

Московский государственный технический университет
имени Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

В.И. Виноградов, М.В. Виноградова

**Постреляционные модели баз данных
и языки запросов**

учебное пособие

Москва
2016 МГТУ им. Н.Э. Баумана

УДК 004.65
ББК 32.973.2
В 41

Издание доступно в электронном виде на портале *ebooks.bmstu.ru*
по адресу: <http://ebooks.bmstu.ru/catalog/>_____

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

*Рекомендовано Редакционно-издательским советом
МГТУ им. Н. Э. Баумана в качестве учебного пособия*

Виноградов В.И.

Постреляционные модели баз данных и языки запросов: учеб.
пособие / В. И. Виноградов, М. В. Виноградова — М.: Изд-во МГТУ им.
Н.Э. Баумана, 2016. — __ с.

В учебном пособии изложены принципы и примеры построения моделей баз данных: сущность–связь, объектная, реляционная, объектно-реляционная, полуструктурированных данных. Приведены основные конструкции и правила построения запросов к базам данных на языках SQL, OQL, объектном расширении SQL, Datalog, XPath и XQuery.

Для студентов, обучающихся по программам специальности «Информатика и вычислительная техника» при изучении дисциплины «Постреляционные базы данных», и для студентов других направлений, изучающих дисциплины, связанные с разработкой программного обеспечения и баз данных.

УДК 004.65
ББК 32.973.2

ISBN

© МГТУ им. Н. Э. Баумана, 2016
© Оформление. Издательство МГТУ им. Н. Э. Баумана, 2016

Предисловие

В настоящее время практически любая программа использует базу данных для хранения информации. Наиболее широко распространены реляционные базы данных, однако все большее распространение получают постреляционные базы данных.

В учебном пособии рассмотрены постреляционные модели баз данных, языки и стандарты их описания, а также языки запросов к базам данных. Кроме того, приведен базовый синтаксис стандартизированных языков ODL и OQL для объектных баз данных, постреляционного расширения языка SQL для объектно-реляционных баз данных, языков XML, DTD, XPath и XQuery для полуструктурированных баз данных. Изложены основы формирования постреляционных моделей баз данных, правила построения моделей и запросов.

Предполагается, что читатели знакомы с реляционной моделью баз данных и языком SQL, тем не менее, в пособии описаны модели сущность-связь и реляционная, а также приведены краткие сведения, касающиеся языка SQL. Для знакомства читателя с логическими языками запросов рассмотрены принципы построения запросов на языке Datalog.

Цель пособия – научить студентов описывать структуру и формировать запросы к постреляционным базам данных. Учебное пособие предназначено для студентов, изучающих курс «Постреляционные базы данных», а также другие курсы, посвященные разработке информационного и программного обеспечения.

В результате изучения учебного пособия студенты приобретут перечисленные далее компетенции. **Общепрофессиональные компетенции (ОП):** способность применять математические методы анализа и моделирования автоматизированных систем организационного управления (ОП-1).

Компетенции в производственно-технологической (ПТ) деятельности: способность реализовать и оценить архитектурные решения и проекты, как компонентов, так и системы в целом на соответствие требованиям технического задания и стандартам (ПТ-1).

Компетенции в проектной (ПР) деятельности:

- способность выявлять потребности конечных пользователей с целью разработки технического задания на проектирование компонентов автоматизированных информационных систем (ПР-1);
- разрабатывать архитектуру системы с указанием технических и программных средств, ручных операций и связей между ними (ПР-4).

Компетенции в научно-исследовательской деятельности (НИ):

- способность выполнять постановку новых задач анализа и синтеза сложных проектных решений (НИ-1);
- готовность применять перспективные методы исследования и решения профессиональных задач с учетом мировых тенденций развития вычислительной техники и информационных технологий (НИ-2).

Учебное пособие состоит из трех глав. В первой главе приведены общие сведения о базах данных и системах управления базами данных, в ее параграфах рассмотрены две модели баз данных: сущность–связь и реляционная. Во второй главе рассмотрены модели постреляционных баз данных: объектная, объектно-реляционная и модель полуструктурированных данных. Третья глава посвящена языкам запросов и формированию запросов к базам данных.

Рассмотренные в учебном пособии материалы помогут студентам выполнять лабораторные и контрольные работы, домашние задания и самостоятельную работу по дисциплине «Постреляционные базы данных».

Авторы надеются, что изучение данного пособия будет полезно всем читателям в их практической деятельности по созданию баз данных.

Глава 1. Создание реляционных баз данных

1.1. Базы данных и системы управления базами данных

База данных (БД) — массив данных, используемый в прикладных программах, который сохраняется после выключения компьютера.

Базы данных предназначены для хранения больших объемов информации в структурированном виде в течение длительного времени.

Системы управления базами данных (СУБД) — компьютерные программы, предоставляющие клиентским приложениям доступ к базам данных.

Из истории развития баз данных

Первое поколение баз данных появилось в конце 60-х годов XX века. При работе с ними СУБД выполняли функции файловой системы для хранения данных и обеспечения доступа к ним. **Базы данных первого поколения** поддерживали сетевые и иерархические модели данных.

Базы данных первого поколения применяли в следующих сферах деятельности:

- бронирование и продажа авиабилетов;
- банковские системы (ведение счетов и проводка платежей);
- корпоративные системы (информация о сотрудниках, товарах и продажах).

Ко *второму поколению* относят **реляционные базы данных**, появившиеся в начале 70-х годов XX века и активно используемые до конца 90-х годов XX века. Считается, что их теорию сформулировал Д. Кодд. Особенность реляционных баз данных состоит в том, что все данные и связи между ними хранятся в таблицах. Для определения структуры данных и манипулирования их значениями используют язык SQL (Structured Query Language – структурированный язык запросов).

Третье поколение баз данных, называемых **постреляционные**, начало развиваться с 90-годов XX века. Тогда появились объектные, объектно-реляционные и полуструктурированные базы данных, которые расширяют возможности реляционных баз данных и позволяют хранить и обрабатывать как атомарные значения, так и объекты со сложной структурой.

Объектные базы данных, основанные на объектно-ориентированной парадигме, — альтернатива реляционному подходу. *Объектно-реляционные* базы данных поддерживают обратную совместимость с реляционными базами и расширяют их возможности. *Полуструктурированные* базы данные развиваются параллельно на

основе сетевых и иерархических баз данных и позволяют работать с частично структурированными данными.

Проектирование баз данных

Процесс создания информационной системы, содержащей БД, можно разделить на два этапа:

- проектирование и реализация базы данных;
- разработка приложения для работы с базой данных.

Основными задачами проектирования БД являются:

- выбор модели для описания структуры данных;
- определение элементов данных, их типов и связей между ними;
- формирование структур данных в терминах выбранной модели.

При проектировании БД сначала формируют концептуальную модель, а затем на ее основе — логическую. Концептуальная модель не зависит от средств реализации. Для ее построения обычно используют подход сущность–связь, который позволяет формализовать объекты предметной области и описать их как наборы сущностей, их атрибутов и связей.

Логическая модель строится с учетом последующей реализации БД в конкретной СУБД. Выделяют четыре основных модели построения логических моделей баз данных:

- реляционная модель,
- объектная модель,
- объектно-реляционная модель,
- модель полуструктурированных данных.

Перечисленные модели, языки их описания и принципы построения определяются стандартами. Поскольку стандарты обуславливают возможности и ограничения моделей, для физической реализации в конкретной СУБД логическая модель БД должна быть адаптирована к ее особенностям.

Требования к системе управления базами данных

Программу считают полноценной СУБД, если она обладает следующими возможностями:

- поддерживает язык определения данных (Data Definition Language (DDL)) для создания баз данных и описания их структуры;
- поддерживает язык манипулирования данными (Data Manipulation Language (DML)) для построения запросов к данным (добавление, удаление, изменение и чтение данных в базе данных);
- длительно (долговременно и постоянно) может хранить большие объемы структурированных данных;
- поддерживает работу в многопользовательской среде (параллельная работа нескольких пользователей);
- восстанавливает информацию после сбоев (как программных, так и аппаратных);
- выполняет транзакции — набор действий, которые выполняются либо все, либо ни одно;
- защищает информацию от несанкционированного доступа;
- оптимизирует запросы и осуществляет эффективный поиск информации в огромных объёмах данных.

Далее в параграфах и пунктах первой главы будут подробно рассмотрены модели данных, принципы их построения и языки описания.

Вопросы для самопроверки

1. Что называют базой данных и системой управления базами данных?
2. Что такое модель базы данных?
3. Перечислите поколения баз данных.
4. Назовите известные модели баз данных. Какие из них относятся к третьему поколению?
5. Чем отличаются логические и концептуальные модели баз данных?
6. В какой последовательности проектируют базы данных?
7. Сформулируйте определение системы управления базами данных? Какие требования предъявляют к системам управления базами данных?

1.2. Модель сущность–связь


Модель сущность–связь, называемая также ER-моделью (entity-relationship model) позволяет описывать структуру БД на концептуальном уровне. Для построения такой модели применяют графическую нотацию, задающую наборы графических символов, с помощью которых можно описывать объекты предметной области, их атрибуты и связи между ними. Существует несколько нотаций описания модели сущность–связь, например, нотации Чена, Мартина, IDEF1X, Баркера и др.

Основные элементы модели

При построении модели сущность–связь с помощью нотации Чена базовыми элементами являются сущности, их атрибуты, ключи и связи между сущностями (табл. 1.1).

Таблица 1.1.

Базовые элементы модели сущность–связь в нотации Чена

Элемент модели	Графическое обозначение	Описание элемента
Сущность (набор сущностей)		Определяет объект предметной области или набор подобных объектов
Атрибут		Характеризует сущность. Может относиться к связи
Ключ (первичный ключ)		Соответствует атрибуту(ам), идентифицирующему(им) сущность
Связь		Соединяет две или более сущности. Связи могут быть 1–1, 1–М, М–1 и М–М.

Сущность или *набор сущностей* соответствует объекту предметной области или набору подобных объектов, с одинаковыми характеристиками (например, сущность Персона). Сущность в нотации Чена имеет имя, уникальное в пределах модели, и обозначается прямоугольником (см. табл. 1.1).

Атрибут характеризует сущность (например, атрибут возраст). Ему присваивают имя, уникальное среди всех атрибутов сущности. Атрибут может иметь атомарное значение (например, число или строка), множество атомарных значений или структуру, состоящую из атомарных значений. Если атрибут описывает более сложную структуру, возможно, его следует выделить в отдельную сущность. Атрибут может

относиться не к сущности, а к связи. Каждый атрибут принадлежит только одной сущности, что в нотации Чена обозначается линией между сущностью и ее атрибутом (рис. 1.1).

Ключ (первичный ключ) (см. табл. 1.1) представляет собой атрибут(ы), идентифицирующий(ие) сущность. Его значение позволяет выделить один конкретный объект предметной области из множества подобных (например, номер ИНН). Составной ключ определяет несколько атрибутов (например, серия и номер паспорта). Каждая сущность должна иметь ключ. В нотации Чена все атрибуты, входящие в ключ (простой или составной), подчеркиваются. Альтернативные ключи на модели сущность–связь не изображаются.

Связь (см. табл. 1.1) соединяет две сущности и более между собой. Связи могут быть 1–1 (один к одному), 1–М (один ко многим), М–1 (многие к одному) и М–М (многие ко многим). Связь между двумя сущностями называют бинарной, между тремя – тернарной, между N сущностями — N -арной. Ей присваивают имя, уникальное в пределах модели.

В нотации Чена конец связи, имеющий множественность 1 (один), обозначают линией со стрелкой, а конец связи, имеющий множественность М (много), – линией без стрелки. Множественность можно указывать явно (1 или М).

Построение простых моделей

Рассмотрим примеры построения простых моделей сущность–связь.

Пример 1.1. *Построение модели для описания фильмов, их актеров и студий.*

Определим предметную область, связанную с фильмами. Для начала выделим основные сущности, их атрибуты, ключи и связи между сущностями. Указываем только информацию, которая будет храниться в БД и представлять интерес для пользователя информационной системы.

Каждый фильм имеет название, год выпуска и длительность. Идентификацию фильма проведем на основе его названия и года. В фильме играет множество актеров. При этом каждый актер имеет Ф.И.О, ИНН и образование. Идентификатором актера считаем его ИНН. Актер может играть в нескольких фильмах. Полагаем, что фильм снят одной студией, которая имеет название и адрес, идентификатором выберем название.

Проанализировав предметную область, выделим сущности Фильм, Актер, Студия со следующими атрибутами:

Фильм (Название, Год выпуска, Жанр и Длительность, составной ключ (Название и Год))

Актер (Ф.И.О, ИНН, Образование, ключ (ИНН))

Студия: Название и Адрес, ключ (Название)

Кроме того, определены следующие связи между сущностями:

- связь бинарная Играл, объединяющая Фильм и Актера, типа М–М (в фильме много актеров, актер играл во многих фильмах);
- связь бинарная Снят, объединяющая Фильм и Студию, типа М–1. Фильм снят одной студией, на студии выпущено много фильмов.

По итогам построения модели сущность–связь получим схему, представленную на рис.1.1.

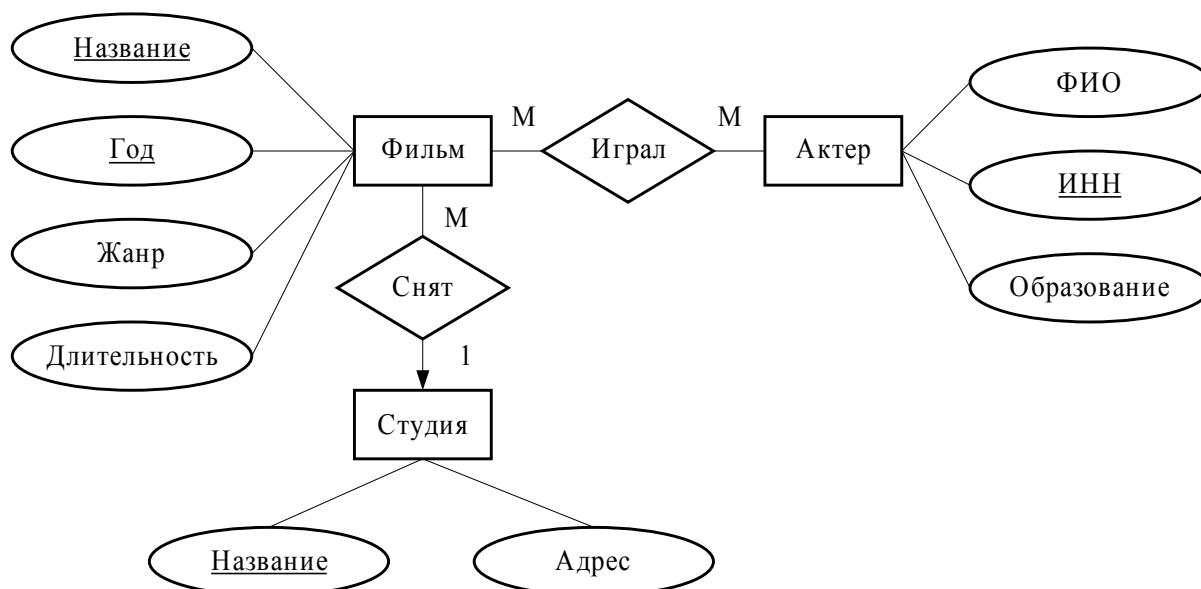


Рис. 1.1. Схема модели фильмов

Пример 1.2. Построение модели учебных занятий.

Определим предметную область, связанную с проведением лекций. Лекции читают в аудитории, по дисциплине, для группы. Группа имеет шифр, который ее идентифицирует, и относится к кафедре. Дисциплина имеет название и ее читает лектор. Идентификатором дисциплины выберем ее название. Аудитория имеет номер и расположена в учебном корпусе. Номер аудитории примем за ее идентификатор.

В аудитории читается много дисциплин для многих групп. Группа слушает много дисциплин во многих аудиториях. Дисциплина читается для многих групп во многих аудиториях.

Проанализировав предметную область, выделим сущности со следующими атрибутами:

Дисциплина (Название, Лектор, ключ (Название))

Группа (Шифр, Кафедра, ключ (Шифр))

Аудитория: Номер, Корпус, ключ (Номер)

Кроме того, определим тернарную (от лат. *ternarius* — «тройной») связь Изучает, которая соединяет все сущности между собой (рис. 1.2).

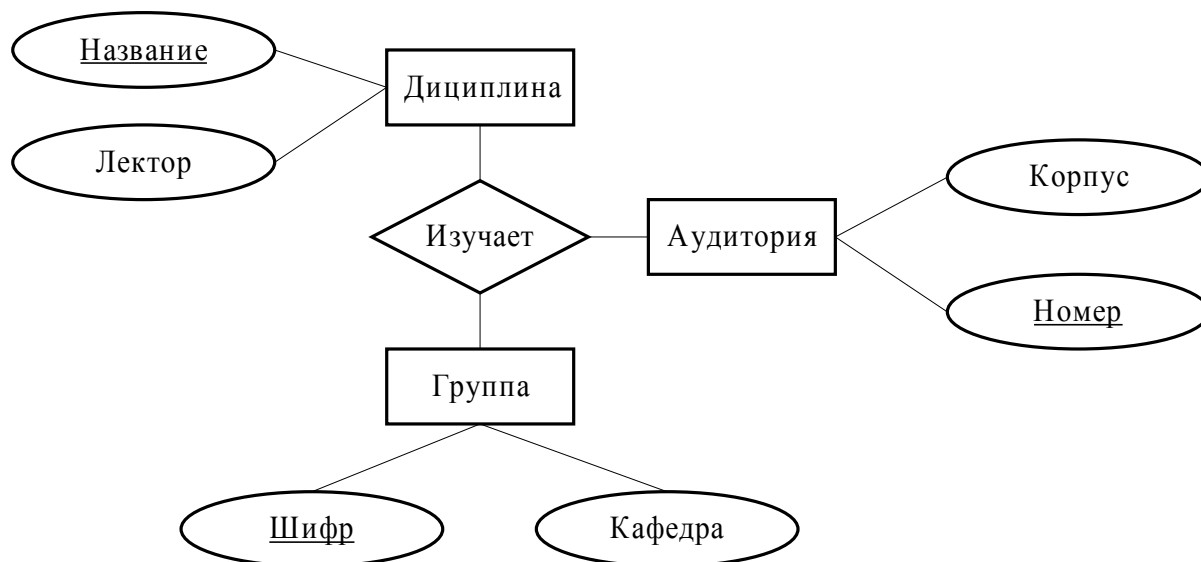


Рис. 1.2. Схема модели учебных занятий

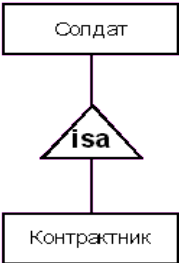
Дополнительные элементы модели

Кроме базовых элементов в модели сущность–связь допускается использование следующих дополнительных элементов (табл. 1.2).

Таблица 1.2.

Дополнительные элементы модели сущность–связь в нотации Чена

Элемент модели	Графическое обозначение	Описание элемента
Слабая сущность		Не может существовать без поддерживающей сущности
Поддерживающая связь		Соединяет зависимую сущность с поддерживающими ее

Связь ISA		Расширяет базовую сущность дополнительными атрибутами. Аналог наследования
-----------	---	---

Слабая (зависимая) сущность — сущность, которая не может функционировать без другой сущности, называемой в данном случае поддерживающей (независимой) (см. табл. 1.2). Например, студент не может существовать без вуза. Может быть несколько поддерживающих сущностей для одной слабой, если все они необходимы для ее существования. Например, для покупки необходимо наличие продавца и покупателя. *Ключ слабой сущности* состоит из ее собственного ключа и ключевых атрибутов всех сущностей, от которых она зависит.

Идентифицирующая (поддерживающая) связь (см. табл. 1.2) объединяет слабую сущность с сущностями, от которых она зависит. Она является бинарной связью типа 1–1 или 1–М. Со стороны поддерживающей сущности всегда указывается 1 (один). Например, слабая сущность Кафедра (рис. 1.3) не может существовать отдельно от поддерживающей ее сущности Вуз. Ключом кафедры будет составной ключ (Название кафедры, Название вуза).

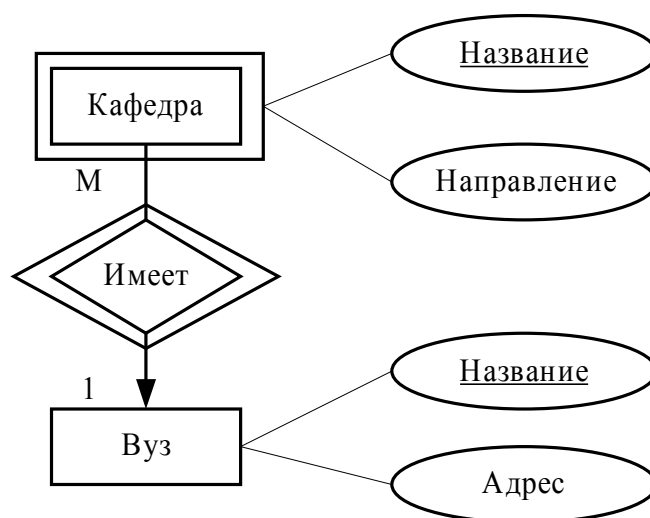


Рис. 1.3. Схема модели кафедры

Связь ISA — аналог наследования (см. табл. 1.2). Она расширяет базовую (родительскую) сущность дополнительными атрибутами производной (дочерней) сущности. В нотации Чена обозначается треугольником, направленным в сторону

родительской сущности. Ключ должен быть определен в базовой сущности. Дочерняя сущность может иметь только атрибуты, не входящие в ключ.

Роль — название сущности в пределах связи. Поскольку одна и та же сущность может участвовать в одной связи несколько раз, необходимо явно указать играемые ею роли. В нотации Чена роль обозначается надписью около линии связи (рис. 1.4). Сущность Сотрудник играет в связи Руководит две роли: начальника и подчиненного.

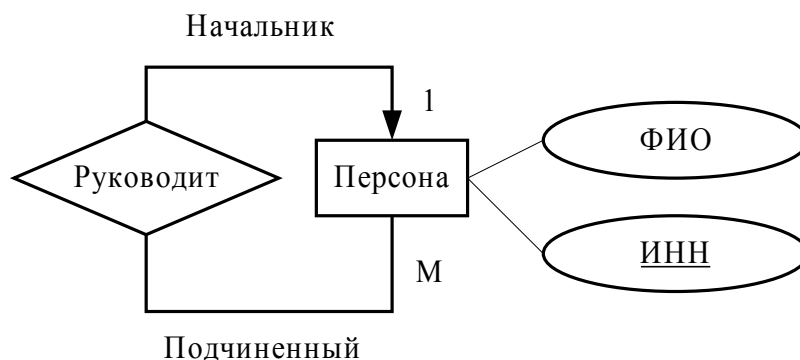


Рис.1.4. Схема модели сотрудников

Ограничения целостности показывают дополнительные требования к данным и их структуре. К ограничениям относят:

- ограничение связи типом множественности (1–М, 1–1, М–М);
- ограничения, накладываемые слабой сущностью, ключами или связью ISA;
- ограничения домена, уникальности или общего вида (например, в связи участвует не более трех сущностей).

Обычно ограничения добавляют на диаграмму как комментарий около элемента, к которому они относятся (если не предусмотрено иное).

Пример модели сущность–связь со связью ISA

Более сложным примером построения модели является модель для описания киновыпусков.

Разделим фильмы на три категории: обычный фильм, мультфильм и драма (экранизация книги). Мультфильмы имеют дополнительные атрибуты: имена привлеченных художников. *Драма* — фильм, для которого необходимо указать название книги, по которой он поставлен, и ее авторов. К построенной ранее модели фильмов (см. рис. 1.1) добавим сущности Мультфильм и Драма. *Мультфильм* — производная

сущность от сущности Фильм. Содержит все атрибуты фильма и дополнительный атрибут Художники. Ключ сущности Мультфильм совпадает с ключом сущности Фильм: Название и Год выпуска. *Драма* — производная сущность от сущности Фильм. Содержит все атрибуты фильма и дополнительные атрибуты: книга (ее название) и авторы (книги). Ключ драмы совпадает с ключом фильма: название и год.

Определим связи типа ISA между базовой сущностью (Фильм) и производными сущностями (Мультфильм и Драма) (рис. 1.5).

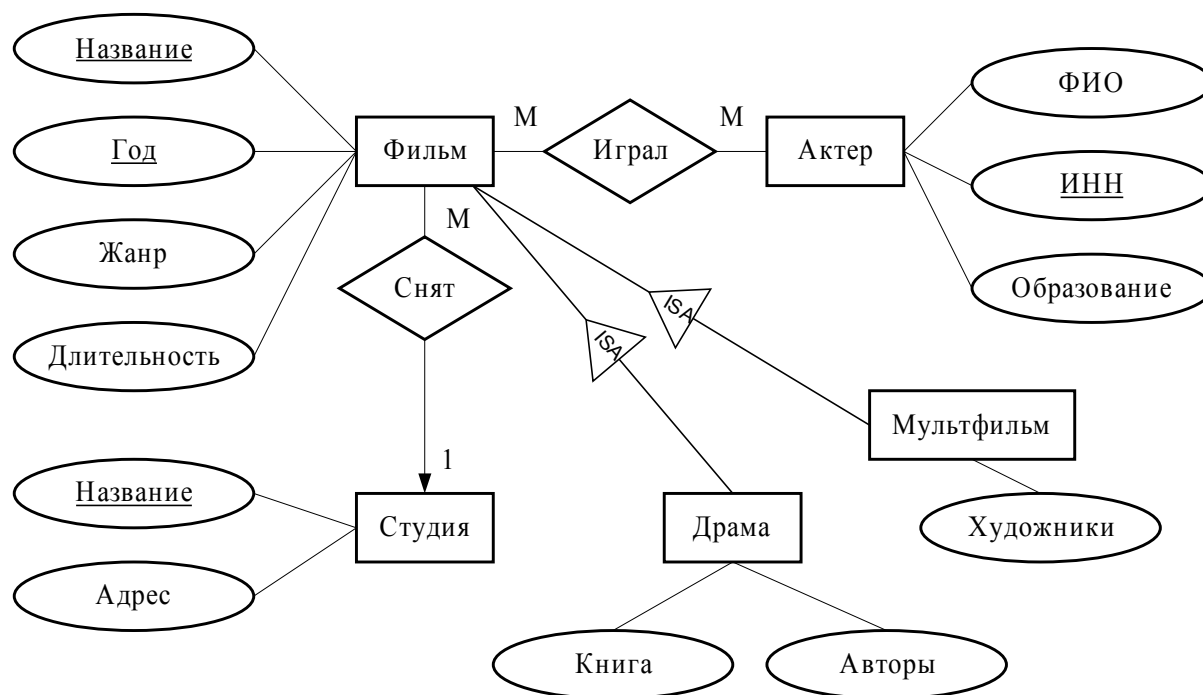


Рис.1.5. Схема модели киновыпусков

Сущности Мультфильм и Драма являются производными от сущности Фильм (объединены с ней связью ISA). В результате они обе наследуют ключ и все атрибуты сущности Фильм и дополняют их собственными атрибутами.

Вопросы для самопроверки

1. Для чего применяют модель сущность-связь?
2. Как в нотации Чена изображают сущность, атрибут, ключ и связь?
3. Как в нотации Чена изображают слабую сущность, поддерживающую связь и связь ISA?
4. Что такое ключ?
5. Назовите типы связей. Что такое арность и множественность связи?

6. Что такое слабая сущность и поддерживающая связь?

7. Для чего применяют связь ISA?

1.3. Реляционная модель

Основные понятия

Реляционная модель БД — логическая модель, базирующаяся на концептуальной модели, например, модели сущность–связь, и используемая для построения схемы (структуры) реляционной БД. Управление реляционной БД осуществляется реляционной СУБД.

Реляционная модель БД позволяет представлять данные в виде отношений, для которых заданы:

- множество отношений (таблиц), состоящих из множества кортежей (строк таблицы);
- схема отношений — названия отношений и набор атрибутов (столбцов таблицы), входящих в отношение.

На основе схемы отношений можно создать экземпляр отношений — конкретную таблицу в БД для определенного момента времени. В таких таблицах хранятся как атрибуты сущностей, так и информация о связях между ними. Рассмотрим элементы реляционной модели БД и принципы ее построения на основе модели сущность–связь. Согласно реляционной модели БД, все данные хранятся в таблицах. Каждая таблица содержит сведения о некотором множестве сущностей (табл. 1.3).

Таблица 1.3.

Отношение Студия

Название	Адрес
Дисней	Калифорния, США
Ленфильм	Санкт-Петербург, Россия
Мосфильм	Москва, Россия

Строку таблицы, соответствующей одной сущности, называют *кортежем*. Пример кортежа: («Ленфильм», «Санкт-Петербург, Россия»). Столбцы отражают названия атрибутов сущности и их значения. Все атрибуты являются атомарными (несоставными), например, целыми или действительными числами, символами или строками, логическими значениями. Каждый атрибут принадлежит домену, который определяет его тип данных. Домен задает множество допустимых значений конкретного

атрибута. Таблицу реляционной модели БД называют отношением или экземпляром отношений (см. табл. 1.3).

Схему отношения описывает название таблицы и названия ее столбцов. Например, Студия (Название, Адрес). В схеме отношения можно указывать домены атрибутов, например, Студия (Название: varchar(30), Адрес: varchar(50)). Схема БД — множество схем отношений, входящих в нее. Каждое отношение должно иметь первичный ключ — один или несколько столбцов, значения которых однозначно определяют значения всех прочих атрибутов каждого кортежа. Атрибуты, входящие в первичный ключ, называют ключевыми и на схеме их отображают подчеркиванием, например, Студия (Название, Адрес).

К основным преимуществам использования реляционной модели относят следующие возможности:

- нормализации схемы БД, что позволяет избежать дублирования данных, избыточности, аномалий включения, обновления и удаления;
- оптимизации запросов к таблицам, что существенно повышает их производительность.

Описание структуры БД при ее реализации выполняют на языке SQL.

Правила преобразования модели сущность–связь в реляционную модель

При переходе к реляционной модели БД основные элементы в модели сущность–связь преобразуют:

- сущность — в отношение;
- атрибут сущности — в атрибут отношения;
- ключ сущности — в ключ отношения;
- связь 1–М — в атрибут сущности со стороны многих (М), который содержит значение ключевого(ых) атрибута(ов) сущности со стороны одного (1);
- связь М–М — в отношение, содержащее ключевые атрибуты связанных сущностей;
- N -арная связь ($N > 2$) — в отношение, содержащее ключевые атрибуты объединенных сущностей;

- слабая сущность и поддерживающие ее связи — в отношении, содержащее атрибуты слабой сущности и ключевые атрибуты всех поддерживающих ее сущностей; ее ключ – совокупность собственного ключа и ключей поддерживающих ее сущностей.

Примеры построения реляционных моделей приведены далее.

Пример 1.3. Построение схемы отношений на основе модели фильмов.

В результате преобразования модели сущность–связь фильмов (рис.1.1) в реляционную модель получена схема отношений:

Актёр (ИНН, Ф.И.О, Образование)

Фильм (Название, Год, Длительность, Жанр, Студия)

Студия (Название, Адрес)

Актёр–Фильм (Название, Год, ИНН)

В примере 1.3 сущности и их атрибуты преобразуют в отношения и их атрибуты. Связь 1–М между студией и фильмом транслируется в атрибут Студия в таблице фильмов. Этот атрибут содержит название студии, которая является ключом для таблицы Студия. Связь М–М между актерами и фильмами преобразуется в отдельное отношение Актер–фильм, содержащее все ключевые атрибуты как актера, так и фильма.

Примеры экземпляров отношений приведены далее в табл. 1.4 – 1.6.

Таблица 1.4.

Отношение Фильм				
Название	Год	Длительность	Жанр	Студия
Чапаев	1934	93	Драма	Ленфильм
Полосатый рейс	1961	83	Комедия	Ленфильм
Пираты Карибского моря	2003	150	Фантастика	Дисней

Таблица 1.5.

Отношение Актер		
ИНН	Ф.И.О	Образование
12345	Иванов И.И.	Щукинское училище
22222	Леонов Е.М.	Студия Захарова
33333	Бабочкин Б.А.	Студия Певцова

Таблица 1.6.

Отношение Актер–Фильм		
Название	Год	ИНН

Чапаев	1934	33333
Чапаев	1934	12345
Полосатый рейс	1961	22222
Полосатый рейс	1961	12345

Пример 1.4. Построение схемы отношений на основе модели учебных занятий.

В результате преобразования модели сущность–связь учебных занятий (см. рис. 1.2) в реляционную модель получена схема отношений:

Дисциплина (Название, Лектор)

Группа (Шифр, Кафедра)

Аудитория (Номер, Корпус)

Занятие (Дисциплина, Группа, Аудитория)

В этом примере тернарная связь между дисциплиной, группой и аудиторией преобразуется в дополнительное отношение, содержащее ключевые атрибуты связываемых сущностей:

- Дисциплина содержит название дисциплины;
- Группа — шифр группы;
- Аудитория — номер аудитории.

Пример 1.5. Построение схемы отношений на основе модели кафедры.

В результате преобразования модели сущность–связь кафедры (см. рис. 1.3) в реляционную модель получена схема отношений:

Кафедра (Название, Направление, Вуз)

Вуз (Название, Адрес)

В ключ слабой сущности Кафедра добавлен ключ поддерживающей сущности Вуз, который содержит ее ключ — название вуза.

Правила преобразования связи ISA в реляционную модель

Для преобразования связи ISA на модели сущность–связь к реляционной модели может быть использован один из трех подходов: сущностный, объектный, пустых значений.

При *сущностном подходе* каждую сущность преобразуют в отношение, содержащее собственные атрибуты и ключи базовой сущности. Для модели киновыпусков (см. рис. 1.5) после преобразования сущностей Фильм, Мультфильм и Драма и объединяющих их связей ISA получаем схему отношений:

Фильм (Название, Год, Длительность, Жанр, Студия)

Мультфильм (Название, Год, Художники)

Драма (Название, Год, Книга, Авторы)

При *объектном подходе* для каждой производной сущности введем отношение, включающее все атрибуты как базовой сущности, так и ее собственные. Тогда получим схему отношений:

Фильм (Название, Год, Длительность, Жанр, Студия)

Мультфильм (Название, Год, Длительность, Жанр, Студия,
Художники)

Драма (Название, Год, Длительность, Жанр, Студия, Книга,
Авторы)

При *подходе пустых значений* создадим одну сущность, содержащую атрибуты базовой и всех производных сущностей. Если при заполнении таблицы данными атрибут сущности отсутствует, запишем вместо его значения пустое значение NULL. Тогда получим схему отношений: Фильм (Название, Год, Длительность, Жанр, Студия, Художники, Книга, Авторы).

Вопросы для самопроверки

1. Приведите основные понятия реляционной модели баз данных.
2. Чем отличаются схема отношения и экземпляр отношения?
3. Что такое кортеж, атрибут, домен?
4. Назовите правила преобразования модели сущность – связь в реляционную.
5. Во что преобразуют сущности, их атрибуты и ключи?
6. Как в реляционной модели определяют связи 1–М и М–М, N-арные связи?
7. Охарактеризуйте три подхода для преобразования связи ISA к реляционной модели.
8. В чем преимущества реляционной модели?

1.4. Описание реляционной модели на языке SQL

Конструкции языка для создания таблиц

Для практической реализации реляционной модели и создания реляционной базы данных используется язык SQL (Structured Query Language — язык структурированных запросов). Создание базы данных по реляционной модели определяется стандартом SQL-92.

Язык SQL позволяет:

- описывать схемы отношений (таблицы и их поля);
- задавать ограничения, в том числе первичный и внешние ключи, ссылочную целостность, ограничение уровня атрибута или кортежа, ограничение общего вида.

Для создания таблицы используется оператор `CREATE TABLE`, имеющий следующее формальное описание:

```
CREATE TABLE _
(
  { column | [table_constraint] } ,
  ...
);
```

где `column` — перечень столбцов таблицы; `table_constraint` — ограничения целостности.

После имени таблицы в круглых скобках через запятую указывают список полей (называемых также столбцами) и ограничений. Определение столбца таблицы задают следующей конструкцией (в квадратных скобках указаны необязательные элементы конструкции):

```
_ [ ] [ ограничение ] [ DEFAULT ]
```

Каждое поле таблицы имеет имя (уникальное в пределах таблицы) и тип. Тип может быть определен как любой допустимый тип языка SQL, например:

`integer` — целое поле;

`char` (`количество_символов`) — строка из фиксированного количества символов;

`varchar` (`число_символов`) — строка символов, размером не более указанного;

`smallint` — целое число;

`float` — действительное число;

`date` — дата;

`time` — время.

Конструкция `DEFAULT` определяет значение по умолчанию.

Ограничения целостности

Система управления базами данных проверяет ограничения целостности при выполнении операторов редактирования, добавления и удаления записей в таблицах.

Если проверка не пройдена, СУБД отменяет выполнение оператора и откатывает транзакцию.

Ограничения целостности бывают следующих уровней применения: для столбца, для кортежа (таблицы), общего вида.

Ограничения для столбца указывают непосредственно после описания столбца, а ограничения для таблицы — через запятую после описания любого столбца. Ограничения для столбца могут указываться следующими фразами (одно из перечисленных или несколько через пробел):

NOT NULL — в любой добавляемой или изменяемой строке столбец всегда должен иметь значение, отличное от NULL;

UNIQUE — все значения столбца должны быть уникальны;

PRIMARY KEY — устанавливает столбец как первичный ключ и одновременно подразумевает, что все значения столбца будут уникальны;

CHECK (условие) — указываемое в скобках условие используется для проверки значения столбца. В скобках может быть условие, выборка, запрос любой сложности.

Ограничения для таблицы могут указываться следующими конструкциями:

CHECK — ограничение на значения полей кортежа;

FOREIGN KEY — ограничение ссылочной целостности.

Конструкция CHECK (условие) задает условие на значение полей кортежа, например,

```
CHECK( FIO<>' ' AND address<>' ' )
```

Ограничение ссылочной целостности FOREIGN KEY гарантирует, что все значения, указанные во внешнем ключе, будут соответствовать значениям родительского ключа, обеспечивая ссылочную целостность. Оно определяется следующим образом:

```
FOREIGN KEY ( _ _ _ [ , . . . n ] )  
REFERENCES _ _ _ [ , . . . n ] ,  
[ ON UPDATE { CASCADE | SET NULL | SET DEFAULT | NO ACTION } ]  
[ ON DELETE { CASCADE | SET NULL | SET DEFAULT | NO ACTION } ]
```

Конструкция REFERENCES таблица (список_полей) определяет имя родительской таблицы и столбцы ее первичного ключа; задает ограничение, которое требует совпадения значений внешнего ключа с указанными столбцами родительской таблицы. Типы данных столбцов внешнего ключа и первичного ключа родительской таблицы должны совпадать. При определении внешнего ключа можно указать действия,

выполняемые при внесении ограничений в родительский ключ, конструкциями ON UPDATE и ON DELETE:

```
ON UPDATE { CASCADE | SET NULL | SET DEFAULT | NO ACTION }
ON DELETE { CASCADE | SET NULL | SET DEFAULT | NO ACTION }
```

где CASCADE распространяет изменения в родительском ключе на совпадающие строки внешнего ключа; SET NULL устанавливает значения внешнего ключа в NULL; SET DEFAULT изменяет значение внешнего ключа на значение по умолчанию; NO ACTION выполняется по умолчанию и указывает, что изменять или удалять запись, на которую есть ссылка (внешний ключ) из другой таблицы, запрещено.

Пример ссылочной целостности с каскадным удалением записей и установкой пустых полей при изменении ключа родительской таблицы:

```
FOREIGN KEY (toPerson) references Person(id)
ON DELETE CASCADE ON UPDATE SET NULL
```

Минусы ограничений уровня кортежа или таблицы в том, что они срабатывают при редактировании и добавлении записи в создаваемой таблице. Если изменение происходит в другой таблице, то ограничение может быть нарушено.

Для задания ограничения общего вида, которое проверяется при любом изменении данных, используют конструкцию:

```
CREATE ASSERTION
CHECK ( )
```

Например, запишем, что нельзя хранить в таблице фильмов Movie фильм с пустым описанием жанра (полем type):

```
CREATE ASSERTION NeedType
CHECK (NOT EXISTS ( SELECT * FROM Movie WHERE type = ' ' ))
```

Пример создания схем отношений

Пример 1.6. Построение схем отношений для описания модели фильмов на языке SQL.

На основе модели фильмов (см. рис.1.1) построим схемы отношений, схожие со схемами отношений из примера 1.3:

```
Actor (inn, fio, edu)
Film (title, year, length, type, stud)
Stud (sid, sname, addr)
FA (act, fname, fyear)
```

Составим конструкции на языке SQL для создания таблиц на основе перечисленных схем отношений:

```
CREATE TABLE Stud
```


REFERENCES Stud(sid) — внешний ключ для связи фильма и студии с каскадным обновлением и установкой пустых значений для фильмов, чьи студии удалены;

CHECK (length>0 AND year>1894) — ограничение уровня кортежа на длину фильма и год выпуска.

Для **таблицы Actor** указали ограничение PRIMARY KEY при задании первичного ключа по полю FIO.

При определении **отношения FA**, раскрывающего связь М–М между фильмом и актером, использовали следующие ограничения:

FOREIGN KEY — два внешних ключа из внешних таблиц Film и Actor, один из них составной;

PRIMARY KEY — составной первичный ключ, состоящий из атрибутов act и fname, fyear. Внешний ключ входит в первичный в данном случае;

ON DELETE CASCADE — каскадное удаление;

ON UPDATE CASCADE — каскадное обновление при изменении ключа родительской таблицы;

ON UPDATE SET NULL — установка значения поля в ноль при изменении ключа родительской таблицы.

Вопросы для самопроверки

1. Как на языке SQL определить схему отношения?
2. Какие типы атрибутов применяют в реляционной БД?
3. Как задать ограничение уровня столбца? Перечислите основные конструкции.
4. Как задать ограничение уровня кортежа? Перечислите основные конструкции.
5. Как задать ограничение общего вида?
6. Что такое первичный ключ?
7. Что такое внешний ключ и ссылочная целостность?

Глава 2. Постреляционные модели баз данных

2.1. Объектно-реляционная модель

Типы данных

Основной недостаток *реляционной модели* связан с ограничением поддерживаемых типов данных. Не предназначена для использования коллекций, структур или составных данных. Это ограничение критическое, поскольку в современных базах данных необходимо хранить сложные типы данных и информацию об их структурах. Например, фото- и видео – материалы, их атрибуты, графические документы, внедренные объекты. Поэтому по мере развития реляционная модель БД трансформировалась в объектно-реляционную.

Объектно-реляционная модель БД предоставляет пользователю следующие *возможности*:

- хранить и обрабатывать сложные типы данных (структуры и коллекции);
- наряду с предопределенными типами данных использовать пользовательские типы данных;
- включать в пользовательские типы данных атрибуты и методы;
- поддерживать уникальные идентификаторы кортежей и ссылки на них;
- определять иерархию (наследование) типов данных.

Поддерживает следующие *типы данных*:

- атомарные;
- структуры, содержащие набор именованных полей;
- массивы, хранящие упорядоченный набор элементов;
- множества и мультимножества, хранящие неупорядоченный набор элементов;
- пользовательские типы данных;
- ссылки на объекты (кортежи) пользовательских типов данных.

Элементами структур и коллекций могут быть любые типы данных, в том числе составные.

Пользовательские типы данных

Пользовательские типы данных содержат атрибуты любых типов, кроме уже существующих. Существует три метода работы с пользовательскими типами данных:

- конструктор — используют при создании объекта данного типа;

- метод экземпляра — вызывают для конкретного объекта и имеет доступ к его полям;
- метод класса (статический) — применяют не к экземпляру, а ко всему классу.

Можно создавать иерархию типов на базе наследования. Производный тип наследует все атрибуты и методы базового типа и может добавлять собственные. При наследовании возможно переопределение методов, поддерживаются виртуальные вызовы и полиморфизм.

Пользовательский тип данных может содержать атрибуты любых типов, кроме него самого, и методы. При создании пользовательского типа данных системой автоматически генерируются три вида методов:

- метод-генератор, т. е. конструктор (название совпадает с названием типа, статический метод);
- метод-обозреватель — вызывается при чтении значения атрибута (название совпадает с названием атрибута, метод экземпляра без параметров);
- метод-модификатор — вызывается при записи значения атрибута (название совпадает с названием атрибута, метод экземпляра с одним параметром).

Типизированные и обычные таблицы

В объектно-реляционной, как и в реляционной модели, данные хранятся в таблицах, которые могут быть обычными либо типизированными.

Обычная таблица аналогична таблице реляционной базы данных, состоит из множества строк (кортежей) и множества столбцов (атрибутов). Таблица имеет первичный ключ, допускается использование внешних ключей, ограничений и всех возможностей, определенных реляционной моделью. *Атрибуты* могут быть любых типов, например, атомарные значения (строки, числа, дата), массивы или коллекции, ссылки на кортежи в других таблицах. Атрибуты могут быть пользовательского типа или множеством структур, *вложенной таблицей* (см. пример в табл. 2.1), в которой атрибут Работа отношения Персона является вложенным отношением со своими атрибутами Организация и Адрес. Значением атрибута Работа является экземпляр отношения, содержащий записи с местами работы данной персоны.

При описании схемы отношений указываются все атрибуты и, если атрибуты составных типов, — входящие в них вложенные атрибуты. Структура таблицы Персона

задается следующей схемой отношений: Персона (Фамилия, Работа (Организация, Адрес)).

Таблица 2.1.

Экземпляр отношения Персона

Фамилия	Работа	
Иванов	Организация Адрес ООО Москва, Ордынка ЗАО Тверь, Ленина	
Петров	Организация Адрес МММ Самара, Волжская ЗАО Тверь, Ленина	

Типизированную таблицу создают на основе пользовательского типа, каждая ее строка содержит один объект пользовательского типа. При этом ключи и прочие ограничения целостности относятся к таблице, а не к пользовательскому типу.

Для последующей ссылки на кортеж типизированной таблицы при ее создании определяют специальный атрибут: идентификатор (ИД), который генерируется системой или основан на первичном ключе.

Ссылки на кортежи

В объектно-реляционной модели вместо внешних ключей используют ссылки. Внутренняя реализация ссылки не раскрывается. Для обозначения ссылок используют записи вида:

$A(*R1)$ — атрибут A, являющийся ссылкой на кортеж отношения R1;

$(\{ *R1 \})$ — атрибут B, являющийся множеством ссылок на кортежи отношения R1.

В данном примере хранение данных (см. табл. 2.1) является избыточным. Кроме того, несколько персон могут работать в одной организации, что ведет к повтору записей. Эффективнее использовать ссылки. Схемы отношений, имеющие ссылки, записывают следующим образом:

Персона (Фамилия, Работа({ *Организация }))

Организация (Название, Адрес)

Приведем пример экземпляров отношений при использовании ссылок. Сведения хранят о персонах (табл. 2.2), об организациях (табл. 2.3).

Таблица 2.2.

Отношение Персона со ссылками

Фамилия	Работа
Иванов	#, ##
Петров	##, ###

Таблица 2.3.

Отношение Организация

(ИД, скрыт)	Организация	Адрес
#	ООО	Москва, Ордынка
##	ЗАО	Тверь, Ленина
###	МММ	Самара, Волжская

Объектно-реляционная модель БД основана на реляционной и для ее реализации используют язык SQL (стандарты SQL-99, SQL-2003), поэтому при описании связей между таблицами применяют те же правила, что и для реляционной модели, но заменяют внешние ключи на ссылки. Поэтому связь М–М реализуют через дополнительную таблицу, хранящую ссылки на связанные таблицы.

Правила преобразования модели сущность–связь в объектно-реляционную модель

При переходе к объектно-реляционной модели БД элементы модели сущность-связь преобразуют по следующим правилам:

- сущность — в отношение;
- атрибут сущности — в атрибут отношения, допускается использовать сложные типы;
- ключ сущности — в ключ отношения;
- связь 1–М — в атрибут сущности со стороны многих (М), который содержит ссылку на кортеж таблицы со стороны одного (1);
- связь М–М — в отношение, содержащее ссылки на кортежи связанных таблиц;
- N -арная связь ($N > 2$) — в отношение, содержащее ссылки на кортежи связанных таблиц;
- слабая сущность и поддерживающие ее связи — в отношение, содержащее атрибуты слабой сущности и ссылки на кортежи всех поддерживающих ее сущностей. Ее ключом будет совокупность собственного ключа и ссылок на поддерживающие ее сущности.

Примеры преобразования модели сущность–связь в объектно-реляционную модель

Пример 2.1. *Построение объектно-реляционной модели фильмов.*

В результате преобразования модели сущность–связь фильмов (рис. 1.1) в объектно-реляционную модель получена схема отношений:

Актёры: ИНН, Ф.И.О (Фамилия, Имя, Отчество), Образование

Фильмы: Название, год, Длительность, Жанр, Студия (*Студии)

Студии: Название, Адрес (Город, Страна)

Актёр–Фильм: Фильм (*Фильмы), Актер (*Актеры)

В этом примере сущности и их атрибуты преобразуются в отношения и их атрибуты. Атрибуты Ф.И.О и Адрес становятся структурами. Связь 1–М между студией и фильмом транслируется в ссылку на студию в таблице Фильмы. Связь М–М

между актерами и фильмами преобразуется в отдельное отношение Актер–Фильм, содержащее ссылки на кортеж Фильма и кортеж Актера.

В табл. 2.4 – 2.7 приведены примеры экземпляров отношений.

Таблица 2.4.

Экземпляр отношения Студии

Название	Адрес (Город, Страна)	ИД
Дисней	Калифорния, США	#2
Ленфильм	Санкт-Петербург, Россия	#1
Мосфильм	Москва, Россия	#3

Таблица 2.5.

Экземпляр отношения Фильмы

Название	Год	Длительность	Жанр	Студия	ИД
Чапаев	1934	93	Драма	#1	#21
Полосатый рейс	1961	83	Комедия	#1	#22
Пираты Карибского моря	2003	150	Фантастика	#2	#23

Таблица 2.6.

Экземпляр отношения Актеры

ИНН	Ф.И.О (Фамилия, Имя, Отчество)	Образование	ИД
12345	Иванов, Иван, Иванович	Щукинское училище	#31
22222	Леонов, Евгений, Михайлович	Студия Захарова	#32
33333	Бабочкин, Борис, Алексеевич	Студия Певцова	#33

Таблица 2.7.

Экземпляр отношения Фильм–Актер

Фильм	Актер
#21	#33
#21	#31
#22	#32
#22	#31

При графическом изображении экземпляров отношений допускается использовать стрелки для указания ссылок, поскольку реальная реализация ссылок скрыта от пользователя.

Пример 2.2. Построение объектно-реляционной модели учебных занятий.

В результате преобразования модели сущность–связь учебных занятий (см. рис.

1.2) в объектно-реляционную модель получена схема отношений:

Дисциплина (Название, Лектор)

Группа (Шифр, Кафедра)

Аудитория (Номер, Корпус)

Занятие (Дисц (*Дисциплина), Группа (*Группа), Ауд (*Аудитория))

В этом примере для реализации тернарной связи дисциплин, групп и аудиторий введено дополнительное отношение, содержащее ссылки на кортежи связываемых таблиц.

Пример 2.3. Построение объектно-реляционной модели кафедры.

В результате преобразования модели сущность–связь кафедры вуза (см. рис. 1.3) в объектно-реляционную модель получена схема отношений:

Кафедра (Название, Направление, вуз (*Вуз))

Вуз (Название, Адрес)

В ключ слабой сущности Кафедра добавляется ссылка на кортеж поддерживающей сущности Вуз.

**Правила преобразования связи ISA
в объектно-реляционную модель**

При преобразовании модели сущность – связь в объектно-реляционную модель связь ISA задают через наследование типов. При этом каждой производной сущности присваивают свое отношение, включающее все атрибуты базовой сущности и ее собственные, и указывают название базовой сущности.

Пример 2.4. Построение объектно-реляционной модели киновыпусков.

В результате преобразования модели киновыпусков (см. рис. 1.5) в объектно-реляционную модель получим набор схем отношений:

Актёры (ИНН, Ф.И.О (Фамилия, Имя, Отчество), Образование)

Фильмы (Название, год, Длительность, Жанр, Студия (*Студии))

Студии (Название, Адрес (Город, Улица))

Актёр–Фильм (Фильм (*Фильмы), Актер (*Актеры))

Мультфильмы:Фильмы (Название, Год, Длительность, Жанр, Студия (*Студия), Художники))

Драмы:Фильмы (Название, Год, Длительность, Жанр, Студия (*Студия), Книга, Авторы)

Вопросы для самопроверки

1. В чем отличия объектно-реляционной модели БД от реляционной?
2. Что такое пользовательский тип данных? Из чего он состоит?


```
FINAL | NOT FINAL
[           ]
```

Имя определяемого пользователем **типа данных** имеет, в общем случае, трехзвенную структуру:

```
имя_каталога.имя_схемы.имя_типа
```

Наследование типов задается в синтаксисе UNDER. Если этот раздел присутствует, то в нем указывается имя ранее определенного UDT, атрибуты и методы которого будут наследоваться определяемым структурным типом.

При определении **пользовательского типа** перечисляют его атрибуты с указанием их типов. Имя определяемого атрибута должно быть уникальным. Тип данных атрибута может быть любым допустимым в SQL типом данных (включая UDT), кроме самого определяемого структурного типа и его базовых типов.

Для атрибута можно объявить значение по умолчанию. Если типом данных атрибута является встроенный тип данных, то значение атрибута объявляется в том же синтаксисе, что и значение столбца по умолчанию в определении таблицы. Если UDT, тип ROW (структура) или ссылочный тип, то единственным допустимым значением по умолчанию является неопределенное значение (NULL). Если же тип ARRAY (массив), то значением по умолчанию может быть NULL или пустое значение, массив указывается как ARRAY[].

Можно определить *инстанцируемый* (instantiable) или *неинстанцируемый* (not instantiable) *пользовательский тип*. Для неинстанцируемого типа конструктор не определяется, и поэтому создать значение этого типа невозможно. Неинстанцируемые типы могут быть типами атрибутов других структурных типов, типами столбцов, переменных и т. д. При отсутствии явной спецификации по умолчанию тип считается *инстанцируемым*. Обязательный раздел FINAL | NOT FINAL указывает на возможность или невозможность определения производных типов определяемого пользовательского типа.

При определении **индивидуального** (атомарного) **типа** всегда требуется указывать FINAL. При определении структурного (из нескольких атрибутов) типа в SQL-1999 необходимо указать NOT FINAL.

При составлении спецификации пользовательского типа задают **сигнатуры** его **методов** и **механизм генерации ссылок** уникальных идентификаторов. Поддерживаются три различных механизма присваивания уникальных

идентификаторов экземплярам пользовательских типов, которые задаются как спецификации ссылочного типа:

- значения, генерируемые системой автоматически (REF IS SYSTEM GENERATED);
- значения некоторого встроенного типа SQL, которые должны генерироваться приложением при сохранении экземпляра структурного типа как строки типизированной таблицы (REF USING название_типа);
- значения, порождаемые из одного или нескольких атрибутов структурного типа (REF USING (список атрибутов)).

По умолчанию предполагается наличие REF IS SYSTEM GENERATED.

В сигнатуре метода указывают имя, по которому этот метод будет вызываться. Кроме того, можно указать точное имя метода, которое может использоваться для уникальной идентификации метода, если его вызывное имя перегружено.

Если у метода имеются какие-либо параметры, отличные от неявного параметра SELF, то в определении должен присутствовать заключенный в скобки список пар:

имя_параметра тип_параметра

Поскольку методы являются функциями, требуется указать тип возвращаемого значения. Методы могут возвращать значения любого допустимого в SQL типа, даже пользовательского типа. Методы могут быть написаны на языке SQL/PSM или на любом из языков программирования, поддержка которых предусмотрена в стандарте SQL.

Вместо внешних ключей в объектно-реляционной модели используют ссылки. Для объявления ссылочного типа используется следующий синтаксис:

REF () [SCOPE _]

где () — имя UDT типа, на экземпляры которого будут указывать значения ссылочного типа; в необязательном разделе SCOPE задается имя типизированной таблицы, на которую ссылаются.

Ссылочный тип может использоваться в качестве типа атрибута пользовательского типа данных.

Пример пользовательских типов данных

Пример 2.5. Описание пользовательских типов данных киновыпусков на языке SQL с объектным расширением.

Определим пользовательские типы для модели киновыпусков (см. рис. 1.5) на основе схем отношений объектно-реляционной модели из примера 2.4.

Определим тип адреса `Addr`, в нем определим метод `fullad`, возвращающий строку с полным адресом. Используем тип адреса при описании других типов:

```
CREATE TYPE Addr AS
(
    city CHAR(20),
    street VARCHAR(100)
)
method fullad() RETURNS VARCHAR(130);
```

Тело метода описываем отдельно по следующему образцу:

```
CREATE method fullad() RETURNS VARCHAR(130)
FOR Addr
BEGIN
    RETURN SELF.city || SELF.street
END;
```

Указываем сигнатуру метода, тип, для которого он задан, и код тела метода. Метод вызывают для определенного кортежа, обращение к кортежу из метода через ключевое слово `self`.

Определим пользовательские типы студии `StudT` и фильма `FilmT`, в котором в качестве атрибута указана ссылка на студию:

```
CREATE TYPE StudT AS
(
    sname VARCHAR(50),
    addr Addr
);

CREATE TYPE FilmT AS
(
    title varchar(100),
    year integer,
    length integer,
    type char(20) ,
    stud REF(StudT) SCOPE Stud
);
```

Типы `AnimationT` (мультфильм) и `SerialT` (драма) наследуются от типа `FilmT` и содержат все его атрибуты и свои собственные:

```
CREATE TYPE AnimationT under FilmT AS
(
    drawers varchar(70) ARRAY[15]
);
```

```

CREATE TYPE SerialT under FilmT AS
(
    book varchar(100),
    avtors varchar(20) ARRAY[10]
);

```

В заключение определим тип актера ActorT с атрибутом fio, который является структурой и состоит из трех полей:

```

CREATE TYPE ActorT AS
(
    inn char(10),
    fio ROW ( f varchar(10), i varchar(10), o varchar(10)),
    edu varchar(15)
);

```

Типизированные таблицы и ссылки

Типизированные таблицы определяют следующей конструкцией:

```

CREATE TABLE      _      OF UDT_
[ UNDER          _      ]
[                | _      ]

```

При создании типизированной таблицы обязательно наличие раздела OF, в котором указывают имя ранее определенного структурного типа. Раздел UNDER отвечает за указание базовой таблицы, от которой данная таблица наследует все столбцы, ограничения целостности и т.д.

В определении **типизированной таблицы** разрешается указывать табличные ограничения целостности. Если определяемая таблица является производной от некоторой таблицы, то в ней не допускается определение ограничения первичного ключа (PRIMARY KEY). При определении базовой таблицы допускается спецификация PRIMARY KEY (с указанием одного или нескольких столбцов) или спецификация ограничения UNIQUE (с указанием одного или нескольких столбцов) в комбинации с указанием ограничения NOT NULL. В определении типизированной таблицы могут содержаться спецификации ссылочных ограничений целостности.

Тип ссылки задает самоссылающийся столбец, что специфицируется следующим синтаксисом:

```

REF IS      _
{ SYSTEM GENERATED | USER GENERATED | DERIVED }

```



```
CREATE TABLE Serial of SerialT under TF
( );
```

Определим типизированную таблицу Actor на основе типа ActorT и добавим в нее следующие ограничения:

- Ссылочный тип Ida, генерируемый системой, для внешних ссылок;
- Первичный ключ, состоящий из атрибута inn.

```
CREATE TABLE Actor OF ActorT
(
    PRIMARY KEY (inn),
    REF IS Ida system generated
);
```

Определим типизированную таблицу TSt на основе типа StudT и добавим в нее ссылочный тип Ids, генерируемый системой, для внешних ссылок:

```
CREATE TABLE TSt of StudT
(
    REF IS Ids system generated
);
```

Для организации связи M–M создаем обычную нетипизированную таблицу FA, содержащую ссылки на фильм и актера:

```
CREATE TABLE FA
(
    act REF(ActorT) SCOPE Actor,
    film REF(FilmT) SCOPE TF
);
```

Как альтернатива, можно создать тип для этой связи и на нем определить типизированную таблицу.

Правила сравнения пользовательских типов данных

Для операций сравнения, сортировки и удаления дубликатов необходимо для любых пользовательских типов задать правила сравнения: на равенство или полное, поэлементное или через функцию.

Сначала создадим поэлементное сравнение на равенство для объектов типа StudT:

```
CREATE ORDERING FOR StudT EQUALS ONLY BY STATE
```

где фраза CREATE ORDERING используется для задания правила сравнения; после FOR необходимо указать пользовательский тип, для которого создаем данное правило; фраза EQUALS ONLY определяет сравнение на равенство; BY STATE — поэлементное сравнение объектов данного типа.

Теперь создадим полное правило сравнения через функцию для пользовательского типа SerialT:

```
CREATE OREDERING SerialT ORDER FULL BY RELATIVE WITH Fun
```

где фраза ORDER FULL определяет полное сравнение; BY RELATIVE WITH — сравнение через функцию, следующую за WITH.

Функция сравнения должна возвращать целое число, интерпретируемое СУБД как результат сравнения объектов. Функция полного сравнения возвращает 0, если сравниваемые объекты равны; значение более нуля, если первый объект больше второго; значения меньше нуля, если первый объект меньше второго. Функция сравнения на равенство возвращает 0, если сравниваемые объекты равны и ненулевое значение в противном случае.

Определим функцию полного сравнения Fun:

```
CREATE FUNCTION Fun (IN S1 SerialT, IN S2 SerialT)
RETURNS integer
IF S1.length()<S2.length() THEN RETURN (-1)
ELSEIF S1.length()>S2.length() THEN RETURN (1)
ELSEIF S1.year()<S2.year() THEN RETURN (-1)
ELSEIF S1.year()>S2.year() THEN RETURN (1)
ELSEIF S1.title()<S2.title() THEN RETURN (-1)
ELSEIF S1.title()>S2.title() THEN RETURN (1)
ELSE RETURN(0)
ENDIF;
```

Мы определили функцию Fun, в которой сначала сравниваем драмы по длительности серии (length), затем по году выпуска (year) и названию сериала (title). Если по всем этим полям кортежи совпадают, то результатом будет 0, что означает, что кортежи эквивалентны.

Вопросы для самопроверки

1. Что такое UDT тип и как его описать?
2. Как определить и описать метод для типа?
3. Что такое типизированная таблица и как ее описать?
4. Как задается наследование? Где определяют ключевые поля?
5. Как реализуют связи 1–М и М–М?
6. Зачем нужны правила сравнения типов? Как их задать?
7. Как задают ссылки?

2.3. Объектная модель

Языки и компоненты объектной модели

Объектный подход к созданию баз данных ориентирован на объединение возможностей объектно-ориентированного языка программирования и базы данных для совместной работы. Он предполагает постоянное хранение объектов программы в базе данных и формирование запросов к базе данных на языке написания программы.

В соответствии со стандартом ODMG (object database management group) выделяют следующие компоненты объектного подхода к построению баз данных:

- ODL (object definition language) — язык определения объектов, используется для описания объектной БД;
- OML (object manipulation language) — язык манипулирования объектами, используется для работы с объектами;
- OQL (object query language) — язык объектных запросов для построения запросов к БД;
- объектная модель;
- хранилище объектов;
- инструментальные средства для создания баз данных и библиотеки.

Язык ODL — абстрактный язык, не имеющий практической реализации. Он определяет типовые конструкции для создания объектной базы данных и ее компонентов и задает стандарт описания интерфейсов объектной БД. При создании реальной БД применяют объектно-ориентированный язык высокого уровня, который должен соответствовать данному стандарту и реализовывать возможности языка ODL.

Язык OML — объектно-ориентированный язык высокого уровня, на котором пишут программу и который имеет конструкции по созданию, изменению и удалению объектов. **Язык OQL** — расширение объектно-ориентированного языка высокого уровня, которое позволяет включать в код программы запросы к объектной базе данных. Языки OQL и ODL задают стандарт и определяют возможности по формированию объектных запросов. Их практическая реализация зависит от конкретного языка программирования.

Объектная модель определяет ряд терминов и понятий, которые описывают основные концепции объектного подхода и объединяют все остальные компоненты в единую систему. Хранилище объектов — объектная СУБД, реализующая как функции СУБД второго поколения, так и возможности работы с объектами. Инструментальные средства предназначены для описания структур объектных баз данных и управления

объектными СУБД (ОСУБД). Библиотеки обеспечивают взаимодействие программ и ОСУБД. С 70-х по 90-е годы XX века разработчики программных приложений активно использовали реляционные базы данных и программы на различных языках для обращения к ним. С появлением объектно-ориентированных языков программирования появилась необходимость длительного хранения объектов, создаваемых и используемых в программах, в базах данных.

Объектные базы данных и объектные СУБД, их возможности и свойства определены стандартами, вырабатываемыми консорциумом OMG (object data group, ранее ODMG – object database management group). Наиболее распространены стандарты ODMG-93 и ODMG-2003. *Объектная модель баз данных* соответствует парадигме объектно-ориентированных языков программирования, которые поддерживают инкапсуляцию, наследование и полиморфизм. В объектной базе данных хранят объекты классов. Класс объекта определяет его структуру, включая атрибуты, методы и связи.

Объектная СУБД обладает следующими возможностями:

- поддерживает сложную систему типов, в том числе атомарные, структуры, коллекции и ссылки и предоставляет средства для их описания;
- разделяет работу с литералами (неизменяемыми значениями любых типов) и объектами классов;
- обеспечивает идентификацию объектов;
- позволяет определять иерархии классов и интерфейсов.

Концепции построения объектной модели

Рассмотрим основные положения **объектной модели**.

В объектной базе данных хранят объекты. *Объект* – экземпляр некоторого класса. *Класс* — специальный тип данных, называемый также объектным типом. Он задает интерфейс относящихся к нему объектов. Объект имеет атрибуты, методы и связи. Их перечень, названия, сигнатуры и типы указаны в описании класса. Атрибуты и связи называют свойствами, а свойства и операции — характеристиками объектного типа или объектов данного типа.

Объектные типы поддерживают наследование, инкапсуляцию и полиморфизм. Каждый объект имеет *уникальный идентификатор объекта* — OID (object identifier), который создается и поддерживается системой автоматически. OID скрыт от пользователя и выполняет роль первичного ключа. В некоторых случаях объекты можно однозначно идентифицировать значениями заданного набора его свойств (атрибутов,

связей или методов). Такие свойства объявляют *ключевыми*. Допускается явно указывать ключи, в том числе составные или альтернативные, при определении класса. Система поддерживает уникальность всех ключей класса. Значения данных разделяют объекты и литералы. Объект может изменять значение и характеризуется своим OID. *Литерал* – константа, которая может иметь сложную структуру, но не может изменять значение. Если значение литерала изменить, то получим другой литерал.

Связи классов определяют между объектными типами. В объектной модели поддерживаются только бинарные связи, т. е. связи между двумя типами.

Связи могут быть разновидностей 1–1 (один-к-одному), 1–М (один-ко-многим) и М–М (многие-ко-многим) в зависимости от того, сколько экземпляров соответствующего объектного типа может участвовать в связи. Для реализации N -арной связи ($N > 2$) создают отдельный класс, содержащий ссылки на объединяемые классы.

Экстент — множество объектов данного типа в базе данных. Если класс A является производным классом от базового класса B , экстент A — подмножество экстента B . Объект производного класса также является объектом базового класса. Если объект – аналог кортежа, экстент – аналог таблицы. *Класс* — объектный тип, на основе которого можно создавать объекты. Для класса допускается указывать экстент и ключи. *Интерфейс* — абстрактный объектный тип, на основе которого нельзя создавать объекты. Его использует при формировании иерархии классов. Интерфейс не имеет экстента и ключей.

Классы образуют ациклический граф на основе *наследования*. Все операции и свойства супертипа наследуются подтипом, возможно множественное наследование.

Супертип — базовый класс, от которого наследуется текущий класс, называемый производным или *подтипом* (подклассом). Подтип может переопределять, уточнять свойства и операции супертипа, вводить новые свойства и операции. Механизм наследования от интерфейсов называют наследованием IS-A (как есть), а механизм наследования от классов — расширением (extends). При порождении подкласса от некоторого суперкласса подкласс наследует экстент и набор ключей суперкласса.

Атомарные и составные типы данных

Объектная модель данных поддерживает две категории значений: объекты и литералы. Литералы являются константами (например, {"Hello", {1, 2, 3, 4}}).

Среди литеральных типов данных можно выделить атомарные и составные типы. *Атомарные* — стандартные типы, например: `integer`, `float`, `string`, `boolean` и т. д. К *составным* относят структуры и коллекции. Их описание на языке ODL приведено далее:

`Set <T>` — множество, где `T` — некоторый тип данных;

`Bag <T>` — мультимножество (в значениях которого допускается наличие повторяющихся элементов), где — некоторый тип данных;

`List <T>` — упорядоченный список значений (среди них допускаются дубликаты), где `T` — некоторый тип данных;

`Array < , N>` — массив с заранее определенным N количеством элементов, где `T` — некоторый тип данных;

`Dictionary <S, T>` — словарь, поименованный массив элементов (обращение к элементам происходит по именам), где `S` — тип ключа, `T` — тип значения;

`Struct { , 2 2, ... }` — структура, обращение идет по именам полей. У каждого поля свое название.

Описание классов на языке ODL

Язык ODL используют для описания интерфейса базы данных, а не для программирования ее реализации. Реализация базы данных выполняется на языке OML.

Описание объектного типа (класса) состоит из следующих элементов:

- имя (название) класса;
- набор супертипов (базовых классов или интерфейсов);
- имя поддерживаемого системой экстенда (`extent`);
- один или более ключей (`key` или `keys`);
- набор атрибутов, каждый из которых может быть объектом или литеральным значением;
- набор связей, каждая из которых указывает на некоторый другой объект или множество объектов;
- набор методов, определяемых своими сигнатурами.

Для описания класса на языке ODL используют конструкцию вида:

```
class (extent — key )
{
```

```
};
```

Для возможности создания объектов класса обязательно указывают его экстент. Ключи указывать не обязательно, поскольку по умолчанию создается OID, который назначается всегда, даже если объявлен свой ключ.

Составной ключ задают перечислением полей внутри скобок

```
key ( 1 , 2 , ... )
```

где — атрибут, метод или связь данного класса.

Альтернативные ключи задают через запятую

```
key 1 , 2 , ...
```

Объектный тип можно определить с помощью двух разных синтаксических конструкций языка ODL: `interface` и `class`.

Определение класса отличается от определения интерфейса наличием необязательных разделов: `extends` (наследование), `extent` (экстент) и `key` (ключ(и)). В остальном описание класса и интерфейса совпадают:

```
interface
{

};
```

Наследование класса А от класса В задают с помощью ключевого слова

```
Class A extends B (...) { ... }
```

Множественное наследование задают перечислением базовых классов через запятую

```
Class A extends B: C, D, E (...) { ... }
```

Только первый из базовых типов может быть классом (В), остальные (С, D, E) — только интерфейсами.

Определение атрибутов, связей и методов

Для задания атрибута указывают ключевое слово `attribute`, его название и тип, например:

```
Class      ( extent      key ( , ) )
{
attribute string      ;
attribute integer    ;
```

```

attribute Bag<string>          ;
attribute Struct{ string      , List<string>
}
}

```

Типом атрибута может быть любой составной или атомарный тип. Для задания связи используются ключевые слова:

`relationship` — ключевое слово для определения связи;

`inverse` — ключевое слово для отображения обратной связи.

Связь обязательно прописывают в обоих классах и указывают два именованных конца. Поэтому перемещаться по ней можно в обоих направлениях (в отличие от ссылок в объектно-реляционной модели, по которым можно перемещаться только в одном направлении), кроме того, необходимо отследить соответствие описаний связей в обоих классах.

Пример описания связи:

```

relationship          1
inverse      ::      2

```

где — класс, с которым данный класс связан; `1` — поле текущего класса, являющегося ссылкой на объект другого класса; `2` — поле связанного класса, являющегося ссылкой на объект данного класса.

Если ссылка указывает на один объект класса, то записывают название класса. Если ссылка указывает на множество объектов, то это коллекция ссылок и записывают конструктор коллекции, примененный к типу класса, например, `Set <тип>`. Допускается использовать любую коллекцию, один раз примененную к классу.

Рассмотрим пример определения связи типа 1–М между классами `Фильм` и `Студия`:

```

class      (... )
{
.....
relationship      made
inverse      :: owns;
}

class      (... )
{
...
relationship Set<      > owns
inverse      :: made
}

```

Фильм может быть снят (made) на одной студии, студии может принадлежать (owns) множество различных фильмов. Прямая связь made описывается в классе Фильм, обратная связь owns описывается в классе Студия.

При определении класса задают сигнатуры его методов (операций). Сигнатура содержит название метода, тип возвращаемого значения, перечень параметров и перечень исключений, например:

```
string GetActorName(in integer RoleTime)
                    raises (noAct,manyAct)
```

где raises — перечисляет исключения, вызываемые методом.

При описании параметров метода указывают способ их передачи:

In — по значению (входной), по умолчанию;

Inout — по ссылке;

Out — по ссылке (выходной).

Тело метода описывают при реализации в среде программирования.

Правила преобразования модели сущность–связь в объектную модель

При переходе к объектной модели БД элементы модели сущность–связь преобразуют по следующим правилам:

- сущность — в класс;
- атрибут сущности — в атрибут класса;
- ключ сущности — в ключ класса;
- бинарная связь (1–M, 1–1 или M–M) — в связь классов, причем в каждый класс добавляется одна ссылка или коллекция ссылок на связанный класс, в зависимости от того, со сколькими объектами другого класса устанавливается эта связь;
- N-арная связь ($N > 2$) — в новый класс, который содержит по одной ссылке на каждый класс, участвующий в связи, а те классы будут содержать коллекции ссылок на новый класс;
- слабая сущность и поддерживающие ее связи – в класс, содержащее атрибуты слабой сущности и ссылки (связи) на все поддерживающих ее классы. Ее ключом будет совокупность собственного ключа и этих ссылок;

- связь ISA — в иерархию наследования классов, причем, базовая сущность станет суперклассом, а производная — подклассом.

Примеры преобразования модели сущность–связь в объектную модель

Пример 2.7. Построение объектной модели базы данных киновыпусков на языке ODL.

В результате преобразования модели сущность–связь киновыпусков (см. рис. 1.5) в объектную модель построена система классов:

```
class Film(extent Films key(name, year))
{
  attribute string name;
  attribute integer year;
  attribute integer len;
  attribute enum Ftype {bw, color} type;
  relationship Studia stud
    inverse Studia::fs;
  relationship Set<Actor> acts
    inverse Actor::infs;
  integer ActCount() raises(noActors);
  void FilmsInYear(in integer year, out Set<Films>)
  raises(noFilms, badYear);
};

class Studia(extent Studies)
{
  attribute string sname;
  attribute Struct Addr
    {
      string city,
      string street
    } addr;
  relationship Set<Film> fs
    inverse Film::stud;
};

class Actor(extent Actors
  key inn)
{
  attribute string inn;
  attribute string fio;
  attribute List<string> edu;
  relationship Set<Film> infs
    inverse Film::acts;
};

class MF extends Film(extent MFS)
{
  attribute string drawer;
```

```

};

class Drama extends Film (extent Dramas)
{
  attribute Struct {
    Set<string> authors,
    string bname
  } book;
};

```

Пример 2.8. Построение объектной модели базы данных кафедры на языке ODL.

В результате преобразования модели сущность – связь кафедры (рис.1.3) в объектную модель построена система классов:

```

class Vuz(extent Vs)
{
  attribute string name;
  relationship Set<Kaf> Kafs
  inverse Kaf::vuzz;
};

class Kaf(extent Ks key(kname, vuzz))
{
  attribute string kname;
  relationship Vuz vuzz
  inverse Vuz::Kafs;
};

```

Вопросы для самопроверки

1. Сформулируйте основные понятия объектной модели.
2. Чем отличаются интерфейс и класс?
3. Как задают наследование?
4. Какие элементы содержит класс и как его определяют на языке ODL?
5. Что такое экстенд?
6. В чем отличие ключей и OID?
7. Как задаются связи 1–М и М–М в объектной модели?
8. Как объявляют сигнатуры методов и где определяют их код?

2.4. Модель полуструктурированных данных

Применение полуструктурированных данных

Преимущество баз данных с фиксированной схемой, например, реляционных, — высокая эффективность обработки и хранения больших объемов данных. Основным

преимуществом полуструктурированных баз данных является их гибкость. Модель полуструктурированных данных позволяет хранить внутри данных не только значения, но и их структуру и изменять ее со временем.

Полуструктурированные данные изображают с помощью графа. Вершины графа — информационные элементы, а дуги — связи между ними. Для линейной записи графа применяют язык XML. Для задания структуры XML-документов используют либо DTD-определения, либо XML-схемы. Примером полуструктурированных данных является программный пакет Lotus Notes.

Полуструктурированные данные применяют:

- для описания схожей информации в БД с отличающимися схемами данных при их интеграции;
- для представления документов (по аналогии с языком XML для веб-сайтов);
- для работы с унаследованными БД при укрупнении, поглощении компаний (описывают обобщенную структуру исходных БД).

Граф полуструктурированных данных

Для графического изображения полуструктурированных данных и их структуры составляют ориентированный граф. Пример графа полуструктурированных данных для модели фильмов (см. рис. 1.1) приведен на рис. 2.1.

Граф имеет следующие **категории вершин**:

- корневая вершина (нет входных дуг) только одна, символизирует всю БД;
- конечная вершина (нет исходящих дуг) хранит данные атомарных типов;
- промежуточная вершина ассоциируется с объектом БД или его элементом.

Каждая *вершина* графа достижима из корня. Граф не обязательно является деревом. Каждая *дуга* имеет метку и показывает определенное соотношение соединяемых ею вершин:

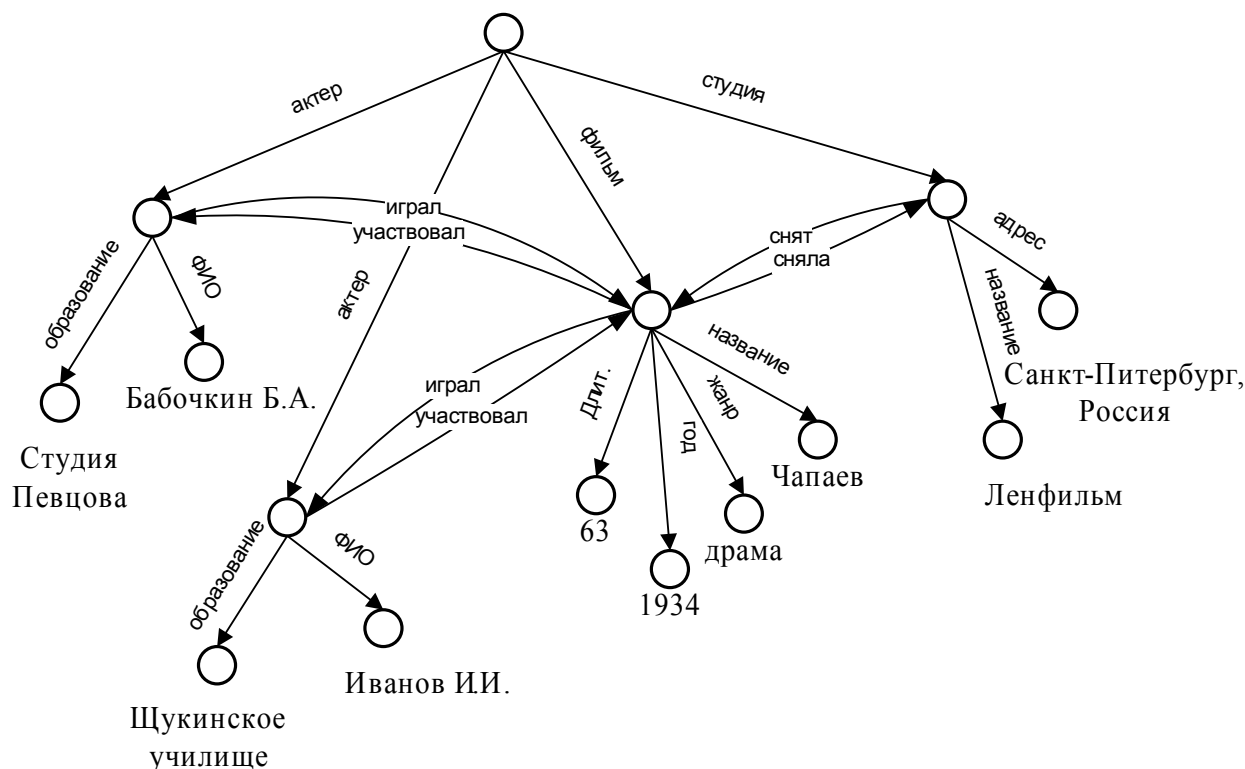


Рис.2.1. Граф полуструктурированных данных для модели фильмов

- если дуга соединяет объект (структуру) с атрибутом (полем структуры), дуга означает принадлежность атрибута объекту, метка дуги — имя атрибута (поля);
- если дуга соединяет два объекта, дуга означает связь между объектами, метка дуги — название связи.

Описание модели на языке XML

Язык XML (Extensible Markup Language) — расширяемый язык разметки. Основан на языке SGML. Документ в формате XML содержит текстовые данные и тэги. Тэги формируют семантическую разметку документа, определяют смысловое назначение данных.

Язык XML задает в линейной форме полуструктурированные данные, тэги XML-метки дуг графа полуструктурных данных.

Пример документа на языке XML (XML-документа):

```

<?xml version="1.0" standalone="yes" encoding="
Windows-1251" ?>
<book>
<avtor> <fio>          </fio> </avtor>
<izdanie city="Mos">          </izdanie>
<bestbook/>

```

```
<dt year="2000" />
</book>
```

Рассмотрим синтаксис подробнее. XML-документ состоит из *элементов*. Каждый элемент задают тэгом:

```
< > </ >
```

где `< >` — начальный (открывающий) тэг; `</ >` — завершающий (закрывающий) тэг.

Название тэга соответствует названию элемента, например, элемент `fio`, содержащий Текст:

```
<fio> </fio>
```

Возможна запись, когда начальный и завершающий тэги соединены, тогда это пустой элемент (без содержимого):

```
< />
```

Например, элемент `bestbook`:

```
<bestbook/>
```

Вложенность тэгов произвольная, но строго иерархическая.

Корневой элемент ровно один на весь документ и включает все прочие элементы. Он обязателен. В примере это элемент `book`, содержащий весь документ:

```
<book> ... </book>
```

Атрибуты характеризуют элемент. Атрибуты указывают в открывающем тэге, их значения пишут в одинарных или двойных кавычках (атрибуты `city` и `year`):

```
<izdanie city="Mos"> ... </izdanie>
<dt year="2000" />
```

XML-документ начинается строкой:

```
<?xml version="1.0" standalone="yes" encoding=" Windows-1251" ?>
```

где `version` — версия документа (версии 1.0 или 1.1); `standalone` — самостоятельность документа (не требуются дополнительные файлы); `encoding` — кодировка элемента.

Комментарии указывают так:

```
<!-- комментарий -->
```

Для отображения служебных символов используют их коды:

`<` — знак `<`;

`>` — знак `>`;

& ; — знак &;

" ; — знак кавычки;

&# ; — символ с указанным кодом.

Для отображения неформатированного текста используют конструкцию:

```
<![CDATA[          ]]>
```

Для включения инструкций обработки используют конструкцию:

```
<?app . . . . ?>
```

Различают правильный и действительный XML-документы.

Правильный XML-документ (well formed):

- содержит любые авторские тэги;
- не имеет определения схемы;
- записывает в линейной форме полуструктурированный граф.

Действительный XML-документ (valid) соответствует некоторой схеме, которая задает допустимый набор тэгов, их атрибутов и их вложенность. Такой документ является средним между жесткой схемой (реляционной БД) и отсутствием схемы (полуструктурированными данными), поскольку в нем могут быть поля, которые не используются.

Определение типа документа DTD

DTD (Document Type Definition) — определение типа документа для автоматизированной обработки XML-документа. Документ в формате DTD содержит набор допустимых тегов, их атрибутов и правил вложения.

Каждый элемент XML-документа должен быть описан в DTD конструкцией вида:

```
<!ELEMENT          >
```

Содержимое элемента указывают из следующего списка:

(#PCDATA) — текст, нет вложенных элементов;

(e11, e12, e13) — содержит вложенные элементы e11, e12 и e13 по одному в указанной последовательности;

EMPTY — пустой элемент;

ANY — любое содержимое.

Если необходимо задать сложную структуру вложенных элементов, то используют правила записи вложений:

* — ноль или более раз;

- + — один или более раз;
- ? — ноль или один раз;
- | — один из указанных элементов.

Пример описания элемента , который содержит либо текст, либо набор элементов `street` и `city` (от нуля до бесконечности):

```
<!ELEMENT E ( #PCDATA | (street*,city*))
```

Если элемент содержит атрибуты, то их состав и допустимые значения задают конструкцией вида:

```
<!ATTLIST
  _1
  _2
  ... >
```

Можно определять атрибуты в одной строке или в нескольких. Тип атрибута указывают из следующего списка:

`CDATA` — текст;

(1 | 2 | 3) — одно из указанных значений;

`ID` — уникальное значение (в пределах документа);

`IDREF` (IDREFS) — ссылка на уникальное значение типа `ID` (ссылки через пробел);

`ENTITY` (ENTITIES) — название сущности/макроса (или несколько через пробел);

`NMTOKEN` (NMTOKENS) — одно слово (набор слов через пробел).

Параметры атрибутов могут быть следующие:

`#REQUIRED` — обязательный;

`#IMPLIED` — необязательный

`#FIXED` " " — фиксированный, имеет указанное значение.

Допускается указывать значение атрибута по умолчанию.

Приведем пример описания обязательного атрибута `at1` элемента `elem` (со значением по умолчанию `red`):

```
<!ATTLIST elem at1 ("red" | "green") #REQUIRED 'red' >
```

В DTD можно применять макросы. Приведем пример описания макроса с названием `hello`:

```
<!ENTITY hello system " _ _ " > — внешняя подстановка  
(тело макроса находится в указанном файле);
```

`<!ENTITY hello " " >` — внутренняя подстановка (телом макроса является указанный текст).

При обращении к макросу перед его именем пишут знак `&`:

```
&hello;
```

Для определения макросов в DTD запишем:

```
<!ENTITY % " " >
```

или

```
<!ENTITY % SYSTEM " " >
```

Для использования определенного ранее макроса запишем:

```
<!ATTLIST at2 % ; >
```

В DTD можно подключать внешние обработчики. Приведем пример определения внешнего обработчика. Объявим обработчик `nn` из файла `nn.exe`:

```
<!NOTATION nn SYSTEM "nn.exe" >
```

Зададим подключение обработчика для макроса `ff`, ссылающегося на данные в файле `ff.nn`:

```
<!ENTITY ff System "ff.nn" NDATA nn>
```

Пропишем включение макроса с обработчиком в документ:

```
<a src = "ff">
```

Задание структуры XML-документа посредством DTD

Для указания, что структура XML-документа задана посредством DTD, используют два варианта: внутренний и внешний DTD.

Вариант 1 (внутренний) — DTD помещают внутрь XML документа:

```
<?xml version = "1.0" standalone = "yes" >
<!DOCTYPE root [
<!ELEMENT root ( elems* ) >
...
]>
<root>
<elems> ...</elems>
<elems> ... </elems>
</root >
```

Вариант 2 (внешний) — DTD хранят в отдельном файле (с расширением `dtd`, например, `"root.dtd"`). В XML-документе размещают ссылку на файл DTD:

```
<?xml version = "1.0" standalone = "no">
<!DOCTYPE root SYSTEM "root.dtd">
<root > ... < root >
```

Возможно сочетание внутреннего и внешнего DTD. В этом случае доминирует внутренний DTD.

Правила построения документа на языке XML по графу

При составлении документа на языке XML-элементы графа полуструктурированных данных преобразуют по следующим правилам:

- корень (вся БД) — в корневой элемент;
- промежуточная вершина (объект) — во вложенный элемент (метка дуги станет тэгом);
- конечная вершина (атомарное значение) — в содержимое элемента (текст);
- дуга связи между объектами — в набор атрибутов тэга.

Если на графе вершина n — сущность (структура), вершина m — его атрибут (поле), то в XML помещают вложенные тэги:

```
<n> <m> </m> </n>
```

Связь между объектами задают через атрибуты, идентификаторы и ссылки на идентификаторы.

Если n и m — объекты (вершины) на графе и имеется связь между ними от n к m , то в XML помещают элементы:

```
<m mid="m1">....</m>  
<n tom="m1">....</n>
```

где mid — атрибут типа ID (идентификатор объекта m); tom — атрибут типа IDREF (ссылка на идентификатор объекта m).

В DTD-определении записывают:

```
<!ELEMENT n ... >  
<!ELEMENT m ... >  
<!ATTLIST m mid ID>  
<!ATTLIST n tom IDREF>
```

Пример 2.9. Построение документа на языке XML на базе графа полуструктурированной БД фильмов.

В результате записи на языке XML графа полуструктурированной БД фильмов (рис. 2.1) получен XML-документ:

```
<?xml version="1.0" encoding="UTF8" standalone="no"?>  
<!DOCTYPE db SYSTEM "file.dtd">  
<db>  
  <film idf="f1" toact="a1 a2" tos="s1">  
    <name>          </name>  
    <year>1934</year>
```

```

        <len>63</len>
        <type>          </type>
    </film>
    <actor ida="a1" tof="f1">
        <fio>          . . </fio>
        <edu>          </edu>
    </actor>
    <actor ida="a2" tof="f1">
        <fio>          . .</fio>
        <edu>          </edu>
    </actor>
    <studio ids="s1" tof="f1">
        <name>          </name>
        <addr>          -          ,          </addr>
    </studio>
</db>

```

DTD для рассмотренного XML-документа запишем в файл file.dtd:

```

<!ELEMENT db (film*, actor*, studio+)>
<!ELEMENT film (name, year, len?, type?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT len (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT actor (fio, edu+)>
<!ELEMENT fio (#PCDATA)>
<!ELEMENT edu (#PCDATA)>
<!ELEMENT studio (name, addr )>
<!ELEMENT addr (#PCDATA)>
<!ATTLIST film idf ID #REQUIRED toact IDREFS tos IDREF>
<!ATTLIST actor ida ID tof IDREFS>
<!ATTLIST studio ids ID tof IDREFS>

```

Корневым элементом объявим db, который содержит ноль и более фильмов и актеров и одну или более студий. Элементы len и type для фильма не обязательны. Актеры, фильмы и студии имеют идентификаторы. Фильм ссылается на множество актеров и одну студию, актер и фильм ссылаются на множество фильмов.

Схемы документов на языке XML

Кроме DTD-определений для описания структуры XML-документов применяют XML-схемы, составленные по спецификации XSD (XML schema definition).

XML-схемы имеют следующие преимущества по сравнению с DTD:

- добавлена работа со стандартными типами данных (int, bool, string и т.д.);
- позволяют накладывать ограничения на значения (например, год не может быть меньше 1500);
- сами являются XML-документами;

- позволяют задавать и использовать пространства имён.

Схемы более функциональны, чем DTD, но соответственно более громоздки и потому удобны для автоматической, а не для ручной обработки документов.

Вопросы для самопроверки

1. В чем особенность полуструктурированных баз данных?
2. Как составляют граф полуструктурированных данных? Что является вершинами и дугами?
3. Как по графу построить документ на языке XML?
4. Что такое элемент и атрибут в языке XML? В чем особенность корневого элемента?
5. Как в DTD-определении описывают элементы?
6. Как в DTD-определении описывают атрибуты? Каких типов они бывают?
7. Как в XML-документах хранят сведения о сущностях, атрибутах и связях?

Глава 3. Формирование запросов к базам данных

3.1. Запросы на языке SQL

Для формирования запросов к табличным данным используют язык SQL.

Основные категории команд и разделов языка SQL:

- язык определения данных;
- язык манипулирования данными;
- язык запросов;
- команды управления данными;
- команды администрирования данных;
- команды управления транзакциями.

Язык определения данных (data definition language – DDL) позволяет создавать и изменять структуру объектов базы данных, например, создавать и удалять таблицы.

Язык манипулирования данными (data manipulation language – DML) используют для манипулирования информацией внутри объектов реляционной базы данных посредством трех основных команд: INSERT, UPDATE, DELETE.

Язык запросов (data query language – DQL) используют для запросов на извлечение данных. Он включает всего одну команду SELECT. Эта команда вместе со своими опциями и предложениями используется для формирования запросов к реляционной базе данных.

Добавление, удаление и изменение записей

Добавление записи (кортежа) в таблицу выполняет оператор INSERT:

```
insert into      ( 1 , 2 , ... )
values(         1 , 2 , ... )
```

Если поля не указать, то необходимо перечислить все значения кортежа.

Так же добавление в таблицу результата запроса:

```
insert into      ( 1 , 2 , ... )
select ....
```

Изменение записи (кортежа) выполняет оператор UPDATE:

```
update
set      1 =      1 ,      2 =      2 , ...
where
```

Он изменяет те записи, которые удовлетворяют условию.

Удаление записей выполняет оператор DELETE:

delete where

Он удаляет те записи, которые удовлетворяют условию.

Запросы на извлечение данных

Оператор SELECT позволяет формировать запрос к базе данных. В результате выполнения этого оператора СУБД формирует результирующий набор. Оператор SELECT имеет в стандарте SQL-92 следующее формальное описание (фрагмент):

```
SELECT [DISTINCT]
|
| *
FROM   _ [AS] [ ]
|
| [AS]
|
| [WHERE ]
| [GROUP BY { _ | } . ]
| [HAVING ]
| [ORDER BY _ [ASC|DESC] ];
```

После фразы SELECT указывают список выражений, определяющий значения, формируемые запросом. В самом простом случае список выражений является списком полей таблицы.

Символ * показывает необходимость извлечения всех полей, например:

```
SELECT * FROM Actor;
```

Имя поля может быть квалифицировано именем таблицы, указываемым через точку, например:

```
SELECT Subject.name, Teacher.FIO FROM Subject, Teacher
```

Секция FROM определяет одну или несколько таблиц или подзапросов, используемых для извлечения данных.

Условия в запросах

Секция WHERE определяет условие, которому должны удовлетворять все строки, используемые для формирования результирующего набора.

Кроме стандартных операторов сравнения, таких как =, <>, >, <, >=, <= могут быть использованы следующие операторы:

BETWEEN возвращает TRUE, если значение находится в указанном диапазоне, например:

```
length BETWEEN 5 AND 10
```

IN проверяет совпадение с одним из перечисленных в списке, например:

```
Actor.FIO IN ( '          ' , '          ' ,  
              '          ' )
```

LIKE используют для контекстного поиска в текстовых полях. Возвращает TRUE для значений, совпадающих с указанной подстрокой символов, использует следующие литеры:

'_' — произвольная единичная литера (символ);

'%' — любая последовательность литер (символов),

например:

```
FIO LIKE ' % '
```

IS NULL возвращает TRUE, если значение равно NULL, например:

```
type IS NULL
```

IS NOT NULL возвращает TRUE, если значение не равно NULL, например:

```
type IS NOT NULL
```

EXISTS возвращает значение TRUE, если указанный в нем подзапрос содержит хотя бы одну строку. Например, выбрать фильмы из таблицы Film, заголовки которых присутствуют в таблице FA:

```
SELECT title, year FROM Film  
WHERE EXISTS (SELECT * FROM FA  
              WHERE fname=title )
```

UNIQUE возвращает значение TRUE, если указанный в нем подзапрос не содержит одинаковых строк.

SOME, ANY или ALL — сравнивают с одним (SOME, ANY) из коллекции или всеми (ALL), например год, меньший, чем год фильма от студии Дисней:

```
year < ANY (SELECT year FROM Film WHERE stud='Disney');
```

Или год, меньший, чем года всех фильмов от студии Дисней:

```
year < ALL (SELECT year FROM Film WHERE stud='Disney')
```

Агрегирование и группировка

В результате выполнения оператора SELECT возможно получение не только значений полей исходных таблиц, но и значений, вычисленных на основе всех или части записей исходных таблиц. Для этого используют функции агрегирования.

Язык SQL поддерживает следующие функции агрегирования:

COUNT — вычисляет количество значений столбцов, при этом игнорирует значения NULL. Допускается указывать ALL (считать все значения) или DISTINCT (считать только различающиеся значения);

COUNT (*) — вычисляет количество записей;

AVG — определяет среднее значение;

SUM — суммирует значения, отличные от NULL;

MAX — определяет максимальное значение, игнорирует NULL;

MIN — определяет минимальное значение, игнорирует NULL.

Например, вычислить количество фильмов в 2000 году, их максимальную длительность и количество различных студий:

```
SELECT COUNT(*), MAX(length), COUNT(DISTINCT stud)
FROM Film
WHERE year = 2000;
```

Вычислять агрегатные значения можно не только для всех записей таблицы сразу, но и для отдельных ее групп.

Ключевое слово GROUP BY применяют для разделения исходного множества записей на группы на основе группирующих атрибутов. Группирующие атрибуты указывают после ключевого слова GROUP BY. В каждой группе содержатся те исходные записи, для которых значения группирующих атрибутов одинаковы. В результирующий набор попадет по одному кортежу для каждой группы. Кортеж, построенный на основе группы, может содержать группирующие атрибуты, а также результат агрегирования относительно прочих атрибутов (агрегируемых). Включать в результат группировки значения не группирующих атрибутов запрещено (и невозможно, поскольку в одной группе их может быть несколько).

Приведем пример группировки. Сгруппируем с помощью конструкции GROUP BY таблицу Film по атрибуту type. В результирующий набор войдут: значение столбца type (жанр), минимальный и максимальный год выпуска для данного жанра фильмов и средняя продолжительность фильмов такого типа:

```
SELECT type, MIN(year), MAX(year), AVG(length)
FROM Film
GROUP BY type;
```

Ключевое слово HAVING оператора SELECT позволяет отбросить из рассмотрения группы, которые не удовлетворяют заданному условию. Конструкция

HAVING работает аналогично WHERE, но применяется не к исходным записям, а к полученным группам.

Для фильтрации результатов запроса из предыдущего примера используем конструкцию HAVING. В данном случае количество фильмов в каждой группе должно быть более пяти:

```
SELECT type, MIN(year), MAX(year), AVG(length)
FROM Film
GROUP BY type
HAVING COUNT(*) >5;
```

Упорядочение результирующего набора

Ключевое слово ORDER BY применяют для упорядочивания (сортировки) записей результирующего набора. Поля, по которым выполняется сортировка, указывают после конструкции ORDER BY через запятую. Допускается указывать ASC (сортировка по возрастанию, используется по умолчанию) или DESC (по убыванию).

Например, сортировка по названию студии в обратном порядке:

```
SELECT sname, addr
FROM Stud
ORDER BY sname DESC;
```

Соединение таблиц

В операторе SELECT после ключевого слова FROM указывают источники записей для выполнения запроса. Это одна или более таблиц, подзапросов или представлений, или их сочетание. Источники указывают списком через запятую, для каждого источника можно указать псевдоним (локальное имя, действующее в пределах данного запроса), а для подзапросов псевдоним обязателен.

При соединении таблиц с одноименными столбцами именно псевдонимы (алиасы) позволяют различать одноименные атрибуты (столбцы) разных источников, например, выбрать названия фильмов с совпадающими годами и длительностью:

```
SELECT m1.title, m2.title, m1.year, m1.length
FROM Film m1, Film m2
where m1.year= m2.year AND m1.length=m2.length;
```

Если после FROM перечислено несколько таблиц или подзапросов, то все эти таблицы (их кортежи) соединяются. Будут созданы и помещены в результирующий набор кортежи с новой схемой, содержащие поля кортежей первого источника и поля кортежей второго источника (при соединении двух таблиц, при большем количестве источников аналогично). Строка результирующего набора будет являться конкатенацией строк исходных таблиц.

Если не указано иное, то соединение источников происходит по принципу декартового произведения (каждая запись каждого источника конкатенируется с каждой записью прочих источников).

После FROM можно использовать следующие операторы соединений:

CROSS JOIN — декартовое произведение;

NATURAL JOIN — естественное соединение;

INNER JOIN — внутреннее соединение, используется по умолчанию;

LEFT JOIN [OUTER] — левое внешнее соединение;

RIGHT JOIN [OUTER] — правое внешнее соединение;

FULL JOIN [OUTER] — полное внешнее соединение.

Следующие два оператора эквивалентны:

```
SELECT Subject.name, Teacher.FIO
FROM Subject, Teacher;
```

и

```
SELECT Subject.name, Teacher.FIO
FROM Subject CROSS JOIN Teacher;
```

Декартовое произведение создает результирующий набор со всеми возможными комбинациями строк источников и позволяет выполнить промежуточное сочетание данных для дальнейшей фильтрации.

При *естественном соединении* конкатенируются только те строки соединяемых таблиц, которые имеют одинаковые значения во всех одноименных атрибутах.

При *внутреннем соединении* конкатенируются только те строки, значения которых по соединяемым атрибутам совпадают. Соединяемые атрибуты указывают явно после ключевого слова ON. Строки исходной таблицы, не имеющие пары для конкатенации в прочих таблицах, участвующих в соединении, отбрасываются.

При *внешнем левом соединении* в результирующий набор будут выбраны все строки из левой таблицы (указываемой первой), независимо от наличия для них пары во второй таблице. Если во второй таблице имеются строки, в которых значения

соединяемых атрибутов равны значениям их же в первой таблице, то они конкатенируются с соответствующими строками первой таблицы. В противном случае вместо значений полей из второй таблицы в результирующий кортеж проставляются значения NULL.

При *внешнем правом соединении* в результирующий набор будут выбраны все строки из правой таблицы (указываемой второй). При наличии совпадающих значений в соединяемых атрибутах кортежи первой таблицы добавляются в результирующий набор в соответствующие строки (конкатенируются со строками второй таблицы). При отсутствии совпадений вместо значений полей из первой таблицы в результирующий кортеж заносятся значения NULL.

При *полном внешнем соединении* в результирующий набор попадут все строки, как из правой, так и из левой таблицы. При совпадении значений по соединяемым атрибутам в исходных таблицах результирующий кортеж будет содержать значения из левой и из правой таблиц. При отсутствии совпадений вместо отсутствующей части (левой или правой) результирующего кортежа заносятся значения NULL.

Внутреннее и все виды внешнего соединения можно использовать при естественном соединении (по одноименным атрибутам) или при соединении по условию.

Соединение по условию выполняют с применением ключевого слова ON. После него записывают условие, на основе которого выбираются кортежи источников, подлежащих конкатенации. В результирующий набор выбираются только кортежи, удовлетворяющие заданному условию. Этот способ соединения аналогичен соединению по условию, указываемому после WHERE, например, выберем названия и годы выпуска фильмов и имена игравших в них актеров:

```
SELECT m.title, m.year, r.act
FROM Film m JOIN FA r
ON m.title= r.fname AND m.year=r.fyear;
```

Операции над множествами

Объединение двух запросов записывают так:

```
(select ... ) UNION (select ... )
```

Пересечение двух запросов записывают так:

```
(select ... ) INTERSECT (select ... )
```

Разность двух запросов записывают так:

```
(select ... ) EXCEPT (select ... )
```


Во всех случаях набор полей запросов (количество и типы) должны совпадать.

Примеры запросов на языке SQL

Пример 2.10. Построение запросов на языке SQL к реляционной БД фильмов.

Предположим, имеется реляционная БД фильмов, содержащая схемы отношений (см. пример 1.6):

```
Actor (inn, fio, edu);
Film (title, year, length, type, stud);
Stud (sid, sname, addr);
FA (act, fname, fyear).
```

Составим конструкции на языке SQL для выборки данных из таблиц. Для этого необходимо выполнить следующие действия.

Выбрать фильмы, снятые в 60-е года:

```
SELECT *
FROM Film
WHERE year >= 1950 AND year < 1960;
```

Выбрать фильмы (в которых снялся хоть один актер и название студии оканчивается на *s* или *a*) с указанием их длительности в часах и с сортировкой по названию и году:

```
SELECT title, year, len/60 AS HOUR
FROM Film
WHERE (stud LIKE '%s' OR stud LIKE '%a')
AND EXISTS (SELECT * FA WHERE fname = title and fyear = year)
ORDER BY title, year DESC;
```

Фильмы студий Лос-Анджелеса:

```
SELECT name, year, sname
FROM Film JOIN Stud ON stud = sid
WHERE addr = 'Los Angeles';
```

Актёры фильма «The Matrix»:

```
SELECT fio
FROM Actor JOIN FA ON inn = act
WHERE fname = 'The Matrix';
```

Актёры, которые не снимались ни в одном фильме:

```
SELECT fio
FROM Actor
WHERE inn NOT IN (SELECT act FROM FA);
```

Актёры, которые снялись хотя бы в одном фильме:

```
SELECT fio
FROM Actor
```

```
WHERE inn IN (SELECT act FROM FA);
```

Актёр, игравший во всех фильмах:

```
SELECT *
FROM Actor
WHERE NOT EXISTS
(
    (
        SELECT title, year
        FROM Film
    )
    EXCEPT
    (
        SELECT title, year
    FROM FA
        WHERE FA.act = actor.inn
    )
)
```

Получить список по фильмам: название, количество актеров в фильме с сортировкой по названию фильма:

```
SELECT fname, COUNT(act)
FROM FA
GROUP BY fname
ORDER BY fname;
```

Вопросы для самопроверки

1. Назовите операторы SQL для добавления, удаления и изменения записей таблицы.
2. Как составить запрос на выборку? Из каких частей он состоит?
3. Что указывают в секции FROM?
4. Что указывают в секции WHERE?
5. Что указывают в секции SELECT?
6. Как выполнить группировку и каков ее результат?
7. Какие способы соединения таблиц бывают?

3.2. Запросы на языке SQL с объектным расширением

Сложные типы данных

Объектно-реляционные СУБД (ОРСУБД) для построения запросов к объектно-реляционным БД используют объектное расширение языка SQL (стандарты SQL-1999,

SQL-2003 и старше), и при этом поддерживают обратную совместимость с реляционными БД.

В языке SQL с объектным расширением применимы все конструкции реляционной версии SQL и предоставлены дополнительные возможности:

- работа с коллекциями и структурами;
- создание хранимых функций и процедур;
- переход по ссылкам.

Рассмотрим способы описания составных типов данных: массива, мультимножества и структуры и обращения к ним из запросов.

Массив — упорядоченная коллекция значений некоторого типа (атомарного или составного), к элементам которой можно обращаться по номеру (индексу). Массив задают оператором ARRAY, например:

```
create table tab
(
...
X int ARRAY []
);
```

где — атрибут таблицы tab, являющийся массивом целых значений.

Для вставки значения в поле массива указывают константу в виде массива значений, например:

```
INSERT INTO tab (x) VALUES ( ARRAY [10, 20, 30] )
```

где ARRAY [10, 20, 30] — массив константных значений.

Для обращения к элементу массива используют квадратные скобки с указанием индекса элемента:

```
SELECT x[1] FROM tab;
```

Для перевода массива в набор значений используют оператор UNNEST, например:

```
SELECT r.id, a.name FROM tabl as r, UNNEST (tabl.x) as a(name).
```

Мультимножество — множество значений (с повторяющимися элементами) некоторого типа (атомарного или составного). Мультимножество задают оператором MULTISSET, например:

```
CREATE TABLE tab
(
...
Y int MULTISSET;
)
```

где Y — атрибут таблицы tab, являющийся мультимножеством целых чисел.

Для формирования значений мультимножества применяют следующие конструкции:

`multiset ()` — пустое множество;

`multiset (1,3,7,8)` — перечисление констант;

`multiset (select col from tab where...)` — данные указанного запроса.

Приведем пример вставки значения в поле, являющееся мультимножеством:

```
INSERT INTO tab (y) VALUES (MULTISET ( 1,3,7,8 ) )
```

Для работы с мультимножеством (коллекцией) применяют следующие функции:

`CARDINALITY ()` — вычисляет количество элементов;

`SET ()` — преобразует коллекцию в множество;

`ELEMENT ()` — преобразует коллекцию из 1 элемента в элемент;

`UNNEST () as tab (c1, c2,...)` — преобразует коллекцию в виртуальную таблицу `tab (c1, c2, ...)`.

С мультимножествами можно выполнять операции:

- объединение `MULTISET UNION`;
- пересечение `MULTISET INTERSECT { DISTINCT ALL }`;
- разность `MULTISET EXCEPT`.

Например, объединим два мультимножества без удаления дубликатов:

```
_1 MULTISET UNION ALL _2
```

Для сравнения мультимножеств используют конструкции:

- значение `[NOT] MEMBER [OF]` — проверяет вхождение элемента в коллекцию;
- `_1 [NOT] SUBMULTISET [OF] _2` — проверяет вхождение коллекции `_1` в коллекцию `_2`;
- `IS[NOT] A SET` — проверяет, что мультимножество (не) является множеством

Структура — набор именованных полей различных типов.

Структуру задают оператором `ROW`, например:

```
create table tab
( ...
z ROW ( 1 int [ ], 2 char(10) [ ], ... )
)
```

где Z — атрибут таблицы tab , являющийся структурой и состоящий из полей: 1 , 2 и т. д. с указанием типов полей.

При вставке значения в поле, являющееся структурой, значения полей записывают в круглых скобках через запятую, например:

```
INSERT INTO tab(z) VALUES ( (10, 'text') );
```

Приведем пример таблицы, содержащей атрибуты сложных типов:

```
create table (  
    c1 int,  
    c2 int multiset,  
    c3 row ( F1 int, F2 int )  
);
```

Для вставки записи в эту таблицу запишем:

```
INSERT INTO tab(c1,c2,c3)  
VALUES ( 10, MULTISSET(1,2,2,3), (100,200) )
```

Составим запрос для выборки данных из таблицы:

```
SELECT c1, CARDINALITY(c2), 3.F1, c3.F2 FROM tab
```

и получим значение атомарного поля 1 , количество элементов коллекции 2 и поля структуры 3 .

Запросы к типизированным таблицам

Типизированные таблицы содержат в качестве кортежей объекты пользовательских типов. При выполнении запросов к связанным типизированным таблицам вместо их соединения используют переход по ссылке. Поскольку переход по ссылке возможен только в одном направлении, в секции FROM запроса указывают таблицу, содержащую ссылку, а в секции SELECT — ее поля и поля связанной таблицы.

В стандарте SQL-1999 операция «->» называется операцией разыменования или операцией перехода по ссылке. Ссылочные значения можно трактовать как указатели на строки типизированных таблиц. Выполнение операции разыменования фактически приводит к выполнению соединения таблиц, при этом в запросе столбец связанной таблицы становится «видимым».

Само значение ссылки смысловой нагрузки не несет, поэтому в запросе используется только для перехода. В запросе допускаются многократные переходы по ссылкам.

Пример 2.11. Построение запросов на языке SQL с объектным расширением к объектно-реляционной БД фильмов.

Предположим, имеется объектно-реляционная БД фильмов, содержащая пользовательские типы и типизированные таблицы (см. примеры 2.5 и 2.6). Составим конструкции на языке SQL с объектным расширением для выборки данных из типизированных таблиц:

Запрос к таблице фильмов на получение данных о фильме Матрица и его студии демонстрирует переход по ссылочному полю stud, обращение к полям структуры (адрес студии) через точечный синтаксис и вызов метода:

```
SELECT year, length, stud->sname, stud->addr.city, stud->addr.fullad()  
FROM TF  
WHERE title = 'The Matrix'
```

Если в запросе нужно указать таблицы, связанные связью М–М, запрос выполняется к таблице, реализующей эту связь. Построим запрос на выборку фильмов (название, год, студия), где снимался Иванов:

```
SELECT film->title, film->year, film->stud->name  
FROM FA  
WHERE act->fio='      '
```

Можно использовать функцию Deref(ссылка), она возвращает все значения полей кортежа, на который указывает ссылка, без его уникального идентификатора.

Приведем пример разыменования ссылок – запрос выведет всех актёров фильма матрица:

```
SELECT deref( act )  
FROM FA  
WHERE film->title='The Matrix'
```

Приведем пример группировки и агрегирования – получить список фильмов (название, количество ролей в фильме) с сортировкой по названию фильма:

```
SELECT film->title(), COUNT(*)  
FROM FA  
GROUP BY film->title, film->year  
ORDER BY film->title;
```

Автоматически генерируемые методы

Поскольку кортеж типизированной таблицы является объектом и доступ к его полям скрыт, то для каждого его атрибута система автоматически генерирует два метода, имена которых совпадают с именем атрибута:

- обозреватель — вызывается без явных параметров и выдает значение указанного атрибута;

- модификатор — вызывается с одним явным параметром – значением атрибута, и этот вызов приводит к тому, что значение атрибута заменяется новым значением.

Нельзя обращаться к атрибутам по конкретным именам, вместо этого используют метод-обозреватель (он вызывается при обращении к атрибуту):

```
SELECT title()
FROM TF
WHERE type() < > '      '
```

В данном примере `type()` является **методом-обозревателем** и обеспечивает инкапсуляцию.

Метод-модификатор используют для изменения значения атрибута картежа.

Существуют **методы-генераторы**. Они создается системой автоматически для каждого пользовательского типа, не имеют параметров и их название совпадает с названием типа. Их вызывают при создании новых объектов пользовательского типа.

Приведем пример использования автоматически генерируемых методов для создания объекта новой студии и добавления его в типизированную таблицу БД фильмов:

```
DECLARE newA Addr;           — объявим переменные
DECLARE newS Stud;
SET newA = Addr();          — вызовем генератор для создания объекта
newA.city('      ');       — вызовем модификатор
newA.street('      , 2');
SET newS = Studia();        — вызовем генератор
newS.sname('      ');      — вызовем модификаторы
news.addr(newA);
INSERT INTO TSt VALUES(newS); — добавим объект в таблицу
```

Вопросы для самопроверки

1. Как объявить и инициализировать коллекцию (массив и множество)?
2. Как объявить и инициализировать структуру?
3. Как выполнить запрос к типизированной таблице?
4. Как перейти по ссылке в запросе?
5. Что такое разыменованная ссылка?

6. Как создать и добавить в типизированную таблицу объект пользовательского типа?

3.3. Запросы на языке OQL

Точечная нотация

Язык OQL (object query language) обеспечивает сходную с языком SQL нотацию для выполнения запросов к объектной БД из программы на объектно-ориентированном языке высокого уровня. Он задает стандарт для расширения объектно-ориентированного языка программирования функциями по извлечению данных из БД и связыванию их с переменными программы. Конкретный язык программирования, соответствующий стандарту, должен обеспечить заданные стандартом возможности, но может иметь собственный синтаксис построения запросов.

Основным оператором языка OQL является команда `SELECT` для выборки данных. Команды добавления, изменения и удаления данных в язык OQL не входят и реализуются функциями объектно-ориентированного языка программирования (языком OML).

Для доступа к компонентам переменных со сложным типом используют точечную нотацию. Общее правило имеет следующий вид: если `p` обозначает объект, принадлежащий классу `C`, а `a` — некоторое свойство класса (атрибут, связь или метод), для доступа к свойству объекта пишут `p.a`.

К примеру, имеется описание класса:

```
Class C ( extent PD )
{
  Attribute integer a;
  Relationship classB myb inverse classB :: myT;
  string fun();
  classV fun2();
};
```

В данном примере, если `p` — объект класса `C`, то в запросе на языке OQL можно записать:

- если `a` — атрибут, записываем `p.a` — значение этого атрибута в объекте `p`;
- если `myb` — связь, записываем `p.myb` — объект (или множество объектов), соединенных с `p` связью `myb`;
- если `fun()` — метод (возможно, с параметрами), пишем `p.fun()` или `p.fun(параметры)` — результат выполнения метода `fun`.

При использовании точечной нотации допустим любой уровень вложенности, например:

`p.myb.b1` — атрибут `b1` объекта класса `classB`, на который ссылается атрибут `p` по ссылке `myb`;

`p.myb.b2()` — вызов метода `b2()` объекта класса `classB`, на который ссылается атрибут `p` по ссылке `myb`;

`p.fun2().c1` — атрибут `c1` объекта класса `classV`, возвращаемого методом `fun2()`.

В точечной нотации для перехода к связанному объекту допускается указывать точку («.») или стрелку («->»), что одно и то же. В языке OQL стрелка — синоним точки.

Запрос выборки данных с условием

Язык OQL позволяет записывать выражения с помощью конструкции `SELECT-FROM-WHERE`, аналогичной известной форме запроса на языке SQL. Если исключить двойные кавычки, в которые заключена строка-константа, этот запрос будет запросом на языке SQL, а не на языке OQL.

Единственным несоответствием этих языков является то, что секция `FROM` запроса на языке SQL обычно записывается как

```
FROM Movies AS m
```

где `Movies` — таблица, `m` — ее алиас.

В языке OQL ключевое слово `AS` применяется по выбору. Это слово можно пропускать, так как выражение `Movies m` означает, что `m` — переменная, указывающая на каждый объект из экстенда `Movies`, т.е. из текущего множества объектов класса.

Приведем пример выборки всех объектов фильмов из экстенда `Films` (здесь и далее запросы построены к объектной БД киновыпусков из примера 2.7):

```
SELECT m
FROM Films m
```

В общем случае в выражение `SELECT-FROM-WHERE` входят следующие элементы:

- ключевое слово `SELECT`, за которым следует список выражений для формирования результирующего набора;

- ключевое слово FROM, за которым следует список описаний переменных: выражения для указания источника выборки (значение которого имеет тип коллекции, например множества или мультимножества); применяемого по выбору ключевого слова AS; имени переменной (для обращения к элементу коллекции);
- ключевое слово WHERE, за которым следует условие выборки элементов из коллекции.

В секции WHERE, как и в выражении, следующем за SELECT, в качестве операндов используют только константы или переменные, указанные в секции FROM.

Операторы сравнения те же, что и в языке SQL, только для оператора «не равно» применяется символ !=, а не <>.

Приведем пример выборки названий фильмов 2008 г.:

```
SELECT m.title
FROM Films m
WHERE m.year = 2008
```

При запросе к нескольким связанным экстентам вместо их соединения используют переход по ссылке. Если ссылка указывает на один объект, то используем точечный синтаксис, например, выберем названия фильмов и выпустивших их студий, расположенных в Москве:

```
SELECT f.name, f.stud.sname
FROM Films f
WHERE f.stud.addr.city = "      "
```

Если ссылка указывает на коллекцию объектов, то ссылку помещают в секцию FROM, например, выберем названия Ф.И.О актеров, игравших в фильме остров:

```
SELECT a.fio
FROM Actors a, a.infs r
WHERE r.name = "      "
```

Такой запрос можно рассматривать как выполнение вложенных циклов. Система проходит во внешнем цикле по всем объектам экстента Actors и для каждого из них выполняет вложенный цикл по объектам фильмов, на которые имеются ссылки в переменной infs.

Изменение типа результатов запроса

В языке OQL результатом выражения типа SELECT– FROM– WHERE является мультимножество или множество (если применяется оператор DISTINCT). Результат можно сделать списком и при этом задать определенный порядок его

элементов, применяя оператор ORDER BY в конце выражения SELECT-FROM-WHERE. Конструкция ORDER BY в языке OQL аналогична такой же в языке SQL.

Рассмотрим примеры получения различных коллекций данных:

- Bag<string> (мультимножество)

```
SELECT m.name FROM Films m
```

- Set<string> (множество)

```
SELECT DISTINCT m.name FROM Films m
```

- List<string> — если нужен упорядоченный список, то используется фраза ORDER BY:

```
SELECT m.name FROM Films m ORDER BY m.year
```

Необязательно, что выражения в секции SELECT являются простыми переменными. В них могут входить другие выражения, построенные конструкторами типов. Например, можно применить конструктор Struct к нескольким выражениям и получить запрос, порождающий множество или мультимножество структур (выбор ремейков фильмов):

```
SELECT distinct Struct (A1:S1, A2:S2)
FROM Films S1, Films S2
WHERE S1.name = S2.name AND S1.year != S2.year
```

Задаем названия полей A1 и A2 и получаем в результате структуру:

```
Set <Struct { Film A1, Film A2} >
```

Логические множественные условия

Язык OQL позволяет формулировать логические множественные условия, которые дают возможность проверить, удовлетворяют ли элементы x коллекции S определенному условию $C(x)$:

```
FOR ALL  $x$  IN  $S$ :  $C(x)$ 
```

результат вычисления выражения равен TRUE, если каждый элемент x в коллекции S удовлетворяет условию $C(x)$, и FALSE в противном случае (квантор всеобщности);

```
EXISTS  $x$  IN  $S$ :  $C(x)$ 
```

результат вычисления выражения равен TRUE, если существует по меньшей мере один элемент, для которого условие $C(x)$ выполняется, и FALSE в противном случае (квантор существования).

Операторы агрегирования и группировки

В языке OQL используют те же операторы агрегации, что и в языке SQL: AVG, COUNT, SUM, MIN и MAX. Но в языке SQL их применяют к столбцу таблицы, а в языке OQL — к коллекции с членами подходящего типа:

COUNT применяют к любой коллекции;

SUM и AVG — к коллекции арифметических типов,

MIN и MAX — к коллекции любых типов, которые можно сравнивать, например, к строкам.

Например, для вычисления средней продолжительности всех фильмов создадим мультимножество длительности всех фильмов:

```
AVG( SELECT m.length FROM Films m )
```

Здесь используем подзапрос, извлекающий значения продолжительности фильмов. Он порождает мультимножество продолжительностей фильмов, а применение AVG к этому мультимножеству дает желаемый результат.

Для группировки в языке OQL используют оператор GROUP BY, после которого записывают список группирующих атрибутов (или выражений) с указанием их алиасов (псевдонимов). В группирующих выражениях должны упоминаться переменные, объявленные после FROM.

Оператор GROUP BY следует в запросе типа SELECT— FROM— WHERE после части WHERE. Приведем пример выборки ФИО актеров и суммарной длительности фильмов, в которых они играли:

```
SELECT act, yr, sumlen: SUM (SELECT p.r.length FROM partition p)
FROM Films r, r.acts a
GROUP BY act: a.fio, yr: r.year
```

Для каждого фильма r из экстенда Films и связанных с ним актеров (a из r.acts), удовлетворяющих условию секции WHERE (если бы оно было указано), составим группы на основе совпадений ФИО актеров и названий фильмов. Все объекты, для которых значения группирующих выражений совпадают, помещают в одну группу. В результате выполнения группировки будет получено множество структур. Элементы этого множества имеют вид:

```
Struct(a.fio:act, r.year:yr, partition : P)
```

Все поля кроме последнего поля, обозначающего группу, содержат значения группирующих выражений для данной группы. Последнее поле имеет специальное

имя `partition` и содержит мультимножество исходных объектов, попавших в эту группу.

Из части `SELECT` запроса, содержащего оператор `GROUP BY`, можно обращаться к группирующим выражениям или всей группе, а именно `act`, `yr` и `partition`. С помощью `partition` можно ссылаться на объекты группы. Ссылаться на переменные `r` или `a` можно только внутри функции агрегирования, примененной к `partition`.

Приведем пример выборки Ф.И.О актеров и суммарной длительности фильмов, в которых они играли:

```
SELECT act, yr , sumlen: SUM (SELECT p. r. length FROM partition p)
FROM Films r, r.acts a
GROUP BY act: a.fio, yr: r.year
Partition
Bag <STRUCT {Actor a, Film r}>
```

В языке OQL за оператором `GROUP BY` может следовать оператор `HAVING` с таким же значением, как и `HAVING` в языке SQL. Оператор вида

```
HAVING < >
```

служит для устранения некоторых групп, созданных оператором `GROUP BY`. Условие относится к значению `partition` каждой группы, полученной в результате выполнения оператора `GROUP BY`. При выполнении условия эта группа передается в результирующий набор; в противном случае она не используется в результате запроса.

Приведем пример ограничения списка актеров теми, кто снимался в фильмах длительностью `>120`:

```
SELECT act, yr , sumlen: SUM (SELECT p. r. length FROM
partition p)
FROM Films r, r.acts a
GROUP BY act: a.fio, yr: r.year
HAVING MAX (SELECT p.r.length FROM partition p) > 120
```

Операции над коллекциями

К двум объектам типа множеств или мультимножеств можно применять операторы объединения, пересечения и разности, которые, как и в языке SQL, выражаются ключевыми словами `UNION`, `INTERSECT` и `EXCEPT` соответственно.

Примеры запросов на языке OQL

Пример 2.12. Построение запросов на языке OQL к объектной БД киновыпусков.

Предположим, имеется объектная БД киновыпусков, содержащая экстенды классов (см. пример 2.7). Составим запросы на языке OQL для выборки данных.

Фильмы, выпущенные до 1950г., где снимался Иванов И.И.:

```
SELECT f.name
FROM Films f, f.acts a
WHERE a.fio="      . ." AND f.year < 1950
```

Актеры, которые снимались только на студии Мосфильм:

```
SELECT a.fio
FROM Actors a
WHERE for all f in a.infs: f.stud.sname="      "
```

Получить список по студиям (название, количество фильмов, количество актеров), выпустившим более 10 фильмов, с сортировкой по названию:

```
SELECT name, cntFilm : COUNT(SELECT p.f FROM partition p),
      cntAct : COUNT(SELECT p.a FROM partition p)
FROM Studies s, s.fs f, f.acts a
WHERE COUNT(s.fs)>10
GROUP BY s.sname:name
ORDER BY name
```

Вопросы для самопроверки

1. Поясните назначение языка OQL.
2. В чем отличия синтаксиса запросов на языках SQL и OQL?
3. Как в запросе на языке OQL указать условие?
4. Как указать группировку и условие на группу?
5. Как выполняют агрегирование?
6. Как изменить тип результата?
7. Как выполнить переход по ссылке и коллекции ссылок?

3.4. Запросы на языках XPath и XQuery

Основы языка XPath

Языки XPath и XQuery используют для составления запросов к полуструктурированным данным в формате XML. Оба языка разработаны консорциумом W3C. Язык XPath (XML Path Language) — язык запросов к элементам XML документа. Предназначен для навигации по XML-документу, позволяет указать путь к элементу, условие выборки и извлекаемое значение.

Язык XQuery — язык запросов к XML-документам, включает в себя язык XPath 2.0. Кроме навигации по XML-документу позволяет генерировать новые элементы,

определять и выполнять пользовательские функции. Содержит конструкции для определения циклов, условий, сортировки, работы с переменными.

Язык XPath применим для XML-документов любой структуры. Он ориентирован на построение маршрута по графу полуструктурированных данных. Версия языка 1.0 имеет максимальную поддержку в браузерах и открытых библиотеках. Версию 2.0 широко используется с 2007г.

Каждый XML-документ может быть представлен как дерево. Элементы и атрибуты документа будут узлами дерева, а отношение вложенности элементов или принадлежности атрибутов — дугами. Такое дерево называют объектной моделью документа (DOM — document object model).

При анализе любого XML-документа строится его объектная модель. Язык XPath предназначен для перемещения по дереву и позволяет указать подмножество узлов документа, удовлетворяющих заданному условию. Кроме элементов и атрибутов узлами считают комментарии, текстовое содержимое элементов, инструкции, пространства имен.

В языке XPath выражение — запрос на языке XPath. Его результатом будет подмножество узлов XML-документа, соответствующих запросу. XPath-выражение (путевое выражение) можно представить как маршрут движения по дереву документа. Маршрут состоит из набора шагов. Каждый шаг определяет направления движения и условия на выбираемые узлы. Результат предыдущего шага (подмножество узлов) является точкой отсчета следующего шага.

Рассмотрим правила построения XPath-выражения.

Путевые выражения на языке XPath

В запросе на языке XPath указывают направление движения по дереву документа от некоторого узла. Это направление называется осью.

Указывают одну из следующих осей:

- `ancestor` — предки, то есть узлы выше текущего по иерархии вложенности;
- `parent` (родитель) — элемент, в который вложен данный;
- `self` — сам элемент;
- `child` — дети, то есть узлы, непосредственно вложенные в текущий элемент (по умолчанию);

`descendant` — потомки, то есть узлы ниже текущего по иерархии вложенности;

`ancestor-or-self` — предки или текущий;

`descendant-or-self` — потомки или текущий;

`attribute` — атрибут элемента;

`namespace` — элемент с пространством имен (с атрибутом `xmlns`);

`following` — следующие элементы после текущего по тексту документа, кроме его потомков;

`following-sibling` — элементы после текущего по тексту документа на его уровне вложенности;

`preceding` — предыдущие элементы до текущего, кроме его предков;

`preceding-sibling` — предыдущие элементы на его уровне вложенности.

Рассмотрим XML-документ, описывающий БД фильмов (см. пример 2.9). Пример путевого выражения для извлечения названия фильма по его идентификатору (атрибуту `idf`):

```
/db/film[@idf="f1"]/name
```

Это краткий синтаксис. Полный синтаксис выглядит так:

```
/ hild::db/ hild::film[attribute::idf="f1"]/ hild::name
```

Приведенное путевое выражение расшифровывается как набор шагов:

`/ hild::db/` — шаг первый, от корня документа взять дочерний элемент с названием `db`,

`hild::film[attribute::idf="f1"]/` — шаг второй, от полученного результата взять дочерний элемент с названием `film`, для которого значение атрибута `idf` равно `"f1"`,

`hild::name` — шаг третий, от полученного результата взять дочерний элемент с названием `name`.

В путевом выражении выделяют:

- шаги адресации;
- предикат;
- условие проверки узлов.

Шаги адресации задают слева направо и разделяются знаком `/`.

Шаг содержит:

- контекст — результат выполнения предыдущего шага (множество узлов). На первом шаге равен корню документа. От него вычисляют новый результат шага;
- ось (обязательна) — определяет направление движения на шаге;
- условие (обязательно) — условие выбора узлов по оси: либо имена узлов, либо знак * (все узлы);
- предикат (необязательный) — фильтр или вычисляемое выражение для фильтрации множества узлов.

В кратком синтаксисе используют следующие обозначения для осей:

descendant (потомки) — «//»;

parent (родитель) — «.»;

self (сам элемент) — «.»;

child (дети) — по умолчанию без обозначения;

attribute — атрибут элемента «@» .

Запросы к элементам возвращают их тэги и содержимое, включая дочерние элементы. Например, запрос вида

```
/root/actor
```

вернет все элементы actor, расположенные внутри элемента root:

```
<actor ida="a1" tof="f1">
  <fio> . . </fio>
  <edu> </edu>
</actor>
<actor ida="a2" tof="f1">
  <fio> . .</fio>
  <edu> </edu>
</actor>
```

Запросы к атрибутам возвращают их значения, например, запрос

```
//film/@idf
```

вернет значения атрибутов idf всех элементов film, в данном случае это одно значение f1.

Запросы к содержимому элемента возвращают текстовые значения указанных элементов без их тэгов, например, запрос:

```
//name/text()
```

вернет значения:

Примеры формирования запросов на языке XPath

Пример 2.13. Формирование запросов на языке XPath к XML-документу БД фильмов.

Приведем примеры запросов к XML-документу БД фильмов (см. пример 2.9).

Запросы к элементам:

`/db` — корень (возвращает весь документ);

`/root/film` — все фильмы (возвращает все элементы `<film>..</film>` вместе с их содержимым и тэгами);

`//film` — все фильмы (аналог предыдущего в сокращенном варианте);

`//film/name` — элементы `name` внутри элементов `film`;

`//name` — любые элементы `name` (дочерние для элементов `film` и `studio`);

`//*` — все элементы;

`/*/*/name` — элементы `name` 3-ого уровня;

`//actor | //studio` — объединение нескольких путей (актеров и студий);

Запросы к содержимому элемента:

`//name/text()` — содержимое элементов `name`;

`//fio [. = " . ."]` — элемент `fio`, значение (текст) которого равно " . .";

Запросы к элементам с условием:

`//film[1]` — первый элемент `film`;

`//film[last ()]` — последний элемент `film`;

`//fio[position() mod 2 = 0]` — каждый второй элемент `fio`;

`//film[@idf]` — все элементы `film`, имеющие атрибут `idf`;

`//film[@*]` — все элементы `film`, имеющие атрибуты;

`//film[not (@*)]` — все элементы `film`, не имеющие атрибутов;

`//film[(len or type) and name]` — все все элементы `film`, имеющие вложенные элементы `len` или `type` и `name`;

`//film[@idf = "f1"]` — элементы `film`, для которых атрибут `idf` равен `f1` (в данном случае один фильм);

`//actor[count(edu) > 3]` — элементы `actor`, имеющие более 3-х вложенных элементов `edu`;

`//*[start-with(name()), "a"]` — все элементы, названия которых начинаются с “a”;

`//film[3[year="1950"]]` — третий фильм, если он снят в 1950 году;

`//film[year="1950"][3]` — все фильмы, снятые в 1950, и взять из них третий;

Запросы к атрибутам:

`//*[@*]` — все атрибуты всех элементов (возвращает набор значений);

`//film/@idf` — значения атрибутов `idf` всех элементов `film`;

`//film[year/text()>2000]/@tos` — значения атрибутов `tos` тех элементов `film`, где вложенный элемент `year` больше 2000 (идентификаторы всех студий, снявших фильмы после 2000 года);

Запросы на соединение элементов:

`//studio[@ids = //film[year>2012]/@tos]/name` — названия студий, снявших фильмы после 2012 года (студия, идентификатор которой указана в атрибуте фильма, снятого после 2012, и взять ее название);

`//studio[not (@ids = //film[year>2012]/@tos)]/name` — названия студий, не снимавших фильмы после 2012 года (студия, такая, что не существует фильма, снятого после 2012, в атрибуте которого указан идентификатор данной студии, и взять название студии).

Функции языка XPath

В языке XPath определен набор стандартных функций. Ниже приведены некоторые из них.

Функции работы с множествами:

`node()` — множество всех узлов в т.ч. текстовых;

`text()` — текстовый узел (`//text ()` — все текстовые узлы);

`current()` — текущий элемент;

`last()` — номер последнего элемента в множестве;

`position()` — номер элемента в множестве;

`count()` — количество элементов в множестве;

`name ()` — имя первого элемента в множестве;

`name()` — имя элемента.

Логические функции:

- or — ИЛИ, and — И, not — отрицание;
- = равно, != не равно;
- > больше, < меньше, >= больше или равно, <= меньше или равно;
- boolean() — возвращает true или false;
- true() — истина; false() — ложь.

Приведем примеры условий:

```
[ count ( ) > 10 or @x = "..." ]
[ not ( ) ]
```

Числовые функции:

- + сумма, - разность, * умножение, div деление, mod остаток от деления по модулю;
- sum() — сумма элементов множества;
- round() — округление значения;
- number() — приведение значения к числу.

Строковые функции:

string(объект) — приведение к строке;

string-length(строка) — длина строки;

contains(исходная_строка, подстрока) — проверка вхождения подстроки в строку;

substring(строка, начало, количество) — извлечь подстроку начиная с символа (начало) и взять количество символов;

starts-with(строка_исходная, подстрока) — проверка начала строки с подстроки;

concat (s1, s2...) — конкатенация строк;

translate(, ,) — замена в 1-ой строке символов из букв 2-й строки на символы из 3-й строки.

Системные функции:

document(объект) — возвращает документ;

document(xml-документ) — возвращает множество узлов;

collection (набор элементов без корня) — возвращает множество узлов.

Язык XQuery базируется на языке XPath и расширяет его функционал, позволяя формировать условия, функции и вложенные циклы.

Язык XQuery используют для работы с последовательностями. *Последовательность* — упорядоченный набор объектов. *Объект* — узел или атомарное значение. Последовательности задают так: 1, 2, 3 или (1, 2, 3) или ((1, 2), 3) или 1 to 3; () — пустая последовательность.

Типы литералов соответствуют типам XML-схем, например, xs:string, xs:date, xs:time и т.д. Значения литералов (констант): 4, 4.7, "hello", true, false. Конструктор простого типа: date("2001-01-01")

Комментарии записывают так:

```
{- комментарий -}
```

Используют строчные символы для ключевых слов. Язык XQuery чувствителен к регистру.

Вызов *функций* задают как:

```
( )
```

Например, вызов функции из базовой библиотеки:

```
Substring( "hello", 1, 3 )
```

Путь определяется как XPath-выражение. Результат любого шага — последовательность узлов, указывают контекст, ось, условие выбора, предикат. Поддерживаются краткий и полный синтаксис осей.

Встроенная функция document() возвращает узел — весь документ, далее к нему можно применять путевые выражения, например:

```
document( "a.xml" )/db/actor[@ida = "a1"]/fio
```

Пространство имен дает возможность применять одноименные элементы с различным содержанием и семантикой. В языке XML пространство имен задают в тэге с помощью атрибута xmlns, например:

```
<root xmlns = ' http://my.site.ru/mynn.xsd'>
```

где http://my.site.ru/mynn.xsd — URL-адрес XML-схемы с описанием пространства имен.

Язык XQuery позволяет определять и использовать пространства имен в своих конструкциях. Пространство имен в языке XQuery объявляют так:

```
declare namespace c= " http://my.site.ru/mynn.xsd"
```

где — имя (название) пространства имен.

Для отнесения элемента или атрибута к конкретному пространству имен его имя записывают перед названием тэга, например:

```
doc( ) // : / : 2
```

Формирование запросов на языке XQuery

Основу языка XQuery составляют команды, называемые FLWOR:

`for` — цикл;
`let` — присваивание переменной;
`where` — условие;
`order by` — сортировка;
`return` — возврат значений.

Цикл (итерация) записывается следующим образом

```
for in
```

Присваивание переменной

```
let =
```

Возврат значения

```
return
```

Переменные записывают как «\$ » , например:

```
let $a=10, $b=15
```

Для подстановки вычисленного значения используют { }:

```
return { $a + $b }
```

Возврат элемента:

```
return $x
```

Возврат последовательности:

```
return {$x, $c}
```

Возврат нового элемента (будет создан элемент `act` с атрибутом `fio`, значение которого берут из переменной `$x/pers/@fio`):

```
return <act fio={$x/pers/@fio}>
```

Возврат значения:

```
return data($x/pers)
```

Приведем пример запроса на языке XQuery на извлечение Ф.И.О актеров, имеющих более 2-х образований и не проживающих в Москве (здесь и далее запросы составлены к XML-документу БД киновыпусков, см. пример 2.9):

```
for $x in document("filename.xml")/db/actor
```

```
let $c = $x/count(edu)
where $c > 2 and $a/addr != "      "
return $x
```

Приведем примеры простых циклов.

```
for $x in ( 2,3 )
return $x + 1
```

Данный цикл возвращает последовательность (3,4).

```
for $x in ( 2,3 ), $y in (5, 10 )
return { $x* $y }
```

Здесь вложенный цикл, возвращает последовательность (10, 20, 15, 30).

Язык XQuery дополнительно поддерживает *операции*:

+, -, * , div, mod, sum(), avg(), max(), min(), count()

Для задания *условия* используют оператор if -then - else, например:

```
return { if ( $x/@at= "ABC " )
          then <a1>{data($x/t2 )}< /a1 >
          else <a1/> }
```

Сравнение *идентичности* узлов XML-документа осуществляют с помощью операторов is, is not, <<, например:

\$n1 is \$ n2 — узлы \$n1 и \$n2 идентичны;

\$n1 is not \$ n2 — не идентичны;

\$1 << \$2 — n1 встречается ранее в документе, чем n2.

Для подстановки значений (констант) в XML-документ (генерации элементов «на лету») применяют *конструктор элементов*:

- из существующих узлов { } . Например, из переменной \$a взять атрибут id и вложенный элемент b:

```
{ $a/@id, $a/b }
```

- с указанием имени нового элемента {element } . Например для элемента из переменной \$e взять ее название и добавить все атрибуты, и ее содержимое, умноженное на 2:

```
element { name($e) } { $e/@*, data( $e ) *2 }
```

Пример создания элемента A с атрибутами id и n и вложенным элементом:

```
element A { attribute id { 1}, attribute n {"name"}, element B { 1234} }
```

результатом будет новый элемент:

```
<A id="1" n="name"><B>1234</B></A>
```

Для объявления функции в языке XQuery используют оператор `declare function`. При объявлении функции указывают ее название, перечень и типы параметров и тип возвращаемого значения, например:

```
declare function fun($var as xs:integer) as element(actor)*
{
    return <actor age={$var}/>
}
```

Эта функция принимает на вход целое число и возвращает ноль или более элементов `actor`.

Для вызова функции указывают ее имя и фактические параметры, например:

```
return { fun(20) }
```

Примеры запросов на языке XQuery

Пример 2.14. *Формирование запросов на языке XQuery к XML-документу БД фильмов.*

Приведем примеры запросов к XML-документу БД фильмов (см. пример 2.9), который содержится в файле `filename.xml`.

Запрос, который ищет актеров, снявшихся более чем в трех фильмах, и возвращает Ф.И.О актера, его 1-ое образование и количество его фильмов с сортировкой по Ф.И.О:

```
for $a in document( "filename.xml" )//actor
let $f:= //film[ contains($a/@tof, @fid) ]
where count($f) >3
order by $a/fio
return { $a/fio, $a/edu[1], { count ($f ) } }
```

Запрос на формирование списка Ф.И.О всех актеров:

```
< ul >
{ for $x in doc ( "filename.xml" )/fio/text ( )
order by $x
return <li>{ $x} </li> }
< ul >
```

Вопросы для самопроверки

1. Поясните назначение и базовый синтаксис языка XPath.
2. Назовите оси языка XPath и их обозначения.
3. Как вычислить путевое выражение?
4. Назовите основные функции XPath.

5. В чем состоят назначение и базовый синтаксис языка XQuery?
6. Что означает аббревиатура FLOWR?
7. Поясните работу операторов FLOWR.
8. Как генерировать новые элементы и атрибуты?

4.5. Запросы на языке Datalog

Предикаты и атомы

Язык Datalog является языком логических запросов. Он позволяет формировать запросы к таблицам реляционной базы данных в терминах предикатов. Запросы составляются к реляционным таблицам. Это альтернатива реляционной алгебре. Язык логических запросов Datalog состоит из правил вида «если-то». Рассмотрим далее базовые конструкции языка. Отношения в Datalog представляют с помощью предикатов. Каждый *предикат* обладает фиксированным количеством аргументов, предикат с аргументами называется *атомом*:

$P(x_1, x_2)$ — предикат P с аргументами x_1 и x_2 .

Предикат является аналогом вызова функций в языках программирования. В качестве аргументов могут выступать не только константы, но и переменные. Если аргументами атома являются одна или несколько переменных, атом представляет собой функцию, которая получает значения этих переменных и возвращает TRUE или FALSE.

Атом $R(a_1, a_2, \dots, a_n)$ вернет значение TRUE, если список (a_1, a_2, \dots, a_n) является кортежем отношения R , в противном случае вернет значение FALSE. Такие атомы называют *реляционными атомами*, они определяют множество кортежей, содержащееся в отношении R , которое может изменяться со временем.

Арифметический атом представляет собой оператор сравнения двух арифметических выражений, например, $(x < y)$ или $((x+1) <= (y+4*z))$. Он определяет бесконечное множество кортежей, которое неизменно. В арифметических атомах используют операторы: $>$ (больше); $<$ (меньше); $>=$ (больше или равно); $<=$ (меньше или равно); $=$ (равно).

NOT (оператор отрицания, ставится перед любым сравнением) используется во всех атомах.

Правила языка Datalog

Запросы к БД в языке Datalog описываются с помощью *правил*, каждое из которых включает в себя следующие компоненты:

- атом-заголовок, который является результатом выполнения правила;
- символ \leftarrow , который называется условием;
- тело, состоящее из одного или нескольких атомов, называемых подцелями, которые могут быть как реляционными, так и арифметическими атомами.

Подцели объединяют с помощью операторов AND и каждой из них может предшествовать оператор NOT.

Допустим, в реляционной БД хранится отношение Фильм со следующей схемой:

Фильм (название, год, жанр, описание, продолжительность, студия, количество серий)

Приведем пример запроса на языке Datalog к таблице Фильм на выборку названий и лет выпуска фильмов, выпущенных студией «Нева-Фильм»:

```
(
    ,
) <- (
    ,
    ,
    ,
    )
AND = ' - '
```

где (,) — результат выполнения запроса; (, , ,) — реляционный атом; = ' - ' — арифметический атом.

Часто в правилах некоторые переменные упоминаются однократно, имена для них особого смысла не имеют. Такие переменные называют *анонимными*. Если переменная присутствует в списке только одного атома, то ее принято заменять символом подчеркивания. Все экземпляры этого символа служат для обозначения различных переменных. Перепишем предыдущий пример:

```
(
    ,
) <- (
    ,
    ,
    ,
    )
AND = ' - '
```

Переменные жанр, описание, продолжительность, количество серий в нашем случае встречается только один раз в одном атоме и нигде более не используется и мы заменяем их на знак ().

Для того, чтобы правило позволяло однозначно определить конечное множество кортежей, составляющих итоговое отношение, необходимо соблюдать *условие безопасности*: «любая переменная, адресуемая в правиле Datalog, должна упоминаться

и в контексте некоторой реляционной подцели этого правила, не подвергаемой операции отрицания».

Пример небезопасного правила:

$$P(x, y) \leftarrow Q(x, a, b) \text{ AND NOT } R(y, z) \text{ AND } a < y \text{ AND NOT } R(y, z)$$

В этом правиле невозможно построить конечное результирующее множество, так как поле y присутствует в операциях отрицания.

Поясним, как происходит вычисление правил. Переменные из текста правила «пробегают» по множествам всех допустимых значений. Всякий раз, когда переменные получают значения, в совокупности удовлетворяющие все подцелям, определяется содержимое атома-заголовка правила для текущей комбинации значений переменных и соответствующий кортеж добавляется в итоговое отношение, на которое ссылается заголовок. Результатом вычисления правила должно служить конечное отношение.

Для вычисления итоговых отношений используют два метода:

- метод перебора всех возможных комбинаций значений переменных;
- метод проверки кортежей отношений, соответствующих реляционным подцелям без операторов отрицания, на соответствие прочим подцелям правила.

Второй метод является предпочтительным. Если кортежи, назначаемые реляционным подцелям без операторов отрицания, образуют согласованный набор, то есть, одноименным переменным присваиваются одни и те же значения, то необходимо рассмотреть значения всех переменных правила. Для любого согласованного набора следует проверить, обращаются ли все реляционные подцели с отрицанием и арифметические подцели в TRUE. Если набор данных доставляет всем подцелям TRUE, необходимо выяснить, какой вид принимает кортеж заголовка правила с учетом значений переменных, и включить этот кортеж в итоговое отношение, соответствующее предикату заголовка.

Операции реляционной алгебры на языке Datalog

Операторы реляционной алгебры могут быть преобразованы к одному или нескольким правилам на языке Datalog. Ниже приведены операторы реляционной алгебры, выраженные правилами Datalog:

- пересечение;
- объединение;
- разность;

- проекция отношения;
- выбор;
- декартово произведение;
- естественное соединение.

Для *пересечения* отношений R и S составляют правило вида:

$$I(n, a, g, b) \leftarrow R(n, a, g, b) \text{ AND } S(n, a, g, b)$$

Операция пересечения определяет отношение, которое содержит кортежи, присутствующие как в отношении R, так и в отношении S. Отношения R и S должны быть совместимы, т. е. иметь одинаковое количество полей с совпадающими типами данных. Пересечением двух таблиц R и S является таблица I, содержащая все строки, присутствующие в обеих исходных таблицах одновременно.

Для *объединения* отношений R и S составляют правила вида:

$$\begin{aligned} U(n, a, g, b) &\leftarrow R(n, a, g, b) \\ U(n, a, g, b) &\leftarrow S(n, a, g, b) \end{aligned}$$

Объединение отношений R и S можно получить в результате их конкатенации с образованием одного отношения с исключением кортежей-дубликатов. При этом отношения R и S должны быть совместимы. Объединением двух таблиц R и S является таблица U, содержащая все строки, которые имеются в первой таблице R, во второй таблице S или в обеих таблицах сразу.

Для *разности* отношений R и S составляют правило вида:

$$D(n, a, g, b) \leftarrow R(n, a, g, b) \text{ AND NOT } S(n, a, g, b)$$

Разность двух отношений R и S состоит из кортежей, которые имеются в отношении R, но отсутствуют в отношении S. Причем отношения R и S должны быть совместимы. Разностью двух таблиц R и S является таблица D, содержащая все строки, которые присутствуют в таблице R, но отсутствуют в таблице S.

Для *проекции* отношения R(n, a, g, b,) на множество атрибутов (n, a, g) составляют правило вида:

$$P(n, a, g) \leftarrow R(n, a, g, b, p)$$

Операция проекции определяет новое отношение, которое содержит подмножество отношения R, создаваемое посредством извлечения значений указанных атрибутов и исключения из результата строк-дубликатов.

Операция *выбора* — построение подмножества кортежей, обладающих заданными свойствами. Операция выборки определяет результирующее отношение,

которое содержит только те кортежи (строки) исходного отношения R, которые удовлетворяют заданному условию F (предикату).

Пример выбора фильмов, снятых на студии «Нева-Фильм»:

```
(
    ,
) <- (
    ,
    ,
    ,
    ,
    )
AND
    = '
    -
    '
```

Если в запросе выборки используются условия, соединенные оператором «или», то можно создать несколько правил, каждое из которых задает одно из указанных условий.

Для декартова произведения отношений R и S составляют правила вида:

$$I(n, a, g, b, c, w, s, k) \leftarrow R(n, a, g, b) \text{ AND } S(c, w, s, k)$$

Декартово произведение R и S двух отношений (двух таблиц) определяет новое отношение — результат конкатенации (т. е. сцепления) каждого кортежа (каждой записи) из отношения R с каждым кортежем (каждой записью) из отношения S.

Для естественного соединения отношений R и S составляют правила вида:

$$I(n, a, g) \leftarrow R(n, a) \text{ AND } S(a, g)$$

Естественным соединением называется соединение по двух отношений R и S, выполненное по всем одноименным атрибутам, из результатов которого исключается по одному экземпляру каждого одноименного атрибута.

Агрегатные функции SUM, MIN, MAX, AVG и COUNT не являются предикатами и поэтому не могут использоваться при построении запросов Datalog.

Рекурсивные запросы

Язык Datalog поддерживает рекурсивные запросы. Допустим, в БД хранят таблицу дорог с атрибутами:

$$\text{Road}(, , ,)$$

Приведем пример запроса на построение маршрутов (перемещение между городами по дорогам). Рекурсивный запрос будет иметь вид:

$$\text{Path}(,) \leftarrow \text{Road}(, , _, _)$$

$$\text{Path}(,) \leftarrow \text{Road}(, 1, _, _) \text{ AND } \text{Path}(1,)$$

где Path(,) — вычисляемое выражение, содержащее все возможные маршруты.

Рекурсивный запрос состоит из двух правил: *базы* и *индукции*. База рекурсии вычисляется один раз и содержит запрос к хранимой таблице:

$$\text{Path}(,) \leftarrow \text{Road}(, , _, _)$$

Индукция рекурсии вычисляется много раз и содержит запрос, выполняющий соединение хранимой таблицы и вычисляемого выражения:

```
Path( , ) <- Road( , 1, _, _ ) AND Path( 1, )
```

В результате выполнения индукционного правила вычисляемое выражение изменяется (в него добавляют новые кортежи). Рекурсия (индукция) выполняется, пока есть изменения вычисляемого выражения.

Выделяют следующие виды рекурсии:

- правая рекурсивная форма, если рекурсивная часть справа;
- левая рекурсивная форма, если рекурсивная часть слева;
- нелинейная рекурсия, если вычисляемый предикат применяется в правиле несколько раз.

Примеры запросов на языке Datalog

Пример 2.15. *Формирование запросов на языке Datalog к БД фильмов.*

Допустим, в реляционной БД фильмов хранят набор отношений:

```
Актер (inn, fio, edu);
Фильм (title, year, length, type, stud);
Студия (sid, sname, addr);
ФА (act, fname, fyear).
```

Приведем далее примеры запросов к БД фильмов. Фильмы, снятые на студии Мосфильм:

```
( , ) <- ( , , _, _ )
AND = " "
```

Актёры, работавшие в 1990г. на студии Мосфильм:

```
( . . ) <- ( i, . . , _ )
AND ( , 1990, i )
AND ( , 1990, _, _ , " " )
```

Актёры, никогда не работавшие на студии Мосфильм:

```
( . . ) <- ( i, . . , _ )
AND NOT ( , , i )
AND ( , , _, _ , " " )
```

Студия, снявшая более одного фильма:

```
( ) <- ( , , _, _ )
AND ( 2, 2, _, _ )
AND ( 2 != )
```

Вопросы для самопроверки

1. Поясните назначение языка Datalog.
2. Что такое реляционные и арифметические атомы?
3. Из каких частей состоит правило на языке Datalog?
4. Как составлять рекурсивные запросы?
5. Как задать условия и выборку?
6. Как реализовывать операции реляционной алгебры на языке Datalog?
7. Что такое анонимные переменные?

Литература

Баженова И.Ю. SQL и процедурно-ориентированные языки [Электронный ресурс]: Курс Интернет-университета информационных технологий . М.: Интернет-университет информационных технологий. URL: <http://www.intuit.ru/studies/courses/4/4/info>, (дата обращения 20.12.2014).

Гарсиа-Молина Г., Ульман Д., Уидом Д. Системы баз данных. Полный курс: Пер. с англ. – М.: Издательский дом «Вильямс», 2004 г. – 1088 с.

Гапанюк Ю.Е., Ревунков Г.И. Введение в XML-технологии. Учебное пособие. [Электронный ресурс] - М.: МГТУ им. Н.Э. Баумана. - 2010г.

Кузнецов С.Д. Базы данных. Модели и языки. М.: Бином-Пресс. 2008г. 720 с.

Кузнецов С.Д. Введение в модель данных SQL: Информация [Электронный ресурс]: Курс Интернет-университета информационных технологий. М.: Интернет-университет информационных технологий. URL: <http://www.intuit.ru/studies/courses/75/75/info> (дата обращения 20.12.2015).

Кузнецов С.Д. Основы современных баз данных [Электронный ресурс] URL: <http://www.ciforum.ru/database/osbd/contents.shtml> (дата обращения 20.12.2015).

Кузнецов С.Д. Три манифеста баз данных: ретроспектива и перспективы [Электронный ресурс] URL: <http://citforum.ru/database/articles/manifests/> (дата обращения 20.12.2015).

Объектно-ориентированные СУБД [Электронный ресурс] URL: <http://bourabai.ru/alg/oor12.htm> (дата обращения 20.12.2015).

Печерский А.А. Язык XML - практическое введение [Электронный ресурс] URL: <http://citforum.ru/internet/xml/index.shtml> (дата обращения 20.12.2015).

Чемберлин Дон. XQuery: язык запросов XML [Электронный ресурс] URL: <http://citforum.ru/internet/articles/xqlzxml.shtml> (дата обращения 20.12.2015).

Язык XML Path (XPath) версия 1.0. Рекомендация W3C от 16 ноября 1999 года [Электронный ресурс] URL: <http://citforum.ru/internet/xpath/index.shtml> (дата обращения 20.12.2015).

XML-схема. Часть 0: пример. Рекомендации W3C, 2 мая 2001 года. [Электронный ресурс] URL: <http://citforum.ru/internet/xml/scheme/> (дата обращения 20.12.2015).

Оглавление

Глава 1. Создание реляционных баз данных.....	5
1.1. Базы данных и системы управления базами данных.....	5
Из истории развития баз данных.....	5
Проектирование баз данных.....	6
Требования к системе управления базами данных.....	6
1.2. Модель сущность–связь.....	7
Основные элементы модели.....	8
Построение простых моделей.....	9
Дополнительные элементы модели.....	11
Пример модели сущность–связь со связью ISA.....	13
1.3. Реляционная модель.....	15
Основные понятия.....	15
Правила преобразования модели сущность–связь	16
в реляционную модель.....	16
Правила преобразования связи ISA в реляционную модель.....	18
1.4. Описание реляционной модели на языке SQL	19
Конструкции языка для создания таблиц.....	19
Ограничения целостности.....	20
Пример создания схем отношений.....	22
Глава 2. Постреляционные модели баз данных.....	25
2.1. Объектно-реляционная модель	25
Типы данных.....	25
Пользовательские типы данных.....	25
Типизированные и обычные таблицы.....	26
Ссылки на кортежи.....	28
Правила преобразования модели сущность–связь	29
в объектно-реляционную модель	29
Примеры преобразования модели сущность–связь	29
в объектно-реляционную модель.....	29
Правила преобразования связи ISA	31
в объектно-реляционную модель.....	31
2.2. Описание объектно-реляционной модели на языке SQL	32
с объектным расширением.....	32
Синтаксис определения типов данных.....	32
Пример пользовательских типов данных.....	34
Типизированные таблицы и ссылки.....	36
Пример типизированных таблиц киновыпусков.....	37

Правила сравнения пользовательских типов данных.....	38
2.3. Объектная модель.....	39
Языки и компоненты объектной модели.....	40
Концепции построения объектной модели	41
Атомарные и составные типы данных.....	42
Описание классов на языке ODL.....	43
Определение атрибутов, связей и методов.....	44
Правила преобразования модели сущность–связь в объектную модель	46
Примеры преобразования модели сущность–связь в объектную модель.....	47
2.4. Модель полуструктурированных данных	48
Применение полуструктурированных данных.....	48
Граф полуструктурированных данных.....	49
Описание модели на языке XML.....	50
Определение типа документа DTD.....	52
Задание структуры XML-документа посредством DTD.....	54
Правила построения документа на языке XML по графу	55
Схемы документов на языке XML.....	56
Глава 3. Формирование запросов к базам данных.....	58
3.1. Запросы на языке SQL.....	58
Добавление, удаление и изменение записей.....	58
Запросы на извлечение данных.....	59
Условия в запросах.....	59
Агрегирование и группировка	60
Упорядочение результирующего набора.....	62
Соединение таблиц.....	62
Операции над множествами.....	64
Примеры запросов на языке SQL.....	65
3.2. Запросы на языке SQL с объектным расширением.....	66
Сложные типы данных.....	66
Запросы к типизированным таблицам.....	69
Автоматически генерируемые методы.....	70
3.3. Запросы на языке OQL.....	72
Точечная нотация.....	72
Запрос выборки данных с условием.....	73
Изменение типа результатов запроса.....	74
Логические множественные условия.....	75
Операторы агрегирования и группировки.....	76

Операции над коллекциями.....	77
Примеры запросов на языке OQL.....	77
3.4. Запросы на языках XPath и XQuery.....	78
Основы языка XPath.....	78
Путевые выражения на языке XPath.....	79
Примеры формирования запросов на языке XPath.....	82
Функции языка XPath.....	83
Основы языка XQuery.....	84
Формирование запросов на языке XQuery.....	86
Примеры запросов на языке XQuery.....	88
.....	89
4.5. Запросы на языке Datalog.....	89
Предикаты и атомы.....	89
Правила языка Datalog.....	89
Операции реляционной алгебры на языке Datalog.....	91
Рекурсивные запросы.....	93
Примеры запросов на языке Datalog.....	94
Литература.....	96
Оглавление.....	97
.....	99
.....	100

Учебное издание

Виноградов Валерий Иванович
Виноградова Мария Валерьевна

Постреляционные модели баз данных и языки запросов

Редактор *Д. С. Пирогова*
Художник
Корректор
Компьютерная верстка

Оригинал-макет подготовлен
в Издательстве МГТУ им. Н.Э. Баумана.

В оформлении использованы шрифты
Студии Артемия Лебедева.

Подписано в печать __. __. 201___. Формат 60×90/16.
Усл. печ. л. __. __. Тираж 50 экз. Изд. №163-2015. Заказ

Издательство МГТУ им. Н.Э. Баумана.
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.
press@bmstu.ru
www.baumanpress.ru

Отпечатано в типографии МГТУ им. Н.Э. Баумана.
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.
baumanprint@gmail.com