Лекция № 8. Визуальное моделирование баз данных.

В этой лекции рассказывается о моделировании схем баз данных с помощью модели сущность-связь. Рассматриваются разные виды схем – концептуальная, логическая и физическая, затрагиваются вопросы автоматической генерации кода, возвратной инженерии и эволюции схем. Приводится сквозной пример фрагмента приложения баз данных, автоматизирующего работу факультета университета. Пример физической модели реализован в Microsoft Visual Studio с целевой платформой – Microsoft SQL Server.

Схемы данных и модельно-ориентированный подход. Приложения баз данных – одни из самых распространенных программных систем. Электронная форма хранения, учет и обработка различной информации стали неотъемлемой частью бизнеса, делопроизводства, библиотечного и музейного дела и т.д. Данные в таких системах хранятся по многу лет, активно изменяются и используются. В связи с этим предъявляются высокие требования к их структуре.

Итак, структура приложения баз данных должна адекватно отражать предметную область и обеспечивать удобство обработки данных теми программными приложениями, которые с ними работают. Здесь важны многие параметры – скорость доступа, надежность обработки и хранения, удобство эволюции структуры данных и т.д. Из этого следует, что структура баз данных должна тщательно проектироваться.

При этом важным оказывается, что хороший результат не достигается «за один присест» – сели и спроектировали хорошую структуру. Здесь оказывается применимой все та же идея, которая положена в основу RUP/USDP – разработку ПО удобно проводить как процесс создания уточняющихся моделей. Именно так мы «добираемся» от предметной области до работающего ПО.

Модель сущность-связь. Итак, при создании структур баз данных принято использовать моделирование, а не сразу писать код, например, на SQL/DLL. Общепринятым способом моделирования структуры данных является модель сущность-связь, предложенная Петером Ченом еще в 1976 году [1].

Сущность (entity) – это "предмет" рассматриваемой предметной области, который может быть идентифицирован некоторым способом, отличающим его от других "предметов". Конкретные человек, компания или событие являются примерами сущности.

Связь (relationship) – это некоторое отношение между двумя и более сущностями, отражающее, что эти сущности участвуют в общей деятельности, взаимодействуют друг с другом, совместно используются некоторой другой сущностью и т.д.

Пример сущностей и связи представлен на рис. 8.1 (а): две сущности – «Студент» и «Кафедра» – связаны отношением «Принадлежит», еще точнее будет сказать, что студент принадлежит кафедре. Это пример направленного отношения между двумя сущностями.

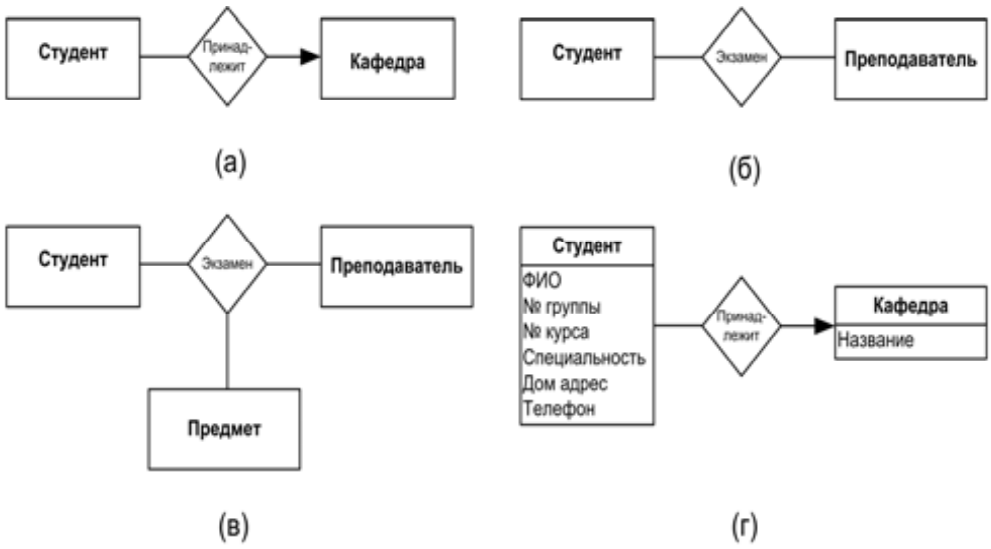

Рис. 8.1.

На рис 8.1. (б) показан пример равноправного отношения между двумя сущностями: преподаватель и студент связаны отношением «Экзамен». На рис. 8.1 (в) показан пример того, как в одном отношении могут участвовать более чем одна сущность.

Необходимо отметить, что модель сущность-связь может показывать как абстрактные сущности, которые мы извлекли из предметной области, анализируя ее в начале процесса создания схемы данных, так и наборы сущностей (типы) и отношения между этими наборами. То есть «Студент», «Преподаватель» и «Предмет» могут быть могут быть как некоторыми абстрактными сущностями, созданными в нашем воображении, а могут определенным образом группировать, классифицировать информацию предметной области, то есть описывать всех студентов, преподавателей, кафедры и экзамены. Для того, чтобы эти описания были осмыслены, типам сущностей приписываются атрибуты, группирующие возможные значения, которыми могут обладать сущности, как показано на рис. 8.1 (г).

Итак, еще одним элементом модели сущность-связь является *атрибут* (attribute) – это функция, отображающая набор сущностей (тип сущностей) в набор значений или декартово произведение наборов значений (в случае сложного атрибута, как например, ФИО у студента на рис 8.1. (г)). На рис. 8.1. (г) показано, какие атрибуты имеют сущности «Студент» и «Кафедра». Например, ФИО – это набор из трех сток на русском языке, представляющих фамилию, имя и отчество студента, номер курса имеет диапазон значений от 1 до 6, атрибут специальность также имеет перечислимы список значений и т.д.

Необходимо отметить, что переход от абстрактных сущностей к типам сущностей означает, что мы переходим от изучения и понимания предметной области (с попутным созданием небольших фрагментов ее описания в виде диаграмм сущность-связь) к формализации предметной области – созданию единой связной модели, исчерпывающе описывающей всю необходимую информацию на определенном уровне абстракции.

Несмотря на простоту модели сущность-связь, она оказалась мощным инструментом при моделировании баз данных, поскольку ее нотация проста в понимании разными специалистами и может служить мостом между аналитиками и заказчиком, экспертами предметной области, для которой создается система, а также ее будущими пользователями. В том виде, в каком эту модель определил Петер Чен [1], к настоящему моменту она не используется – создано большое количество различных нотаций расширяющих и уточняющих эту модель, например, IDEF1x [2]. Кроме того, многие виды диаграмм UML, не имеющие отношения к моделированию схем баз данных (диаграммы развертывания, диаграммы компонент, диаграммы классов, диаграммы объектов и т.д.), фактически, основываются на модели сущность-связь, предлагая разработчикам ПО создавать специальные типы сущностей, их атрибуты и связи. Наконец, отметим, что диаграммы классов UML могут быть с успехом использоваться для моделирования схем баз данных (реляционных, объектно-ориентированных, постреляционных и т.д.) [6,4].

В примерах, которые мы будем рассматривать ниже, мы используем модель сущность-связь на основе диаграмм классов UML. Классы у нас оказываются типами сущностей, их атрибуты – атрибутами типов сущностей, а ассоциации – связями.

Об уровнях абстракции при моделировании данных. Мы рассмотрим следующие уровни описания данных (см. рис. 8.2):

- концептуальная модель – помогает нам выявить основные сущности и связи предметной области, служит средством извлечения знаний из специалистов предметной области и не содержит модельных конструкций, которые эксперты могут не понять; таким образом, эта модель, следуя RUP/USDLP, является моделью уровня разработки требований;
- логическая модель – предназначена уже программистам, содержит различные модельные абстракции, которые могут быть непонятны экспертам предметной области; эта модель служит для уточнения информации о предметной области в виде, удобном для последующей реализации; следуя RUP/USDP, соответствует модели анализа-проектирования;
- физическая модель – описание структуры данных в терминах платформы реализации, содержит информацию об индексах, ключах, типы данных атрибутов определены в терминах целевого языка программирования и т.д.; в терминах RUP/USDP соответствует модели позднего проектирования/реализации; фактически является диаграммным представлением части программного кода, определяющего схему данных;
- полная спецификация структуры данных – спецификация схемы данных на языке программирования, например, SQL/DDL: содержит значения записей по умолчанию, определяет права на таблицы и группы таблиц, кластеризацию и т.д. – то есть информацию, которая отсутствует в физической модели;
- «живая» структура данных – результат исполнения программы (SQL/DDL-скрипта) СУБД (системой управления базой данных – MS Access, MA SQL Server, Oracle и т.д.), генерирующей схему данных для того, чтобы наше приложение могло с ней работать – заносить данные туда и брать их оттуда на обработку данные от туда с выполнением транзакций, разделенного доступа и т.д.; кроме обработки периода исполнения СУБД поддерживает хранение данных в этой структуре после окончания работы приложения – это свойство данных обычно называют персистентностью (persistent).

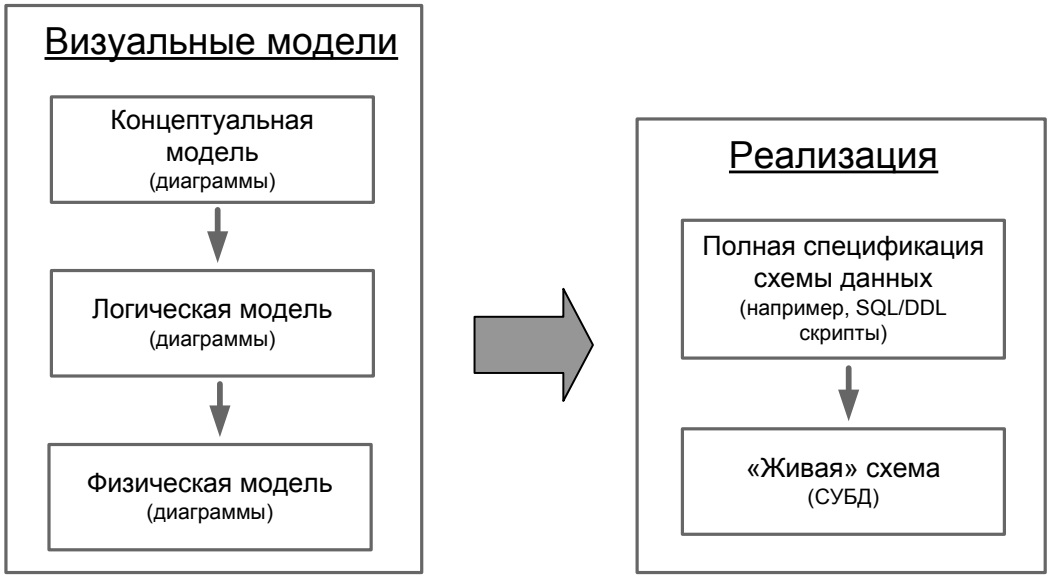

Рис. 8.2

Пример концептуальной модели. Теперь перейдем к примеру. Представим, что мы создаем схему данных для приложения, автоматизирующего работу факультетов университета. Фрагмент концептуальной модели представлен на рис. 8.3.

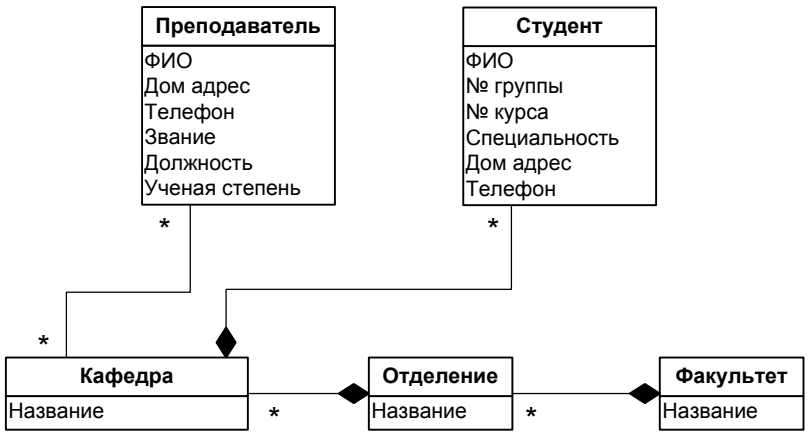

Рис. 8.3.

Анализируя эту предметную область, мы выделили следующие сущности[1]: «Студент», «Преподаватель», «Кафедра», «Отделение» и «Факультет». При этом мы показываем их отношения и атрибуты, для отношений обозначаем множественность. Важно, что в концептуальной модели мы не показываем типы атрибутов, ключи, индексы, не нормализуем сущности (то есть, допускаем наличие сложных атрибутов, например, «Адрес» и «ФИО»). Все это нужно, во-первых, для того, чтобы в первом приближении увидеть всю структуру данных в виде небольшой модели. А во-вторых, такую, незагроможденную реализационными деталями модель, можно легко обсуждать со специалистами в предметной области, для которой мы создаем наше приложение. Если мы добавим реализационной информации, то они, как показывает опыт, сразу перестанут понимать, о чем речь.

[1] Далее, говоря о типах сущностей, мы будем для простоты называть их сущностями.

Пример логической модели. На рис. 8.4. показан тот же фрагмент предметной области, что и нет рис. 8.3, но «расписанный» в терминах логической модели.

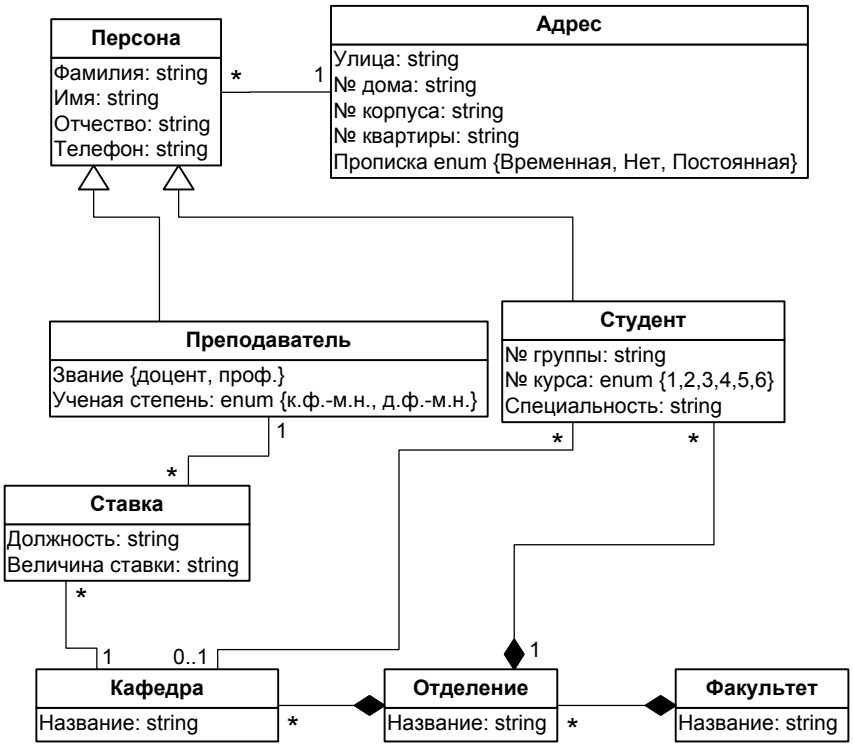

Рис. 8.4.

Каковы отличия моделей, представленных на рис. 8.3 и 8.4? На первый взгляд видно, что у нас появилось больше сущностей, а у атрибутов появились типы. Но это далеко не все. Рассмотрим эти отличия, а также действия разработчиков модели, приведшие к этому, более подробно.

1. Анализ атрибутов: типизация – появление перечислимых типов, возможны также таблицы-справочники, при анализе типов атрибутов некоторые из них – например, «Адрес» – вынесены в отдельную таблицу. Необходимо заметить, что типы атрибутов в логической модели могут не быть типами целевой платформы, а нужны для того, чтобы уточнить нашу схему: ведь, задумываясь о типах атрибутов мы можем создавать новые сущности. Кроме того, эти типы могут использоваться при автоматической генерации физической модели или конечного кода.
2. Использование наследования – это также следствие анализа атрибутов сущностей «Преподаватель» и «Студент» – часть общих атрибутов была вынесена в их общего предка – сущность «Персона».
3. Уточнение связей: значений множественности (не все они были проставлены в концептуальной модели), а также связанных с этим нюансов предметной области. Например, мы поняли, что студены только после второго курса распределяются по кафедрам, а до этого времени учатся все вместе. Но на то или иное отделение факультета они поступают изначально. В связи с этим мы делаем так, что сущность «Студент» агрегируется не кафедрой, а отделением. А с кафедрой у него остается связь, причем ее множественность со стороны кафедры – 0..1 (то есть этой связи может не быть, если студент учится на первом или втором курсе).

4. Раскрытие отношения многие-ко-многим. Данный вид отношения неудобен при реализации схемы данных в реляционной модели, и оно раскрывается с помощью заведения дополнительной сущности и пары новых отношений вида один-к-многим. Более того, оказывается, что даже при моделировании эта сущность оказывается содержательной, а не просто заглушкой – в нашем примере сущность Ставка, раскрывающая отношение многие-ко-многим между кафедрами и преподавателями, имеет атрибуты – должность преподавателя на кафедре и величину ставки, на которой он числится на данной кафедре. Ведь преподаватель может работать на разных кафедрах, на разных должностях и на разных ставках.

Относительно рис. 8.2. и 8.3. интересно отметить следующее.

Фрагмент логической модели, изображенный на рис. 8.3 получился сильно упрощенным. Фрагмент реальной логической схемы данных этого приложения, отвечающий только за адрес, состоит из девяти разных сущностей – учитывается возможность задания сельского и городского адреса, в составе городского адреса включаются возможность задать район и т.д. Преподаватель и студент также описываются с помощью внушительного набора сущностей.

Если же говорить о приведенном выше фрагменте концептуальной модели, то он вполне может быть таким. Но тут, однако, нужно уточнить цель и аудиторию концептуальной модели. Например, если она широко обсуждается с экспертами предметной области, а сами они едва знакомы с UML и моделью сущность-связь. Если же эта модель строится как результат «вытягивания» информации из экспертов области, но сама она с ними не обсуждается, то тогда она может быть посложнее. Например, все значения множественности могут в ней быть проставлены точно, показано наследование.

Пример физической модели. Диаграмма, представленная на рис. 8.5, представляет физическую модель для концептуальной и логической моделей, представленных на рис. 8.3. и 8.4. Эта диаграмма создана в Microsoft Visual Studio 2005 и ориентирована на реализацию схемы базы данных на СУБД Microsoft SQL Server.

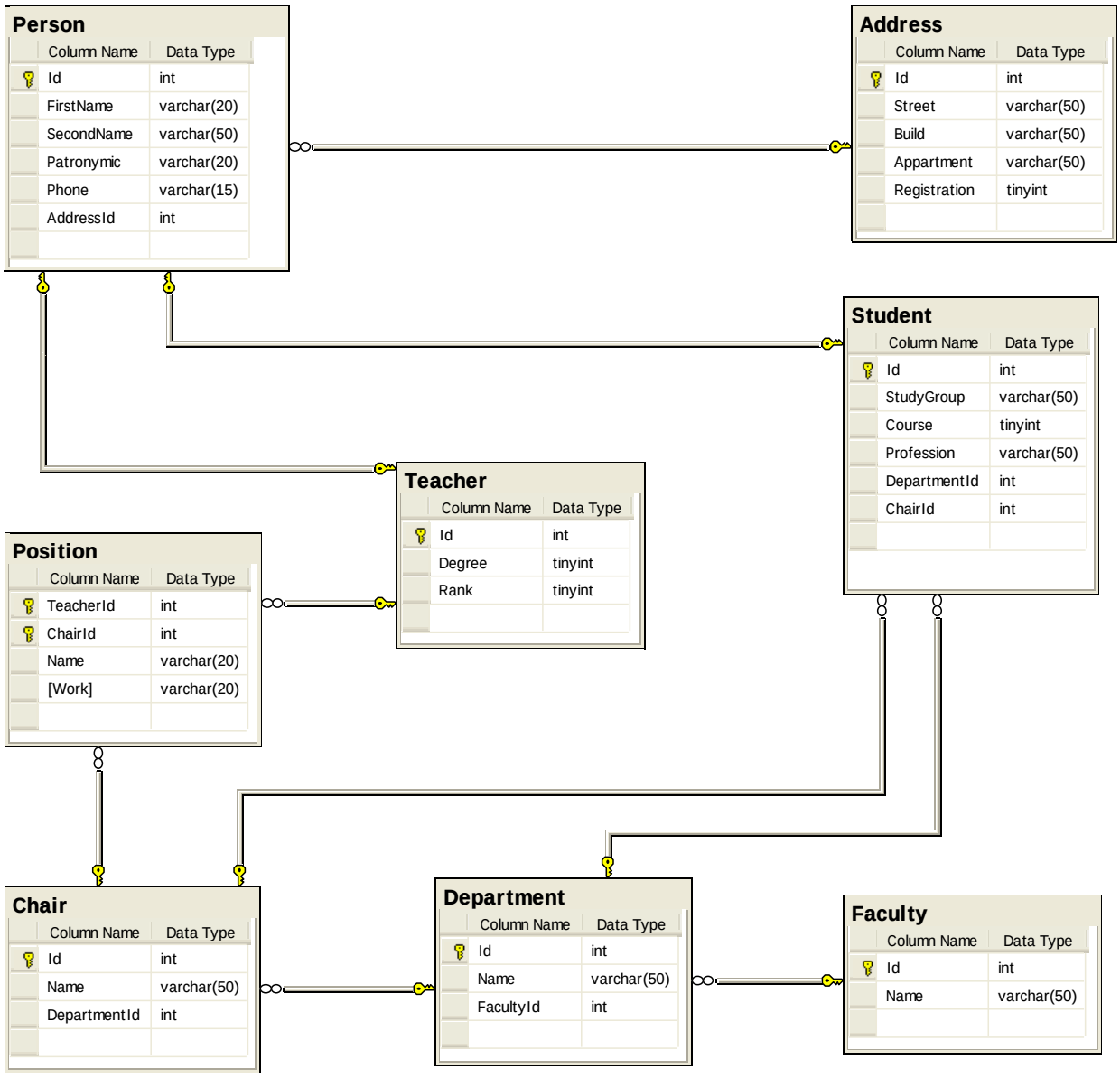

Рис. 8.5

Сущности представлены таблицами, атрибуты – колонками, а их типы имеют типы платформы реализации (в данном случае – Microsoft SQL Server). Все отношения – наследования, один-ко-многим – также реализованы в рамках реляционной модели:

1. О реализации отношения один-ко-многим.
2. О реализации наследования.

Кроме того, в схему добавлены ключи и индексы для таблиц, а также многие другие реализационные детали.

Спецификация структуры данных на SQL/DDL. По физической модели, представленной на рис. 8.5., Microsoft Visual Studio 2005 генерирует код на SQL, который описывает базу данных нашего приложения. Этот код представлен ниже.

```
CREATE TABLE [Faculty](
        [Id] [int] NOT NULL,
        [Name] [varchar](50) NULL,
 CONSTRAINT [PK_Faculty] PRIMARY KEY([Id] ASC)
)

CREATE TABLE [Address](
        [Id] [int] NOT NULL,
        [Street] [varchar](50) NULL,
        [Build] [varchar](50) NULL,
        [Appartment] [varchar](50) NULL,
        [Registration] [tinyint] NULL,
 CONSTRAINT [PK_Address] PRIMARY KEY([Id] ASC)
)

CREATE TABLE [Teacher](
        [Id] [int] NOT NULL,
        [Degree] [tinyint] NULL,
        [Rank] [tinyint] NULL,
 CONSTRAINT [PK_Teacher] PRIMARY KEY([Id] ASC)
)

CREATE TABLE [Student](
        [Id] [int] NOT NULL,
        [StudyGroup] [varchar](50) NULL,
        [Course] [tinyint] NULL,
        [Profession] [varchar](50) NULL,
        [DepartmentId] [int] NOT NULL,
        [ChairId] [int] NULL,
 CONSTRAINT [PK_Student] PRIMARY KEY([Id] ASC)
)

CREATE TABLE [Department](
        [Id] [int] NOT NULL,
        [Name] [varchar](50) NOT NULL,
        [FacultyId] [int] NOT NULL,
 CONSTRAINT [PK_Department] PRIMARY KEY([Id] ASC)
)

CREATE TABLE [Chair](
        [Id] [int] NOT NULL,
        [Name] [varchar](50) NOT NULL,
        [DepartmentId] [int] NOT NULL,
 CONSTRAINT [PK_Chair] PRIMARY KEY([Id] ASC)
)

CREATE TABLE [Position](
        [TeacherId] [int] NOT NULL,
        [ChairId] [int] NOT NULL,
        [Name] [varchar](20) NULL,
        [Work] [varchar](20) NULL,
 CONSTRAINT [PK_Position] PRIMARY KEY([TeacherId] ASC,     [ChairId] ASC)
)

CREATE TABLE [Person](
        [Id] [int] NOT NULL,
        [FirstName] [varchar](20) NULL,
        [SecondName] [varchar](50) NULL,
        [Patronymic] [varchar](20) NULL,
        [Phone] [varchar](15) NULL,
        [AddressId] [int] NULL,
 CONSTRAINT [PK_Person] PRIMARY KEY([Id] ASC)
)

ALTER TABLE [Teacher] ADD CONSTRAINT [FK_Teacher_Person]
        FOREIGN KEY([Id]) REFERENCES [Person] ([Id])

ALTER TABLE [Student] ADD CONSTRAINT [FK_Student_Chair]
        FOREIGN KEY([ChairId]) REFERENCES [Chair] ([Id])

ALTER TABLE [Student] ADD CONSTRAINT [FK_Student_Department]
FOREIGN KEY([DepartmentId]) REFERENCES [Department] ([Id])

ALTER TABLE [Student] ADD CONSTRAINT [FK_Student_Person]
```

```
FOREIGN KEY([Id]) REFERENCES [Person] ([Id])

ALTER TABLE [Department] ADD CONSTRAINT [FK_Department_Faculty]
FOREIGN KEY([FacultyId]) REFERENCES [Faculty] ([Id])

ALTER TABLE [Chair] ADD CONSTRAINT [FK_Chair_Department]
FOREIGN KEY([DepartmentId]) REFERENCES [Department] ([Id])

ALTER TABLE [Position] ADD CONSTRAINT [FK_Position_Position]
FOREIGN KEY([ChairId]) REFERENCES [Chair] ([Id])

ALTER TABLE [Position] ADD CONSTRAINT [FK_Position_Teacher]
FOREIGN KEY([TeacherId]) REFERENCES [Teacher] ([Id])

ALTER TABLE [Person] ADD CONSTRAINT [FK_Person_Address]
FOREIGN KEY([AddressId]) REFERENCES [Address] ([Id])


/* BEGIN HANDLE-WRITTEN CODE */

CREATE PROCEDURE [InsertStudent]
  @Id int, @FirstName varchar(20) null, @SecondName varchar(20) null,
    @Patronymic varchar(20) null, @Phone varchar(15) null, @AddressId int null,
    @StudyGroup varchar(50) null, @Course tinyint null, @Profession varchar(50) null,
    @DepartmentId int null, @ChairId int null
  AS BEGIN

    INSERT INTO Person (Id, FirstName, SecondName, Patronymic, Phone, AddressId)
    VALUES (@Id, @FirstName, @SecondName, @Patronymic, @Phone, @AddressId);

    INSERT INTO Student(Id, StudyGroup, Course, Profession, DepartmentId, ChairId)
    VALUES (@Id, @StudyGroup, @Course, @Profession, @DepartmentId, @ChairId)

  END

/* END HANDLE-WRITTEN CODE */
```

Отметим, что не весь код, задающий схему базы данных, можно генерировать автоматически – например, права на таблицы и колонки, триггеры, хранимые процедуры и т.д. необходимо дописывать «в ручную».

Тут можно заметить, что не все из выше перечисленного относится к определению схемы - например, хранимые процедуры и триггеры можно отнести к бизнес-логике. Однако строгой грани, отделяющей бизнес-логику от схемы базы данных нет. Кроме того, уж определение прав точно не относится к бизнес-логике. Тог есть сгенерированный код нужно все таки дополнять ручными вставками.

В современных средах разработки типа Microsoft Visual Studio, делаются попытки максимально сократить «ручные» изменения сгенерированного кода. Это достигается тем, что прямо в физической модели можно создавать файлы-вставки на целевом языке, и далее все такие вставки при генерации «соберутся» в единую программу. Механизм частичных классов в языке #C, а также возможность определять множество действий над одной и той же таблицей в SQL/DDL помогают осуществить эту стратегию – то есть конечный код представляется в виде независимых модулей, часть из которых генерируется, а часть пишется «руками». Причем эти модули являются достаточно мелкозернистыми единицами – вплоть до отдельных операторов, как в SQL/DDL. Ну, а выгоды «нетронутых руками» сгенерированных файлов очевидны – при изменении исходной визуальной модели их можно просто перегенерировать, а иначе мы будем иметь трудности циклической разработки (round-trip engineering).

Об инструментальных средствах. На настоящий момент почти все средства разработки баз данных поддерживают визуальное моделирование с автоматической генерацией конечного кода – Microsoft Visual Studio, Oracle и т.д. Имеются также специальные модельные средства, поддерживающие кроме физической модели также и логическую. Одним из лидеров в этой нише является пакет IBM Erwin. Наконец, концептуальные модели принято создавать общих, универсальных в средствах визуального моделирования типа IBM Rational Rose.

Кроме генерации различных диалектов SQL эти средства умеют также осуществлять возвратное проектирование схемы данных из кода в диаграммы.

Об эволюции схем данных. Трудности практического использования иерархии моделей при разработке схем баз данных – в итеративности разработки ПО и его

эволюции. Базы данных, как и программные системы в целом, редко сохраняют свой первоначальный дизайн в процессе использования. И хотя оценки различаются, большинство исследователей согласны, что не менее 50% усилий программисты затрачивают на внесение изменений в систему после ее ввода в эксплуатацию. И если визуальные модели еще можно «потерять» (мы говорили при изучении RUP, а также обсуждая психологические аспекты визуального моделирования, что часто визуальные модели служат лишь для извлечения знаний, формирования понимания предметной области программистами, и после достижении этих целей не хранятся и не поддерживаются в проекте), то живые пользовательские данные, существующие в прежней, старой структуре, потерять никак нельзя. Таким образом, появляется практически важная и научно содержательная тема эволюции схем баз данных. К сожалению, широко распространенных методов, реализованных в промышленных средствах разработки и повсеместно используемых, в этой области нет. Эти вопросы на практике, во многом, решаются ad hoc, умение их решать – это скорее искусство, чем выверенная технологическая дисциплина.

Хороший обзор на русском языке подходов к эволюции схем данных представлен в [7]. Интересный метод эволюции различных моделей схем данных (концептуальной, логической и физической), вместе с формальной постановкой задачи, содержится в [8]. Описание подхода, активно используемого на практике, который поддерживает согласованность визуальных моделей и кода при эволюции ПО и затрагивает, в том числе, задачу моделирования баз данных, можно найти в работах [5,6].

Основная литература

1. Петер П.Ш. Чен. Модель «сущность-связь» - шаг к единому представлению о данных. СУБД № 3 1995. http://lib.csu.ru/dl/bases/prg/dbms/1995/03/source/chen.html / Оригинал: in ACM Transactions on Database Systems v.1 N1, 1976, 1976.

Дополнительная литература

2. Integration Definition For Information Modeling (IDEF1X). — Draft Federal Information Processing Standards Publication, 1993. — 87 p.
3. Стригун С.А., Иванов А.Н., Соболев Д.И. Технология REAL для создания информационных систем и ее применение на примере системы «Картотека». – Математические модели и информационные технологии в менеджменте. Выпуск 2. – Изд-во СПбГУ, 2004. – С.120-139.
4. Иванов А.Н. Технологическое решение REAL-IT: создание информационных систем на основе визуального моделирования / Системное программирование. – Изд-во СПбГУ, 2004. – С.89-100.
5. Иванов А.Н. Механизмы поддержки циклической разработки ИС в рамках модельно-ориентированного подхода. – Системное программирование. – СПб, 2004. – С.101-123.
6. A.Ivanov, D.Koznov. REAL-IT: Model-BASED User Interface Development Environment. – Proceedings of IEEE/NASA ISoLA 2005 Workshop on Leveraging Applications of Formal Methods, Verification, and Validation. Loyola College Graduate Center Columbia, Maryland, USA, 23-24 September 2005. – P. 31-41.
7. Кукс С. Эволюция слабоструктурированных данных. – Диссертация на соискание степени к.ф.-м.н. СПбГУ, 2004. – 89 с.
8. J.-M. Hick, J.-L. Hainaut. Strategy for Database Application Evolution: The DB-MAIN Approach – Lecture Notes in Computer Science, Volume 2813/2003 – P. 291-306.