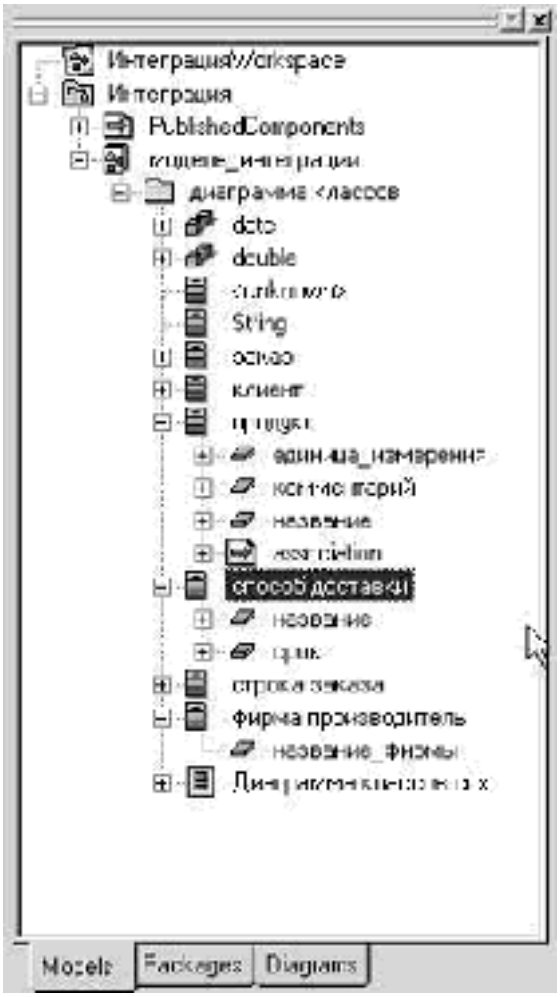


Рисунок 5. Дерево модели АСМ с импортированным и элементами.

На этом простом примере была продемонстрирована возможность интеграции AllFusion ERwin Data Modeler и AllFusion Component Modeler с помощью утилиты AllFusion Component Modeler-ERwin Data Modeler Add-In.

Использование языка макрокоманд в AllFusion ERwin Data Modeler

Козодаев А.А.
технический специалист компании Interface Ltd.

На современном этапе редко какое предприятие имеет единую информационную структуру. Как правило, информационный отдел организации имеет мозаичную структуру, где каждый элемент мозаики является решением отдельной задачи или подразделения и реализован в соответствии с параметрами этой задачи. Такая мозаика во многом определяется историческим развитием организации. К примеру, на гипотетическом предприятии первоначально был автоматизирован бухгалтерский учет при помощи собственной разработки, далее была внедрена сторонняя программа по учету заработной платы и т.д. Не секрет что такая участь была уготована и системам управления баз данных - от настольных баз MS Access до высокопроизводительных баз данных Oracle. В итоге практически перед любой организацией стоит задача унификации процесса создания баз данных с использованием различных СУБД, документирования и сопровождения уже существующих наработок. Именно таким инструментом является AllFusion ERwin Data Modeler (ранее: ERwin). Данный программный продукт выпускается американской компанией Computer Associates. ERwin DM поддерживает более 20 типов СУБД, как настольных, так и реляционных. Среди функциональных возможностей ERwin DM можно выделить следующие:

- *поддержка нескольких методик создания диаграмм сущность-связь*, на основе которых впоследствии создаются системные объекты выбранной СУБД, а именно: американский стандарт IDEF1X, методика IE во многом подобная стандарту IDEF1X и специальная методика, поддерживающая создание хранилищ данных.
- *прямое генерирование (Forward engineer).* Это функция создания системных объектов выбранного сервера базы данных на основе созданной в ERwin DM схемы данных.
- *обратное генерирование (Reverse engineer).* При помощи данной функции на основе существующей базы данных создается схема данных в ERwin, что позволяет эффективно документировать существующие базы данных.
- *полное сравнение (Complete compare).* Функция, которая позволяет сравнивать построенные в ERwin DM схемы данных между собой, либо схему данных с существующей базой данных, вследствие чего сокращается время, необходимое на сопровождение СУБД.

Этот список можно продолжить. Однако целью данной статьи является не перечисление всех функциональных возможностей ERwin DM, а рассказ о вспомогательных средствах, включенных в ERwin DM, которые позволяют существенно расширить функциональность данного продукта. К таковым можно причислить:

- встроенный API (ERwin API)
- поддержка XML
- использование правил преобразования типов данных, стандартов именования объектов схемы данных и глоссария
- использования построителя отчетов для извлечения данных по модели
- использование языка макрокоманд.

Далее мы рассмотрим именно язык макрокоманд, сферы его применения, а также приведем ряд практических примеров, демонстрирующих выгоду от использования макрокоманд.

Любая СУБД подразумевает поддержку определенной модели данных, которая определяет

структуру данных, и правила, по которым эта структура создается. На данный момент наиболее используемой моделью данных является реляционная модель. Первая практическая реализация реляционной модели была предпринята компанией IBM в семидесятые годы (1973-1978) прошлого столетия в продукте System R. Эта система была основой для всех последовавших за ней коммерческих продуктов, реализующих реляционный подход. В данном продукте был предложен язык манипулирования данными, который носил название SEQUEL (structured English query language), в дальнейшем получивший название SQL. В коммерческих продуктах язык SQL дополнялся, и в итоге появилось большое количество диалектов, не совместимых между собой. В 1987 году американским комитетом стандартизации был предложен первый стандарт языка SQL, в 1992 году был создан стандарт SQL-92, дополняющий SQL-87, в 1999 году предложена его третья версия, главным дополнением которой были объектные возможности и хранимые процедуры. Несмотря на процесс стандартизации, версии языка SQL очень сильно различаются от производителя к производителю. Язык макрокоманд ERwin DM позволяет расширить стандартный синтаксис языка SQL до диалектов конкретных баз данных. Однако это не единственная возможность использования макрокоманд в ERwin DM. Макрокоманды могут быть использованы:

- в доменах
- в настройках правил именования объектов схемы
- для создания триггеров и хранимых отображений
- для создания шаблонов скриптов.

Как гласит реляционная теория, домен является областью определения значений атрибута. Однако, несмотря на то, что некоторые производители СУБД заявляют о полной поддержке реляционного подхода в своих продуктах, на самом деле все положения этого подхода не реализованы ни в одной коммерческой СУБД, да и вряд ли когда будут реализованы полностью. Одним из таких нереализованных моментов является понятие домена – в современных СУБД оно подменено понятием типа данных, что является довольно примитивным решением. В ERwin DM тоже существует понятие домена и его можно представить шаблоном атрибута, обладающим набором свойств, в числе которых тип данных, значение по умолчанию, правило проверки и некоторые другие. В процессе работы рекомендуется использовать именно домены, и только потом создавать атрибуты, принадлежащие конкретному домену. Такой подход не лишен определенной гибкости и позволяет более эффективно редактировать схему данных. Действительно, гораздо удобней создать домен ИД, присвоить ему набор свойств и только потом создавать атрибуты на основе данного домена. В случае изменения, к примеру, длины поля домена ИД, гораздо проще исправить данные этого домена - и все входящие в него атрибуты автоматически будут изменены. Использование макрокоманд в доменах является простейшим примером применения макрокоманд в ERwin DM.

Рис. 1. Использование макрокоманд в определении доменов

На приведенном выше рисунке используется ряд макрокоманд, в результате действия которых имя атрибута будет выглядеть следующим образом: название_сущности домен_атрибута.

%Lower – макрокоманда, понижающая регистр
%OwnerEntity – макрокоманда, возвращающая имя сущности, к которой принадлежит атрибут
%AttDomain – макрокоманда, возвращающая имя домена, к которому принадлежит атрибут

При разработке крупных информационных систем, когда над моделью данных работают несколько специалистов, необходимо соблюдать правила именования объектов схемы данных. Например: названия сущностей должны быть представлены в верхнем регистре, названия атрибутов - в нижнем, таблицы на физическом уровне представления должны иметь префикс tbl и т.д. Конечно, можно создать документ, описывающий правила именования объектов схемы данных и затем административными мерами добиваться выполнения положений этого документа. Однако, в случае, когда количество таблиц в схеме начинает исчисляться сотнями, эта задача становится трудновыполнимой. В ERwin DM имеются специальные механизмы, позволяющие задать правила именования как логического, так и физического уровня схемы данных. В общем случае макрокоманды используются для преобразования имен объектов на логическом уровне в соответствующие имена на физическом уровне. Используя различные комбинации макрокоманд, можно добиться нужного поведения при именовании объектов схемы данных.

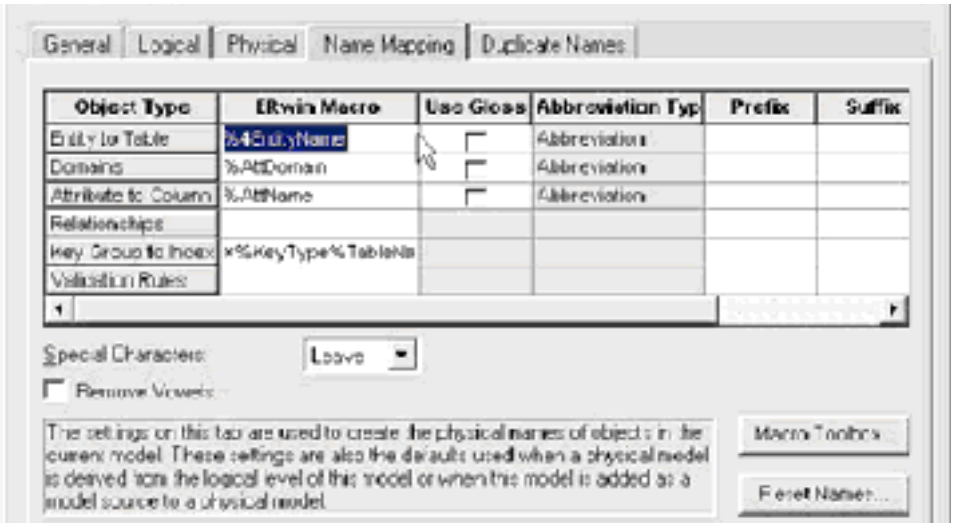


Рис. 2. Пример использования макрокоманд в редакторе именования объектов модели.

В приведенном примере используется макрокоманда вида %4EntityName, которая указывает на то, что таблица на физическом уровне представления будет иметь имя соответствующей сущности, ограниченное четырьмя первыми символами.

Выше были рассмотрены наиболее простые примеры использования языка макрокоманд ERwin DM. Перед тем, как рассмотреть более сложные примеры, познакомимся чуть ближе с самим языком макрокоманд.

Язык макрокоманд состоит из 195 команд.

В ERwin DM включена специальная панель инструментов – Macro Toolbox для работы с макрокомандами. С помощью данной панели можно выбрать нужную макрокоманду из списка, просмотреть синтаксис выбранной макрокоманды и получить справку по ее использованию.

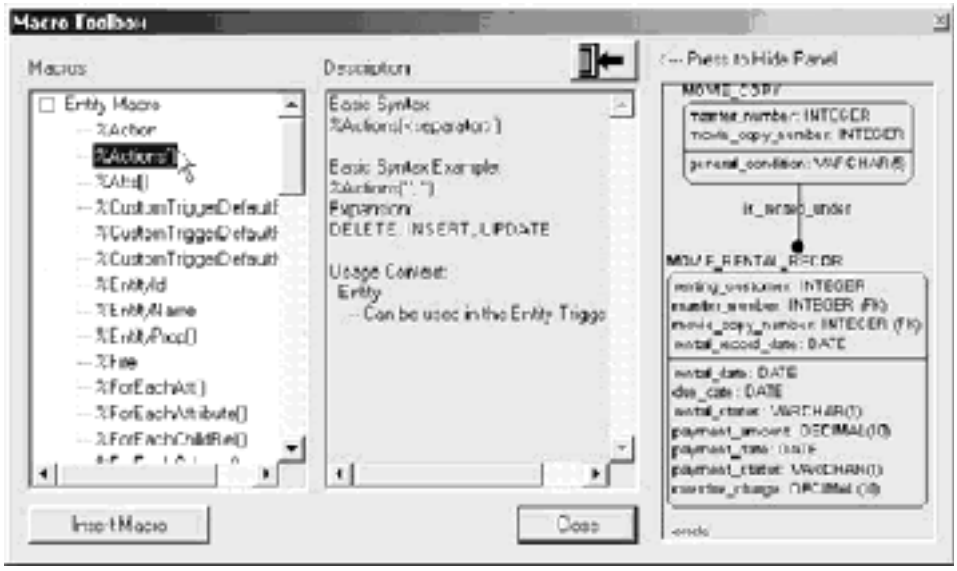


Рис. 3. Панель инструментов Marco Toolbox.

Рассмотрим синтаксис одной из макрокоманд ERwin DM.

%ForEachColumn(<table>,<separator>,<sort order>) { <macro code> }

Фраза **%ForEachColumn** является названием макрокоманды. Все фразы, заключенные в кавычки (<>), являются переменными макрокоманды, скобки и запятые указывают на необходимый синтаксис.

%ForEachColumn(cust,","){%ColName} - в результате действия этой макрокоманды будет выведен список столбцов таблицы cust.

Действия, выполняемые при помощи языка макрокоманд ERwin DM подобны действиям, выполняемым в других языках программирования, и включают в себя:

№	Тип действия	Макрокоманда
1	определение переменных	%ChildFKDecl %ChildNKDec l%ChildParamDecl %ChildPKDecl %Decl %NKDecl %ParamDecl %ParentNKDecl %ParentParamDecl %ParentPKDecl %PKDecl
2	выполнение арифметических операций	%- %* %/ %+
3	использование операций сравнения и логических операций	%!= %< %<= %= = %> %>= %And %Not %0r
4	ветвление	%If %Else %Switch

№	Тип действия	Макрокоманда
5	организация циклов	%ForEachAtt %ForEachAttribute %ForEachChildRel %ForEachColumn %ForEachDefault %ForEachDomain %ForEachEntity %ForEachFKAtt %ForEachFKAttribute %ForEachFKColumn %ForEachIndex %ForEachIndexMem %ForEachKey %ForEachKeyMem %ForEachLogEntity %ForEachParentRel %ForEachTable %ForEachValidation %ForEachValidValue %ForEachView %ForEachViewColumn
6	работа с внешними файлами	%File %Include %Lookup

Перейдем к использованию макрокоманд в триггерах. Как известно, одной из задач, стоящих перед современными СУБД, стоит задача поддержания логической целостности данных. Реляционные базы состоят из таблиц, связанных между собой при помощи механизма ключей. Для поддержания целостности между двумя таблицами используется триггер ссылочной целостности (RI триггер). В общем случае текст триггера зависит от типа связи таблицы, с которой он связан (триггер для родительской таблицы, триггер для дочерней таблицы) и от действия, при котором он срабатывает (добавление, изменение, удаление записи в таблице). В ERwin DM используется набор шаблонов для реализации триггеров ссылочной целостности. Тексты шаблонов зависят от типа сервера базы данных. Шаблоны представляют собой специальные скрипты, в которых используются макрокоманды ERwin DM. При генерации триггеров вместо макрокоманд подставляются имена таблиц, колонок, переменных и других фрагментов кода, соответствующих синтаксису выбранной макрокоманды. Для каждой комбинации ссылочной целостности (к примеру, Parent-Delete Cascade) в ERwin DM существует предопределенный шаблон, однако у пользователя имеется возможность переопределить этот шаблон, причем сделать это можно на трех уровнях: на уровне всей схемы, на уровне отдельной таблицы и на уровне отдельной связи. В дальнейшем на этапе генерации кода DDL будет использоваться шаблон, переопределенный пользователем.

Рассмотрим, каким образом используются макрокоманды в шаблонах триггеров ссылочной целостности. В схеме данных существуют две таблицы: отдел и сотрудник, между ними существует неидентифицирующая связь между полями **отдел.ид_отд** и **сотрудник.ид_отд.** В правилах ссылочной целостности этой связи была выбрана опция Parent-Delete RESTRICT, в результате которой запрещается удалять строку в таблице **отдел**, если существуют записи, ссылающиеся на эту строку из таблицы **сотрудник** .

Шаблон на данный триггер выглядит следующим образом:

```
/* ERwin Builtin %Datetime */
/* %Parent %VerbPhrase %Child 0N PARENT DELETE RESTRICT */
select count(*) into numrows
from %Child
where
/* %%JoinFKPK(%Child,:%%0ld," = "," and") */
%JoinFKPK(%Child,:%0ld," = "," and");
if (numrows > 0)
then
raise_application_error(
-20001,
'Cannot DELETE %Parent because %Child exists.'
);
end if;
```

При генерации будет создан следующий триггер

```
create trigger tD_Отдел after DELETE on Отдел for each row
-- ERwin Builtin Wed 0ct 15 17:06:44 2003
-- DELETE trigger on Отдел
declare numrows INTEGER;
begin
/* ERwin Builtin Wed 0ct 15 17:06:44 2003 */
/* Отдел R/2 Сотрудник 0N PARENT DELETE RESTRICT */
select count(*) into numrows
from Сотрудник
where
Сотрудник.ид_отд = :old.ид_отд;
if (numrows > 0)
then
raise_application_error(
-20001,
'Cannot DELETE Отдел because Сотрудник exists.'
);
end if;
end;
```

Помимо использования стандартных шаблонов пользователь может создать свой собственный шаблон. Приведем пример создания пользовательского шаблона триггера.

```
create trigger %TriggerName
%Fire %Actions(" or ")
on %TableName
%RefClause
%Scope
/* ERwin Builtin %Datetime */
/* default body for %TriggerName */
begin
Insert into Security (OldName, NewName, UserUpdate, UpdateDate)
values (:old1.CustomerName, :new1.CustomerName, User, Sysdate);
end;
/
```

В результате использования этой макрокоманды любое изменение в таблице Customer, будет фиксироваться в таблице Security, а именно прежнее значение имени, новое значение, дата изменения и имя пользователя, производившего изменения. На этапе генерации кода SQL эта макрокоманда будет выглядеть следующим образом.

```
create trigger SecurWrite
BEFORE UPDATE OF
CustomerName
on CUSTOMER
REFERENCING OLD AS old1 NEW AS new1
for each row
/* ERwin Builtin %Datetime */
/* default body for %TriggerName */
begin
Insert into Security (OldName, NewName, UserUpdate, UpdateDate)
values (:old1.CustomerName, :new1.CustomerName, User, Sysdate);
end;
/
```

Перенос логики приложения с клиентской части на серверную часть предоставляет целый ряд преимуществ, среди которых: снижение стоимости сопровождения системы, увеличение уровня безопасности приложения, уменьшение сетевого трафика. Основным средством реализации данного подхода является использование хранимых процедур. Для написания хранимых процедур в ERwin DM можно также использовать язык макрокоманд.

Создание хранимых процедур при помощи языка макрокоманд не отличается от создания триггеров.

Как уже было сказано выше, язык макрокоманд ERwin DM, также может быть использован для создания пре- и пост-скриптов. В ERwin DM есть возможность создания шаблонов скриптов, которые могут быть использованы на этапе прямого генерирования. Могут быть скрипты уровня всей схемы и уровня таблицы. По времени генерации скрипты могут делиться на прескрипты и постскрипты. Генерация кода прескрипта происходит перед генерацией основного объекта, с которым этот скрипт связан, и соответственно генерация кода постскрипта осуществляется после генерации объекта. Проиллюстрируем использование макрокоманд на примере создания прескрипта уровня всей схемы и постскрипта уровня таблицы.

Скрипт уровня схемы будет выполняться для всей схемы всего лишь раз, в отличие от скрипта уровня таблицы, текст которого может дублироваться для ряда - как таблиц, так и представлений.

Следующий прескрипт уровня схемы позволяет перед генерацией SQL кода проверить базу данных на наличие таблиц с именами, совпадающими с таблицами в схеме данных, и в случае если такие имеются, предварительно их удалить.

```
%ForEachTable() {
If Exists (select * from sysobjects where name = '%TableName')
Drop table %TableName
%DMBSDelim
}
```

примечание – скриптнаписан для СУБД MS SQL Server 2000

Этот скрипт легко можно преобразовать до скрипта уровня таблицы, удалив макрокоманду %ForEachTable(), в этом случае скрипт будет выглядеть следующим образом

```
If Exists (select * from sysobjects where name = '%TableName')
Drop table %TableName
%DMBSDelim
```

Следующий скрипт используется для предоставления привилегий на выборку из таблицы определенному пользователю.

```
grant select on %TableName to %TableProp(GrantSelect)
%DBMSDelim
```

В ERwin DM существует механизм свойств, определенных пользователем (UDP – user defined property), которые позволяют существенно расширить функциональность продукта. В приведенном выше примере используется UDP под названием GrantSelect, в котором перечислены все пользователи базы данных. На этапе моделирования каждой таблице присваивается данный UDP и в нем указываются пользователи, которые будут иметь привилегию на выборку из таблицы. На этапе генерации кода SQL выбранным пользователям будет предоставлена привилегия на выбор из соответствующих таблиц.

AllFusion ERwin Data Modeler является мощным средством, позволяющим создавать, документировать и сопровождать базы данных. Язык макрокоманд, включенный в состав данного продукта, позволяет генерировать SQL-код, приближенный к коду диалектов SQL конкретных производителей СУБД.

AllFusion ERwin Data Modeler API и ERwin Spy Utility

© Зайцев С.Л.

AllFusion ERwin Data Modeler API (ERwin API) предлагает усовершенствованные возможности для задания пользовательских настроек, которые позволяют приложениям заказчиков иметь доступ к моделям AllFusion ERwin Data Modeler (ранее: ERwin) и управлять ими. Клиенты ERwin API имеют возможность доступа к информации модели в открытых моделях ERwin, в файлах с расширением "er1" и в моделях, сохраненных в AllFusion Model Manager. В этой статье обсуждается утилита ERwin Spy – приложение, которое по выбору может быть установлено с ERwin. ERwin Spy демонстрирует функциональные возможности ERwin API, а также представляет ряд полезных возможностей для изучения хранения данных моделей в ERwin.

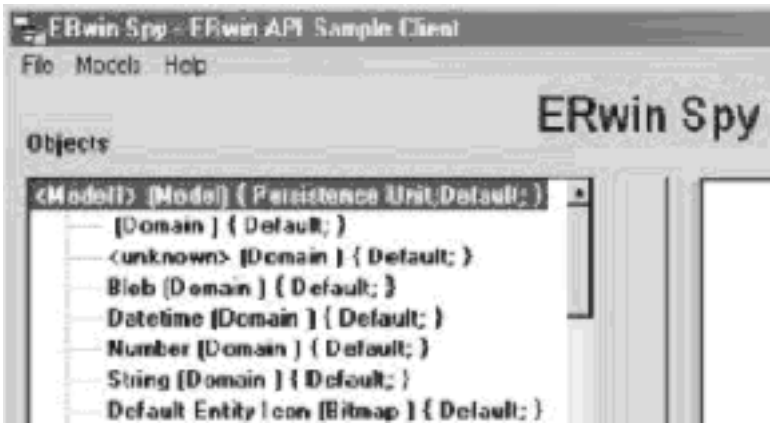
Установка ERwin включает два необязательных компонента API: ERwin API Sample Client и утилиту ERwin Spy. Эти компоненты предлагают пользователю ERwin пример программы-браузера ERwin API и саму утилиту ERwin Spy. Детали об установке данных компонентов и краткие инструкции о пользовании утилитой находятся в файле "… Doc\ERwinAPI_Readme.html".

В данной статье обсуждается только утилита ERwin Spy, важность которой трудно переоценить. Она позволяет заглянуть в устройство моделей ERwin и является необходимой для любого, кто хочет разрабатывать приложения с помощью ERwin API. Любая модель ERwin – это сложная сеть объектов модели, свойств и перекрестных ссылок. ERwin Spy считывает метамодель ERwin и упрощает понимание запутанных деталей любой модели ERwin.

В каталоге ERwin Spy имеется две доступные версии утилиты: автономный исполняемый файл ERwinSpy.exe и дополнительная динамически подсоединяемая библиотека ERwinSpy_AddIn.dll. Эти версии функционально эквивалентны и отличаются только способом запуска данного приложения. Автономная версия работает независимо от ERwin и имеет доступ к моделям, которые хранятся в файлах с расширением "er1". Вторая версия запускается из меню Tools программы ERwin и имеет доступ к моделям, находящимся в памяти ERwin или в er1-файлах. В дальнейшем обсуждается только библиотечная версия ERwin Spy. Детальное описание установки и регистрации утилиты как дополнительной библиотеки ERwin находится в файле ERwinAPI_Readme.html каталога DOC.

Начнем с пустой логико-физической модели. Для этого следует запустить ERwin и создать новую пустую логико-физическую модель. Затем нужно запустить ERwin Spy с помощью опции Add-Ins в меню Tools. В меню Модель утилиты ERwin Spy необходимо выбрать верхний пункт с названием новой модели.

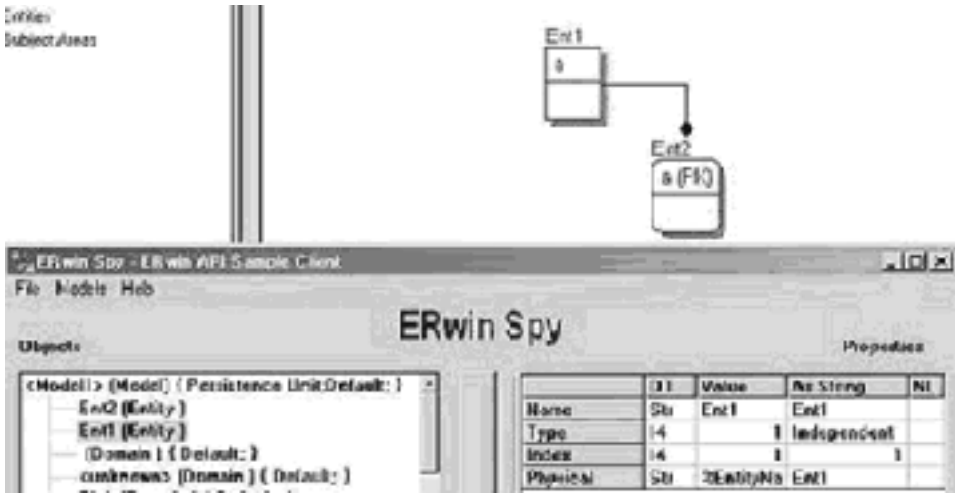
Далее нужно развернуть (двойным щелчком мыши) объект Model в левосторонней панели утилиты.



В результате этих действий должна появиться картинка, подобная той, что находится слева.

Как можно заметить, имеется довольно большое количество объектов, перечисленных в ERwin Spy. Несмотря на то, что модель является "пустой", эти объекты необходимы для представления параметров ERwin, заданных по умолчанию. Обратите внимание, что такие задаваемые по умолчанию параметры, как Domains, Main Subject Area, Trigger Templates и так далее, являющиеся объектами, отмечены флажком {Default; } справой стороны от типа объекта модели.

Теперь добавим некоторую информацию к данной модели. Для этого нужно закрыть ERwin Spy, вернуться к ERwin и создать пару сущностей, скажем, Ent1 и Ent2. Затем следует добавить атрибут первичного ключа к сущности Ent1 и переместить его к Ent2, путем создания взаимосвязи отождествления. Теперь снова откроем ERwin Spy. Следует обратить внимание на возникшее отличие. Теперь при развертывании объекта Model (двойной щелчок мышью) можно будет увидеть две сущности, которые были только что созданы. Выделим Ent1 и щелкнем на кнопку, расположенную в центре экрана. После этого, на правосторонней панели утилита покажет свойства выделенных объектов.



Ниже перечислено то, что теперь должен показывать ERwin Spy.

- В первом столбце показаны имена свойств, такие как Name, Physical Name и т.д.
- Второй столбец, озаглавленный DT, показывает типы свойств данных, такие как "Str" для параметра типа строка, I4 для числа, Bool для логического параметра, Id для ссылки на другой объект и т.д.
- Третий столбец, Value, отображает значение свойства в исходном формате.
- Четвертый столбец, As String, показывает значение свойства заново интерпретированного как параметр типа строка. Чтобы лучше понять, посмотрим на

четвертую линию диаграммы, представленной выше. Свойству с именем Physical в столбце Value соответствует макрос %EntityName, тогда как в As String содержится расширение макроса, "Ent1".

- Остальные столбцы на панели в правой части экрана (для того, чтобы их увидеть, возможно, понадобиться воспользоваться полосой прокрутки) представляют флаги свойств. Имеются следующие флажки:
    - *NL* для свойств с Null (без значения), данный флаг всегда неактивен для ERwin Spy;
    - *UD* для определенных пользователем свойств;
    - *VC* для векторных свойств;
    - *TL* для свойств, которые поддерживаются ERwin и не могут быть изменены с помощью API напрямую;
    - *RO* для свойств, предназначенных только для чтения;
    - *DR* для вторичных (derived) свойств, значения которых были унаследованы (из родительского домена, например).

Рассмотрим еще один пример. В левосторонней панели находим сущность Ent2, разворачиваем ее, чтобы увидеть ее дочерние объекты, затем находим атрибут a. Выделяем объект атрибута и смотрим его свойства на панели в правой части экрана.

Заслуживает внимание свойство Parent Relationship. Поскольку рассматриваемый атрибут является результатом перемещения внешнего ключа, это свойство показывает, какой из связанных объектов используется для хранения данных об этом. Значения Id в колонке DT показывает, что свойство является ссылкой. Это означает, что данное значение является уникальным идентификатором связанного объекта.

Для того чтобы отследить связанный объект, нужно просто посмотреть на его имя в колонке As String или найти объект с помощью его уникального идентификатора. Чтобы посмотреть идентификаторы объектов, необходимо сделать эту возможность доступной, поставив флажок Show Ids в Options из меню File. Тогда, если курсор расположен над объектом в левосторонней панели, то уникальный идентификатор появится во всплывающем окне.

Интересно сравнить свойство Parent Relationship со свойствами Parent Attribute и Master Attribute. Нужно заметить, что последнее свойство имеет доступ только на чтение, так как оно служит исключительно для информационных целей и не может быть изменено с помощью API. Откроем модель, чтобы посмотреть, как эти свойства используются в ERwin для представления различных взаимосвязей в данной модели.

Клиент ERwin XML – еще один пример приложения, основанного на ERwin API. В качестве интересного примера и для лучшего понимания того, как данные моделей представлены в языке XML, экспортируем одну из моделей в XML, используя опцию SaveAs из меню File и выбрав xml как тип. Затем открываем полученный файл в Internet Explorer и сравниваем результат с информацией, предоставляемой утилитой ERwin Spy.

В заключение стоит повторить, что использование преимуществ утилиты ERwin Spy является прекрасным способом для того, чтобы просмотреть и понять необходимые детали данных в модели ERwin, доступной при использовании ERwin API. Для того, чтобы понять, как конкретные данные представлены в модели ERwin, нужно просто следовать сценарию, описанному выше. Необходимо начать с пустой модели, создать минимальную модель необходимую для представления рассматриваемой особенности и затем воспользоваться утилитой ERwin Spy для просмотра деталей представления данных.

AllFusion ERwin Data Modeler API – это проще, чем кажется

Зайцев С.Л.

Обновленный программный интерфейс приложения (API) продукта AllFusion ERwin Data Modeler (ранее: ERwin) (версии 4.0 SP2 или более поздней) намного проще в использовании по сравнению со своим предшественником. В этой статье даются некоторые основные советы о начале работы с данным API, которые можно рассматривать как введение в "Справочное руководство API".

Перед чтением данной статьи следует убедиться в том, что установлены все компоненты, необходимые для использования AllFusion ERwin Data Modeler API. Устанавливая AllFusion ERwin Data Modeler версии 4.0 SP2 или более поздней, нужно выставить флажок для установки утилиты ERwin Spy Utility и ERwin API Sample Client. Эти компоненты устанавливаются в подкаталог "\Samples" основного каталога установки.

Более простая и согласованная модель объектов делает AllFusion ERwin Data Modeler API версии 4 проще в эксплуатации, чем ERwin версии 3.5. Ниже перечислены некоторые улучшения.

- Парные объекты (таблицы, столбцы и индексы) удалены, остались только категории, атрибуты и ключевые группы.
- Категории, атрибуты и ключевые группы теперь имеют логические и физические свойства, подобно доменам в API версии 3.5.
- В API версии 3.5 использовались свойства, атрибуты и признаки для представления информации, связанной с объектами модели. API версии 4 использует только свойства.
- Проблемы с обработкой многозначных свойств были разрешены благодаря новому, более простому и единообразному методу.

Это только несколько основных отличий между API версий 3.5 и 4. Однако код для версии 3.5 может быть беспрепятственно перенесен в версию 4. Такой переход к новому API выполняется очень несложно и код, получающийся в версии 4, будет проще.

Основная структура API версии 4 столь же проста и однородна, как и в версии 3.5.

1. Работа ведется с объектами и коллекциями объектов.
2. Объекты подразделяются на родительские и дочерние. Последние являются коллекцией объектов, для которых корневыми объектами (называемыми контекстом) являются родительские.
3. Создание объектов осуществляется добавлением их в коллекцию с одним из родительских объектов в качестве контекста.
4. При работе с уровнями модели нужно понимать, что объект на одном уровне дает коллекцию объектов на следующем уровне.

Одним из сходств API версий 4 и 3.5 является то, что оба они основаны на модели компонентных объектов Microsoft (COM) и фактически являются COM-серверами. Это означает, что любые инструменты, совместимые с COM, могут быть использованы для разработок с использованием API. Например, можно применять такие инструменты как Visual Basic, Visual C++, Delphi и другие. В примерах данной статьи используется VB, чтобы

сделать представление материала более простым и понятным. Перед началом работ с AllFusion ERwin Data Modeler API, используя VB, необходимо добавить в текущем проекте ссылку на файл SCAPI.dll, который является COM-сервером динамически подсоединяемых библиотек (DLL).

Как и в API версии 3.5, можно использовать версию 4 для разработки дополнительных функций. Дополнительные функции активируются с помощью опции Add-In в меню Tools программы AllFusion ERwin DM и могут быть использованы для работы с моделями, открытыми в программе. API можно также использовать для разработки автономных программ, работающих с моделями, которые открывают сами программы.

Фактически нет ограничений для расширений и интеграции, которые можно разрабатывать с помощью ERwin API.

Модель объектов в API версии 4

В этой статье будут обсуждаться три уровня API версии 4:

уровень приложения;
уровень сессии;
уровень модели.

Уровень приложений содержит четыре объекта, с которыми будет вестись работа:

объект приложения;
коллекция постоянных модулей;
объект постоянного модуля;
пакет свойств.

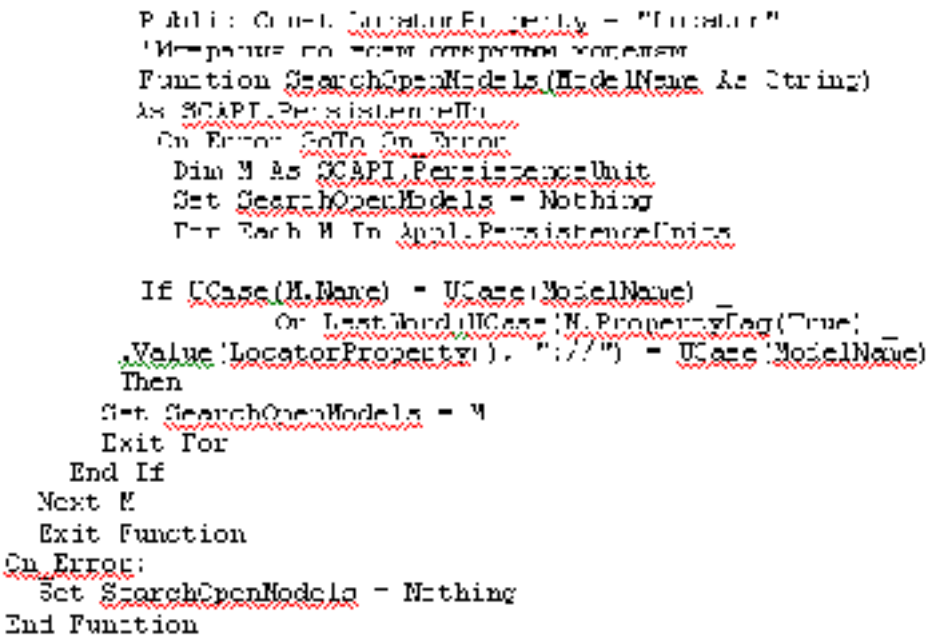
Объект приложения является точкой входа – необходимо задать хотя бы один объект данного типа. Для уровня приложения нужно использовать следующий код:

```
Dim Appl As SCAPI.Application
```

Постоянным модулем является либо модель ERwin DM, либо предметная область модели ERwin DM. Таким образом, коллекция постоянных модулей представляет уже открытые модели и используется либо для выбора открытой модели, либо для задания новой модели посредством добавления нового объект в коллекцию. Для того чтобы добавить новый объект в коллекцию можно использовать следующий код:

```
' Открытие новой модели через добавление объекта в коллекцию постоянных модулей
Private Model As SCAPI.PersistenceUnit
Set Model = Appl.PersistenceUnits _
.Add("erwin://" + ModelName, "RDO=No;OWT=No")
```

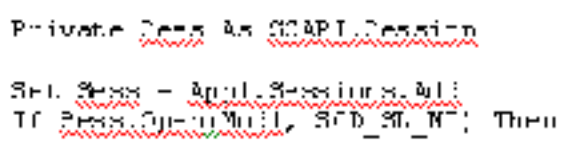
Однако если попытаться добавить объект к уже открытой модели, то возникнет ошибка "модель уже открыта в памяти". Для проверки того, открыта модель или нет, можно использовать следующую функцию:



Эта функция помимо того, что показывает, насколько легко проводить итерацию по всей коллекции, также реализует нужную операцию. Имя открытой модели содержит только имя файла, а не весь путь к нему. Это является некоторой проблемой, так как имя модели не уникально. Поэтому нужно использовать свойство Locator, которое задается префиксами "erwin://" или "mmart://" перед полным именем. Сама модель имеет свойство "file name", которое в принципе можно было бы использовать, но оно требует открытия модели. Поэтому использование локатора более эффективно.

После задания постоянного объекта необходимо начать сессию, в которой нужно открыть модель для работы с ней.

Уровень сессии содержит объект Session и коллекцию Sessions. Для уровня сессии можно использовать следующий код:



Если метод Open был выполнен успешно, то практически можно начинать работать с моделью. (Параметр SCD_SL_MO указывает уровень модели. M1 определяет уровень метамодели для доступа к данным метамодели. Для манипулирования моделями всегда используется параметр M0).

Чтобы работать с моделью дальше, нужно инициировать транзакцию. Это очень важный шаг. Если транзакция не будет начата, то при первой же попытке обновить любое из свойств возникнет ошибка. Активизация транзакции не будет успешной, если сессия закрыта или недействительна. Так же нужно следить за тем, чтобы идентификатор транзакции возвращался методом Open. Для безопасного и простого обращения с транзакциями лучше использовать следующие функции (здесь также используются некоторые особенности VB):

```
Private Trans As Variant

Public Function BeginTransaction() As Boolean
  If Not Sess Is Nothing Then
    If Sess.IsValid And Sess.IsOpen Then
      Trans = Null
      Trans = Sess.BeginTransaction
      BeginTransaction = Not IsNull(Trans)
    Or IsEmpty(Trans)
    Else
      BeginTransaction = False
    End If
  Else
    BeginTransaction = False
  End If
End Function

Public Function CommitTransaction() As Boolean
  CommitTransaction = False
  If Not Sess Is Nothing Then
    If Not IsNull(Trans)
    And Sess.IsValid Then
      If Sess.IsOpen Then
    'IsOpen cannot be used on invalid session
        CommitTransaction =
      Sess.CommitTransaction(Trans)
      End If
    End If
  End If
  Trans = Null
End Function

Public Function RollBackTransaction()
  As Boolean
  RollBackTransaction = False
  If Not Sess Is Nothing Then
    If Not IsNull(Trans) And Sess.IsVal
      And Sess.IsOpen Then
      RollBackTransaction =
    Sess.RollBackTransaction(Trans)
    End If
  End If
  Trans = Null
End Function

Public Function ActiveTransaction() As Boolean
  ActiveTransaction = False
  If Not Sess Is Nothing Then If Sess.IsValid
    Then If Sess.IsOpen Then
    ActiveTransaction = Not IsNull(Trans)
End Function
```

Следует помнить о том, что целостность модели после обновлений поверяется во время фиксации изменений. Например, если создается взаимосвязь между родительской и дочерней сущностями, не указав тип, который является обязательным свойством, то во время фиксации возникнет ошибка, сообщающая о том, что задание типа является обязательным, и создание взаимосвязи будет отклонено.

Производительность может пострадать, если будет слишком много обновлений не

прошедших фиксацию. Тест на портативном компьютере с частотой процессора 500 МГц и ОЗУ 512 MB показал, что проведение фиксации после 100-200 обновлений является оптимальным. Наличие 5000 и более обновлений без фиксации может существенно увеличить время прогона программы, в некоторых случаях вплоть до 5-10 раз. Оптимальный выбор проведения фиксации может зависеть от конкретного приложения.

Теперь обратимся к уровню модели, который содержит активные объекты. Ниже перечислены объекты, принадлежащие уровню модели.

Объект модели и коллекции объектов модели.
Объект свойств модели и коллекции свойств объектов модели.
Объект значения свойства модели и коллекции объектов значений свойств модели.

Доступ к объекту осуществляется с помощью коллекции объектов модели, возвращаемой методом ModelObjects уровня модели, или через коллекцию подмножеств, которую создает программный код, использующий метод Collect. Например, доступ к предметной области можно получить следующим образом:

```
Public Const SubjectAreaClass = "Subject Area"

Public Function SubjectAreas() As SCA.ModelObjects
        Set SubjectAreas _
        = Scsa.ModelObjects.Collect
 (Scsa.ModelObjects.Root, SubjectAreaClass)
End Function
```

Метод ModelObjects возвращает коллекцию объектов модели целиком, тогда как ModelObjects.Root дает только родительские или контекстные объекты из всей коллекции. Для того чтобы создать новую предметную область, нужно просто добавить новый объект к данной коллекции:

```
Public Const NameProperty = "Name"

Public Function CreateSubjectArea(Name As String)
  | As SCAPI.ModelObject
  Dim SA As SCAPI.ModelObject
  On Error Resume Next
  Set SA = Nothing
  Set SA = Scsa.ModelObjects.Add(SubjectAreaClass)
  SA.Properties.Item(NameProperty).Value(0) = Name
  Set CreateSubjectArea = SA
End Function
```

Добавление нового атрибута к сущности в точности соответствует той же парадигме. Следующие функции осуществляют это совместно:

```
Public Function Entity(Name As String)
    As ModelObject
  On Error Resume Next Set Entity = Nothing
 Set Entity = Sess.ModelObjects
  (Name, EntityClass)
End Function

Public Function Attributes
(Entity As SCAPI.ModelObject)
 As SCAPI.ModelObjects
  If ParmCheck("Entity", EntityClass, Entity)
  Then Exit Function On Error Resume Next
  Set Attributes = Nothing
  Set Attributes = Sess.ModelObjects.Collect
   (Entity, AttributeClass)
End Function

Public Function CreateAttribute
   (Entity As SCAPI.ModelObject,
    Name As String) As SCAPI.ModelObject
  If ParmCheck("Entity", EntityClass, Entity)
 Then Exit Function
  Dim A As SCAPI.ModelObject
  On Error Resume Next Set A = Nothing
  Set A = Attributes(Entity).Add(AttributeClass)
  A.Properties.Item(NameProperty).Value(0) = Name
  Set CreateAttribute = A
End Function
```

Эта функция облегчает использование API. В функции CreateAttribute новый атрибут сущности создается простым добавлением его в коллекцию атрибутов данной сущности.

Доступ к элементам коллекции объектов осуществляется следующим образом:

```
Public Function SubjectArea(Name As String) As SCAPI.ModelObject
  On Error Resume Next
  Set SubjectArea = Nothing
  Set SubjectArea = Sess.ModelObjects(Name, SubjectAreaClass)
End Function

Public Function MainSubjectArea() As SCAPI.ModelObject
  Set MainSubjectArea = SubjectArea("<Main Subject Area>")
End Function
```

Можно заметить, что присвоение значений выполняется для элементов коллекции свойств (например "A.properties.item(NameProperty)"), а не напрямую через "A.Name". Кроме того, присвоение значений свойствам всегда происходит в массивах (например, A.Properties.Item (NameProperty).Value(0) = Name). Это получается из-за того, что все значения свойств потенциально многозначны. Легко проверить, является ли свойство многозначным:

```
        Public Function PropertyIsScalar
(Proper x As SCAPI.ModelProperty) As Boolean
                PropertyIsScalar =
Property.Flags And S_D_MV_SCALAR
        End Function
```

Если значение не является скаляром, то назначение осуществляется нулевому элементу, как было показано выше. Если оно – скаляр, то число элементов можно получить с помощью свойства Count. Например, следующая функция обращает значение массива в коллекцию:

```
Public Function PropertyCollection
(Property As SCAPI.ModelProperty) _
    As Collection
  Dim Coll As New Collection
  If Property.Flags And STD_MEP_SCALAR Then
    Coll.Add Property.Value(0)
  Else
    Dim i As Long
    For i = 0 To Property.Count - _
      Coll.Add Property.Value(i), Property.Value(i)
    Next i
  End If
  Set PropertyCollection = Coll
End Function
```

Для доступа к свойствам используется метод HasProperty:

```
Public Function ObjectProperty(Obj As SCAPI.ModelObject
        , Property As String) As SCAPI.ModelProperty
  'Возвращается Nothing, если свойство отсутствует
  Set ObjectProperty = Nothing
  On Error Resume Next
  If Obj.Properties.HasProperty(Property) Then
    Set ObjectProperty = Obj.Properties(Property)
  End If
End Function
```

Другая ценная возможность API версии 4 – метод FormatAsString. Рассмотрим следующий пример:

```
Dim DSProp As SCAPI.ModelProperty
Set DSProp = ObjectProperty(PAtt, DataSourceProperty)
DSVal = DSProp.FormatAsString
```

в качестве альтернативы коду:

```
DSVal = DSProp.Value(0)
```

В этом случае, в результате будут возвращены идентификаторы объектов источников данных. Тогда, для получения объекта нужно использовать следующую функцию:

```
Public Function ObjectSet(Id As String) As SCAPI.ModelObject
  On Error Resume Next
  Set ObjectRef = Nothing
  Set ObjectRef = Sess.ModelObjects.Item(Id)
End Function
```

Однако если используется метод FormatAsString, API дает нужную интерпретацию даже тогда, когда значение типа "строка" и значения свойств разделены запятой.

В конечном итоге, важно помнить о том, что утилита ERwin SPY снабжена интерфейсом API для того, чтобы облегчить исследование того, как в точности назначены свойства, какие свойства существуют и какие являются обязательными. Эта утилита очень полезна при работе с API. Для получения более подробной информации рекомендуется обратиться к "Справочному руководству API" (ERwinAPI.pdf), которое находится в подкаталоге "\docs"

основного каталога установки.

Генерация отчетов по вашим моделям ERwin

Зайцев С.Л.,
к.ф.-м.н.

Работа с построителем шаблонов отчетов

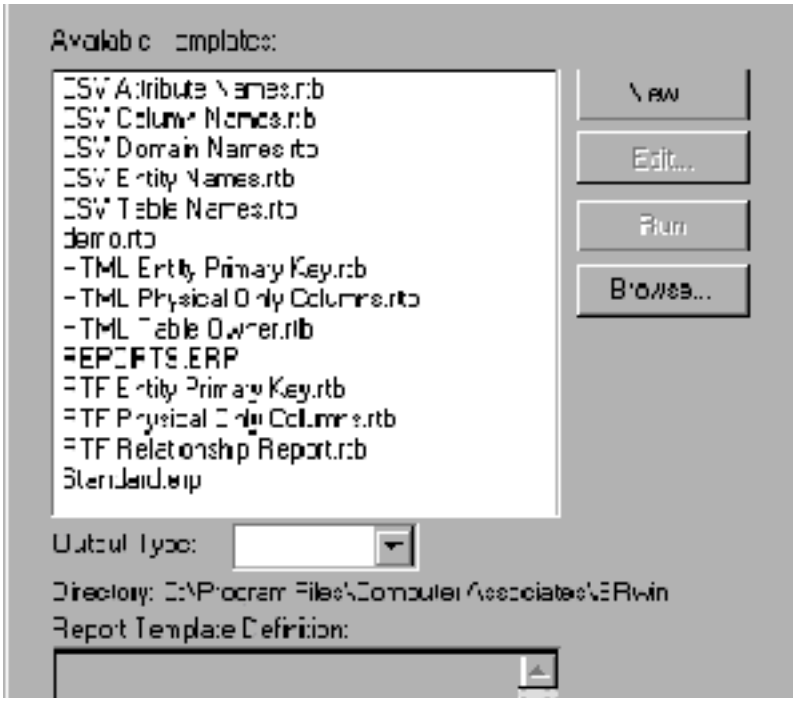
Построитель шаблонов отчетов создает в HTML, RTF и текстовом форматах предварительно определенные шаблоны отчетов, которые затем могут использоваться для генерации отчетов по любой модели. Например, если выбран формат RTF, создаваемый отчет будет автоматически вызывать локально установленное программное обеспечение для обработки текстов. Если выбран формат HTML, то создаваемый отчет будет запускать локально установленный web-браузер. Если же выбрать текстовый формат, то будет запускаться приложение Microsoft Excel.

Созданные отчеты можно просмотреть и сохранить, чтобы сделать их доступными для просмотра с помощью web-браузера другими пользователями.

Знакомство с возможностями построителя шаблонов отчетов

Можно предварительно ознакомится с построителем шаблонов отчетов, прежде чем фактически использовать его. Для этого выполните следующие операции:

1. После того, как построитель шаблонов отчетов будет открыт из меню Tools (Инструменты) в **ERwin**, нажмите F1, чтобы вывести на экран интерактивную справку.
2. Выберите один из разделов, что получить дополнительную информацию о построителе шаблонов отчетов или по какой либо другой теме.



Чтобы создать шаблон отчетов, выполните следующие операции:

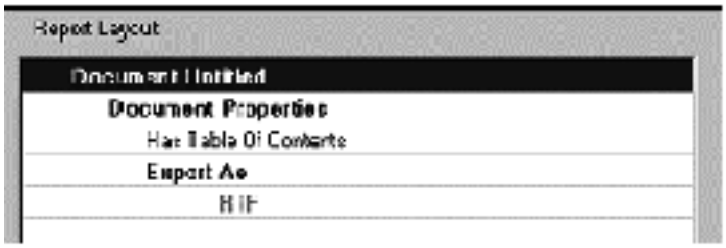
1. Откройте **eMovies.er1** и перейдите в режим логической модели.

2. Перейдите в меню **Tools**, выберите пункт **Report Builder** (Построитель отчетов). До тех пор пока первый шаблон не будет сохранен в папку **Reports** (Отчеты), будет отображаться сообщение, что папка пуста.

Также может отображаться сообщение, предлагающее щелкнуть кнопку браузера в следующем диалоговом окне и выбрать папку, в которую требуется сохранять отчеты.

Щелкните **New** (Создать), чтобы создать новый шаблон отчетов.

> **Подсказка:** *можно прочитать ил и выполнить ERP-файл отчета, созданный в браузере данных ERwin Data Browser, используя диалог Report Templates (Шаблоны отчетов). Выберите свой ERP– отчетиз списка или щелкните Browse (Просмотреть), чтобы выполнить поиск файла.*



3. В построителе шаблонов отчетов на панели **Report Layout** (формат отчета) дважды щелкните **Document Untitled** (Документ не озаглавлен).
4. Подтвердите предлагаемые по умолчанию настройки на вкладке **Property Tree** (Дерево свойств). Перейдите на вкладку **Title** (Заголовок).
5. Введите "**Простейший отчет для модели**: " в поле заголовка документа. Прежде чем выполнять следующий шаг, убедитесь, что в конце заголовка поставлен пробел.
6. Щелкните кнопку **Add Macro** (Добавить макрос), чтобы включить название модели в название отчета.
7. Перейдите на вкладку **Export** (Экспорт) и убедитесь, что в качестве формата для вывода отчета задан HTML. Если нет, выберите HTML из списка **Export AS** (Экспортировать как). Затем в групповом блоке **HTML Export Properties** (Свойства HTML при экспорте), определите для отчетов представление в виде картинок во всплывающих окнах.
8. Закройте это окно и вернитесь в панель построителя шаблонов отчетов.

Чтобы добавить разделы отчетов к шаблону отчетов, выполните следующие операции:

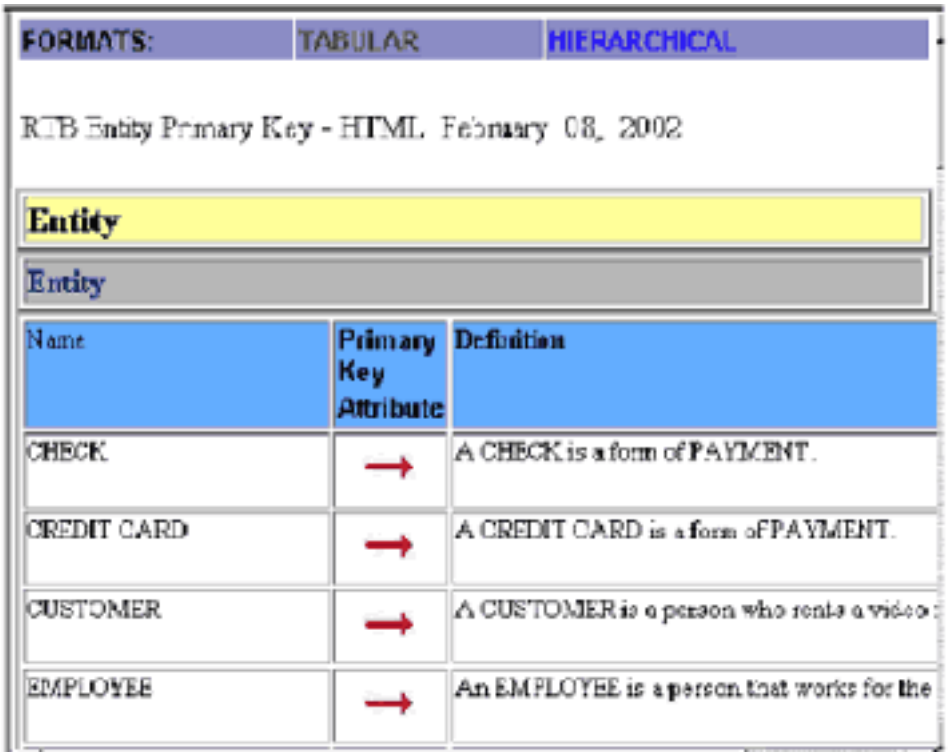
1. В построителе шаблонов отчетов в левой панели **Available Sections** (Доступные разделы) выберите **Picture** (Картинка), а затем щелкните стрелку вправо, чтобы добавить этот раздел на панель разметки отчета.
2. В панели **Available Sections** выберите раздел **Entity** (в Logical Section) и щелкните стрелку вправо, чтобы добавить этот раздел на панель разметки отчета.
3. На панели **Report Layout** дважды щелкните раздел **Entity**, чтобы открыть диалог **Properties** (Свойства) и указать подробные настройки отчета для этого раздела.
4. Щелкните знак "+", чтобы развернуть узел **Entity**, установите флажки для **Name** (Название) и **Definition** (Определение).
5. Щелкните значок "+" чтобы развернуть узел **Attribute**, установите флажок **Name** и закройте окно, чтобы вернуться в построитель шаблонов отчетов.

6. Сохраните это шаблон отчетов с названием **MyReport**.

Чтобы запустить HTML-отчет, выполните следующие операции:

1. Открыв **emovies.er1** в построителе шаблонов отчетов, щелкните на панели инструментов кнопку **Run** (Выполнить). При выполнении отчета построитель шаблонов отчетов запускает веб-браузер с отображаемым в нем отчетом.
2. В левом фрейме в окне браузера найдите ссылки для компонентов отчета. Щелкните ссылки для разделов картинки **Picture** и **Entity**, чтобы просмотреть каждый из них.
3. В разделе **Entity** можно изменить представление отчета с табличного на иерархическое.
4. Закройте веб-браузер.
5. Вернитесь в **ERwin** и закройте построитель шаблонов отчетов. Закройте файл **emovies.er1**.

Отчет, созданный в формате HTML на основе модели **emovies.er1**, выглядит следующим образом:



Чтобы применить шаблон к другой модели выполните следующие операции:

1. Откройте файл **emovies-L1.er1**.
2. Щелкните кнопку построителя шаблона отчетов на панели инструментов.
3. В окне диалога **Report Templates** (Шаблоны отчетов), запущенного из списка **Available Templates** (Доступные шаблоны), выберите шаблон **MyReport**.
4. Из списка **Output Type** (Формат вывода), выберите **RTF**.
5. Щелкните кнопку **Run**, чтобы запустить локально установленный текстовый процессор и отобразить отчет в окне документов.

После того как отчет будет просмотрен, закройте свой текстовый процессор, а затем закройте файл **emovies1.er1**.

Понятие отношения

Зайцев С.Л., к.ф.-м.н.

В предыдущих статьях были описаны основные концепции моделирования данных, сущностей и атрибутов для выявления и определения требований к информации, предъявляемых моделируемой областью бизнеса. Как вы уже знаете, логическая модель представляется на ERD сущностями, атрибутами и отношениями. В этой статье основное внимание будет уделено детальному рассмотрению отношений.

В следующих разделах будут освещены следующие вопросы, касающиеся отношений:

- Роль отношений в процессе моделирования
- Свойства отношений, такие как степень и направление
- Основные виды отношений (идентифицирующие и не идентифицирующие), количество элементов и обязательность
- Распространенные ошибки при выявлении отношений

На ER диаграммах сущности представляют контейнеры для атрибутов, атрибуты представляют интересующие нас факты, а отношения позволяют вам применять реляционную алгебру для связанных групп атрибутов в соответствие с бизнес-правилами. Давайте теперь рассмотрим отношения - сердце реляционной модели данных.

Что такое отношения?

Отношение - это ассоциация или "связь" между двумя сущностями. Отношение представляется в модели линией, соединяющей две сущности и глагольной конструкцией, которая описывает, как две сущности зависят друг от друга. Глагольная конструкция - механизм описания бизнес-правил, определяющих отношение. Хорошая глагольная конструкция описывает отношение в терминах бизнеса, а не на языке технических спецификаций. На Рисунке 4.1 показано отношение между двумя сущностями.

Рис. 4.1. На ER диаграммах отношения представляются линией между двумя сущностями, или замкнутой на саму сущность. Тип и свойства отношения представляют бизнес-правила. Экземпляр сущности БАНАНОВЫЙ ДЕСЕРТ может иметь

один, два, три или ни одного экземпляра сущности
ВЕРХУШКА БАНАНОВОГО ДЕСЕРТА.
Экземпляр сущности ВЕРХУШКА БАНАНОВОГО ДЕСЕРТА
может принадлежать одному, и только одному экземпляру
сущности БАНАНОВЫЙ ДЕСЕРТ.

Отношения используются и в логической и в физической модели, и представляются в них в виде одного или нескольких мигрирующих атрибутов внешнего ключа. Отношения двунаправлены и представляют значимые ассоциации между двумя сущностями или сущности самой с собой.

Отношение обладает следующими свойствами:

- Степень
- Направленность
- Тип
- Количество элементов
- Обязательность

> **ПРИМЕЧАНИЕ**
> Важно иметь в виду, что не все методологии моделирования используют именно эти термины для описания отношений, хотя большинство из них поддерживают сходные концепции.

Выявление отношений

Другим хорошим источником является корпоративная модель.

Проверьте отношения между сущностями в рамках моделируемой предметной области. Внимательно изучите внешние ключи, которые определяют отношения между кодовыми и другими сущностями. Эти отношения могут помочь выявить сущности, которые должны быть добавлены в вашу модель, или определить уровень абстракции, который улучшит ее расширяемость.

Вы можете выявить отношения и во время нормализации при появлении новых сущностей, связанных отношениями. Разрешение отношений **многие-ко-многим** в логической модели, скорее всего, приведет к появлению дополнительных сущностей и отношений.

> ПРИМЕЧАНИЕ
> Редко в приложении все сущности связаны друг с другом отношениями попарно. Если в модели присутствует только несколько сущностей, то они все могут быть связаны отношениями. Однако в более крупных приложениях отношениями могут быть связаны не все сущности.

Чтение отношений

Определив подходящую глагольную конструкцию, вы можете читать отношения справа налево, используя шаблон сущность - глагольная конструкция для отношения - сущность. Для примера, с магазином мороженого вы можете определить отношение:

Потребитель покупаетСмесь

Или

Магазин продаетразные сорта Мороженого

Чтение отношений на основе использования этой структуры - сущность - отношение - сущность (иногда называемой парная связь сущностей) - является полезным механизмом для представления отношений партнерам по бизнесу. Парные связи сущностей двунаправлены. Так что сущности связаны в обоих направлениях. Таким образом, *Потре бите ль покупает Смеси* то же самое, что и *Смеси продаются Потребителю* .

Такой способ прочтения отношений обеспечивает проверку корректности разработанной логической модели. Хотя отношения и не описывают полностью все бизнес-правила, они позволяют бизнес-партнерам просматривать модель для понимания взаимосвязей между сущностями. Я считаю, что это лучший способ убедиться, что каждая глагольная конструкция в модели формирует корректное утверждение. Просмотр модели вместе с бизнес-партнерами - один из основных методов проверки того, что модель правильно отражает бизнес-правила.

Степень отношения

Степень отношения представляет собой число сущностей, ассоциированных с отношением. В основном, отношения имеют степень единица (унарные отношения) или двойка (бинарные отношения). *Унарные*, или *рекурсивные* отношения представляют случаи, когда экземпляр сущности связан с другим экземпляром той же самой сущности. Бинарные отношения представляют случаи, когда одна сущность связана с другой. *Бинарные* отношения отражают наиболее распространенные взаимосвязи, присутствующие в реальном мире. Фактически, большинство разработчиков моделей рассматривают унарные или рекурсивные отношения как бинарные рекурсивные отношения, связывающие экземпляр сущности с другим ее экземпляром. Например, высказывание "Некоторыми сотрудниками руководят другие сотрудники" представляет отношение между экземплярами сущности СОТРУДНИК.

> **ПРИМЕ ЧАНИЕ**
> В некоторых методологиях существуют n-арные отношения (n представляет степень отношения). В n-арных отношениях участвуют более двух сущностей. Например, "Потребитель покупает Смесь в Магазине" является тернарным (степени три) отношением, вовлекающим три сущности. Однако большинство методологий оперируют только бинарными отношениями и преобразуют все n-арные отношения в бинарные.

Направленность отношения

Направленность отношения указывает на исходную сущность в отношении. Сущность, из которой отношение исходит, называется *родите льской сущностью* . Сущность, в которой отношение заканчивается, называется *подчиненной сущностью* .

Направленность отношения определяется взаимосвязью между сущностями. В отношении между независимой и зависимой сущностями, отношение *исходит* из независимой сущности и заканчивается в зависимой сущности. Если обе сущности независимые, отношение симметрично. В отношении **один-ко-многим** родительской является сущность, входящая в

отношение однократно. Отношения **многие-ко-многим** - симметричны.

Определение типов отношений

В ERwin отношение между двумя сущностями, или сущности самой с собой, может принадлежать к одному из следующих типов:

- Идентифицирующее отношение
- Не идентифицирующее отношение
- Типизирующее отношение
- Отношение **многие-ко-многим**
- Рекурсивное отношение

Каждый тип отношений определяет поведение атрибутов первичного ключа, когда они мигрируют из родительской сущности в подчиненную. В следующих разделах будут описаны особенности каждого из типов отношений.

Идентифицирующие отношения

Идентифицирующим является отношение между двумя сущностями, в котором каждый экземпляр подчиненной сущности идентифицируется значениями атрибутов родительской сущности. Это означает, что экземпляр подчиненной сущности зависит от родительской сущности и не может существовать без экземпляра родительской сущности. В идентифицирующем отношении единственный экземпляр родительской сущности связан с множеством экземпляров подчиненной. Атрибуты первичного ключа родительской сущности мигрируют в атрибуты подчиненной, чтобы стать там атрибутами первичного ключа. На Рисунке 4.2 представлено идентифицирующее отношение между сущностями ВКУСОВАЯ ДОБАВКА и МОРОЖЕНОЕ. Заметьте, что атрибуты первичного ключа родительской сущности ВКУСОВАЯ ДОБАВКА мигрировали в сущность МОРОЖЕНОЕ и стали там первичным ключом.

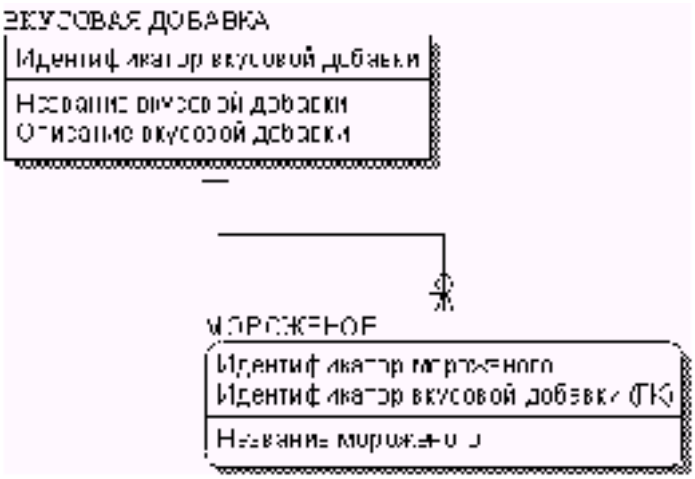


Рис. 4.2. Идентифицирующее отношение между сущностями МОРОЖЕНОЕ и ВКУСОВАЯ ДОБАВКА.

Экземпляр сущности МОРОЖЕНОЕ не существует до тех пор, пока не появится экземпляр родительской сущности ВКУСОВАЯ ДОБАВКА. Обратите внимание на непрерывную линию между двумя сущностями.

Неидентифицирующие отношения

Неидентифицирующим является отношение между двумя сущностями, в котором каждый экземпляр подчиненной сущности не зависит от значений атрибутов родительской сущности. Это означает, что экземпляр подчиненной сущности не зависит от родительской сущности и может существовать без экземпляра родительской сущности. В неидентифицирующем отношении единственный экземпляр родительской сущности связан с множеством экземпляров подчиненной. Атрибуты первичного ключа родительской сущности мигрируют в подчиненную, чтобы стать там не ключевыми атрибутами. На рисунке 4.3 представлено неидентифицирующее отношение между сущностями СМЕСЬ и ТИП СМЕСИ. Обратите внимание, что атрибуты первичного ключа родительской сущности СМЕСЬ мигрировали в подчиненную сущность ТИП СМЕСИ и стали там неключевыми атрибутами.

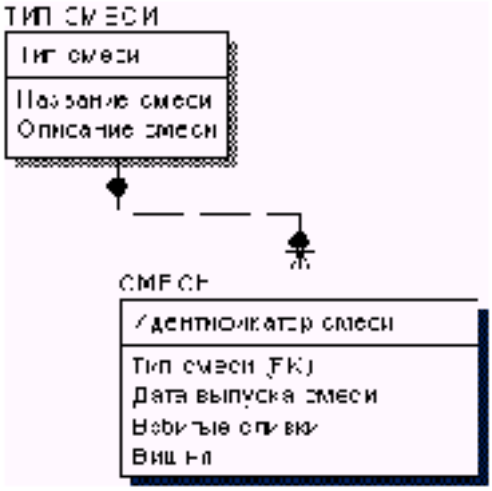


Рис. 4.3. Неидентифицирующее отношение
между сущностями СМЕСЬ и ТИП СМЕСИ.

Обратите внимание, что ERwin изображает не идентифицирующее отношение в виде пунктирной линии.

Типизирующие отношения

Типизирующими являются отношения между родительской и одной или более подчиненными сущностями. Вы должны использовать типизирующие отношения в том случае, если имеет смысл указать на такие отношения, когда экземпляр родительской сущности определяет различные наборы атрибутов в подчиненных сущностях.

Клайв Финклештейн называет характеристические сущности вторичными сущностями. Характеристические сущности всегда имеют одну или более "равноправных" сущностей. Равноправные характеристические сущности связаны с родительской сущностью особым типом отношений, которые могут быть исключающими или включающими.

Исключающие типизирующие отношения

Исключающие типизирующие отношения указывают, что только одна подчиненная сущность идентифицируется родительской сущностью. Другими словами, экземпляр родительской сущности связаны с экземплярами не более чем одной вторичной сущности.

Исключающая характеристическая сущность представляет отношение "является" (is a). На рисунке 4.4 изображено исключающее типизирующее отношение между сущностью верхнего уровня СМЕСЬ и двумя характеристическими сущностями БАНАНОВЫЙ ДЕСЕРТ и СЛИВОЧНАЯ ПОМАДКА.

Рис. 4.4. Исключающее типизирующее отношение
между сущностью верхнего уровня СМЕСЬ
и двумя подчиненными характеристическими сущностями
БАНАНОВЫЙ ДЕСЕРТ и СЛИВОЧНАЯ ПОМАДКА.

Заметьте символ подтипа X, указывающий, что это исключающее типизирующее отношение в нотации IE системы ERwin. Представленное бизнес-правило утверждает, что экземпляр сущности СМЕСЬ может быть сущностью БАНАНОВЫЙ ДЕСЕРТ или СЛИВОЧНАЯ ПОМАДКА, но не ими обеими одновременно.

Включающие типизирующие отношения

Включающее типизирующее отношение указывает, что экземпляром родительской сущности могут определяться более одной вторичной сущности. Другими словами, экземпляр родительской сущности связан с экземплярами нескольких вторичных сущностей. Рисунок 4.5 демонстрирует включающее типизирующее отношение между родительской характеристической сущностью ПЕРСОНА и двумя подчиненными сущностями КЛИЕНТ и СОТРУДНИК.

Отношения многие-ко-многим

Отношения *многие-ко-многим* возникают там, где один экземпляр одной сущности связан с несколькими экземплярами другой, и один экземпляр этой другой сущности также связан с несколькими экземплярами первой сущности. Эти отношения также называют неспецифическими. Отношения **многие-ко-многим** используются на предварительных стадиях разработки логической модели. Обычно они разрешаются за счет использования ассоциативной сущности, содержащей ключи родительских сущностей. Ассоциативные сущности позволяют экземплярам каждой из родительских сущностей быть представленными в виде уникальной пары во вторичной сущности. Рисунок 4.6 показывает отношение многие-ко-многим между сущностями СЛИВОЧНАЯ ПОМАДКА и МОРОЖЕНОЕ.

Рис. 4.5. Включающее типизирующее отношение между
сущностью верхнего уровня ПЕРСОНА и двумя подчиненными
характеристическими сущностями КЛИЕНТ и СОТРУДНИК.

Обратите внимание на отсутствие символа подтипа X, указывающее, что это включающее типизирующее отношение в нотации IE системы ERwin. Представленное бизнес-правило утверждает, что экземпляр сущности ПЕРСОНА может быть одновременно сущностями КЛИЕНТ и СОТРУДНИК.

Рис. 4.6. Отношение **многие-ко-многим**
между сущностями СЛИВОЧНАЯ ПОМАДКА и МОРОЖЕНОЕ.

В нотации IE, ERwin представляет отношения многие-ко-многим непрерывной линией с "птичьей лапкой" на обоих концах.

Рекурсивные отношения

Рекурсивное отношение - это неидентифицирующее отношение между двумя сущностями, которое указывает, что экземпляр сущности может быть связан с другим экземпляром той же самой сущности. При рекурсивном отношении родительская и подчиненная сущности совпадают. На Рисунке 4.7 показаны примеры двух реализаций рекурсивного отношения для сущности СОТРУДНИК, с использованием названия роли и без него. Обратите внимание, что ERwin "унифицирует" атрибуты внешнего ключа и первичного ключа, когда название роли не используется. Использование имени роли приводит к размещению внешнего ключа в качестве неключевого атрибута.

ПРИМЕ ЧАНИЕ
Поскольку рекурсивные отношения могут оказаться непривычными для пользователей и начинающих разработчиков моделей, хорошей практикой является использование названий ролей для мигрирующих атрибутов. В последующих статьях будет описано использование названий ролей и интересные особенности унификации в ERwin.

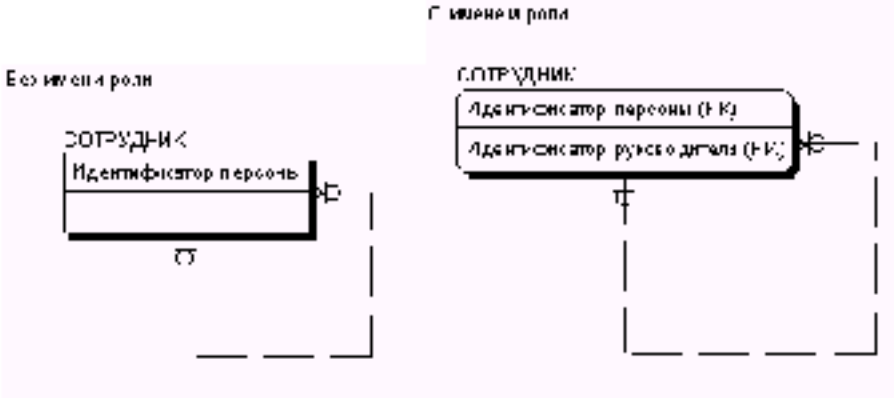


Рис. 4.7. Примеры реализации рекурсивных отношений
с использованием названия роли и без него в сущности ПЕРСОНА.

Заметьте, что без названия роли ERwin комбинирует атрибуты внешнего и первичного ключей.

Количество элементов отношения

Количество элементов отношения задает максимальное число экземпляров одной сущности, которые могут быть связаны с экземплярами другой сущности. Количество элементов определяется для обеих сторон отношения - для исходной и завершающей сущностей. Количество элементов определяет максимальное количество экземпляров сущностей, участвующих в отношении, в то время как обязательность определяет минимальное число экземпляров. Подробнее об обязательности вы прочтете в следующем разделе.
Количество элементов часто выражается как *один* или ***много.*** Один и много могут появляться в трех различных комбинациях:

- **Один-к-одному** (1:1) - один и только один экземпляр сущности связан с одним и только одним экземпляром другой сущности.
- **Один-ко-многим** (1:N) - один и только один экземпляр родительской сущности связан со многими экземплярами подчиненной сущности.
- **Многие-ко-многим** (M:N) - много экземпляров одной сущности связаны с многими экземплярами другой сущности (также называется неспецифическим отношением).

Один-к-одному

В отношении один-к-одному один и только один экземпляр сущности связан с одним и только одним экземпляром другой сущности.

Эти отношения иногда являются результатом нормализации, когда удаляются атрибуты, имеющие значения не для всех экземпляров сущности. Позаботьтесь проверить атрибуты родительской сущности, участвующей в отношении, для определения классификационного

атрибута, чьи значения определяют, существует ли экземпляр в зависимой сущности. Если классификационный атрибут существует, рассмотрите возможность использования типизирующего отношения вместо отношения один-к-одному.

> **Совет**
> В действительности, отношения **один-к-одному** встречаются редко. В большинстве случаев вы должны собрать атрибуты в одной сущности. Вы должны также проверить отношение, чтобы убедиться, что отношение **один-ко-многим** не было ошибочно представлено в модели в качестве отношения **один-к-одному**.

Рисунок 4.8 демонстрирует отношение один-к-одному между сущностями ПЕРСОНА и АДРЕС ПЕРСОНЫ.

Рис. 4.8 Пример отношения один-к-одному
между сущностями ПЕРСОНА и АДРЕС ПЕРСОНЫ.

Представленное бизнес-правило утверждает, что экземпляр сущности ПЕРСОНА может иметь ровно один адрес. Как и многие другие отношения один-к-одному, это отношение в действительности является некорректно представленным в модели отношением один-ко-многим.

Один-ко-многим

В отношении один-ко-многим один и только один экземпляр сущности связан со многими экземплярами другой сущности. Сущность, входящая в отношение единственным экземпляром является родительской или исходной сущностью. Сущность, входящая в отношение многими экземплярами является подчиненной или конечной сущностью.

> ПРИМЕЧАНИЕ
> Большинство разработчиков моделей согласны с мнением, что логическая модель должна содержать только отношения **один-ко-многим**.

Эти отношения часто являются результатом нормализации, когда атрибуты повторяющейся группы переносятся в зависимую сущность. На Рисунке 4.9 показано отношение между сущностями ПЕРСОНА и АДРЕС ПЕРСОНЫ, корректно представленное в модели в виде отношения **один-ко-многим.**

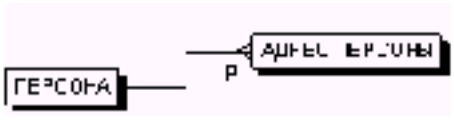


Рис. 4.9. Пример отношения **один-ко-многим**
между сущностями ПЕРСОНА и АДРЕС ПЕРСОНЫ.

*Представленное бизнес-правило утверждает, что экземпляр сущности ПЕРСОНА может иметь более одного адреса. Это отношение **один-ко-многим** исправляет ранее*

(неправильно) представленное в модели отношение ***один-к-одному*** .

Многие-ко-многим

Отношения **многие-ко-многим** возникают там, где один экземпляр одной сущности связан с несколькими экземплярами другой, и один экземпляр этой другой сущности также связан с несколькими экземплярами первой сущности. Эти отношения еще называют неспецифическими.

Отношения **многие-ко-многим** должны использоваться только на предварительных стадиях разработки логической модели. Поскольку отношения **многие-ко-многим** часто скрывают важные бизнес-правила или ограничения, они должны быть полностью преобразованы в ходе процесса моделирования. Разрешение отношения **многие-ко-многим** требует создания ассоциативной сущности, содержащей ключи обеих родительских сущностей и представляющей экземпляры каждой из родительских сущностей в виде уникальной пары во вторичной сущности.

> ПРЕДОСТЕРЕЖЕНИЕ
> Отношения многие-ко-многим являются логическими и не могут быть представлены в физической модели в ERwin. ERwin автоматически создает ассоциативные сущности для разрешения отношений многие-ко-многим когда модель преобразуется из логической в физическую.

Рисунок 4.10 показывает отношение между сущностями ПЕРСОНА и АДРЕС ПЕРСОНЫ, представленное в модели в виде отношения **многие-ко-многим** .

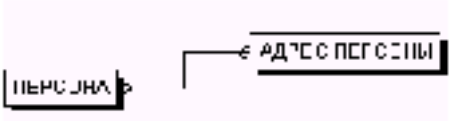


Рис. 4.10. Пример отношения многие-ко-многим
между сущностями ПЕРСОНА и АДРЕС ПЕРСОНЫ.

Это позволяет экземпляру сущности ПЕРСОНА иметь более одного адреса. В свою очередь экземпляр сущности АДРЕС ПЕРСОНЫ может иметь более одной персоны, живущей по этому адресу.

Обязательность отношения

В отличие от количества элементов, обязательность отношения определяет, должны ли экземпляры сущности участвовать в отношении. Обязательность иногда называют *модальностью* или *присутствием* . В то время как количество элементов определяет максимальное количество экземпляров сущностей, которые могут участвовать в отношении, обязательность определяет минимальное число экземпляров, которые должны участвовать в отношении. Значение обязательности равно нулю в том случае, если экземпляр сущности не обязателен или не требуется, и равно единице, если наличие сущности требуется или обязательно.

Принимая решение об обязательности отношения, определите, должен ли экземпляр одной сущности всегда присутствовать, для того чтобы другая сущность участвовала в отношении. Если так - отношение обязательно. Так утверждение "Каждый сотрудник должен иметь

одного руководителя" - пример обязательного отношения. Если экземпляр сущности не требуется - отношение необязательно. Например, утверждение "Некоторые из сотрудников могут иметь одного руководителя" - задает необязательное отношение.

В обязательном не идентифицирующем отношении атрибуты, мигрировавшие в неключевую область подчиненной сущности, являются обязательными для подчиненной сущности. Это значит, что значение внешнего ключа не может быть пустым. Рисунок 4.11 демонстрирует обязательное отношение.

Рис. 4.11. Обязательное отношение между сущностями
ПЕРСОНА и АДРЕС ПЕРСОНЫ.

Представленное бизнес-правило утверждает, что экземпляр сущности ПЕРСОНА должен иметь хотя бы один адрес. Обратите внимание, что в нотации IE системы ERwin терминальная точка отношения не включает символ ноль (кружок) в основании "птичьей лапки" - символа множественности.

В необязательном неидентифицирующем отношении атрибуты, мигрировавшие в неключевую область подчиненной сущности, не являются обязательными для подчиненной сущности. Это значит, что значение внешнего ключа может быть пустым. Рисунок 4.12 демонстрирует необязательное отношение между сущностями ПЕРСОНА и АДРЕС ПЕРСОНЫ.

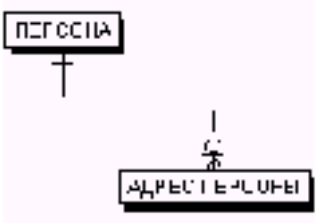


Рис. 4.12. Необязательное отношение между сущностями
ПЕРСОНА и АДРЕС ПЕРСОНЫ.

Представленное бизнес-правило утверждает, что экземпляр сущности ПЕРСОНА не нуждается в адресе. Обратите внимание, что в нотации IE системы ERwin терминальная точка отношения включает символ ноль (кружок) в основании "птичьей лапки" - символа множественности.

Распространенные ошибки, связанные с отношениями

Этот раздел, посвященный распространенным ошибкам при моделировании отношений, не претендует на полноту. В нем описаны некоторые ошибки, которые встречаются у разработчиков моделей.

Отношения являются сердцем реляционной модели. Неправильное представление в модели единственного отношения может нарушить целостность всей модели. ERwin не позволит вам совершить наиболее очевидные ошибки при использовании отношений.

Ненужные отношения

Назначение имени в форме глагольной конструкции, направленности, степени, типа, количества элементов и обязательности данному отношению должно подтвердить его корректность. Если вы не сможете обеспечить все эти характеристики для отношения, скорее всего оно просто вообще не является отношением.

Отношения один-к-одному

Внимательно проверьте атрибуты каждой из сущностей, участвующих в отношении один-к-одному. Проанализируйте атрибуты родительской сущности, участвующей в отношении, для нахождения классификационного атрибута, чьи значения определяют, существует ли экземпляр в зависимой сущности. Если классификационный атрибут существует, рассмотрите возможность использования типизирующего отношения вместо отношения **один-к-одному** . Как показывает опыт, реальные отношения **один-к-одному** очень редки и еще реже они реализуются на практике.

Отношения многие-ко-многим

В отношении **многие-ко-многим** множество экземпляров одной сущности связаны с одним экземпляром другой и наоборот. Отношения многие-ко-многим не могут быть транслированы в физические таблицы и связи базы данных. Это ограничение систем баз данных, а не прикладных программ. Вы должны преобразовать отношения **многие-ко-многим** перед тем, как конструировать физическую модель данных и проектировать базу данных. Разработчиком моделей рекомендуется преобразовывать отношения **многие-ко-м ногим** в ходе рабочих сессий. Участники сессии обычно могут определить подходящую сущность для такого преобразования.

Разрешение отношения **многие-ко-многим** означает преобразование его в два отношения: **один-ко-многим** и **многие-к-одному** . Между двумя исходными создается новая сущность, и эта новая сущность называется *ассоциативной* сущностью или *пересечением*. Добавление этой сущности позволяет создать экземпляр для каждого возможного соответствия между двумя другими сущностями. Иногда ассоциативная сущность определяет границы или точку во времени.

Необходимость или необязательность

При определении необязательности отношения и определении того, требуется ли наличие экземпляра одной сущности для существования экземпляра другой, важно убедиться, что требуемый экземпляр всегда доступен.

Можно попытаться обеспечить значения по умолчанию для требуемых внешних ключей. Однако часто выгода от заполнения необходимых данных фиктивными значениями невелика. Если информация так важна, что корпорация считает ее обязательной, в чем выгода от подставленных значений? Если существует вероятность, что сущность может не быть доступной для всех экземпляров, лучше сделать отношение необязательным.

Заключение

Отношения представляют собой ассоциации или связи между сущностями. Они являются ядром реляционной модели данных. Визуально на ER диаграмме отношение представляется линией, соединяющей две сущности. Глагольная конструкция, описывающая взаимосвязь, позволяет вам "прочесть" отношение в форме *сущность - глагольная конструкция - сущность* . Отношения выявляются в ходе рабочих сессий, в процессе нормализации и путем сравнения с корпоративной моделью данных (если она доступна).

Отношения определяются в терминах степени, направленности, типа, количества элементов и обязательности. Степень определяет число сущностей, участвующих в отношении. По степени можно выделить унарные, бинарные и n-арные отношения. Унарным является отношение между экземплярами одной и той же сущности. Бинарные отношения связывают две сущности. N-арные отношения - это отношения между n сущностями, где **n** является числом больше двух. Наиболее часто используются бинарные отношения. Направление отношения определяется от исходной, или родительской, сущности, к завершающей, или подчиненной, сущности.

Тип отношения определяет поведение мигрирующих вторичных ключей. В ERwin тип отношения может быть одним из следующих:

- Идентифицирующее отношение
- Неидентифицирующее отношение
- Типизирующее отношение
- Отношение монгие-ко-многим
- Рекурсивное отношение

Идентифицирующим является отношение, в котором первичный ключ родительской сущности требуется для идентификации экземпляра подчиненной сущности. В неидентифицирующем отношении первичный ключ родительской сущности становится одним или несколькими неключевыми атрибутами подчиненной сущности. Типизирующее отношение, в котором родительская сущность может иметь одну или несколько подчиненных сущностей, может быть исключающим или включающим. Исключающее типизирующее отношение указывает, что экземпляр родительской сущности может иметь соответствующие экземпляры только в одной из подчиненных сущностей. Включающее типизирующее отношение указывает, что экземпляр родительской сущности может иметь соответствующие экземпляры в нескольких подчиненных сущностях.

Отношения **многие-ко-многим** - это такие отношения, в которых экземпляры сущности связаны со многими экземплярами другой сущности, и наоборот. Это логические отношения, которые не могут быть представлены в физической модели и должны быть преобразованы за счет введения ассоциативной сущности между двумя сущностями. Ассоциативная сущность содержит первичные ключи каждой из исходных сущностей, что позволяет экземплярам ассоциативной сущности быть уникальными. Рекурсивными являются отношения, в которых экземпляр сущности связан с другим экземпляром той же самой сущности.

Количество элементов задает максимальное число экземпляров сущности, участвующих в отношении. Необязательность задает минимальное число. Количество элементов выражается терминами *один* или *много*. Необязательность выражается терминами ноль или один.

Понятия сущности, атрибута и отношений между ними исключительно важны для

эффективного использования ERwin. В следующей статье "С чего начать работу с ERwin" будет описана инсталляция ERwin, настройка среды моделирования, а также будет представлены некоторые расширенные режимы отображения в ERwin.

Проектирование баз данных с ERwin
Основные компоненты диаграммы ERwin – сущности, атрибуты, связи.
Часть 2. Понятие атрибута

Зайцев С.Л., к.ф.-м.н.

В статье "Базовые концепции моделирования данных" были введены основные понятия, связанные с моделированием данных. В статье Основные компоненты диаграммы ERwin - сущности, атрибуты, связи. Часть 1. Понятие сущности были даны первоначальные сведения о сущностях и ключах сущностей. В данной статье рассматриваются атрибуты и более детально описываются нормализация и ключи.

В этой статье вы узнаете как:

- Выявлять атрибуты
- Осуществлять нормализацию в процессе анализа атрибутов
- Именовать и описывать атрибуты, а также узнаете о соглашениях, об именовании атрибутов
- Определять типы и характеристики атрибутов, такие как область определения и логические типы данных, и проверять ключи с точки зрения атрибутов
- Избежать распространенных ошибок при работе с атрибутами

На ER-диаграммах сущности и отношения служат для группировки и объединения атрибутов. Именно атрибуты составляют суть модели. Так что давайте приступим к изучению атрибутов - фактов, составляющих информацию логической модели.

Что такое атрибут?

Атрибут является логическим представлением фактов, данные о которых корпорация заинтересована хранить. Вспомните, что в ERwin сущности служат для визуального представления логической группировки атрибутов. С другой стороны, атрибуты представляют факты, накапливаемые о сущностях в логической модели. Атрибуты представляют собой факты, которые служат для идентификации, характеристики отнесения к категории, числового представления или другого вида описания состояния экземпляра сущности.

Атрибут должен представлять единственную концепцию. Атрибуты формируют логические группы, описывающие каждый экземпляр сущности. Конкретным экземпляром атрибута является значение. Например, атрибут с названием Имя определяет область определения для фактов о сущности с названием ПЕРСОНА. Габриэль, Р.Дж., Уилл и Ванесса - примеры конкретных значений Имени для конкретных экземпляров ПЕРСОНЫ. Конкретные значения для каждого из атрибутов сущности представляют единственный экземпляр.

Корректная модель атрибута обладает следующими признаками:

- Значение атрибута представляет интерес для корпорации.
- В логической модели присутствует единственный экземпляр атрибута.
- Атрибут имеет логический тип данных и область определения.
- Значение атрибута определяется как требуемое или необязательное.
- Атрибут имеет имя и описание.
- Для каждого экземпляра сущности может использоваться только одно значение.

> ПРИМЕЧАНИЕ
> На Рисунке 3.1 представлен пример не очень хорошей модели; это прямолинейное отражение требований к информации. В следующих разделах будет предпринята попытка улучшить эту модель для демонстрации процесса размещения атрибутов в соответствующих сущностях.

Корпорация Торговли мороженым Бетти Уилсон хочет заказывать больше наиболее популярных вкусовых добавок и меньше - наименее популярных. Корпорация Бетти делает специальные предложения по продаже мороженого, и заинтересована знать, мороженое с каким вкусом покупатели выбирают для банановой десерта и сливочной помадки во время специальных предложений. Для соответствия бизнес-требованиям необходимо собирать данные о вкусовых добавках к мороженому для банановой десерта и сливочной помадки и дату.

На рисунке 3.1 две сущности БАНАНОВЫЙ ДЕСЕРТ и СЛИВОЧНАЯ ПОМАДКА. Каждая сущность содержит атрибуты, представляющие компоненты каждого из блюд. Обратите внимание, что для сущности БАНАНОВЫЙ ДЕСЕРТ можно выбрать три вкусовых добавки, три верхушки: банан, взбитые сливки и вишни. Для экземпляра СЛИВОЧНОЙ ПОМАДКИ можно выбрать две вкусовых добавки и банан, взбитые сливки и вишни.

Рис. 3.1. Сущности и атрибуты, представляющие (не очень удачно) две основных концепции: СЛИВОЧНАЯ ПОМАДКА и БАНАНОВЫЙ ДЕСЕРТ

Выявление атрибутов

С чего начинать процесс выявления атрибутов? Большинство атрибутов выявляются в ходе рабочих сессий и интервью во время определения сущностей. Анализ требований к информации, полученных от экспертов в предметной области и конечных пользователей - наилучший источник информации для идентификации атрибутов.

Корпоративная модель тоже является отличной основой для выделения атрибутов. Сравните сущности и атрибуты корпоративной модели с сущностями и атрибутами новой логической модели. В корпоративной модели присутствуют атрибуты, которые были ранее определены

для каждой из сущностей, в особенности для стержневых сущностей. Если атрибут не присутствует в корпоративной модели, дополнительный анализ позволит определить, нужно ли его добавить или он принадлежит другой сущности.

Упорядочивание атрибутов в соответствии с требованиями к информации.

Атрибуты логической модели должны строго соответствовать требованиям к информации. Каждый из присутствующих в модели атрибутов должен служить удовлетворению одного или нескольких требований к информации. Модель должна содержать только те атрибуты, которые необходимы для представления фактов, интересующих корпорацию в рамках рассматриваемой предметной области.

> СОВЕТ
> Атрибуты, выходящие за рамки предметной области и не связанные напрямую с одним или более требованиями к информации, должны быть устранены. Атрибуты, не имеющие большого значения для корпорации, тоже должны устраняться в целях сокращения затрат на программирование и сопровождение.

Каждый факт, интересный с точки зрения корпорации, должен быть точно и полно представлен в логической модели. Требования к информации служат мерой необходимости выделения атрибута. Представляется полезным документирование взаимосвязей между атрибутами и требованиями к информации.

Анализ атрибутов

Вам следует проанализировать каждый из атрибутов для определения его взаимосвязей со всеми остальными атрибутами модели. Корректно выполненный анализ гарантирует, что каждый из атрибутов присутствует в модели в единственном экземпляре и размещен в сущности в соответствии с третьей нормальной формой.

Особенно важно проанализировать каждый первичный ключ и каждую часть составного первичного ключа для проверки того, что их значения существуют для каждого экземпляра сущности. Вы должны также убедиться, что первичный ключ идентифицирует один и только один экземпляр сущности.

С помощью анализа также можно установить, заинтересована ли корпорация накапливать и сопровождать какую-либо информацию собственно об атрибуте. Если атрибут так важен, что требуются дополнительные атрибуты для хранения данных о нем, то следует задуматься о возможности создания новой сущности.

Вы должны проанализировать каждый из атрибутов логической модели, чтобы убедиться, что каждый из атрибутов присутствует в модели в единственном экземпляре и только одно значение атрибута существует для каждого экземпляра сущности. Вы должны поместить атрибут в соответствующей сущности, используя правила нормализации, и определить его характеристики.

Остаться должен только один

Атрибут должен присутствовать в логической модели в единственном экземпляре. "Один факт в одном месте" (Дейт, 1986). Для гарантии того, что каждый факт представлен единственным атрибутом, проверьте атрибуты со сходными именами или описаниями. Кроме

того, вы должны определить, являются ли атрибуты реальными экземплярами или конкретными значениями, которые ошибочно представлены в модели разными атрибутами.

Атрибуты со сходными именами и описаниями могут в действительности представлять одну и ту же концепцию и должны быть представлены одним атрибутом. В естественном языке одно и то же слово может представлять несколько концепций. Но что еще хуже, в английском языке для представления одной и той же концепции может существовать несколько разных слов.

Атрибуты, имеющие в составе своего имени слова "индикатор" или "флаг", скорее всего, представляют конкретное значение из области определения атрибута. Конкретное значение является экземпляром атрибута. Использование в модели экземпляров атрибутов - распространенная ошибка. Например, "Индикатор черных волос" имеет значение "да" если присутствуют черные волосы, и значение "нет" если черные волосы отсутствуют. Более предпочтительным будет использование в модели атрибута "Цвет волос", который может иметь конкретное значение "Черный".

Атрибут должен представлять только одну концепцию бизнеса. Он не должен иметь несколько значений для одного экземпляра сущности. На Рисунке 3.1 показаны две сущности, БАНАНОВЫЙ ДЕСЕРТ и СЛИВОЧНАЯ ПОМАДКА. Обе сущности содержат многозначный атрибут с именем "Дата начала или окончания специального предложения". Имя атрибута показывает, что его значение может представлять дату начала специального предложения или дату окончания специального предложения, и у нас нет возможности их различить! Этот атрибут должен быть разделен на два, каждый из которых будет представлять единственный факт.

> ПРИМЕЧАНИЕ
> Хотя разбиение одного атрибута на два для различения фактов позволяет разрешить проблему с многозначностью атрибута, остается другая проблема: значения атрибутов Дата начала специального предложения и Дата окончания специального предложения не зависят от идентификаторов сущностей БАНАНОВЫЙ ДЕСЕРТ и СЛИВОЧНАЯ ПОМАДКА. Эта проблема связана с нормализацией и будет рассмотрена в следующем разделе.

Если мы разрешим атрибуту иметь несколько значений, это может привести к появлению тесно связанных "скрытых" атрибутов. Предыдущий пример достаточно очевиден. Не все многозначные атрибуты могут быть так легко преобразованы. Для вас может оказаться неожиданностью, что в атрибуте, содержащем фрагмент текста, такой как комментарий или примечание, среди текста спрятано множество важных значений атрибута.

Нормализация: помещение атрибута в соответствующую сущность

Атрибуты определяют количество сущностей, которые будут присутствовать в логической модели, приведенной к третьей нормальной форме. Процесс нормализации заключается в анализе зависимости атрибутов друг от друга и зависимости атрибутов от первичного ключа.

Корректно проведенная нормализация гарантирует, что модель будет масштабируемой и расширяемой за счет помещения атрибутов в соответствующие сущности.

Приведение логической модели к третьей нормальной форме часто приводит к появлению новых сущностей.

ПРЕДОСТЕРЕЖЕНИЕ
Осторожно добавляйте новые атрибуты к сущностям нормализованной модели. Новый атрибут должен зависеть от значения ключа, полного ключа, и ни от чего кроме ключа. Рассмотрим случай существования составного первичного ключа: добавление нового атрибута, значение которого зависит от значения части ключа, нарушает требования второй нормальной формы.

Другими преимуществами нормализации являются:

- Устранение или минимизация избыточности. Избыточные данные могут присутствовать в атрибутах, представляющих одну и ту же концепцию, но по-разному поименованных, или в повторяющихся группах. Однократное представление каждого факта в одном месте минимизирует избыточность.
- Устранение или минимизация аномалий при вставке, удалении или обновлении. Ненормализованные структуры данных позволяют одному и тому же факту присутствовать в нескольких местах при неполной или частичной зависимости от первичного ключа. Аномалии при вставке, удалении или обновлении заключаются в противоречивости данных, которая при этих условиях может привести к неожиданностям или неточностям при доступе к данным.
- Устранение или минимизация использования пустых значений для атрибутов. Повторяющиеся группы атрибутов часто содержат пустые значения для многих экземпляров, так как они представляют факт, для которого одни сущности могут иметь несколько значений, а другие - нет. Наличие сущностей, содержащих экземпляры с пустыми значениями, приводит к слабо заполненным (разреженным) структурам данных.

Когда модель приведена к третьей нормальной форме, каждый атрибут принадлежит соответствующей сущности. При приведении модели к третьей нормальной форме часто обнаруживаются новые атрибуты и сущности.

Функциональная зависимость

Функциональная зависимость служит для описания взаимосвязей между атрибутами в модели. Каждый атрибут сущности должен функционально зависеть от первичного ключа сущности (и не зависеть функционально от любого другого атрибута модели). Если это не так, атрибут должен быть перемещен в новую сущность, где это положение будет соблюдаться.

ПРИМЕЧАНИЕ
В заданном отношении R атрибут Y функционально зависит от атрибута X. В символьном виде R.X -> R.Y (читается как "R.X функционально определяет R.Y") - в том и только в том случае, если каждое значение X в R ассоциируется строго с одним значением Y в R (в каждый конкретный момент времени). Атрибуты X и Y могут быть составными (Дейт, 1986).

Для определения функциональной зависимости между атрибутами сначала сгруппируйте их в наборы, объединенные общей темой. Тщательно проанализируйте темы с точки зрения их сходства. Проверьте атрибуты в темах, для определения наличия функциональной зависимости атрибутов в рамках темы. Если атрибут, или группа атрибутов, не зависят от первичного ключа сущности, они должны быть перемещены в другую сущность.

Атрибуты, принадлежащие к одной теме, могут оказаться избыточными. Избыточные атрибуты могут быть сгруппированы в единую сущность или могут использовать общую абстракцию более высокого уровня в качестве характеристических сущностей родительской сущности. На рисунке 3.1 присутствует, по меньшей мере, две общих темы: Вкусовая добавка к мороженому и Верхушка. Эти атрибуты являются хорошими кандидатами на перенос в другие сущности. Рассмотрим их в аспекте функциональной зависимости. Значение атрибута Вкусовая добавка к мороженому не зависит от значения первичного ключа - Ингредиенты бананового десерта. То же самое касается и ключа Сливочная помадка.

Рисунок 3.2 иллюстрирует решение, в котором Вкусовая добавка к мороженому и Верхушка выделены в сущности, где их значения зависят от первичного ключа. Это решение устраняет некоторые очевидные проблемы, связанные с избыточностью.

Первая нормальная форма

Приведение к первой нормальной форме означает перемещение всех повторяющихся атрибутов в другую сущность. Повторяющиеся атрибуты достаточно легко обнаружить, так как часто они просто пронумерованы как **Верхушка 1** и **Верхушка 2** или **Вкус 1** и **Вкус 2.**

Создайте зависимую сущность, которая будет содержать набор атрибутов для представления повторяющихся атрибутов. Первичный ключ зависимой сущности будет составным первичным ключом, в который войдет первичный ключ родительской сущности и, по меньшей мере, один дополнительный атрибут для гарантии уникальности.

На рисунке 3.2 перенесены повторяющиеся группы **Вкусовая добавка к мороженому** и **Верхушка** в зависимые сущности. Обратите внимание на создание сущности ВКУС.

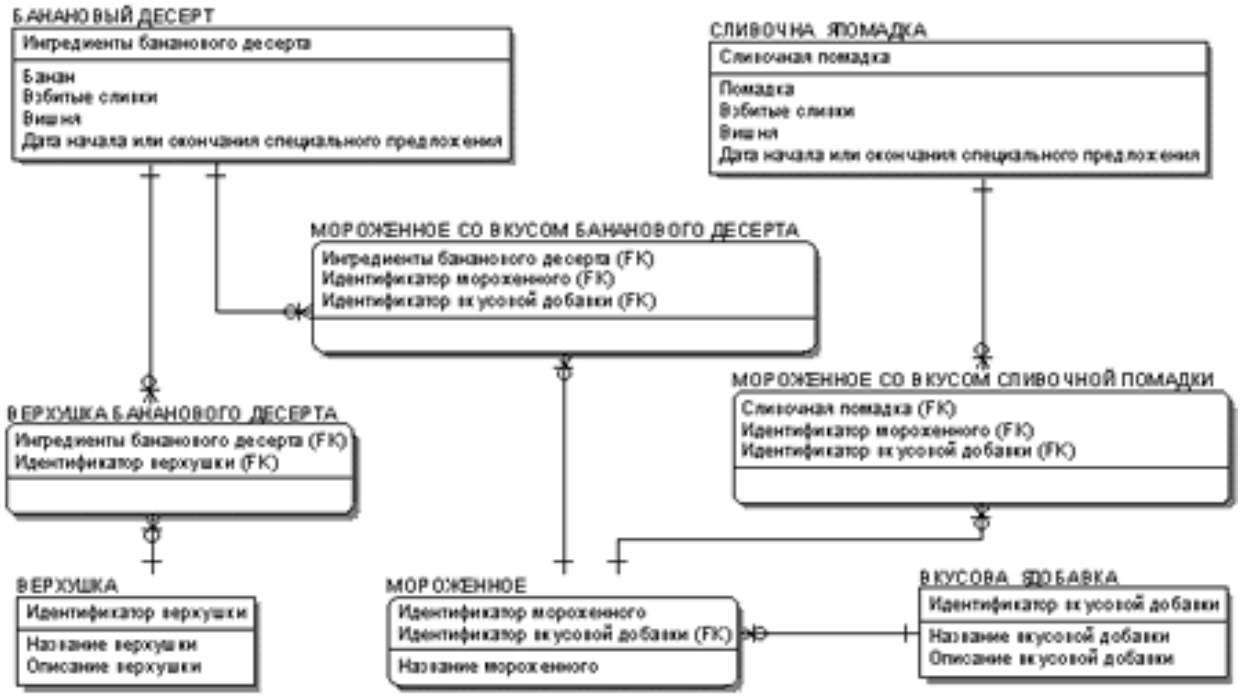


Рис. 3.2. Устранение избыточных атрибутов

Вторая нормальная форма

Приведение ко второй нормальной форме означает удаление избыточных атрибутов. Избыточными атрибутами могут быть:

- Разные атрибуты представляющие одну и ту же концепцию
- Атрибуты различных сущностей, относящиеся к одной и той же теме
- Атрибуты, которые имеют значения не для каждого из экземпляров сущности

> **СОВЕТ**
> *Тщательно проанализируйте сущности со сходными атрибутами. Эти сущности могут оказаться связанными или даже представлять одну и ту же концепцию. Если это так, их нужно объединить.*

Атрибуты, представляющие одну и ту же концепцию, должны быть преобразованы к единому атрибуту. Избыточные атрибуты могут иметь значения не для каждого из экземпляров сущности и, таким образом, их существование не будет зависеть от значения первичного ключа. Переместите эти атрибуты в сущность, где они будут иметь значения для каждого из экземпляров.

Создайте сущность с атрибутами для представления избыточных атрибутов. Новая сущность обладает первичным ключом, идентифицирующим единственный экземпляр. Этот первичный ключ станет внешним ключом в исходной сущности. Внешние ключи будут обсуждены позднее.

Рисунок 3.2 демонстрирует решение для некоторых избыточных атрибутов сущностей БАНАНОВЫЙ ДЕСЕРТ и СЛИВОЧНАЯ ПОМАДКА. Рассмотрим избыточность с точки зрения двух стержневых сущностей. В обеих сущностях присутствуют общие темы: вкусовая добавка к мороженому и верхушка. Это признак того, что стержневые сущности могут быть объединены на более высоком уровне абстракции.

Рисунок 3.3 демонстрирует создание супертипа с именем СМЕСЬ, для которого БАНАНОВЫЙ ДЕСЕРТ и СЛИВОЧНАЯ ПОМАДКА являются его реализациями. Я добавил классификационный атрибут "Тип смеси" в родительскую сущность СМЕСЬ для идентификации является ли СМЕСЬ экземпляром сущности БАНАНОВЫЙ ДЕСЕРТ или СЛИВОЧНАЯ ПОМАДКА. Экземпляр сущности СМЕСЬ может быть экземпляром сущности БАНАНОВЫЙ ДЕСЕРТ или СЛИВОЧНАЯ ПОМАДКА, но не их обеих одновременно.

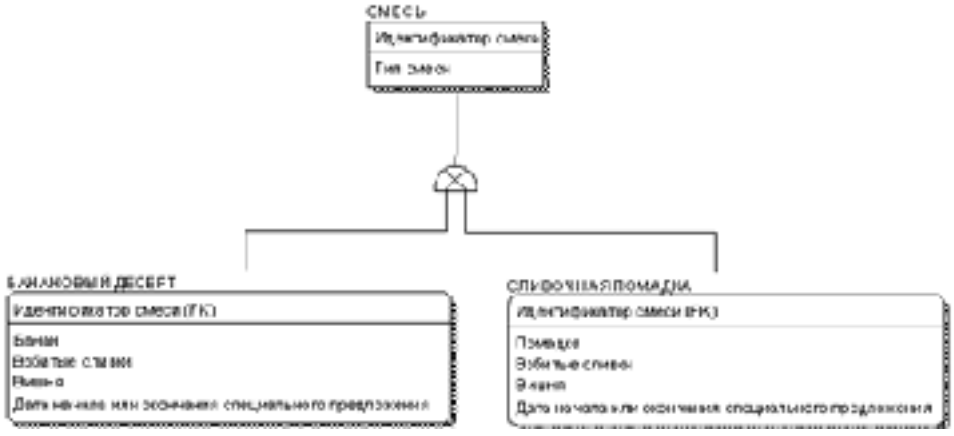


Рис. 3.3. Избыточность стержневых сущностей устранена за счет перемещения общих атрибутов в более общую сущность СМЕСЬ. Обратите внимание, что первичный ключ "Идентификатор смеси" помещен и в сущности БАНАНОВЫЙ ДЕСЕРТ и СЛИВОЧНАЯ

ПОМАДКА.

Третья нормальная форма

Приведение к третьей нормальной форме означает устранение любых атрибутов, которые зависят от значений других атрибутов кроме первичного ключа. Иногда это называют *транзитивной зависимостью* .

Создайте новую сущность и переместите в нее атрибуты, не зависящие от первичного ключа в исходной сущности. Определите первичный ключ для новой сущности так, чтобы он гарантировал уникальность.

На Рисунке 3.3 атрибуты **Взбитые сливки** и **Вишня** не зависят от первичных ключей сущностей БАНАНОВЫЙ ДЕСЕРТ и СЛИВОЧНАЯ ПОМАДКА. Фактически вы должны решить, не являются ли атрибуты **Взбитые сливки** и **Вишня** экземплярами сущности ВЕРХУШКА.

> **ПРИМЕЧАНИЕ**
> *Сущность **БАНАНОВЫЙ ДЕСЕРТ** содержит атрибут **Взбитые сливки** и сущность **СЛИВОЧНАЯ ПОМАДКА** тоже содержит атрибут **Взбитые сливки** . Сравнение описаний этих атрибутов показывает, что они описывают одну и ту же концепцию. Взбитые сливки были выбраны как логическое имя для представления общей концепции и перемещены в более общую сущность **СМЕСЬ.***

На Рисунке 3.4 обратите внимание на дополнительный атрибут Дата смеси, который обеспечивает информацию о том, когда был создан экземпляр сущности СМЕСЬ. Я удалил атрибуты Дата начала и Дата окончания из сущностей БАНАНОВЫЙ ДЕСЕРТ и СЛИВОЧНАЯ ПОМАДКА. Новая сущность СПЕЦИАЛЬНОЕ ПРЕДЛОЖЕНИЕ теперь содержит эти две даты и атрибут Вкусовая добавка к мороженому для указания того, на какой из видов мороженого распространяется предложение.

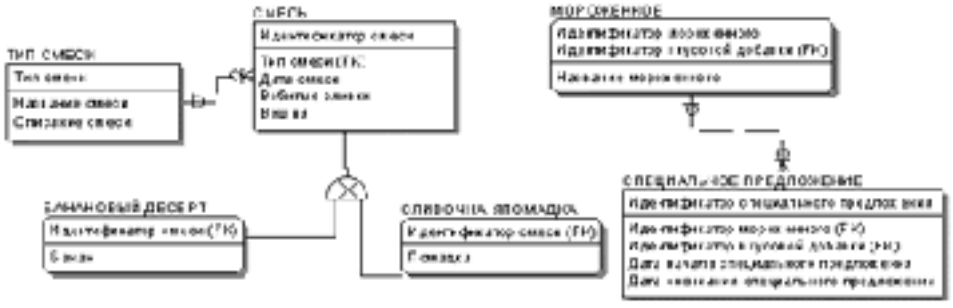


Рис. 3.4. Каждый атрибут зависит от первичного ключа, полного первичного ключа и ни от чего кроме ключа.

Определение характеристик атрибута

Атрибуты делятся на две группы. Атрибут либо является ключом, либо нет. Рисунок 3.5 показывает ключевые атрибуты для логической модели сущности СМЕСЬ. Заметьте, что, в сущности, атрибуты первичного ключа располагаются над линией внутри сущности, а остальные атрибуты - под линией.

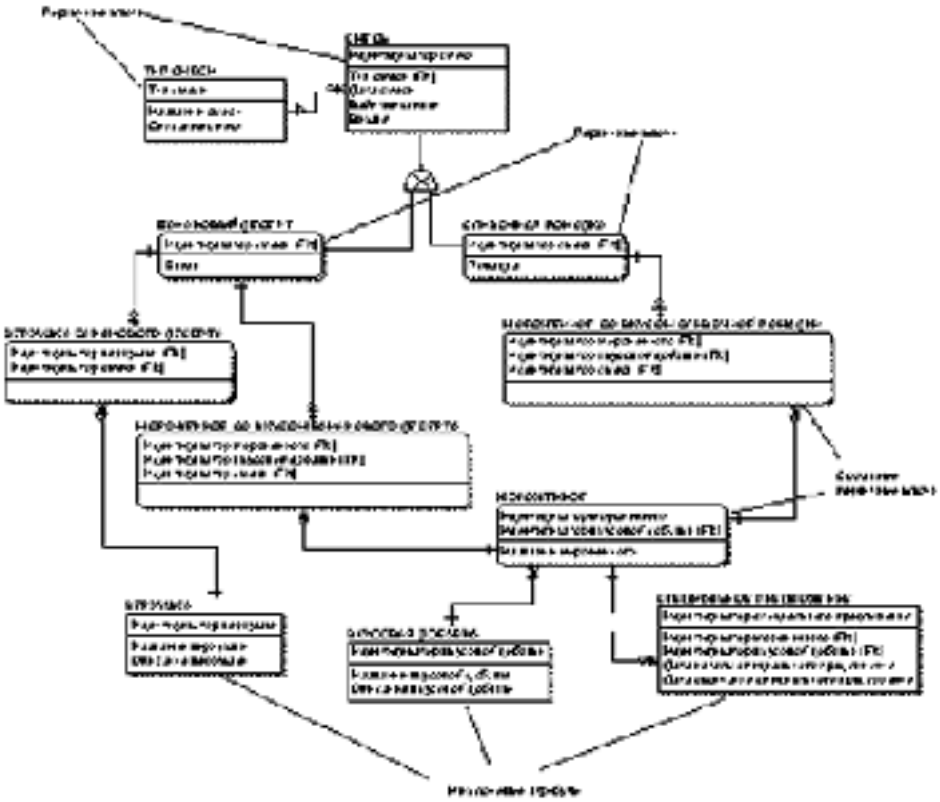


Рис. 3.5. Все атрибуты, не являющиеся частью первичного ключа, располагаются в сущности ниже разделителя. Это могутбыть кандидаты в ключи, внешние и альтернативные к лючи и простые атрибуты.

Проектирование баз данных с ERwin

Основные компоненты диаграммы ERwin - сущности, атрибуты, связи

Часть 2. Понятие атрибута (продолжение)

Зайцев С.Л., к.ф.-м.н.

Ключевые атрибуты

Ключевыми являются атрибуты, значения которых определяют значения других атрибутов. Значения ключевых атрибутов не зависят от значений никаких других атрибутов. Ключ может состоять из единственного атрибута или быть составлен из нескольких атрибутов. Эти атрибуты могут быть первичными ключами, составными первичными ключами, кандидатами в ключи, внешними ключами или альтернативными ключами.

Атрибуты первичного ключа

Является ли первичный ключ единственным атрибутом или группой, его значения определяют значения всех других атрибутов.

Хороший первичный ключ будет обладать следующими признаками:

- Значение гарантирует уникальность для каждого из экземпляров

- Значение не имеет скрытого смысла
- Область определения значений будет оставаться постоянной с течением времени
- Значения существуют для каждого из экземпляров сущности

Искусственные первичные ключи

Искусственные первичные ключи - это атрибуты, созданные с единственной целью идентификации конкретных экземпляров сущности. В некоторых случаях не существует атрибута или группы, однозначно идентифицирующих экземпляр сущности. В других случаях, составной первичный ключ слишком громоздок, и его трудно сопровождать. Искусственный первичный ключ иногда называют псевдоключом или ключом, *сгенерированным системой* . Еще он известен под названием *искусственный уникальный идентификатор* , которое указывает на его назначение.

Искусственный первичный ключ часто формируется простой последовательной нумерацией каждого из экземпляров сущности. Дополнительным преимуществом таких искусственных ключей является то, что не нужно заботиться о смысле связанных с ними экземпляров сущности, кроме гарантии уникальности. Фактически, искусственные первичные ключи, созданные таким способом, гарантированно обладают особенностями хороших первичных ключей.

Обратите внимание, что большинство первичных ключей на Рисунке 3.5 являются искусственными. По большей части, первичный ключ является просто уникальным номером для каждого из экземпляров.

Кандидаты в ключи

Кандидатом в ключи является атрибут или группа атрибутов, идентифицирующих конкретный экземпляр сущности. Кандидат в ключи представляет механизм определения потенциальных первичных ключей для идентификации конкретных экземпляров сущности.

Кандидат в ключи, который не выбран в качестве первичного ключа, еще называют альтернативным ключом. *Альтернативный ключ* - это атрибут или группа атрибутов, которые могут быть использованы при индексировании.

Внешние ключи

Внешним ключом является атрибут или группа атрибутов, составляющих первичный ключ другой сущности. Внешний ключ может быть, а может и не быть, ключевым атрибутом в связанной сущности. Обратите внимание на термин *связанная сущность* . Внешние ключи представляют связи между сущностями, которые более детально будут обсуждаться в следующей статье.

Миграция атрибутов первичного ключа

Внешние ключевые атрибуты являются атрибутами первичного ключа другой сущности, которые мигрировали в данную сущность через связь. Внешние ключи могут быть как идентифицирующими, так и неидентифицирующими. *Идентифицирующие внешние ключи* становятся частью первичного ключа в той сущности, в которую они мигрировали.

Неидентифицирующие внешние ключи становятся не ключевыми атрибутами.

Неключевые атрибуты

Неключевыми являются атрибуты, значения которых зависят от значений первичного ключа или составного первичного ключа. Эти не ключевые атрибуты должны зависеть от значения ключа, полного ключа, и ни от чего кроме ключа.

В своей книге **Strategic Systems Development** К. Финклештейн определяет несколько типов неключевых атрибутов:

- Селективные атрибуты - атрибуты, используемые для идентификации единственного экземпляра сущности, когда ключ не уникален. Также называются *вторичными ключами*.
- Групповой атрибут - атрибут, объединяющий группу более детальных атрибутов.
- Атрибуты повторяющейся группы - атрибуты, представляющие несколько включений одного атрибута в рамках сущности.
- Производные атрибуты - атрибуты, чьи значения определяются из значений других атрибутов.
- Атрибуты основных данных - атрибуты, которые не являются селективными, групповыми, атрибутами повторяющейся группы или производными.

Селективные, групповые и атрибуты повторяющейся группы не должны присутствовать в логической модели, приведенной к третьей нормальной форме. Селективные атрибуты должны стать частью первичного ключа, если они нужны для идентификации единственного экземпляра сущности. Групповые атрибуты являются многозначными. По моему мнению, групповые атрибуты лучше всего представлять в модели кодовыми или классификационными сущностями. Как отмечалось выше, повторяющиеся группы должны быть вынесены в подчиненные сущности.

В третьей нормальной форме не ключевые атрибуты должны быть простыми (основными) или производными атрибутами.

Простые атрибуты

Простые атрибуты - это атрибуты, которые в результате декомпозиции были доведены до наивысшей степени детализации и, таким образом, их значения полностью зависят от первичного ключа и определены для каждого из экземпляров сущности. Они не являются критерием выбора и не могут служить для группировки сущностей. Они представляют простые атомарные факты, в которых заинтересована корпорация.

Производные атрибуты

Производными атрибутами являются атрибуты, значения которых выведены или вычислены на основе значений одного или более других атрибутов. Вопрос допустимости присутствия производных атрибутов в логической модели активно обсуждается. Некоторые эксперты считают, что поскольку значения производных атрибутов зависят от значений исходных атрибутов, производные атрибуты не должны быть представлены в логической модели.

Логическая модель предназначена для полного и точного представления требований к

информации. Вы можете принять решение создать производные атрибуты на уровне физической модели в соответствие с требованиями к использованию.

Хотя понятны аргументы в пользу исключения производных атрибутов, тем не менее, логическая модель - наилучшее место для введения имен и описаний всех атрибутов. Поэтому предпочтительнее включать производные атрибуты в логическую модель. Однако, разработчикам моделей стоит отказаться от использования производных атрибутов в качестве первичных ключей или части составных первичных ключей. Также не забудьте включить правило вывода в описание производного атрибута.

Нахождение области определения атрибута

Область определения атрибута задает список разрешенных значений, которые атрибут может принимать в конкретном экземпляре сущности. Область определения включает, по меньшей мере, область определения универсального типа данных и может включать область определения, заданную пользователем. В завершенной логической модели вы должны найти область определения для каждого из атрибутов.

Можно сказать, что область определения атрибута должна содержать, как минимум, два значения. Атрибут, для которого всегда разрешено только одно значение, вероятно, некорректно отображен в модели. На Рисунке 3.5 присутствует два таких атрибута - Банан и Помадка.

В сущности БАНАНОВЫЙ ДЕСЕРТ присутствует атрибут Банан. Бизнес-правило утверждает, что каждый экземпляр сущности БАНАНОВЫЙ ДЕСЕРТ содержит банан. Поэтому у атрибута Банан может быть только одно значение и, вероятно, этот атрибут не является необходимым. Вместо этого описание сущности БАНАНОВЫЙ ДЕСЕРТ должно быть указание на то, что банан включается в каждый ее экземпляр. То же самое касается атрибута Помадка в сущности СЛИВОЧНАЯ ПОМАДКА.

В Таблице 3.1 приведены области определения логических типов данных сущности СПЕЦИАЛЬНОЕ ПРЕДЛОЖЕНИЕ логической модели СМЕСЬ.

ТАБЛИЦА 3.1. Примеры логических типов данных.

Имя атрибута	*Логический тип данных*
Идентификатор специального предложения	Number (число)
Идентификатор мороженого	Number (число)
Идентификатор вкуса	Number (число)
Начало специального предложения	Date (дата)
Конец специального предложения	Date (дата)

Области определения простых и расширенных типов данных

Область определения типа данных определяет способ представления значений атрибута. В завершенной логической модели область определения типа данных требуется для каждого атрибута. В следующем списке приведено несколько примеров логических типов данных ERwin:

- Datetime - дата/время

- Number - число
- String - строка

Многие из новых платформ баз данных поддерживают более развитые типы данных. Однако важно помнить, что эти сложные типы данных, за некоторым исключением, зависят от платформы. В любом случае, если пользователю нужен атрибут, он должен быть включен в модель, вне зависимости от его типа данных. Некоторые широко используемые расширенные типы данных приведены ниже:

- Image - изображение
- Sound - звук
- Video - видео

Области определения, вводимые пользователем

Области определения, вводимые пользователем - специализированные области определения, которые уточняют набор значений, допустимых для атрибута. Эти области определения часто специфичны для организации и должны определяться и использоваться единообразно в пределах корпорации. Например, атрибут с областью определения типа данных Number может, кроме того, иметь введенную пользователем область определения, которая ограничивает возможные значения пределами от 1 до 100. Принцип целостности дает корпорации возможность внести изменения в одной сущности расширить область определения для каждого из атрибутов, который ее использует.

Определение необязательности атрибута

Значение атрибута может быть обязательным или нет. Если значение требуется, или обязательно, значение должно присутствовать в момент создания экземпляра. Если значение необязательно, вы можете создавать экземпляры без него.

В книге **Information Engineering: Strategic Systems Development** К. Финклештейн определяет свойство обязательности атрибута через серию "правил редактирования":

- Добавляется сразу, изменить позже - нельзя.
- Добавляется сразу, изменяется позже.
- Добавляется позже, изменяется позже.
- Добавляется позже, изменить потом - нельзя.

Внимательно следите за необязательными атрибутами. Если атрибут или набор атрибутов имеет значение только для конкретных экземпляров сущности, рассмотрите возможность его переноса в сущность, где значение будет существовать для каждого из экземпляров.

В Таблице 3.2 указано свойство обязательности для атрибутов сущности СПЕЦИАЛЬНОЕ ПРЕДЛОЖЕНИЕ. Обратите внимание на тот факт, что при создании экземпляра сущности СПЕЦИАЛЬНОЕ ПРЕДЛОЖЕНИЕ значения требуются для всех атрибутов кроме атрибута Дата окончания специального предложения.

Таблица 3.2. Примеры обязательности атрибутов

Имя атрибута	*Обязательность*
Идентификатор специального предложения	Требуется
Идентификатор мороженого	Требуется
Идентификатор вкуса	Требуется
Начало специального предложения	Требуется
Конец специального предложения	Необязательно

В таблице 3.2 представлено бизнес-правило, которое говорит, что для экземпляра сущности СПЕЦИАЛЬНОЕ ПРЕДЛОЖЕНИЕ требуется следующая информация:

- Идентификатор сущности (Идентификатор специального предложения)
- Идентификатор вкусовой добавки специального предложения (Идентификатор мороженого и Идентификатор вкуса)
- Дата начала специального предложения (Начало специального предложения)

Дата окончания для каждого экземпляра сущности СПЕЦИАЛЬНОЕ ПРЕДЛОЖЕНИЕ необязательна. Бизнес-правило утверждает, что СПЕЦИАЛЬНОЕ ПРЕДЛОЖЕНИЕ должно иметь начало, но не обязательно должно иметь конец.

Атрибуты, значения которых обязательны, не могут иметь пустых значений. Некоторые эксперты считают, что значение должно быть обязательным для каждого экземпляра сущности. Естественно, в предположении, что значение каждого атрибута экземпляра сущности найдено или известно, до того, как экземпляр создается.

Атрибуты, чьи значения необязательны, могут иметь пустые значения. Некоторые эксперты считают, что атрибут не должен присутствовать в сущности, если его значение недоступно для каждого из ее экземпляров. Одной из причин является сложность интерпретации пустых значений. Означает ли пустое значение, что это значение неизвестно для экземпляра, или оно просто не было получено?

ПРИМЕЧАНИЕ

Среди разработчиков моделей дискуссия о недостатках и достоинствах требуемых и необязательных значений все еще продолжается. Одни разработчики уверены, что атрибут не должен иметь пустых значений, и утверждают, что область определения должна содержать такие значения, как неизвестен и неполучен. Другие считают обязательность значений и использование области определения также требует и использования значений по умолчанию, что приводит к ненадежным сомнительным значениям.

Предпочтительнее использовать пустые значения и переложить ответственность за работу с пустыми значениями на прикладную программу или средство формирования запросов. Это наиболее адекватное и гибкое решение, поскольку оно позволяет интерпретировать пустые значения по-разному для удовлетворения разных требований бизнеса.

Именование атрибутов

Каждый атрибут должен иметь ясное, точное и непротиворечивое имя. Имя атрибута не должно конфликтовать с его описанием. Имя атрибута должно указывать на значения, собираемые для экземпляров атрибута. Имя атрибута должно быть понятным и общепринятым в корпорации.

СОВЕТ

При выборе имени для атрибута, сохраняйте точку зрения корпорации и позаботьтесь о том, чтобы оно отражало смысл атрибута, а не то, как он будет использоваться. Используйте имена, осмысленные для сообщества пользователей и экспертов предметной области.

Вероятно, что у вас в корпорации есть набор соглашений об именовании атрибутов, разработанные в вашей корпорации или при формировании корпоративной модели данных, которыми вы руководствуетесь. Использование соглашений именования атрибутов гарантирует, что имена конструируются единообразно в рамках корпорации, вне зависимости от того, кто конструирует имя.

Соглашения об именовании атрибутов важны, вне зависимости от того, в маленькой или большой организации вы работаете. Однако, в большой организации с несколькими командами разработчиков и большим количеством пользователей, соглашения об именовании существенно помогают при взаимодействии и понимании элементарных данных. В идеале, вы должны разработать и сопровождать соглашения об именовании атрибутов централизованно

и затем документально оформить и опубликовать их для всей корпорации.

Ниже представлены некоторые положения для формирования начального набора соглашений об именовании атрибутов, просто на случай, если в вашей организации пока такой набор не разработан:

- Имя атрибута должно быть достаточно описательным. Подумайте об использовании словосочетаний на основе существительных в форме объект/ модификатор/ класс.
- По возможности имя атрибута должно включать имя сущности. Используйте "Имя для персоны" вместо просто "Имя".
- Имя атрибута должно указывать на значения конкретных экземпляров атрибута. Использование одинаковых имен для атрибутов, содержащих различные данные, или разных имен для атрибутов, содержащих одинаковые данные, будет без необходимости вводить в заблуждение разработчиков и конечных пользователей.
- Имя атрибута должно использовать язык бизнеса вместо языка технических описаний.
- Имя атрибута не должно содержать специальных символов (таких как !, @, #, $, %, л, &, * и тому подобных) или указывать на принадлежность (Имя, принадлежащее персоне).
- Имя атрибута не должно содержать акронимов или аббревиатур, если только они не являются частью принятых соглашений именования.

Разработчикам моделей предпочтительно использовать хорошие соглашения именования, если таковые существуют, или разработать их, если таких соглашений нет.

Имена атрибутов в форме Объект/ Модификатор/ Класс

Форма *объект/модификатор/класс* - широко распространенное в отрасли соглашение об именовании атрибутов. Это соглашение побуждает использовать имена атрибутов, состоящие из трех частей. Часть объект иногда называют субъектом или основным словом. В качестве объекта обычно используется имя сущности.

Модификатор может быть одиночным термом или группой термов. Хотя списка стандартных модификаторов не существует, разработчикам моделей желательно формировать короткие осмысленные модификаторы. Использование модификаторов позволяет вам создавать наглядные осмысленные имена атрибутов. Если имя становится неприемлемо тяжеловесным для пользователей или широкого применения, как требуется в корпорации, вы можете пойти на компромисс, отказавшись от трехсложных имен атрибутов.

Базовой частью имени атрибута является класс, который определяет тип информации, представляемой атрибутом. Некоторые часто используемые классы:

- Идентификатор
- Код
- Дата
- Число
- Величина
- Количество
- Частота

Примеры имен атрибутов

В рамках корпорации всегда лучше использовать единообразные имена атрибутов. В Таблице 3.3 приведены примеры хороших и плохих имен для атрибутов. Обратите внимание, что слова в имени атрибута отделены пробелами, начинаются с заглавных букв и используют строчные символы для остальных.

ТАБЛИЦА 3.3. Имена атрибутов с пояснениями

Хорошее Имя	Неудачное имя	Пояснение
Person First Name (Имя персоны)	Name (Имя)	Name (Имя) - название класса и нуждается в обозначении объекта Person (персона) и в модификаторе First (первое).
Ice Cream Sales Quantity (Объем продаж мороженого)	The Quantity of Sales (Объем продаж)	Quantity (Количество) - название класса и должно быть на последнем месте (в английском варианте имени атрибута). "The" и "of" не привносят дополнительного смысла.
Item Cost Amount (Величина стоимости позиции)	Cost of Item (Стоимость позиции)	"of" не привносит дополнительного смысла. Название класса "Amount" (величина) указывает пользователю, что должно быть в атрибуте.
Product Identifier (Идентификатор продукта)	Product Identifiers (Идентификаторы продуктов)	"Identifiers" (Идентификаторы) - множественное число. Имя атрибута должно быть существительным в единственном числе.
Point of Sale Location Code (Код местоположения точки продаж)	POS Code (Код POS)	"POS" - аббревиатура. Использованное название класса "Code" (код) нуждается в модификаторе.
Person Birth Date (Дата рождения персоны)	Birthday (День рождения)	Birthday (День рождения) не содержит названия класса Date (Дата). Включение модификатора и имени объекта проясняет смысл имени атрибута.

Описание атрибутов

Описание атрибу та должно быть коротким пояснением смысла атрибута, а не того, как он используется. Описание атрибута не должно противоречить его имени и не должно быть простым повторением имени. Используйте название класса и объекта в утверждении для точного описания данных. Если атрибут выводится или рассчитывается, включайте правила вывода или формулы расчета. Следующие правила касаются описания атрибутов:

- Описание атрибута должно быть ясным, полным и однозначным.
- Описание атрибута должно соответствовать его имени.
- Описание атрибута не должно опираться на описание другого атрибута.

- Описание атрибута должно формулироваться на языке бизнеса, а не на языке технических описаний.
- Имя атрибута должно отражать его смысл, а не то, как он используется.
- В описании атрибута должны быть расшифрованы все аббревиатуры и акронимы, использованные в его имени.

Разработчикам моделей рекомендуется давать хорошие описания для каждого из атрибутов. Хорошие описания атрибутов делают легким использование модели для всех. Те, кто использует модель, созданную хорошим разработчиком, испытывают удовольствие от хорошо сформулированных в модели требований к информации. Сравните примеры из таблицы

3.4.**Таблица 3.4.** Имена и описания атрибутов с пояснениями.

Имя атрибута	***Хорошее описание***	***Неудачное описание***	***Пояснение***
Person First Name (Имяперсоны)	Имя персоны, которое позволяет корпорации общаться с персоной, используя дружеские обращения.	Поле с длиной в 40 символов.	Не используется язык бизнеса. Применены технические термины.
Ice Cream Sales Quantity (Объем продаж мороженого)	Количество мороженого конкретного сорта, проданного в рамках конкретного мероприятия по продаже.	Объем продаж.	Не добавляет нового смысла, а просто перефразирует имя атрибута в расплывчатых терминах.
Item Cost Amount (Величина стоимости позиции)	Величина стоимости конкретной позиции в конкретный период времени. Представляет суммарную стоимость продажи и доставки.	Шестизначное десятичное число с двумя знаками после запятой.	Слишком техническое описание. Почти ничего не значит для пользователей элемента данных.
Product Identifier (Идентификатор продукта)	Искусственный уникальный числовой идентификатор для конкретного продукта.	Идентификаторы продуктов.	Простая перефразировка имени атрибута.
Point of Sale Location Code (Код местоположения точки продаж)	Уникальный код, идентифицирующий географическое положение точки продаж.	Код POS.	Использованный акроним может быть непонятен пользователям. Кроме того, в описании опущен важный модификатор.
Person Birth Date (Дата рождения персоны)	Дата рождения персоны.	День рождения персоны.	В описании опущено название класса “дата”.

Распространенные ошибки при работе с атрибутами

Этот раздел, посвященный распространенным ошибкам при моделировании атрибутов, не претендует на полноту. Цель его - указать на наиболее распространенные ошибки, которые встречаются у разработчиков моделей.

Иногда, при моделировании чего-либо определенным способом, разработчик модели делает сознательный выбор, руководствуясь вполне правильными принципами. Очень важно понимать всю причинно-следственную цепочку принимаемых решений и результаты, к которым они могут привести.

Моделирование в терминах значений

Что понимается под моделированием в терминах значений? Во время рабочей сессии пользователи могут сказать вам, что им нужен набор атрибутов, указывающих на возрастные категории экземпляра сущности ПЕРСОНА. В этом сценарии возникают, по меньшей мере, три проблемы:

1. Способ определения возрастных категорий в корпорации со временем может измениться.
2. Возраст конкретной персоны определенно меняется с течением времени.
3. Все атрибуты будут представлять значения атрибута **Возраст персоны**. Естественно, **Возраст персоны** с течением времени будет меняться, так что лучшим решением будет использование в модели простого атрибута **Дата рождения персоны**.

> **СОВЕТ**
>
> Если приобретенные или производные данные, такие как возрастная категория, это единственное, что доступно, тогда они, конечно, должны быть включены в модель. Если доступны и возрастная категория и дата рождения, включите в модель и категорию, и дату рождения, и выводите возрастную категорию из даты рождения для экземпляров, для которых дата рождения известна. Как часто придется делать вывод и будет ли возрастная категория иметь физическое представление, зависит от требований к использованию.

Моделирование многозначных атрибутов

Многозначными являются атрибуты, имеющие несколько значений для концепции. Проверьте описания атрибутов, которые указывают на наличие нескольких значений для одной и той же концепции.

Иногда эксперты из различных предметных областей в корпорации используют имя атрибута, которое *пишется и произносится одинаково*, но *имеет разные значения* для разных экспертов. Один из способов убедиться в том, что *атрибуты с одинаковыми именами описывают одинаковые объекты*, заключается в проверке описаний. Убедитесь, что значения атрибутов описывают единственную концепцию.

Например, вы можете создать искусственные коды путем соединения одного или нескольких кодов для связи прежде не связанных данных. Текстовые фрагменты могут скрывать

множество ценных атрибутов и значений.

Неудачное разрешение многозначных атрибутов может привести к тому, что некоторые важные бизнес-правила останутся необнаруженными и недокументированными.

Моделирование избыточных атрибутов

Избыточными являются атрибуты с разными именами, но содержащие информацию о сходных концепциях. Во всех языках существует много слов, представляющих одни и те же вещи. Один из способов найти избыточные атрибуты - просмотреть сущности с похожими атрибутами. Сравните описания всех атрибутов для проверки того, не содержат ли эти сущности данные о сходных концепциях. Избыточные атрибуты часто являются результатом тенденции к моделированию значений в качестве атрибутов. Например, сущности УПРАВЛЕНЕЦ и ИСПОЛНИТЕЛЬ могут содержать атрибуты Имя менеджера и Имя исполнителя. Так как и УПРАВЛЕНЕЦ, и ИСПОЛНИТЕЛЬ являются ролями, в которых может выступать экземпляр сущности ПЕРСОНА, вы можете переместить туда этот атрибут и назвать его Имя ПЕРСОНЫ.

Использование неудачных имен для атрибутов

Неясные, неоднозначные или неточные имена атрибутов усложняют для новых пользователей и команд разработчиков повторное использование или развитие существующей модели.

Не используйте аббревиатур или акронимов в качестве части имени атрибута. Аббревиатуры и акронимы открыты для неправильной интерпретации и даже могут иметь разное значение в разных предметных областях.

Не используйте имена собственные, указывающие на значение для конкретного экземпляра. Имя атрибута, использующее имя собственное - индикатор серьезных проблем при моделировании, заключающихся не только в неудачном выборе имени. Не включайте месторасположение в качестве части имени атрибута. Если значение существует для одного месторасположения, оно определенно существует и для другого месторасположения. Имя атрибута с указанием расположения является признаком того, что вы моделируете конкретный экземпляр вместо класса.

Использование неудачных описаний для атрибутов

Не используйте описаний атрибутов, заимствованных только из словаря. Описания из словаря не будут включать информацию, значимую для бизнеса, которая делает атрибут важным для корпорации. Не используйте простое перефразирование имени атрибута. Не используйте имени атрибута в его описании.

Неясное, неопределенное описание атрибута, или что еще хуже - его отсутствие, затрудняет повторное использование или развитие существующей модели. Пользователи не смогут проверить, что модель содержит все требования к информации. Это так же повышает вероятность использования в модели вместо атрибутов конкретных значений и многозначных атрибутов.

Концепции, которые кажутся очевидными для всех участников рабочих сессий, могут перестать быть столь очевидными с течением времени, когда перед новой командой

разработчиков будет поставлена задача развить существующую модель.

Заключение

Сущности представляют собой факты, информацию о которых корпорация заинтересована накапливать и сопровождать. Они составляют существо модели и в основном выявляются во время рабочих сессий. Полное и точное отражение атрибутов в модели требует тщательного анализа, гарантирующего, что атрибуты точно соответствуют требованиям к информации. Атрибут должен присутствовать в модели в единственном экземпляре и должен представлять единственную концепцию бизнеса. Для помещения атрибутов в соответствующие сущности должны использоваться правила нормализации.

Атрибуты могут быть ключевыми и неключевыми. Ключ может быть единственным атрибутом или группой атрибутов. Первичные ключи выбираются из кандидатов в ключи, которые уникально идентифицируют экземпляр сущности. Атрибуты первичного ключа мигрируют из исходной сущности, чтобы стать внешними ключами вторичных сущностей. Значения неключевых атрибутов должны функционально зависеть от значения первичного ключа.

Область определения задает набор значений атрибута. Логические области определения могут быть простыми типами данных, такими как числа или строки. Они так же могут быть сложными типами данных, определяемыми пользователем, которые приспособлены для удовлетворения специфических требований корпорации. Новые СУБД поддерживают расширенные типы данных, такие как изображения и звук.

Значения атрибута могут быть требуемыми или необязательными. Если значение требуемое, атрибут не может иметь пустых значений. Атрибут должен иметь имя и описание. При именовании атрибутов рекомендуется использовать стандарт именования в форме объект/ модификатор/ класс. Каждый атрибут должен включать хорошее описание, использующее терминологию бизнеса для определения сущности атрибута, а не того, как он будет использоваться.

Сущности и отношения служат для группировки связанных атрибутов. В последующих статьях будет рассказано о роли отношений в процессе моделирования и о свойствах и типах отношений, там же вы найдете некоторые советы, помогающие избежать распространенных ошибок при моделировании отношений.

Проектирование баз данных с ERwin Основные компоненты диаграммы ERwin - сущности, атрибуты, связи

Часть 1. Понятие сущности

Зайцев С.Л., к.ф.-м.н.

В статье "Базовые концепции моделирования данных", были введены основные понятия, связанные с моделированием данных. Здесь же мы детально опишем сущности и ключи сущностей. Как вы знаете, сущности - это понятия, информацию о которых следует сохранять для возможности дальнейшей обработки. В ERwin сущности являются графическим представлением логической группировки данных. Сущности могут быть вещественными, реальными объектами или неосязаемыми концептуальными абстракциями. Сущности не предназначены для представления единичного объекта. Скорее они представляют классы, включающие атрибуты, содержащие информацию о множестве экземпляров.

Ниже будут рассмотрены следующие вопросы, касающиеся сущностей:

- Диаграммы Сущность/Отношение (Entity Relational)
- Выделение сущностей
- Определение типов сущностей
- Именование и описание сущностей
- Распространенные ошибки при работе с сущностями

Так как в ERwin для моделирования данных используется методология ER (Entity Relational), давайте начнем с краткого введения в концепции ER. Для начала приступим к изучению сущностей - "контейнеров" для хранения информации логической модели.

Введение в реляционную диаграмму сущности

В этой и других публикациях на эту тему для визуального представления сущностей и отношений между ними используются ERD-диаграмма (Entity Relational Diagram - реляционная диаграммя сущности), основанная на нотации, используемой ERwin. Хотя существуют и другие методологии моделирования данных, такие как расширенный реляционный анализ (Extended Relational Analysis - ERA), объектно-ориентированный подход (Object Oriented - OO) и объектно-ролевое моделирование (Object Role Modeling - ORM), фундаментальные концепции ER методологии присутствуют и в них.

Методология ER-моделирования разработана П. Ченом в конце 1970-х годов. Для представления сущностей в методологии ER используются прямоугольники. В исходной ER-нотации Чена отношения содержат атрибуты. Равная возможность использования атрибутов в сущностях и отношениях делает различие между сущностями и отношениями достаточно сложным.

С течением времени ER-подход изменялся и расширялся, но базовые концепции продолжали обеспечивать надежную основу для грамотного моделирования данных. В статье "Базовые концепции моделирования данных" дано описание двух расширений ER-подхода - IDEF1X и IE. В обеих методологиях сущности представляются прямоугольниками.

Далее даётся детальное описание сущности и представлены предварительные сведения о ключах с особым акцентом на поиск первичных ключей сущности. Также приводится описание типов сущностей, и даются рекомендации по именованию и описанию сущностей. Последний раздел посвящен разбору типичных ошибок, связанных с сущностями и ключами.

Что такое сущность?

Сущность - это физическое представление логической группировки данных. Сущности могут быть вещественными, реальными объектами, такими как ПЕРСОНА или МОРОЖЕНОЕ, или неосязаемыми концептуальными абстракциями как ЦЕНТР ЗАТРАТ или РЫНОК. Сущности не предназначены для представления единичного объекта, они представляют набор экземпляров, содержащих информацию, представляющую интерес с точки зрения их уникальности. Например, сущность ПЕРСОНА представляет собой экземпляр объектов типа Персона. Иван Петров, Мария Русанова и Савелий Богданов - конкретные примеры экземпляров сущности ПЕРСОНА. Конкретный экземпляр сущности представляется строкой таблицы и идентифицируется первичным ключом.

Сущность имеет следующие признаки:

- Она имеет имя и описание.
- Она представляет класс, а не единичный экземпляр абстракции.
- Ее конкретные представители (экземпляры) могут быть уникально идентифицированы.
- Она содержит логическую группировку атрибутов, представляющих информацию, интересную с точки зрения корпорации.

Формальные определения сущности

Ниже приведен список определений сущности признанных авторитетов в области моделирования данных. Обратите внимание на их сходство:

- Чен (1976): "Вещь, которая может быть однозначно идентифицирована".
- Дейт (1986): "Любой различимый объект, который будет представлен в базе данных".
- Финклештейн (1989): "Информационная сущность представляет некую ' вещь'которая сохраняется для дальнейших ссылок. Термин сущность относится к логическому представлению данных".

Выделение сущностей

Как приступить к процессу выделения сущностей? Большинство сущностей выявляются в ходе рабочих сессий и интервью. Анализ требований к информации, полученной от экспертов в предметной области и конечных пользователей - вот наилучший источник информации.

Другим хорошим источником является корпоративная модель.

Обратите внимание на имена существительные и имена объектов - вполне возможно, что они станут логическими сущностями. Старайтесь не представлять единичные экземпляры в виде сущностей, как это часто бывает, когда сущности моделируются в терминах роли.

Моделирование сущностей в терминах роли - достаточно распространенная ошибка. Сущности появляются и в процессе нормализации (см. "Понятие нормализации" в статье "Базовые концепции моделирования данных"). Приведение логической модели к третьей нормальной форме вероятнее всего приведет к появлению нескольких дополнительных сущностей.

Существует две основных группы сущностей: зависимые и независимые. Независимая сущность не нуждается в информации из другой сущности для идентификации уникального экземпляра. Она представляется в ERwin в виде прямоугольника. Первичный ключ независимой сущности не включает в себя первичных ключей других сущностей.
Зависимая сущность должна привлекать информацию из другой сущности для идентификации уникального экземпляра. Она представляется на ER-диаграмме в виде прямоугольника с закругленными углами. Первичный ключ зависимой сущности включает первичные ключи одной или более родительских сущностей.

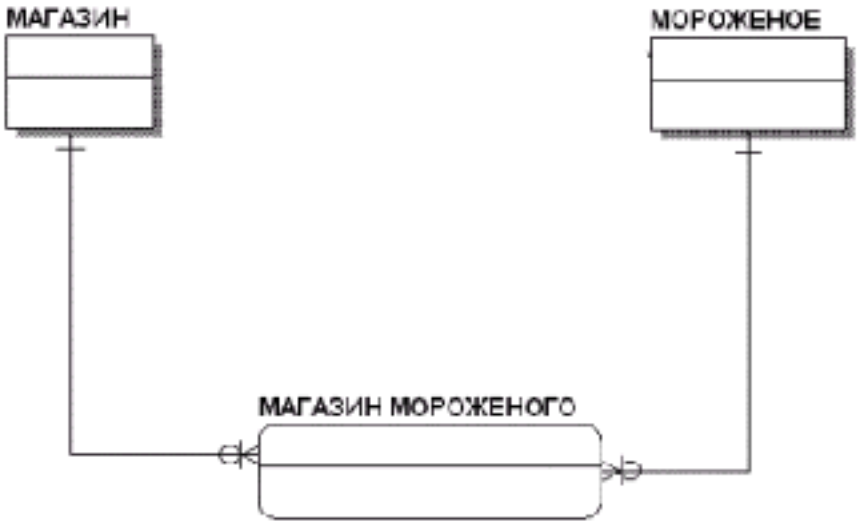


Рис. 2.1. Примеры стержневых сущностей для корпорации, торгующей мороженым.

Обратите внимание на рис. 2.1., где изображены прямые углы независимых сущностей МАГАЗИН и МОРОЖЕННОЕ и скругленные углы зависимой сущности МАГАЗИН МОРОЖЕННОГО.

Определение типов сущностей

И зависимые, и независимые сущности можно разделить на несколько типов:

- Стержневые сущности - их иногда называют основными или первичными сущностями. Они представляют важные объекты, информацию о которых следует хранить.
- Коды/ссылки/классификаторы - эти сущности содержат строки, определяющие набор значений или область определения для атрибута.
- Ассоциативные сущности - эти сущности используются для разрешения отношений многие-ко-многим.
- Характеристические сущности - эти сущности бывают двух типов: исключающие и включающие.

Стержневые сущности

Стержневые сущности представляют наиболее важные корпоративные информационные объекты. Их иногда называют первичными, главными или основными сущностями. Так как эти сущности чрезвычайно важны, то, скорее всего, они используются во многих подразделениях корпорации. Потратьте время на поиск сходных сущностей, поскольку для стержневых сущностей велика вероятность наличия возможности их повторного использования. В рамках корпорации стержневые сущности должны моделироваться единообразно. Хорошие разработчики моделей рассматривают такой подход как исключительно полезный.

Стержневые сущности могут быть как независимыми, так и зависимыми. На рисунке 2.1 представлены примеры стержневых сущностей для корпорации, торгующей мороженым. Сущность МОРОЖЕНОЕ представляет базовый продукт корпорации. Сущность МАГАЗИН является примером канала сбыта или посредника при продаже товара.

Предположим, что дела в корпорации идут хорошо и принимается решение об открытии дополнительного МАГАЗИНА. Для добавления новых экземпляров сущности МАГАЗИН нет необходимости менять модель. То же самое касается и сущности МОРОЖЕНОЕ.

Обратите внимание на стержневые сущности МОРОЖЕНОЕ и МАГАЗИН. Хотя пример может показаться несколько прямолинейным, он иллюстрирует всю мощь концепции, лежащей в основе моделирования стержневых сущностей.
Понимание необходимости моделировать стержневые сущности в виде масштабируемых и расширяемых контейнеров требует от разработчика модели взгляда на сущности как на абстрактные концепции и моделирования информации независимо от текущего способа ее использования. В этом примере модель сущности МОРОЖЕНОЕ полностью вне контекста сущности МАГАЗИН и наоборот. Так что если в корпорации решат продавать МОРОЖЕНОЕ через новый канал сбыта, такой как Интернет или доставка на дом, новый канал сбыта может быть добавлен без изменений в других сущностях.

> **Совет**
> Важность корректного моделирования стержневых сущностей значительно возрастает, когда организации начинают использовать новые пути использования своей информации. Представьте себе, что получение информации, которую вы моделируете сегодня, в будущем, вероятно, будет доступно несколькими различными способами. Например, многие компании должны обрабатывать запросы на информацию, поступающие через Интернет, отдельно, так как внутренние системы до сих пор не способны взаимодействовать с этим новым мощным коммуникационным каналом.

Кодовые сущности

Кодовые сущности всегда являются независимыми. Их часто называют ссылками, классификаторами или сущностями типов, в зависимости от используемой методологии.

Уникальные экземпляры, представляемые кодовыми сущностями, определяют область определения для значений атрибутов, принадлежащих другим сущностям. Отношения между кодовыми сущностями и другими сущностями будут рассмотрены в одной из следующих публикаций на эту тему. У вас может возникнуть искушение использовать единственный атрибут в кодовой таблице. Гораздо лучше включать, по меньшей мере, три атрибута в кодовую сущность: идентификатор, имя (иногда его называют кратким именем) и определение.

На рисунке 2.2 ВЕРХУШКА - независимая сущность (обратите внимание на прямые углы). ВЕРХУШКА является к тому же кодовой сущностью или классификатором. Экземпляры (строки) сущности ВЕРХУШКА определяют список доступных верхушек.

Кодовые сущности обычно содержат ограниченное количество атрибутов. Существуют реализации, где эти сущности имели только один атрибут. Предпочтительно моделировать кодовые сущности с использованием искусственного идентификатора. Искусственный идентификатор вместе с именем и определением позволяют добавлять новые виды ВЕРХУШЕК в качестве экземпляров (строк) в сущность. Обратите внимание на три атрибута сущности ВЕРХУШКА.

Специалисты часто ссылаются на кодовые сущности, как на корпоративные бизнес-объекты. Термин корпоративный бизнес-объект указывает, что сущность определена и совместно используется на корпоративном уровне, а не на уровне единичного приложения, системы или подразделения организации. Эти сущности часто совместно используются многими базами данных для обеспечения целостного подхода к формированию сводных отчетов и при проведении анализа тенденций.

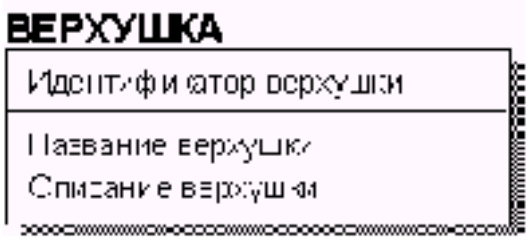


Рис. 2.2. Кодовые сущности позволяют корпорации определять набор значений для централизованного использования в рамках корпорации. Экземпляры кодовой сущности определяют область определения значений для использования в других частях модели.

Ассоциативные сущности

Ассоциативными являются сущности, которые содержат первичные ключи двух или более других сущностей. Ассоциативные сущности всегда зависимы. Они используются для разрешения отношений многие-ко-многим других сущностей. Отношения многие-ко-многим возникают в том случае, когда множество экземпляров одной сущности связаны с множеством экземпляров другой. Ассоциативные сущности позволяют нам моделировать пересечение экземпляров двух сущностей, обеспечивая уникальность каждого экземпляра ассоциации.

> Примечание
> Отношения многие-ко-многим не могут быть реализованы в базе данных на физическом уровне. ERwin автоматически создает ассоциативные сущности для разрешения отношений многие-ко-многим при переходе от логической модели к физической.

На рисунке 2.1 ассоциативная сущность используется для разрешения отношения многие-ко-многим между сущностями МАГАЗИН и МОРОЖЕНОЕ. Введение ассоциативной сущности дает возможность использовать одно и то же МОРОЖЕНОЕ для продажи в нескольких экземплярах МАГАЗИНА, без необходимости продажи в каждом из МАГАЗИНОВ одинаковых сортов МОРОЖЕНОГО. Ассоциативная сущность МАГАЗИН МОРОЖЕНОГО учитывает тот факт, что экземпляр МАГАЗИНА продает множество экземпляров МОРОЖЕНОГО, и экземпляр МОРОЖЕНОГО может продаваться многими экземплярами МАГАЗИНА.

Характеристические сущности

Характеристические сущности всегда являются зависимыми. Вы должны использовать характеристические сущности там, где для экземпляров сущностей имеет смысл хранить различные наборы атрибутов. Финклештейн называет характеристические сущности вторичными сущностями. Характеристические сущности всегда имеют одну или более "равноправных" сущностей. Равноправные характеристические сущности связаны с родительской сущностью особым типом отношений, которые могут быть исключающими или включающими.

> Примечание
> Равноправные характеристические сущности, которые находятся в исключающем отношении к родительской сущности, указывают на то, что только одна из равноправных сущностей содержит экземпляр для любого из экземпляров родительской сущности. Исключающая характеристическая сущность представляет отношение "является" (is a).

На рисунке 2.3 представлена сущность КОНТЕЙНЕР и характеристические сущности РОЖОК и СТАКАНЧИК. Магазин мороженого, судя по всему, торгует не на развес, а отдельными порциями. Обратите внимание, что экземпляр КОНТЕЙНЕРА должен быть РОЖКОМ или СТАКАНЧИКОМ. КОНТЕЙНЕР не может быть одновременно и РОЖКОМ и СТАКАНЧИКОМ. Это исключающие характеристические сущности.

Сущность ПЕРСОНА на рисунке 2.3 имеет две характеристические сущности СОТРУДНИК и КЛИЕНТ. Заметьте, что исключающие характеристические сущности не позволят одному экземпляру ПЕРСОНЫ содержать факты, общие для СОТРУДНИКА и КЛИЕНТА. Естественно, это противоречит реальной практике. СОТРУДНИК определенно может быть КЛИЕНТОМ. ПОСТАВЩИК тоже может выступать в качестве КЛИЕНТА. Это пример включающих характеристических сущностей.

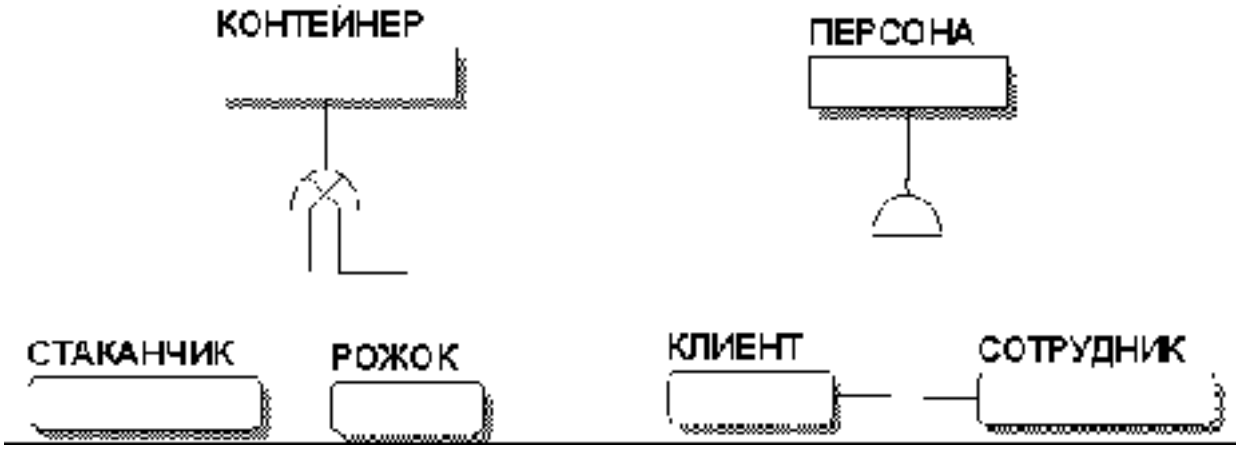


Рис. 2.3. Два примера характеристических сущностей ПЕРСОНА и КОНТЕЙНЕР. Оба примера используютнотацию ERwin IE для представления исключающих и включающих характеристических сущностей. Отсутствие (X) в символе характеристической сущности указываетна включающее отношение.

Структурная сущность

Иногда экземпляры одной и той же сущности связаны. В своей книге 1992-го года "Strategic Systems Development" К. Финклештейн предложил использовать структурные сущности для представления отношений между экземплярами одной и той же сущности. Связи между экземплярами одной и той же сущности называются рекурсивными отношениями. Рекурсивные отношения будут рассмотрены в статье "Понятие отношения". Рекурсивные отношения - это логическая концепция, а концепции не легко воспринимаются пользователями.

На рисунке 2.4 показана дополнительная структурная сущность, описывающая отношение между экземплярами сущности СОТРУДНИК. Диаграмма показывает, что характеристическая сущность СОТРУДНИК сущности ПЕРСОНА имеет две характеристические сущности ИСПОЛНИТЕЛЬ и УПРАВЛЕНЕЦ. Сущность СТРУКТУРА СОТРУДНИКОВ представляет отношение между экземплярами сущности СОТРУДНИК.

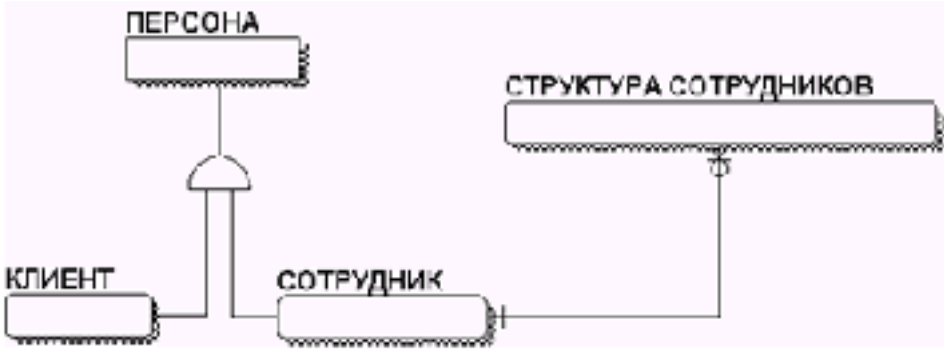


Рис. 2.4. Структурная сущность - иллюстрация подхода К. Финклештейна к разрешению рекурсивных отношений.

Определение первичного ключа

Для идентификации конкретного экземпляра сущности вам необходимо определить

первичный ключ. Первичным ключом служит атрибут или набор атрибутов, уникально идентифицирующих единственный экземпляр сущности. Другими словами, первичный ключ может быть как одним атрибутом, так и состоять из нескольких. Первичный ключ, состоящий более чем из одного атрибута, называется составным или компонентным ключом. Далее мы будем использовать термин составной ключ.

Первичный ключ должен быть статическим (static) и неразрушаемым (non-volatile). Под статичностью и неразрушаемостью подразумевается, что первичный ключ не должен подвергаться изменениям. Изменения первичного ключа трудно сопровождать, что часто приводит к весьма дорогостоящим переделкам, поэтому лучшим считается вариант, когда первичный ключ абсолютно не зависит от экземпляров сущности.

> Примечание
> Далее будут добавлены первичные ключи в базу данных торговли мороженым и рассмотрены кандидаты в ключи. Концепции ключей только введены в данном разделе, а обсуждаться более подробно будут в статье "Понятие атрибута".

Для нахождения первичного ключа требуется проанализировать данные, определяющие сущность. Как правило, первичные ключи для стержневых сущностей определяются во время рабочих сессий и обсуждений. Эксперты предметной области и пользователи - хорошие источники информации для выбора потенциальных первичных ключей. Примеры данных тоже обеспечивают ценный вклад при выборе первичного ключа.

Начинайте процесс выявления первичных ключей с определения всех потенциально ключевых атрибутов, называемых кандидатами в ключи. Кандидатом в ключи может быть и один атрибут, и комбинация нескольких атрибутов. Если кандидатов в ключи не существует, или кандидатом является составной ключ, который слишком велик и громоздок, рассмотрите возможность использования искусственного уникального идентификатора. Ключи, заимствованные из родительской сущности, называются внешними ключами. Внешние ключи будут рассматриваться в одной из последующих публикаций на эту тему. Ниже приведено описание различных типов ключей:

- Кандидат в ключи. Кандидатом в ключи является атрибут или набор атрибутов, идентифицирующих единичный экземпляр сущности. Иногда единичный экземпляр сущности идентифицируется несколькими атрибутами или их комбинацией.
- Составной ключ. Ключ, который состоит более чем из одного атрибута, называется составным, сложным или компонентным. Для составных ключей каждая составляющая ключа должна иметь значение для каждого экземпляра. Ни одна часть ключа не должна быть неопределенной (NULL). Все части ключа являются обязательными и не могут быть опущены.
- Искусственный первичный ключ. Иногда ни единичный атрибут, ни комбинация атрибутов, не определяют экземпляр. В этих случаях вы используете искусственный уникальный идентификатор. Искусственные первичные ключи часто просто нумеруют каждый экземпляр или код.
- Внешние ключи. Когда первичный ключ одной сущности мигрирует в другую таблицу, он называется внешним ключом. Внешние ключи "связывают" сущности, представляя отношения между ними. Внешние ключи будут обсуждаться более подробно в

дальнейших статьях по этой тематике.

Приведение модели к третьей нормальной форме включает проверку на отсутствие функциональных зависимостей и выявление первичных или составных ключей. Функциональные зависимости, обсуждавшиеся в статье "Базовые концепции моделирования данных", играют важную роль при выявлении первичных ключей и кандидатов в ключи.

Именование сущностей

Имя, присваиваемое сущности, должно характеризовать экземпляры сущности. Имя должно быть понятным и общепринятым. При выборе имени руководствуйтесь корпоративной точкой зрения и старайтесь использовать имена, отражающие способ использования данных в рамках корпорации, а не в отдельном подразделении. Используйте имена, осмысленные для сообщества пользователей и экспертов предметной области.

Вероятно, у вас в корпорации есть набор соглашений об именовании, используемых в ходе разработки или при формировании корпоративной модели данных, которыми вы руководствуетесь. Использование соглашений гарантирует, что имена конструируются единообразно в рамках корпорации, вне зависимости от того, кто конструирует имя. В следующих разделах приводится начальный набор соглашений об именовании, и даются примеры хороших и плохих вариантов имен.

Соглашения об именовании сущностей

Соглашения об именовании могут показаться несущественными, если вы работаете в маленькой организации, с небольшим количеством пользователей. Однако, в большой организации с несколькими командами разработчиков и большим количеством пользователей, соглашения об именовании существенно помогают при взаимодействии и совместном использовании данных. В идеале, вы централизованно должны разработать и сопровождать соглашения об именовании, и затем документально оформить их, опубликовав для всей корпорации.

Ниже приведены некоторые положения для формирования начального набора соглашений об именовании, на случай, если в вашей организации пока такой набор не разработан:

- Имя сущности должно быть достаточно описательным. Используйте имена из одного слова, только когда они являются названием широко распространенных концепций. Подумайте об использовании словосочетаний на основе существительных.
- Имя сущности должно быть существительным или словосочетанием на основе существительного в единственном числе. Используйте ПЕРСОНА вместо ПЕРСОНЫ или ЛЮДИ, и КОНТЕЙНЕР - вместо КОНТЕЙНЕРЫ.
- Имя сущности должно быть уникальным. Использование одинаковых имен для сущностей, содержащих различные данные, или разных имен для сущностей, содержащих одинаковые данные, будет без необходимости вводить в заблуждение разработчиков и конечных пользователей.
- Имя сущности должно указывать на данные, которые будут храниться в каждом из экземпляров.
- Имя сущности не должно содержать специальных символов (таких как !, @, #, $, %, &, * и тому подобных) или указывать на принадлежность (МОРОЖЕНОЕ ПЕРСОНЫ).

- Имя сущности не должно содержать акронимов или аббревиатур, если только они не являются частью принятых соглашений об именовании.

Разработчикам моделей рекомендуется использовать хорошие соглашения об именовании, если таковые существуют, или разработать их, следуя приведенным положениям, если таких соглашений нет.

Примеры хороших имен сущностей

Всегда лучше использовать единообразные имена в рамках корпорации. В таблице 2.1 приведены примеры хороших и плохих имен для сущностей.

> Совет
> Обратите внимание, что для имен сущностей используются заглавные буквы, что соответствует рекомендуемому соглашению об именовании сущностей.

ТАБЛИЦА 2.1 Примеры имен сущностей с объяснениями.

Хорошее имя	Неудачное имя	Пояснение
МАТЕМАТИЧЕСКАЯ ФОРМУЛА	ФОРМУЛА	ФОРМУЛА - слишком расплывчато, добавление прилагательного МАТЕМАТИЧЕСКАЯ значительно проясняет смысл.
КНИГА	КНИГИ	КНИГА - существительное в единственном числе.
СОФА	СОФА КУШЕТКА	СОФА и КУШЕТКА имеют одинаковый смысл. Выберите что-то одно.
МОРОЖЕНОЕ	КАКОЕ-ТО МОРОЖЕНОЕ	Местоимение КАКОЕ-ТО не привносит дополнительного значения или смысла к термину. Избегайте излишних дополнений.
ФОТОСНИМОК	ИЗОБРАЖЕНИЕ	ФОТОСНИМОК - достаточно определенно. ИЗОБРАЖЕНИЕ - несколько расплывчато.
ОЖИДАЕМОЕ ВРЕМЯ ПРИБЫТИЯ	ОВП	Аббревиатура ОВП может оказаться непонятной для пользователей.
КОМПАНИЯ	КОМПАНИЯ XYZ	XYZ - конкретный экземпляр компании и должен быть строкой в сущности КОМПАНИЯ.

Описание сущностей

Даже хороших имен, указывающих пользователю, какую информацию стоит ожидать от сущности, обычно недостаточно. Каждая сущность нуждается в ясном, точном и полном описании или определении, чтобы быть однозначно интерпретируемой в рамках корпорации. Описание сущности должно объяснять смысл сущности и ее значение для корпорации.

Хотя описание, определение и назначение часто используются в качестве синонимов, термин описание предпочтительнее, поскольку он побуждает нас описывать сущности в терминах, понятных для пользователя.

Правила формирования хороших описаний

Описание сущности должно объяснять ее смысл, а не то, как будет использоваться информация этой сущности. Вы должны собирать описания сущностей во время идентификации сущностей. Будьте осторожны при включении информации об использовании: подобная информация должна использоваться только в качестве примера или для пояснения. Способ использования информации изменяется более часто, чем информация сама по себе, поэтому информация об использовании непостоянна.

Описание сущности должно быть ясным, точным, полным и непротиворечивым. Оно должно быть сформулировано без привлечения технических терминов, понятно любому, кто хотя бы чуть-чуть знаком с описываемой концепцией. Убедитесь, что описание сформулировано в терминах бизнеса, и включает пояснение значимости сущности.

Примеры хороших описаний

Таблица 2.2 не претендует на полноту, но служит для демонстрации хороших описаний и причин, по которым неудачные описания не отвечают основным положениям.

ТАБЛИЦА 2.2. Описания сущностей с пояснениями

Хорошее описание	Неудачное описание	Пояснение
ПЕРСОНА содержит информацию о физических лицах, которые вступают во взаимодействие с корпорацией. Информация о ПЕРСОНЕ помогает корпорации при планировании, разработке продуктов и рекламной деятельности.	Клиент или сотрудник.	Хорошее описание включает определение сущности и ее значение для корпорации.
	Включает имя, дату рождения и т.п. для персоны.	Простое перечисление атрибутов сущности не несет дополнительной информации о том, что собой представляет сущность и почему она важна для корпорации.
	Информация о клиентах и сотрудниках.	Клиент и сотрудник являются примерами ролей, в которых может выступать ПЕРСОНА. Использование одних только примеров не объясняет, что сущность собой представляет и почему она важна для корпорации.

Хорошее опи сание	Неудачное описание	Пояснение
	Сущность содержит символы и числовые данные, извлеченные из POS (Point Of Sale - торговый терминал), хранящиеся с использованием стандартного сжатия и упакованных десятичных чисел.	Данный искусственный пример призван проиллюстрировать, что технические описания и аббревиатуры с трудом понимаются бизнес-пользователями.

Распространенные ошибки при моделировании сущностей и выборе ключей

Этот раздел, посвященный распространенным ошибкам при моделировании, не претендует на полноту. Его цель - указать на наиболее распространенные ошибки, которые возникают у разработчиков моделей.

> Совет
> Иногда при моделировании разработчик модели делает некий выбор, руководствуясь совершенно правильными принципами. Очень важно понимать всю причинно-следственную цепочку принимаемых решений так, чтобы вы ясно понимали полученное заключение.

Моделирование ролей

Что подразумевается под моделированием ролей? Во время рабочих сессий пользователи могут сказать вам, что им необходимо хранить информацию о сотрудниках. Возникает искушение создать сущность СОТРУДНИК. Более тщательный анализ информации, представляющей интерес для корпорации, например, такой как имя, адрес и номер социального страхования показывает, что эти значения не зависят от сущности СОТРУДНИК. Для конкретного СОТРУДНИКА значение атрибута ИМЯ не зависит от сущности СОТРУДНИК. Это легко понять, если задуматься о том, что ваше имя остается вашим именем вне зависимости от того, являетесь ли вы СОТРУДНИКОМ или нет.

Перегрузка сущностей

Перегруженными являются сущности, содержащие информацию более чем об одном концептуальном объекте. Если некоторые атрибуты сущности описывают одну и ту же концепцию, такие сущности следует проверить. Перегруженные сущности имеют значения не для каждого из атрибутов.

Иногда эксперты из разных предметных областей в корпорации используют имя сущности, которое звучит и пишется одинаково, но имеет разный смысл для разных экспертов. Единственный способ убедится, что одинаковые имена описывают одинаковые объекты, это проверка описаний. Убедитесь, что сущность содержит данные, описывающие единственную

концепцию.

Например, сущность ОБОРУДОВАНИЕ может иметь совершенно разное значение для подразделений информационных технологий и для отдела средств массовой информации и коммуникаций.

Избыточные сущности

Избыточными являются сущности, имеющие различные имена, но содержащие информацию о сходных концепциях. Английский язык включает много слов для представления одних и тех же вещей. Один из способов обнаружить такие сущности - это поиск сущностей, содержащих сходные атрибуты. Сравните описания каждой из таких сущностей, чтобы определить, не представляют ли они сходные концепции. Избыточные сущности часто появляются в результате тенденции к моделированию ролей в качестве сущностей.

Например, сущности УПРАВЛЕНЕЦ и СОТРУДНИК могут содержать сходную информацию, поскольку обе являются ролями, которые может играть экземпляр сущности ПЕРСОНА.

Выбор неправильного первичного ключа

Выбор неправильного первичного ключа означает, что вы выбрали первичный ключ, не выдерживающий тестирования. Распространенными ошибками, связанными с первичным ключом, являются:

- Не уникальность: первичный ключ не является уникальным для каждого из экземпляров. Например, разработчик модели может считать, что номер социального страхования является уникальным для каждой ПЕРСОНЫ. Однако номер социального страхования может повторно использоваться в том случае, если первоначальный его владелец скончался.
- Требуемое значение/неопределенность: первичный ключ не имеет значения для некоторых из экземпляров. Например, не каждый экземпляр сущности ПЕРСОНА будет иметь номер социального страхования. Иностранцы и маленькие дети - вот две категории людей, у которых он будет отсутствовать.

Использование неудачных имен сущностей

Непонятные, неоднозначные или неточные имена затрудняют для новых пользователей и команд разработчиков повторное использование или расширение существующей модели.

Не используйте аббревиатуры или акронимы в качестве части имени. Аббревиатуры и акронимы открыты для неправильной интерпретации и даже могут иметь разное значение в разных предметных областях.

> Предостережение
> Не используйте имена собственные, указывающие на конкретный экземпляр, такие как, например, Компания Интерфейс. Это неудачный выбор имени сущности, который в дальнейшем приведет к серьезным проблемам при моделировании.

Не включайте месторасположение в качестве части имени. Как правило, вам неизбежно потребуется и другое месторасположение. Имя с указанием расположения является признаком того, что вы моделируете конкретный экземпляр вместо класса сущностей.

Использование неудачных описаний сущностей

Не используйте описаний, заимствованных только из словаря. Описания из словаря не будут включать значимую для бизнеса информацию.

Не пытайтесь перефразировать имя сущности. Не используйте имя сущности в ее описании.

Неясные, расплывчатые или, что еще хуже, неполные описания затрудняют повторное использование и расширение существующей модели. Пользователь не сможет проверить, содержит ли сущность всю необходимую информацию.

При этом значительно повышается риск возникновения перегруженных сущностей и использования их для хранения информации о разных объектах.

Концепции, которые кажутся очевидными для всех участников рабочих сессий, могут перестать быть столь очевидными с течением времени, когда перед новой командой разработчиков будет поставлена задача расширения существующей модели.

Заключение

Сущности представляют собой объекты, информацию о которых следует накапливать и сопровождать. Они являются "контейнерами" для организации и группировки бизнес-фактов. Наиболее важные сущности обычно выявляются и фиксируются в документах во время рабочих сессий или интервью, а также в результате процесса нормализации.

Сущности делятся на две основные группы: зависимые и независимые. Зависимым сущностям для уникальной идентификации экземпляра требуется информация из других сущностей, независимым - нет. В рамках двух основных групп сущностей выделяются более специализированные типы, с особенностями для поддержки конкретных видов отношений между основными и подчиненными сущностями.

Каждая сущность должна включать один или несколько наборов атрибутов, являющихся кандидатами в ключи. Кандидаты в ключи уникально идентифицируют конкретные экземпляры сущности. Кандидаты в ключи могут состоять из одного атрибута или из группы атрибутов. Если кандидатов в ключи не существует, или их трудно сопровождать, вам может потребоваться создать искусственный первичный ключ. Анализ и исследования играют важную роль в определении первичных ключей, которые будут сохранять уникальность и надежность с течением времени.

Для сущностей необходимы хорошие имена и описания. Стандарты и соглашения об именовании обеспечивают целостный подход к разработке имен и описаний. Характеристики сущности определяются содержащимися в ней атрибутами. Атрибуты сущности представляют факты, касающиеся сущности, которые корпорация заинтересована накапливать и сопровождать.

В следующей статье данной серии будет описан процесс выявления атрибутов и их характеристик, определения ключевых и не ключевых атрибутов, областей определения и необязательных данных, а также сформулированы соглашения для формирования хороших

имен и описаний атрибутов.

Проектирование баз данных с ERwin
Базовые концепции моделирования данных
(Часть 1)

Зайцев С.Л., к.ф.-м.н.

Моделирование данных представляет собой деятельность по обнаружению и документированию требований к информации. Требования к информации описывают данные и бизнес-правила, необходимые для поддержки бизнеса. Модель данных может выражать как сложные информационные потребности целой корпорации, так и конкретные информационные потребности одной единственной программы. ***ERwin - это графический инструментарий для моделирования данных,*** основной целью которого является помощь аналитику в использовании бизнес-правил и требований к информации при создании логических и физических моделей данных. Как и многие другие инструментальные средства, ERwin следует использовать тому, кто понимает, для чего именно этот инструмент предназначен, и кто способен использовать его наиболее продуктивно.

Здесь описаны концепции и процедуры моделирования данных, принятые в подходе ER (Entity Relational - отношения между сущностями), на основе рассмотрения следующих вопросов:

- Роль моделирования данных
- Формирование модели данных
- Принятие корпоративного представления
- Основы методологии моделирования

Понимание процесса моделирования начинается с понимания концепций моделирования данных - семантики языка моделирования. Первая часть раскрывает концепции с точки зрения начинающего аналитика. Эти концепции вводятся последовательно, без попытки объяснить мощную математическую базу реляционной алгебры.

Роль моделирования данных

Моделирование данных обеспечивает наибольшую отдачу при применении на ранних стадиях жизненного цикла разработки. Модель предоставляет критически важную информацию для понимания функциональных границ проекта на итерационных фазах разработки. Начало фазы реализации без ясного понимания требований к данным может привести к тому, что ваш проект быстро выйдет за рамки отведенного бюджета или закончит свое существование на куче недоделанных бета-версий.

Проект разработки ПО: общее представление

Множество публикаций посвящено обсуждению организации проекта разработки, и данный текст не претендует на детальное рассмотрение этой области. Данный раздел включен в статью только для того, чтобы помочь аналитикам осознать роль и место моделирования данных в процессе работы над проектом.

Большинство компаний руководствуется какой-либо методологией, которая очерчивает выбранный жизненный цикл для направления процесса разработки. Как правило, на

концептуальном уровне большинство придерживается сходной последовательности шагов:

1. Определение проблемы
2. Анализ требований
3. Концептуальное проектирование
4. Детальное проектирование
5. Реализация
6. Тестирование

Этот подход к разработке известен как метод водопада. Как видно из рисунка 1.1, каждая фаза завершается перед переходом к следующей, создавая эффект "водопада".

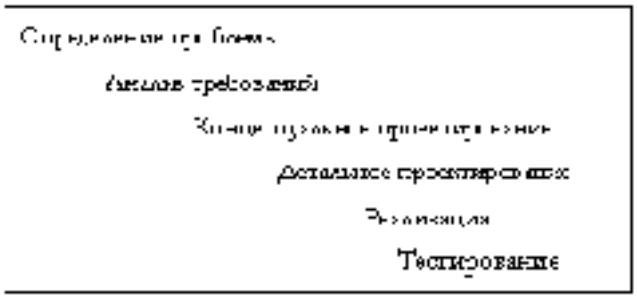


Рис. 1.1. Метод водопада для проекта разработки ПО.

Обратите внимание, что результаты каждой предыдущей фазы являются основой для следующей.

Многие проекты реализуются на основе итераций или фаз. Итеративная разработка снижает риск путем разбиения проекта на отдельные управляемые фазы, каждая из которых включает анализ, детальное проектирование, реализацию и тестирование. Последующие фазы строятся на базе функциональных возможностей, реализованных на предыдущих фазах. При этом внутри каждой фазы применяется все тот же метод водопада.

Когда именно следует заняться моделированием данных

Логическое моделирование должно производиться в процессе разработки как можно раньше, часто уже на этапе определения проблемы. Кроме того, логическое моделирование данных является мощным средством как для определения и документирования требований к данным, так и для выявления бизнес-правил, описывающих способы использования данных. Фактически, многие профессионалы считают, что модель данных должна служить фундаментом проекта разработки ПО.

Формирование модели данных

Модель данных является визуальным представлением структур данных, данных и бизнес-правил для СУБД. Обычно она разрабатывается как часть более крупного проекта по разработке ПО.
Модель данных состоит из двух компонент - логической и физической моделей. В большинстве случаев первой создается логическая модель, а уже потом модель физическая. Иногда модель данных получается путем реконструкции из существующей базы данных. Процесс реконструкции (обратное проектирование) будет обсуждаться в последующих публикациях на эту тему.

В большинстве случаев для построения модели данных вам потребуется выполнить следующие операции:

1. Определение проблемы и функциональных границ
2. Сбор требований
3. Анализ
4. Формирование логической модели
5. Формирование физической модели
6. Генерация базы данных

Рисунок 1.2 показывает, как каждый из шагов обеспечивает исходные данные для следующего шага.

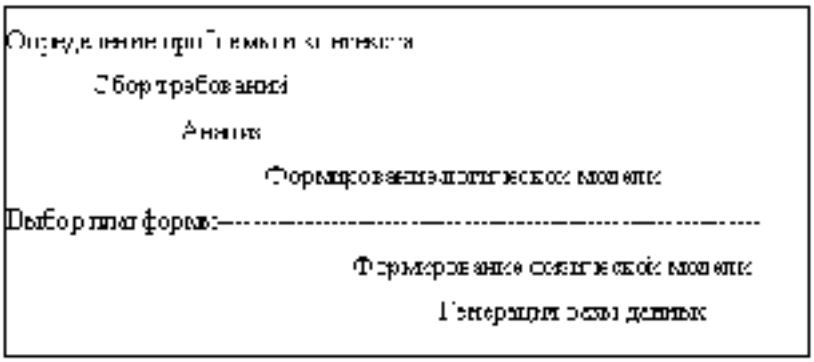


Рис. 1.2. Формирование логической модели может предшествовать выбору платформы СУБД.

Определение проблемы и функциональных границ

Логическое моделирование данных начинается с определения проблемы. Этот шаг иногда определяют как формулирование цели или функциональных границ проекта. Определение проблемы может состоять из одного параграфа или представлять собой сложный документ, отражающий систему бизнес-целей. Эта операция задает функциональные границы модели данных, точно так же, как землемер устанавливает границы частного владения.

Сбор требований к информации

Многие из профессионалов согласны, что наиболее важной задачей проекта разработки ПО является точное и полное определение требований. В самом деле, неполное или неточное понимание требований может привести к выходу за рамки отведенного бюджета и существенному срыву срока завершения проекта.

Сбор требований к информации представляет собой процедуру обнаружения и документирования информации, необходимой для идентификации и определения сущностей, атрибутов и бизнес-правил, составляющих логическую модель. Существует два хорошо известных метода сбора требований: рабочие сессии и интервью. Большинство методологий разработки рекомендуют практику рабочих сессий. В следующем разделе приведены краткие рекомендации по сбору требований к информации в ходе рабочих сессий. Последующее упражнение демонстрирует использование собранной информации при формировании модели данных с помощью ERwin.

Подготовка к рабочей сессии

Важным фактором успеха проведения рабочей сессии является подготовка к ней. Она начинается с правильного подбора участников. Естественно вы должны предусмотреть решение организационных вопросов, таких как расписание заседаний и наличие необходимого оборудования. Распространение описания проблемы и других информационных материалов среди участников до проведения сессии может значительно повысить ее продуктивность. Но нет ничего более важного, чем обеспечение присутствия на сессии всех необходимых специалистов.

Рекомендации по подбору участников сессии

Определение правильного списка участников - одна из важнейших задач при подготовке сессии. Участники должны иметь не только теоретические знания, но и практический опыт в предметной области рассматриваемой проблемы. Наиболее успешно совещания проходят с участием специалистов, лично заинтересованных в успешной реализации проекта.

Хороший председатель - еще один критически важный фактор. Председатель гарантирует, что все участники имеют возможность поделиться знаниями, опытом и высказать свое мнение. Сессия, на которой выступили лишь несколько человек, вероятнее всего, приведет к неадекватному набору требований к информации.

По меньшей мере, один профессиональный секретарь должен вести записи и документировать требования к информации. Так как логическая модель будет строиться на основе информации, собранной во время сессии, хорошая документация является важным фактором успеха.

Аналитик должен документировать все сущности, атрибуты и отношения по мере сбора информации. При необходимости он должен задавать уточняющие вопросы и проверять любые предположения. Хотя многие из аналитиков могут так же председательствовать на заседании, однако одновременно вести заседание и строить модель достаточно сложно.

Представители высшего руководства должны участвовать в заседаниях для принятия окончательных решений в спорных случаях. Если представитель руководства не присутствует, убедитесь в том, что конфликтные точки зрения зафиксированы для получения последующей резолюции.

Помещение и оборудование

В наше время, постоянно проводимых видео- и телеконференций, может оказаться достаточно сложно найти подходящее помещение и оборудование для проведения сессии. Кроме того, мы все-таки люди, и значительную часть информации передаем мимикой и языком жестов. Поэтому, по возможности, не пренебрегайте личными встречами на заседаниях сессии.

Комната со столами и стульями, расположенными полукругом - оптимальный вариант. Комната должна быть прохладной, свободной от предметов, отвлекающих внимание, и достаточно большой, чтобы вместить всех участников. Предоставьте всем блокноты, ручки или карандаши, а также позаботьтесь о карточках с именем каждого участника. И не забудьте о таких немаловажных мелочах как вода, прохладительные напитки и кофе. Существенным

дополнением могут оказаться конфеты или другие сладости.

Наличие доски значительно облегчает создание набросков концепций и информации. Электронные доски даже удобнее, так как вы можете распечатать копии для всех участников. Убедитесь, что в комплекте достаточно запасных маркеров.

Председателю полезно использовать два лекционных плаката: один для записи обсуждаемых вопросов, а другой для фиксации отложенных проблем, требующих для своего разрешения привлечения помощи извне. Нужно назначить по одному или несколько ответственных за каждый из вопросов, как запланированных, так и отложенных.

Распространение материалов перед заседанием

С появлением электронной почты почти в любой компании распространение материалов перед очередным заседанием стало гораздо проще. Участники сессии должны получить, как минимум, описание проблемы, программу, правила проведения сессии и список целей. Предварительный список сущностей и атрибутов с первоначальными определениями и скелетной моделью будут очень полезны.

> ПРИМЕЧАНИЕ
> Скелетная модель предоставляет стартовую позицию для процесса моделирования. Она содержит ограниченный набор сущностей и атрибутов, и создается на основе описания проблемы и какой-либо предыдущей работы, относящейся к данной предметной области.
>
> Вы можете использовать скелетную модель и как дополнение к описанию концепций моделирования для участников сессии.

Убеждайте участников готовиться к заседаниям! Запланируйте перерывы, по меньшей мере, раз в два часа.

С чего начать

Председатель начинает с представления участников сессии и обзора целей сессии в терминах описания проблемы. Программа сессии обсуждается, чтобы убедиться в ее соответствии поставленным целям. Покажите участникам, где находятся места общего пользования и телефоны. Это может показаться смешным, но иногда отсутствие информации даже относительно таких мелочей может значительно испортить настроение участников и, как следствие, уменьшить отдачу от всей сессии.

Правила проведения сессии призваны сохранить ее продуктивность. Ниже приведен список наиболее общих правил:

- Начало - в назначенное время
- Говорит один человек, а не все сразу
- Выступающий должен иметь возможность выразить свои соображения до конца, не перебивайте его
- Равноправие всех участников
- Если спорный вопрос не может быть решен в разумное время, зафиксируйте проблему или отложенный вопрос

- После перерыва вовремя приступайте к работе
- Заканчивать заседание нужно в назначенное время

Каждый участник должен представиться и кратко сообщить, зачем его привлекли к работе или что он хочет вынести из сессии.

Хороший председатель фокусирует внимание участников на поставленных целях и достижении консенсуса там, где это возможно. Делайте перерывы вовремя. Можно, например, попросить кого-либо из добровольцев подать мне знак, когда подходит время перерыва. Подводя итог сессии, распределите все задачи между участниками и назначьте даты их завершения. Необходимо запланировать следующую встречу для обзора требований к информации и предварительной логической модели. Не забудьте поблагодарить каждого из участников. Результаты встречи опубликуйте в течение трех рабочих дней, предоставьте возможность участникам высказать свои замечания.

Анализ

Для формирования целостной логической модели, вы должны проанализировать и исследовать требования к данным и бизнес-правила. В результате проведения анализа должны быть созданы точные и полные определения для всех сущностей, атрибутов и отношений. В течение фазы анализа собираются и документируются метаданные (данные о данных).

Анализ может проводиться как разработчиком модели, так и бизнес-аналитиком. И тот, и другой работают с пользователями, документируя их намерения по использованию данных. Эта работа должна привести к формированию корпоративных бизнес-объектов, необходимых для ведения требований к информации. Корпоративные бизнес-объекты приводят к коду, ссылкам и классификации структур данных. Здесь кроется возможность для документирования кодовых значений, которые будут использоваться. Вы должны тщательно документировать все производные данные (данные, которые создаются путем комбинации или манипуляции над одним или более элементами данных) и исходные элементы данных.

Данный раздел посвящен более детальному рассмотрению отображения и трансформации данных, предметных областей и значений по умолчанию.

Отображение и трансформация данных

Часто вам необходимо некоторым образом отображать или трансформировать данные. Отображение значений данных из источника в базу данных является важной задачей анализа. Важно помнить, что моделирование данных не должно зависеть от их источника. То есть способ получения данных не должен влиять на подход к их моделированию.

В таблице 1.1 приведен пример использования правил трансформации и отображения для определения того, что исходное кодирование данных действительно содержит три отдельных и различимых значения. Кроме того, соответствие определяет бизнес-правила для отображения исходных кодов на три самостоятельных атрибута, которые они представляют.

Таблица 1.1. Правила отображения для исходного кода.

Исходный код	Значение атрибута 1	Значение атрибута 2	Значение атрибута 3
1234567890	123	456	7890
0987654321	098	765	4321
12345	123	450	0000

Бизнес-правила, показанные в таблице 1.1, включают руководство для отображения значений исходного кода в его компоненты. Значения исходного кода содержат атрибут 1, атрибут 2 и атрибут 3. Новые значения исходного кода используются, начиная с 1-го января 1998, и состоят из 10 цифр, в то время как старые значения состояли из 5 цифр. Первые три цифры значения исходного кода содержат атрибут 1. Следующие три цифры составляют атрибут 2. Последние четыре цифры содержат значение атрибута 3. Для значений из пяти цифр в исходном коде вы используете ноль (0) в качестве третей цифры атрибута 2 и заполняете нулями значение атрибута 3.

Область определения данных

Область определения данных должна быть проанализирована и документирована. Область определения накладывает ограничения на значения атрибутов. Например, в России в качестве рабочего дня рассматривается один из следующих: понедельник, вторник, среда, четверг или пятница. Этот список является областью определения для значений атрибута с именем **День Недели.**

Значения по умолчанию

Некоторым атрибутам значения назначаются по умолчанию. Значениями по умолчанию для атрибутов могут быть значения, которые атрибут имеет в большинстве случаев. Значения по умолчанию могут устанавливаться для атрибутов, чтобы избежать неинициализированных значений. Например, значением по умолчанию для Количества Покрышек должно быть 4, так как большинство транспортных средств имеет четыре покрышки на колесах. Часто для числовых полей принимают 0 в качестве значения по умолчанию, так как неинициализированные значения могут привести к непредвиденным последствиям.

> ***ПРИМЕЧАНИЕ***
> *Аналитик должен с самого начала уделять внимание неинициализированным значениям и значениям по умолчанию. Тщательно обсудите возможное отсутствие значения и значения по умолчанию с экспертами предметной области. Не забывайте с осторожностью относиться к генерации ответов по полям, если эти поля могутзаполняться значениями по умолчанию, так как это можетпривести к неожиданным результатам.*

Вы должны определить и документировать все значения по умолчанию для атрибутов. Значения атрибутов по умолчанию должны быть едиными для всей корпорации. Проверьте все значения по умолчанию на соответствие корпоративной модели данных.

Логическая модель данных

Логическая модель данных является визуальным представлением структур данных, их атрибутов и бизнес-правил. Логическая модель представляет данные таким образом, чтобы они легко воспринимались бизнес-пользователями. Проектирование логической модели должно быть свободно от требований платформы и языка реализации или способа дальнейшего использования данных.

Разработчик модели использует требования к данным и результаты анализа для формирования логической модели данных. Разработчик приводит логическую модель к третьей нормальной форме и проверяет ее на соответствие корпоративной модели данных, если она существует. Последующие разделы описывают формирование логической модели данных, приведение логической модели к третьей нормальной форме, обзор корпоративной модели и дают некоторые рекомендации по проверке логической модели на соответствие корпоративной модели.

После сравнения логической и корпоративной моделей данных и внесения всех необходимых изменений важно повторно рассмотреть модель с точки зрения точности и полноты. Для достижения хороших результатов полезно проводить экспертный анализ, а также совместный анализ с бизнес партнерами и командой разработчиков. Анализ логической модели будет детально обсуждаться в одной из последующих статей этой серии.

Последующие разделы описывают логическую и физическую модели данных.

Компоненты логической модели данных

Логическая модель использует сущности, атрибуты и отношения для представления данных и бизнес-правил. ***Сущности*** представляют собой объекты, о которых корпорация заинтересована хранить данные. ***Атрибуты*** - это данные, которые корпорация заинтересована хранить. ***Отношения*** описывают взаимосвязи между сущностями в терминах бизнес-правил.

Сущности

Сущности представляют собой объекты, данные о которых корпорация заинтересована сохранять. Сущностями могут быть вещественные объекты, такие как персона или книга, но они могут представлять и абстрактные концепции, такие как центр затрат или производственная единица. Сущности для ясности и обеспечения целостности обозначаются существительными в единственном числе, например, **Потребитель (CUSTOMER)** а не **Потребители (CUSTOMERS).**

Вы должны описать сущность, используя фактографические подробности, которые уникально ее идентифицируют. Каждый экземпляр сущности должен быть отдельным и отличным от всех других экземпляров этой сущности. Например, модель данных для хранения информации о клиентах должна обеспечивать способ, позволяющий отличить одного клиента от другого.
На рисунке 1.3 представлено несколько примеров сущностей.

Рис. 1.3. Пример использования ERwin для отображения сущностей в простейшей форме.

В одной из последующих публикаций будет представлена детальная информация о сущностях и их типах, а так же об обнаружении, именовании, определении и других аспектах сущностей.

Атрибуты

Атрибуты представляют данные об объектах, которые необходимо иметь корпорации. Атрибуты представляются именами существительными, которые описывают характеристики сущностей.

Рисунок 1.4 иллюстрирует несколько примеров атрибутов.

Рис. 1.4. В качестве атрибутов могут выступать дата рождения клиента, модель автомобиля или код ISBN книги.

В одной из следующих статей будет представлена детальная информация об атрибутах и их типах, а так же об обнаружении, именовании, определении и других аспектах, связанных с атрибутами.

Отношения

Отношения *представляют взаимосвязи между объектами, о которых корпорация заинтересована хранить данные.* ***Отношения*** *выражаются глаголами или глагольными фразами, которые описывают взаимосвязь.* На рис. 1.5 приведено несколько примеров отношений, представленных в нотации IE (Information Engineering - информационная инженерия) системы ERwin.

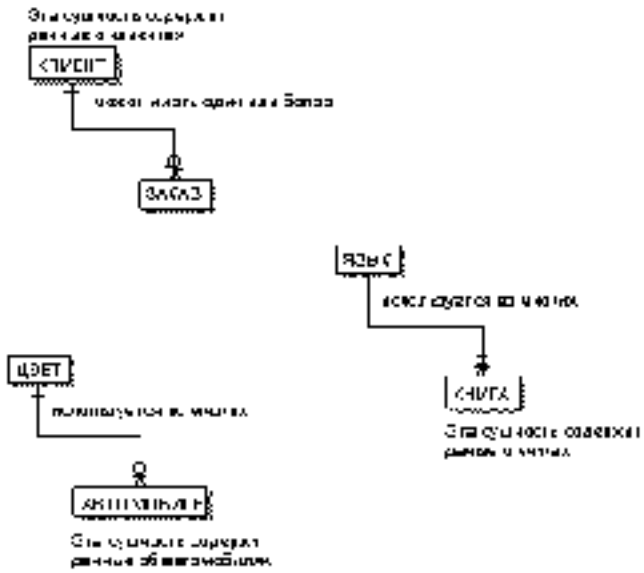


Рис. 1.5. Примеры отношений используютнотацию IE системы ERwin, в которой "коровьи копыта" или "пре зубцы" отображаютмногие с тороны отношений.

Понятие нормализации

Нормализация является операцией перемещения атрибутов в подходящие сущности в соответствие с требованиями нормальных форм. Нормализацию обычно представляют как набор сложных правил, из-за чего эта концепция кажется трудной для понимания. В действительности, нормализация достаточно очевидна: "Один факт в одном месте", как пишет К.Д. Дейт в своей книге ("Введение в системы баз данных, 6-е издание") изд. Вильямс: 1999, 848 с.).

Нормализация данных означает проектирование структур данных таким образом, чтобы удалить избыточность и ограничить несвязанные структуры.

Широко используются пять нормальных форм. Эти формы называются просто: первая нормальная, вторая нормальная, третья нормальная, четвертая нормальная и пятая нормальная формы. На практике многие логические модели приведены только к третьей нормальной форме.

В конце этой статьи приведено формальное определение Дейта для первых пяти нормальных форм, включая нормальную форму Бойса/Кодда (BCNF - Boyce/Codd normal form), которая представляет собой более строго ограниченную третью нормальную форму.

Термин "бизнес-нормальная форма", введенный Клайвом Финклештейном (Finklestein C. An Introduction to Information Engineering: From Strategic Planning to Information Systems. Reading, Massachusetts: Addison-Wesley, 1989), дает менее формальное определение нормальных форм и рекомендаций по приведению моделей к каждой форме. Хотя существуют незначительные различия в способах определения нормальных форм, приводимых разными экспертами, конечный результат будет один - модель, где каждый атрибут "зависит от ключа, полного ключа, и только ключа, да поможет мне Кодд".

В повседневной практике, логическая модель нормализуется до третьей нормальной формы. Ниже приведены простые правила нормализации:

- Размещайте повторяющиеся атрибуты в зависимых сущностях.

- Убедитесь, что каждый факт в модели представлен только один раз.
- Размещайте атрибуты, независящие от первичного ключа, в зависимых сущностях.
- Устраняйте отношения ***многие-ко-многим***

Примеры, использованные ниже для иллюстрации процесса нормализации, начинаются с единственной ненормализованной сущности, которая нарушает требования всех нормальных форм. Затем, применяя простые правила, мы приведем ее к третьей нормальной форме.

Проектирование баз данных с ERwin
Базовые концепции моделирования данных
(Часть 2)

Зайцев С.Л., к.ф.-м.н.

Повторяющиеся группы

Повторяющимися группами являются атрибуты, для которых единственный экземпляр сущности может иметь более одного значения. Например, персона может иметь более одного навыка. Если, с точки зрения требований бизнеса, нам нужно знать уровень владения навыком для каждого, и каждая персона может иметь только два навыка, мы можем создать сущность, показанную на рис. 1.6. Здесь представлена сущность **ПЕРСОНА** с двумя атрибутами для хранения навыков и уровня владения навыками для каждого.

Рис. 1.6. В данном примере используются повторяющиеся группы .

Проблема повторяющихся групп заключается в том, что мы не можем точно знать, сколько навыков может иметь персона. В реальной жизни у некоторых людей есть один навык, у некоторых - несколько, а у некоторых - пока ни одного. На рисунке 1.7 представлена модель, приведенная к первой нормальной форме. Обратите внимание на добавленный ***Идентификатор навыка*** , который уникально определяет каждый **НАВЫК.**

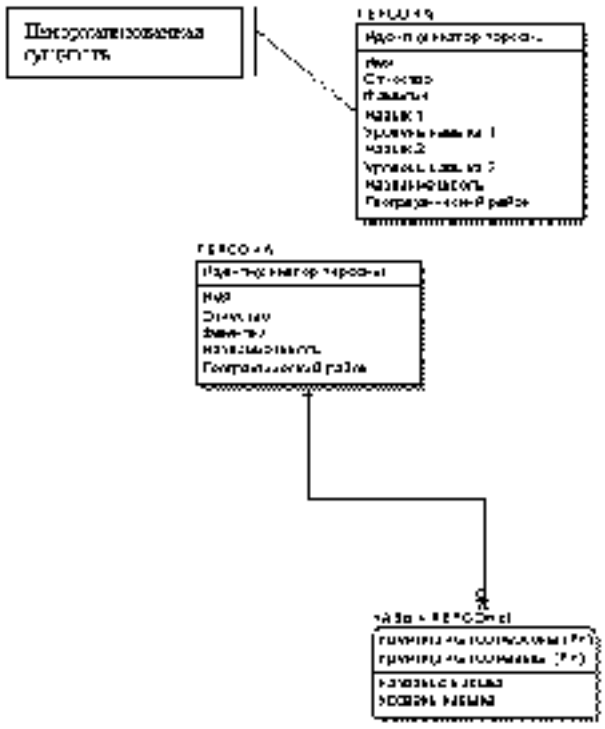


Рис. 1.7. Модель, приведенная к первой нормальной форме.

Один факт в одном месте

Если один и тот же атрибут присутствует более чем в одной сущности и не является внешним ключом, то этот атрибут рассматривается как избыточный. Логическая модель не должна содержать избыточных данных.

Избыточность требует дополнительного пространства, однако, хотя эффективность использования памяти немаловажна, действительная проблема заключается в другом. Гарантированная синхронизация избыточных данных требует накладных расходов, и вы всегда работаете в условиях риска возникновения конфликтных значений.

В предыдущем примере **НАВЫК** зависит от **Идентификатора персоны** и от **Идентификатора навыка.** Это значит, что у вас не появится **НАВЫК** до тех пор, пока не появится **ПЕРСОНА,** обладающая этим навыком. Это так же усложняет изменение Названия навыка. Необходимо найти каждую запись с Названием навыка и изменить ее для каждой Персоны, владеющей этим навыком.

На рисунке 1.8 представлена модель во второй нормальной форме. Заметьте, что добавлена сущность **НАВЫК** , и атрибут **НАЗВАНИЕ** навыка перенесен в эту сущность. Уровень навыка остался, соответственно, на пересечении **ПЕРСОНЫ и НАВЫКА.**

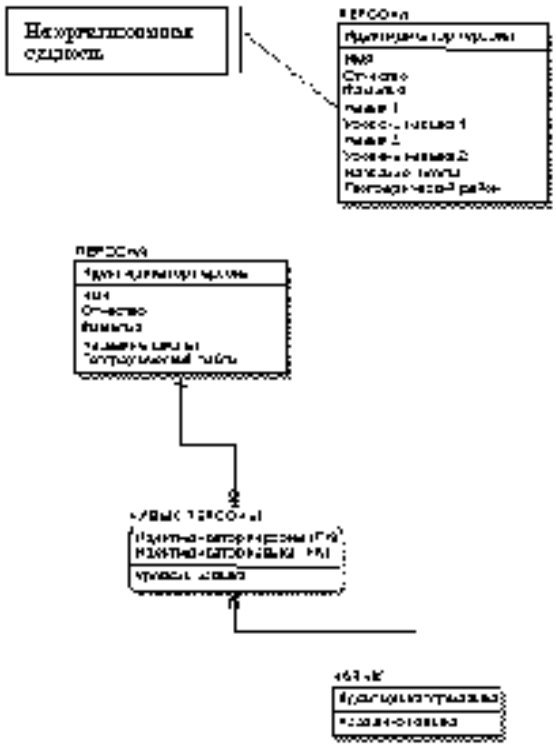


Рис. 1.8. Во второй нормальной форме повторяющаяся группа вынесена в другую сущность. Это обеспечивает гибкость при добавлении необходимого количества Навыков и изменении Названия навыка или Описания навыка в одном месте.

Каждый атрибут зависит от ключа

Каждый атрибут сущности должен зависеть от первичного ключа этой сущности. В предыдущем примере **Название школы** и **Географический район** присутствуют в таблице **ПЕРСОНА**, но не описывают персону. Для достижения третьей нормальной формы необходимо переместить атрибуты в сущность, где они будут зависеть от ключа. Рисунок 1.9. показывает модель в третьей нормальной форме.

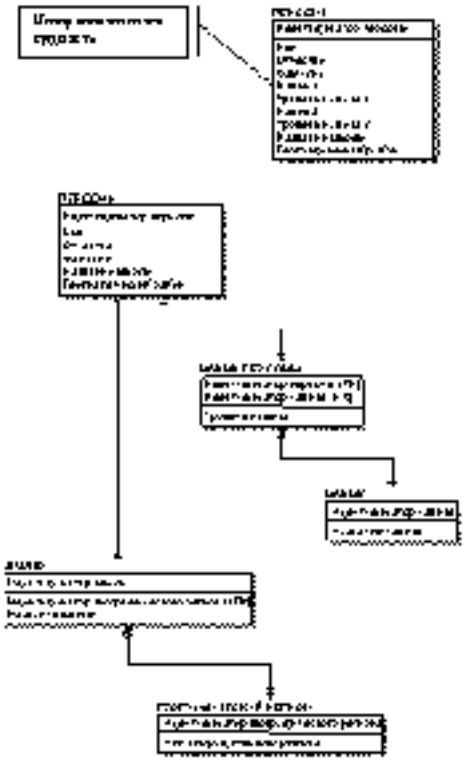


Рис. 1.9. В третьей нормальной форме ***Название школы*** *и* ***Географический регион*** *перенесены в сущность, где их значения зависятотключа.*

Отношения многие-ко-многим

Отношения **многие-ко-многим** отражают реальность окружающего мира. Обратите внимание, что на рисунке 1.9 существует отношение многие-ко-многим между **ПЕРСОНОЙ** и **ШКОЛОЙ** . Отношение точно отражает тот факт, что **ПЕРСОНА** может учиться во многих **ШКОЛАХ** и в **ШКОЛЕ** может учиться много **ПЕРСОН.** Для достижения четвертой нормальной формы создается ассоциативная сущность, которая устраняет отношение моногие-ко-многим за счет формирования отдельной записи для каждой уникальной комбинации школы и персоны. На рисунке 1.10 представлена модель в четвертой нормальной форме.

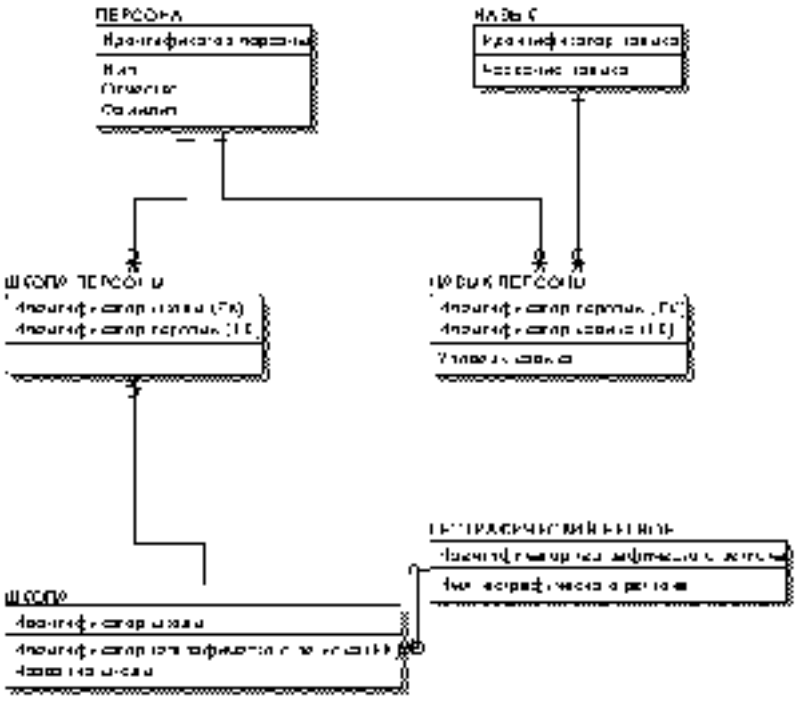


Рис. 1.10. В четвертой нормальной форме отношение многие-ко-многим между ***ПЕРСОНОЙ*** *и* ***ШКОЛОЙ*** *разрешается за счет введения ассоциативной сущности, в которой отводится отдельная запись для каждой уникальной комбинации* ***ШКОЛЫ*** *и* ***ПЕРСОНЫ*** *.*

Формальные определения нормальных форм

Следующие определения нормальных форм могут показаться устрашающими. Рассматривайте их просто как формулы для достижения нормализации. Нормальные формы основаны на реляционной алгебре и могут интерпретироваться как математические преобразования. Хотя эта книга и не посвящена детальному обсуждению нормальных форм, разработчикам моделей рекомендуется более глубоко изучить этот вопрос.

В заданном отношении R атрибут Y функционально зависит от атрибута X. В символьном виде R.X -> R.Y (читается как "R.X функционально определяет R.Y") - в том и только в том случае если каждое значение X в R ассоциируется строго с одним значением Y в R (в каждый конкретный момент времени). Атрибуты X и Y могут быть составными (Дейт К. Дж. Введение в системы баз данных. 6-е издание. Изд. Вильямс: 1999, 848 с.).

Отношение R соответствует первой нормальной форме (1NF) тогда и только тогда, когда все принадлежащие ему домены содержат только атомарные значения (Дейт, там же).

Отношение R соответствует второй нормальной форме (2NF) тогда и только тогда, когда оно соответствует 1NF, и каждый неключевой атрибут полностью зависит от первичного ключа (Дейт, там же).

Отношение R соответствует третьей нормальной форме (3NF) тогда и только тогда, когда оно соответствует 2NF, и каждый неключевой атрибут не транзитивно зависит от первичного ключа (Дейт, там же).

Отношение R соответствует нормальной форме Бойса-Кодда (BCNF) тогда и только тогда, когда каждый детерминант является кандидатом на использование в качестве ключа.

ПРИМЕЧАНИЕ
Ниже приводится краткое объяснение некоторых аббревиатур, используемых в определениях Дейта.

MVD (multi-valued dependency) - многозначная зависимость. Используется только для сущностей с тремя и более атрибутами. При многозначной зависимости значение атрибута зависит только от части первичного ключа.

FD (functional dependency) - функциональная зависимость. При функциональной зависимости значение атрибута зависит от значения другого атрибута, который не является частью первичного ключа.

JD (join dependency) - зависимость по объединению. При зависимости по объединению первичный ключ родительской сущности прослеживается до потомков, по меньшей мере, третьего уровня сохраняя способность использоваться в объединении по исходному ключу.

Отношение соответствует четвертой нормальной форме (4NF) тогда и только тогда, когда в R существует MVD, например A®®B. При этом все атрибуты R функционально зависят от A. Другими словами, в R присутствуют только зависимости (FD или MVD) формы K®X (т.е. функциональная зависимость атрибута X от кандидата на использование в качестве ключа K). Соответственно R отвечает требованиям 4NF, если оно соответствует BCNF и все MVD фактически являются FD (Дейт, там же).

Для пятой нормальной формы отношение R удовлетворяет зависимости по объединению (JD) *(X, Y, …, Z) тогда и только тогда, когда R эквивалентно его проекциям на X, Y,..., Z, где X, Y,..., Z подмножества множества атрибутов R.

Существует много других нормальных форм для сложных типов данных и специфических ситуаций, которые выходят за рамки нашего обсуждения. Каждому энтузиасту разработки моделей желательно было бы изучить и другие нормальные формы.

Бизнес-нормальные формы

В своей книге Клайв Финклештейн (Finklestein Cl. An Introduction to Information Engineering: From Strategic Planning to Information Systems. Reading, Massachusetts: Addison-Wesley, 1989) применил другой подход к нормализации. Он определяет бизнес-нормальные формы в терминах приведения к этим формам. Многие разработчики моделей, считают этот подход интуитивно более ясным и прагматичным.

Первая бизнес-нормальная форма (1BNF) выносит повторяющиеся группы в другую сущность. Эта сущность получает собственное имя и первичные (составные) ключевые атрибуты из исходной сущности и ее повторяющейся группы.

Вторая бизнес-нормальная форма (2BNF) выносит атрибуты, которые частично зависят от первичного ключа в другую сущность. Первичный (составной) ключ этой сущности является первичным ключом сущности, в которой он исходно находился, вместе с дополнительными ключами, от которых атрибут полностью зависит.

Третья бизнес-нормальная форма (3BNF) выносит атрибуты, не зависящие от первичного ключа, в другую сущность, где они полностью зависят от первичного ключа этой сущности.

Четвертая бизнес-нормальная форма (4BNF) выносит атрибуты, которые зависят от значения первичного ключа или являются необязательными, во вторичную сущность, где они полностью зависят от значения первичного ключа или где они должны (обязательно) присутствовать в этой сущности.

Пятая бизнес-нормальная форма (5BNF) проявляется как структурная сущность, если есть рекурсивная или другая зависимость между экземплярами вторичной сущности, или если рекурсивная зависимость существует между экземплярами ее первичной сущности.

Завершенная логическая модель данных

Завершенная логическая модель должна удовлетворять требованиям третьей бизнес-нормальной формы и включать все сущности, атрибуты и связи, необходимые для поддержки требований к данным и бизнес-правил, ассоциированных с данными.

Все сущности должны иметь имена, описывающие содержание и ясное, краткое, полное описание или определение. В одной из следующих публикаций будет рассмотрен исходный набор рекомендаций для правильного формирования имен и описаний сущностей.

Сущности должны иметь полный набор атрибутов, так, чтобы каждый факт относительно каждой сущности мог быть представлен ее атрибутами. Каждый атрибут должен иметь имя, отражающее его значения, логический тип данных и ясное, короткое, полное описание или определение. В одной из следующих публикаций мы рассмотрим исходный набор рекомендаций для правильного формирования имен и описаний атрибутов.

Связи должны включать глагольную конструкцию, которая описывает отношение между сущностями, наравне с такими характеристиками, как множественность, необходимость существования или возможность отсутствия связи.

> **ПРИМЕЧАНИЕ**
> **Множественность** связи описывает максимальное число экземпляров вторичной сущности, которые могут быть связаны с экземпляром исходной сущности.
> **Необходимость существования** или **возможность отсутствия** связи служит для определения минимального числа экземпляров вторичной сущности, которые могут быть связаны с экземпляром исходной сущности.

Физическая модель данных

После создания полной и адекватной логической модели вы готовы к принятию решения о выборе платформы реализации. Выбор платформы зависит от требований к использованию данных и стратегических принципов формирования архитектуры корпорации. Выбор платформы - сложная проблема, выходящая за рамки данной книги.

В ERwin физическая модель является графическим представлением реально реализованной базы данных. Физическая база данных будет состоять из таблиц, столбцов и связей. Физическая модель зависит от платформы, выбранной для реализации, и требований к использованию данных. Физическая модель для IMS будет серьезно отличаться от такой же модели для Sybase. Физическая модель для OLAP-отчетов будет выглядеть иначе, чем модель для OLTP (оперативной обработки транзакций).

Разработчик модели данных и администратор базы данных (DBA - database administrator) используют логическую модель, требования к использованию и стратегические принципы

формирования архитектуры корпорации для разработки физической модели данных. Вы можете денормализовать физическую модель для улучшения производительности, и создать представления для поддержки требований к использованию. В последующих разделах детально рассматривается процесс денормализации и создания представлений.

В этом разделе приведен обзор процесса построения физической модели, сбора требований к использованию данных, дано определение компонентов физической модели и обратного проектирования. В следующих публикациях эти вопросы освещены более подробно.

Сбор требований к использованию данных

Обычно вы собираете требования к использованию данных на ранних стадиях в ходе интервью и рабочих сессий. При этом требования должны максимально полно определять использование данных пользователем. Поверхностное отношение и лакуны в физической модели могут привести к внеплановым затратам и затягиванию сроков реализации проекта. Требования к использованию включают:

- Требования к доступу и производительности
- Волюметрические характеристики (оценку объема данных, которые предстоит хранить), которые позволяют администратору представить физический объем базы данных
- Оценка количества пользователей, которым необходим одновременный доступ к данным, которая помогает вам проектировать базу данных с учетом приемлемого уровня производительности
- Суммарные, сводные и другие вычисляемые или производные данные, которые могут рассматриваться в качестве кандидатов для хранения в долговременных структурах данных
- Требования к формированию отчетов и стандартных запросов, помогающих администратору базы данных формировать индексы
- Представления (долговременные или виртуальные), которые будут помогать пользователю при выполнении операций объединения или фильтрации данных.

Кроме председателя, секретаря и пользователей в сессии, посвященной требованиям к использованию, должны принимать участие разработчик модели, администратор базы данных и архитектор базы данных. Обсуждению должны подвергаться требования пользователя к историческим данным. Длительность периода времени, в течение которого хранятся данные, оказывает значительное влияние на размер базы данных. Часто более старые данные хранятся в обобщенном виде, а атомарные данные архивируются или удаляются.

Пользователям следует принести с собой на сессию примеры запросов и отчетов. Отчеты должны быть строго определены и должны включать атомарные значения, использующиеся для любых суммарных и сводных полей.

Компоненты физической модели данных

Компонентами физической модели данных являются таблицы, столбцы и отношения. Сущности логической модели, вероятно, станут таблицами в физической модели. Логические атрибуты станут столбцами. Логические отношения станут ограничениями целостности связей.
Некоторые логические отношения невозможно реализовать в физической базе данных.

Обратное проектирование

Когда логическая модель недоступна, возникает необходимость воссоздания модели из существующей базы данных. В ERwin этот процесс называется обратным проектированием. Обратное проектирование может производиться несколькими способами. Разработчик модели может исследовать структуры данных в базе данных и воссоздать таблицы в визуальной среде моделирования. Вы можете импортировать язык описания данных (DDL - data definitions language) в инструмент, который поддерживает проведение обратного проектирования (например, Erwin). Развитые средства, такие как ERwin, включают функции, обеспечивающие связь через ODBC с существующей базой данных, для создания модели путем прямого чтения структур данных. Обратное проектирование с использованием ERwin будет подробно обсуждаться в одной из последующих публикаций.

Использование корпоративных функциональных границ

При построении логической модели для разработчика модели важно убедиться, что новая модель соответствует корпоративной модели. Использование корпоративных функциональных границ означает моделирование данных в терминах, использующихся в рамках корпорации. Способ использования данных в корпорации изменяется быстрее, чем сами данные. В каждой логической модели данные должны быть представлены целостно, не зависимо от предметной области бизнеса, которую она поддерживает. Сущности, атрибуты и отношения должны определять бизнес-правила на уровне корпорации.

> **ПРИМЕЧАНИЕ**
> Некоторые из моих коллег называют эти корпоративные функциональные границы моделированием реального мира. Моделирование реального мира побуждает разработчика модели рассматривать информацию в терминах реально присущих ей отношений и взаимосвязей.

Использование корпоративных функциональных границ для модели данных, построенной соответствующим образом, обеспечивает основу поддержки информационных нужд любого числа процессов и приложений, что дает возможность корпорации эффективнее эксплуатировать один из ее наиболее ценных активов - информацию.

Что такое корпоративная модель данных?

Корпоративная модель данных (EDM - enterprise data model) содержит сущности, атрибуты и отношения, которые представляют информационные потребности корпорации. EDM обычно подразделяется в соответствие с предметными областями, которые представляют группы сущностей, относящихся к поддержке конкретных нужд бизнеса. Некоторые предметные области могут покрывать такие специфические бизнес-функции, как управление контрактами, другие - объединять сущности, описывающие продукты или услуги.

Каждая логическая модель должна соответствовать существующей предметной области корпоративной модели данных. Если логическая модель не соответствует данному требованию, в нее должна быть добавлена модель, определяющая предметную область. Это сравнение гарантирует, что корпоративная модель улучшена или скорректирована, и в рамках корпорации скоординированы все усилия по логическому моделированию.

EDM также включает специфические сущности, которые определяют область определения значений для ключевых атрибутов. Эти сущности не имеют родителей и определяются как независимые. Независимые сущности часто используются для поддержания целостности связей. Эти сущности идентифицируются несколькими различными именами, такими как кодовые таблицы, таблицы ссылок, таблицы типов или классификационные таблицы. Мы будем использовать термин "корпоративный бизнес-объект". Корпоративный бизнес-объект это сущность, которая содержит набор значений атрибутов, не зависящих ни от какой другой сущности. Корпоративные бизнес-объекты в рамках корпорации следует использовать единообразно.

Построение корпоративной модели данных путем наращивания

Существуют организации, где корпоративная модель от начала до конца была построена в результате единых согласованных усилий. С другой стороны, большинство организаций создают достаточно полные корпоративные модели путем наращивания.

Наращивание означает построение чего-либо последовательно, слой за слоем, подобно тому, как устрица выращивает жемчужину. Каждая созданная модель данных обеспечивает вклад в формирование EDM. Построение EDM этим способом требует выполнения дополнительных действий моделирования для добавления новых структур данных и предметных областей или расширения существующих структур данных. Это дает возможность строить корпоративную модель данных путем наращивания, итеративно добавляя уровни детализации и уточнения.

Понятие методологии моделирования

Существует несколько методологий визуального моделирования данных. ERwin поддерживает две:

- IDEF1X (Integration Definition for Information Modeling - интегрированное описание информационных моделей).
- IE (Information Engineering - информационная инженерия).

IDEF1X - хорошая методология и использование ее нотации широко распространено

Интегрированное описание информационных моделей

IDEF1X- высоко структурированная методология моделирования данных, расширяющая методологию IDEF1, принятую в качестве стандарта FIPS (Federal Information Processing Standards - федеральный орган стандартов обработки информации). IDEF1X использует строго структурированный набор типов конструкций моделирования и приводит к модели данных, которая требует понимания физической природы данных до того, как такая информация может стать доступной.

Жесткая структура IDEF1X принуждает разработчика модели назначать сущностям характеристики, которые могут не отвечать реалиям окружающего мира. Например, IDEF1X требует, чтобы все подтипы сущностей были эксклюзивными. Это приводит к тому, что персона не может быть одновременно клиентом и сотрудником. В то время как реальная практика говорит нам другое.

Разработчикам моделей рекомендуется изучить IDEF1X и сформировать собственное мнение

о ней. Документы FIPS доступны в Web, существует несколько хороших публикаций по IDEF1X.

Информационный инжиниринг

Клайва Финклештейна часто называют отцом информационного инжиниринга, хотя подобные же концепции излагал вместе с ним и Джеймс Мартин (Martin, James. Managing the Database Environment. Upper Saddle River, New Jersey: Prentice Hall, 1983.). Информационный инжиниринг использует для управления информацией подход, направляемый бизнесом, и применяет другую нотацию для представления бизнес-правил. IE служит расширением и развитием нотации и базовых концепций методологии ER, предложенной Питером Ченом.

IE обеспечивает инфраструктуру поддержки требований к информации путем интеграции корпоративного стратегического планирования с разрабатываемыми информационными системами. Подобная интеграция позволяет более тесно увязать управление информационными ресурсами с долговременными стратегическими перспективами корпорации. Этот подход, направляемый требованиями бизнеса, приводит многих разработчиков моделей к выбору IE вместо других методологий, которые, в основном, концентрируют внимание на решении сиюминутных задач разработки.

IE предлагает последовательность действий, приводящую корпорацию к определению всех своих информационных потребностей по сбору и управлению данными и выявлению взаимосвязей между информационными объектами. В результате, требования к информации ясно формулируются на основе директив управления и могут быть непосредственно переведены в информационную систему управления, которая будет поддерживать стратегические потребности в информации.

Разработчикам моделей рекомендуется познакомиться с работами Джеймса Мартина и Клайва Финклештейна для более глубокого понимания подхода IE к управлению информационными ресурсами.

Заключение

Понимание того, как пользоваться инструментом моделирования данных, подобным ERwin, составляет только часть проблемы. Кроме этого вы должны понимать, когда решаются задачи моделирования данных и как осуществляется сбор требований к информации и бизнес-правил, которые должны быть представлены в модели данных. Проведение рабочих сессий обеспечивает наиболее благоприятные условия для сбора требований к информации в среде, включающей экспертов предметной области, пользователей и специалистов в области информационных технологий.

Для построения хорошей модели данных требуется анализ и исследование требований к информации и бизнес-правилам, собранных в ходе рабочих сессий и интервью. Результирующую модель данных необходимо сравнить с корпоративной моделью, если это возможно, для гарантии того, что она не конфликтует с существующими моделями объектов и включает в себя все необходимые объекты.

Модель данных состоит из логической и физической моделей, отображающих требования к информации и бизнес-правила. Логическая модель должна быть приведена к третьей нормальной форме. Третья нормальная форма ограничивает, добавляет, обновляет и удаляет

аномалии структур данных для поддержки принципа "один факт в одном месте". Собранные требования к информации и бизнес-правила должны быть проанализированы и исследованы. Их необходимо сравнить с корпоративной моделью для гарантии того, что они не конфликтует с существующими моделями объектов, и включают в себя все необходимые объекты.

В ERwin модель данных включает как логическую, так и физическую модели. ERwin реализует подход ER и позволяет вам создавать объекты логических и физических моделей для представления требований к информации и бизнес-правил. Объекты логической модели включают сущности, атрибуты и отношения. К объектам физической модели относятся таблицы, столбцы и ограничения целостности связей.

В одной из следующих публикаций будут рассмотрены вопросы идентификации сущностей, определения типов сущностей, выбора имен сущностей и описаний, а так же некоторые приемы, позволяющие избежать наиболее распространенных ошибок моделирования, связанных с использованием сущностей.

Сущности должны иметь полный набор атрибутов, так, чтобы каждый факт относительно каждой сущности мог быть представлен ее атрибутами. Каждый атрибут должен иметь имя, отражающее его значения, логический тип данных и ясное, короткое, полное описание или определение. В одной из следующих публикаций мы рассмотрим исходный набор рекомендаций для правильного формирования имен и описаний атрибутов.
Связи должны включать глагольную конструкцию, которая описывает отношение между сущностями, наравне с такими характеристиками, как множественность, необходимость существования или возможность отсутствия связи.

> **ПРИМЕЧАНИЕ**
> Множественность связи описывает максимальное число экземпляров вторичной сущности, которые могут быть связаны с экземпляром исходной сущности. Необходимость существования или возможность отсутствия связи служит для определения минимального числа экземпляров вторичной сущности, которые могут быть связаны с экземпляром исходной сущности.

Проектирование баз данных с ERwin Базовые концепции моделирования данных (Часть 3)

Зайцев С.Л., к.ф.-м.н.

Физическая модель данных

После создания полной и адекватной логической модели вы готовы к принятию решения о выборе платформы реализации. Выбор платформы зависит от требований к использованию данных и стратегических принципов формирования архитектуры корпорации. Выбор платформы - сложная проблема, выходящая за рамки данной книги.

В ERwin физическая модель является графическим представлением реально реализованной базы данных. Физическая база данных будет состоять из таблиц, столбцов и связей. Физическая модель зависит от платформы, выбранной для реализации, и требований к использованию данных. Физическая модель для IMS будет серьезно отличаться от такой же

модели для Sybase. Физическая модель для OLAP-отчетов будет выглядеть иначе, чем модель для OLTP (оперативной обработки транзакций).

Разработчик модели данных и администратор базы данных (DBA - database administrator) используют логическую модель, требования к использованию и стратегические принципы формирования архитектуры корпорации для разработки физической модели данных. Вы можете денормализовать физическую модель для улучшения производительности, и создать представления для поддержки требований к использованию. В последующих разделах детально рассматривается процесс денормализации и создания представлений.

В этом разделе приведен обзор процесса построения физической модели, сбора требований к использованию данных, дано определение компонентов физической модели и обратного проектирования. В следующих публикациях эти вопросы освещены более подробно.

Сбор требований к использованию данных

Обычно вы собираете требования к использованию данных на ранних стадиях в ходе интервью и рабочих сессий. При этом требования должны максимально полно определять использование данных пользователем. Поверхностное отношение и лакуны в физической модели могут привести к внеплановым затратам и затягиванию сроков реализации проекта.

Требования к использованию включают:

- Требования к доступу и производительности
- Волюметрические характеристики (оценку объема данных, которые предстоит хранить), которые позволяют администратору представить физический объем базы данных
- Оценка количества пользователей, которым необходим одновременный доступ к данным, которая помогает вам проектировать базу данных с учетом приемлемого уровня производительности
- Суммарные, сводные и другие вычисляемые или производные данные, которые могут рассматриваться в качестве кандидатов для хранения в долговременных структурах данных
- Требования к формированию отчетов и стандартных запросов, помогающих администратору базы данных формировать индексы
- Представления (долговременные или виртуальные), которые будут помогать пользователю при выполнении операций объединения или фильтрации данных.

Кроме председателя, секретаря и пользователей в сессии, посвященной требованиям к использованию, должны принимать участие разработчик модели, администратор базы данных и архитектор базы данных. Обсуждению должны подвергаться требования пользователя к историческим данным. Длительность периода времени, в течение которого хранятся данные, оказывает значительное влияние на размер базы данных. Часто более старые данные хранятся в обобщенном виде, а атомарные данные архивируются или удаляются.

Пользователям следует принести с собой на сессию примеры запросов и отчетов. Отчеты должны быть строго определены и должны включать атомарные значения, использующиеся для любых суммарных и сводных полей.

Компоненты физической модели данных

Компонентами физической модели данных являются таблицы, столбцы и отношения. Сущности логической модели, вероятно, станут таблицами в физической модели. Логические атрибуты станут столбцами. Логические отношения станут ограничениями целостности связей.

Некоторые логические отношения невозможно реализовать в физической базе данных.

Обратное проектирование

Когда логическая модель недоступна, возникает необходимость воссоздания модели из существующей базы данных. В ERwin этот процесс называется обратным проектированием. Обратное проектирование может производиться несколькими способами. Разработчик модели может исследовать структуры данных в базе данных и воссоздать таблицы в визуальной среде моделирования. Вы можете импортировать язык описания данных (DDL - data definitions language) в инструмент, который поддерживает проведение обратного проектирования (например, ERwin). Развитые средства, такие как ERwin, включают функции, обеспечивающие связь через ODBC с существующей базой данных, для создания модели путем прямого чтения структур данных. Обратное проектирование с использованием ERwin будет подробно обсуждаться в одной из последующих публикаций.

Использование корпоративных функциональных границ

При построении логической модели для разработчика модели важно убедиться, что новая модель соответствует корпоративной модели. Использование корпоративных функциональных границ означает моделирование данных в терминах, использующихся в рамках корпорации. Способ использования данных в корпорации изменяется быстрее, чем сами данные. В каждой логической модели данные должны быть представлены целостно, не зависимо от предметной области бизнеса, которую она поддерживает. Сущности, атрибуты и отношения должны определять бизнес-правила на уровне корпорации.

> ПРИМЕЧАНИЕ
> Некоторые из моих коллег называют эти корпоративные функциональные границы моделированием реального мира. Моделирование реального мира побуждает разработчика модели рассматривать информацию в терминах реально присущих ей отношений и взаимосвязей.

Использование корпоративных функциональных границ для модели данных, построенной соответствующим образом, обеспечивает основу поддержки информационных нужд любого числа процессов и приложений, что дает возможность корпорации эффективнее эксплуатировать один из ее наиболее ценных активов - информацию.

Что такое корпоративная модель данных?

Корпоративная модель данных (EDM - enterprise data model) содержит сущности, атрибуты и отношения, которые представляют информационные потребности корпорации. EDM обычно подразделяется в соответствие с предметными областями, которые представляют группы

сущностей, относящихся к поддержке конкретных нужд бизнеса. Некоторые предметные области могут покрывать такие специфические бизнес-функции, как управление контрактами, другие - объединять сущности, описывающие продукты или услуги.

Каждая логическая модель должна соответствовать существующей предметной области корпоративной модели данных. Если логическая модель не соответствует данному требованию, в нее должна быть добавлена модель, определяющая предметную область. Это сравнение гарантирует, что корпоративная модель улучшена или скорректирована, и в рамках корпорации скоординированы все усилия по логическому моделированию.

EDM также включает специфические сущности, которые определяют область определения значений для ключевых атрибутов. Эти сущности не имеют родителей и определяются как независимые. Независимые сущности часто используются для поддержания целостности связей. Эти сущности идентифицируются несколькими различными именами, такими как кодовые таблицы, таблицы ссылок, таблицы типов или классификационные таблицы. Мы будем использовать термин "корпоративный бизнес-объект". Корпоративный бизнес-объект это сущность, которая содержит набор значений атрибутов, не зависящих ни от какой другой сущности. Корпоративные бизнес-объекты в рамках корпорации следует использовать единообразно.

Построение корпоративной модели данных путем наращивания

Существуют организации, где корпоративная модель от начала до конца была построена в результате единых согласованных усилий. С другой стороны, большинство организаций создают достаточно полные корпоративные модели путем наращивания.

Наращивание означает построение чего-либо последовательно, слой за слоем, подобно тому, как устрица выращивает жемчужину. Каждая созданная модель данных обеспечивает вклад в формирование EDM. Построение EDM этим способом требует выполнения дополнительных действий моделирования для добавления новых структур данных и предметных областей или расширения существующих структур данных. Это дает возможность строить корпоративную модель данных путем наращивания, итеративно добавляя уровни детализации и уточнения.

Понятие методологии моделирования

Существует несколько методологий визуального моделирования данных. ERwin поддерживает две:

- IDEF1X (Integration Definition for Information Modeling - интегрированное описание информационных моделей).
- IE (Information Engineering - информационная инженерия).

DEF1X - хорошая методология и использование ее нотации широко распространено.

Интегрированное описание информационных моделей

IDEF1X - высоко структурированная методология моделирования данных, расширяющая методологию IDEF1, принятую в качестве стандарта FIPS (Federal Information Processing Standards - федеральный орган стандартов обработки информации). IDEF1X использует строго структурированный набор типов конструкций моделирования и приводит к модели

данных, которая требует понимания физической природы данных до того, как такая информация может стать доступной.

Жесткая структура IDEF1X принуждает разработчика модели назначать сущностям характеристики, которые могут не отвечать реалиям окружающего мира. Например, IDEF1X требует, чтобы все подтипы сущностей были эксклюзивными. Это приводит к тому, что персона не может быть одновременно клиентом и сотрудником. В то время как реальная практика говорит нам другое.

Разработчикам моделей рекомендуется изучить IDEF1X и сформировать собственное мнение о ней. Документы FIPS доступны в Web, существует несколько хороших публикаций по IDEF1X.

Информационный инжиниринг

К. Финклештейна часто называют отцом информационного инжиниринга, хотя подобные же концепции излагал вместе с ним и Д. Мартин (Martin, James. Managing the Database Environment. Upper Saddle River, New Jersey: Prentice Hall, 1983.). Информационный инжиниринг использует для управления информацией подход, направляемый бизнесом, и применяет другую нотацию для представления бизнес-правил. IE служит расширением и развитием нотации и базовых концепций методологии ER, предложенной П. Ченом.

IE обеспечивает инфраструктуру поддержки требований к информации путем интеграции корпоративного стратегического планирования с разрабатываемыми информационными системами. Подобная интеграция позволяет более тесно увязать управление информационными ресурсами с долговременными стратегическими перспективами корпорации. Этот подход, направляемый требованиями бизнеса, приводит многих разработчиков моделей к выбору IE вместо других методологий, которые, в основном, концентрируют внимание на решении сиюминутных задач разработки.

IE предлагает последовательность действий, приводящую корпорацию к определению всех своих информационных потребностей по сбору и управлению данными и выявлению взаимосвязей между информационными объектами. В результате, требования к информации ясно формулируются на основе директив управления и могут быть непосредственно переведены в информационную систему управления, которая будет поддерживать стратегические потребности в информации.

Разработчикам моделей рекомендуется познакомиться с работами Д. Мартина и К.Финклештейна для более глубокого понимания подхода IE к управлению информационными ресурсами.

Заключение

Понимание того, как пользоваться инструментом моделирования данных, подобным ERwin, составляет только часть проблемы. Кроме этого вы должны понимать, когда решаются задачи моделирования данных и как осуществляется сбор требований к информации и бизнес-правил, которые должны быть представлены в модели данных. Проведение рабочих сессий обеспечивает наиболее благоприятные условия для сбора требований к информации в среде, включающей экспертов предметной области, пользователей и специалистов в области информационных технологий.

Для построения хорошей модели данных требуется анализ и исследование требований к информации и бизнес-правилам, собранных в ходе рабочих сессий и интервью. Результирующую модель данных необходимо сравнить с корпоративной моделью, если это возможно, для гарантии того, что она не конфликтует с существующими моделями объектов и включает в себя все необходимые объекты.

Модель данных состоит из логической и физической моделей, отображающих требования к информации и бизнес-правила. Логическая модель должна быть приведена к третьей нормальной форме. Третья нормальная форма ограничивает, добавляет, обновляет и удаляет аномалии структур данных для поддержки принципа "один факт в одном месте". Собранные требования к информации и бизнес-правила должны быть проанализированы и исследованы. Их необходимо сравнить с корпоративной моделью для гарантии того, что они не конфликтует с существующими моделями объектов, и включают в себя все необходимые объекты.

В ERwin модель данных включает как логическую, так и физическую модели. ERwin реализует подход ER и позволяет вам создавать объекты логических и физических моделей для представления требований к информации и бизнес-правил. Объекты логической модели включают сущности, атрибуты и отношения. К объектам физической модели относятся таблицы, столбцы и ограничения целостности связей.

В одной из следующих публикаций будут рассмотрены вопросы идентификации сущностей, определения типов сущностей, выбора имен сущностей и описаний, а так же некоторые приемы, позволяющие избежать наиболее распространенных ошибок моделирования, связанных с использованием сущностей.