Проектирование РБД с использованием E/R-диаграмм и диаграмм классов языка UML

С.Д. Кузнецов. Базы данных. Тема 7

План (1)

- Семантические модели данных
- Семантическая модель Entity-Relationship (Сущность-Связь)
  - Основные понятия ER-модели
  - Уникальные идентификаторы типов сущности
  - Нормальные формы ER-диаграмм
  - Более сложные элементы ER-модели
    - Наследование типов сущности и типов связи
    - Взаимно исключающие связи
  - Получение реляционной схемы из ER-диаграммы
    - Базовые приемы
    - Представление в реляционной схеме супертипов и подтипов сущности
    - Представление в реляционной схеме взаимно исключающих связей

План (2)

- Диаграммы классов языка UML
  - Основные понятия диаграмм классов UML
    - Классы, атрибуты, операции
    - Категории связей. Связь-зависимость
    - Связи-обобщения и механизм наследования классов в UML
    - Связи-ассоциации: роли, кратность, агрегация
  - Ограничения целостности и язык OCL
    - Общая характеристика языка OCL
    - Инвариант класса
    - Примеры инвариантов
    - Плюсы и минусы использования языка OCL при проектировании реляционных баз данных
  - Получение схемы реляционной базы данных из диаграммы классов UML
- Заключение

Введение (1)

- Широкое распространение реляционных (SQL-ориентированных) СУБД и их использование в самых разнообразных приложениях показывает, что реляционная модель данных достаточна для моделирования разнообразных предметных областей
- Однако проектирование реляционной базы данных в терминах отношений на основе кратко рассмотренного нами в двух предыдущих лекциях механизма нормализации часто представляет собой очень сложный и неудобный для проектировщика процесс.
- При использовании в проектировании ограниченность реляционной модели проявляется в следующих аспектах

Введение (2)

- Модель не обеспечивает достаточных средств для представления смысла данных
  - Семантика реальной предметной области должна независимым от модели способом представляться в голове проектировщика
  - В частности, это относится к проблеме представления ограничений целостности, выходящих за пределы ограничений первичного и внешнего ключа
- Во многих прикладных областях трудно моделировать предметную область на основе плоских таблиц
  - В ряде случаев на самой начальной стадии проектирования дизайнеру приходится нелегко, поскольку от него требуется описать предметную область в виде одной (возможно, даже ненормализованной) таблицы

Введение (3)

- Хотя весь процесс проектирования происходит на основе учета функциональных и других зависимостей, реляционная модель не предоставляет какие-либо формализованные средства для представления этих зависимостей
- Несмотря на то, что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области («сущностей») и выявления связей между этими сущностями, реляционная модель данных не предлагает какого-либо механизма для разделения сущностей и связей

Семантические модели данных (1)

- Потребность проектировщиков баз данных в более удобных и мощных средствах моделирования предметной области привела к появлению семантических моделей данных
- Основным назначением семантических моделей является обеспечение возможности выражения семантики данных.
- Чаще всего на практике семантическое моделирование используется на первой стадии проектирования базы данных
- В терминах семантической модели производится концептуальная схема базы данных, которая затем вручную преобразуется к реляционной (или какой-либо другой) схеме
- Этот процесс выполняется под управлением методик, в которых достаточно четко оговорены все этапы такого преобразования

Семантические модели данных (2)

- Основным достоинством данного подхода является отсутствие потребности в дополнительных программных средствах, поддерживающих семантическое моделирование
- Требуется только знание основ выбранной семантической модели и правил преобразования концептуальной схемы в реляционную схему.
- Следует заметить, что многие начинающие проектировщики баз данных недооценивают важность семантического моделирования вручную
- Зачастую это воспринимается как дополнительная и излишняя работа

Семантические модели данных (3)

- Эта точка зрения абсолютно неверна
- Во-первых, построение мощной и наглядной концептуальной схемы БД позволяет более полно оценить специфику моделируемой предметной области и избежать возможных ошибок на стадии проектирования схемы реляционной БД
- Во-вторых, на этапе семантического моделирования производится важная документация (хотя бы в виде вручную нарисованных диаграмм и комментариев к ним), которая может оказаться очень полезной
  - не только при проектировании схемы реляционной БД, но и
  - при эксплуатации, сопровождении и развитии уже заполненной БД

Семантические модели данных (4)

- Неоднократно приходилось и приходится наблюдать ситуации, в которых отсутствие такого рода документации существенно затрудняет внесение даже небольших изменений в схему существующей реляционной БД
- Конечно, это относится к случаям, когда проектируемая БД содержит не слишком малое число таблиц
- Скорее всего, без семантического моделирования можно обойтись, если число таблиц не превышает десяти, но оно совершенно необходимо, если БД включает более сотни таблиц
- Для справедливости заметим, что процедура создания концептуальной схемы вручную с ее последующим преобразованием в реляционную схему БД затруднительна в случае больших БД (содержащих несколько сотен таблиц)
- Причины, по всей видимости, не требуют пояснений

Семантические модели данных (5)

- История систем автоматизации проектирования баз данных (CASE-средств проектирования БД) началась с автоматизации процесса
  - рисования диаграмм,
  - проверки их формальной корректности,
  - обеспечения средств долговременного хранения диаграмм и другой проектной документации
- Конечно, компьютерная поддержка работы с диаграммами для проектировщика БД очень полезна
  - Наличие электронного архива проектной документации помогает при эксплуатации, администрировании и сопровождении базы данных.
- Но система, которая ограничивается поддержкой рисования диаграмм, проверкой их корректности и хранением, напоминает текстовый редактор, поддерживающий
  - ввод, редактирование и проверку синтаксической корректности конструкций некоторого языка программирования,
  - но существующий отдельно от компилятора

Семантические модели данных (6)

- Кажется естественным желание расширить такой редактор функциями компилятора, и это действительно возможно, поскольку известна техника компиляции конструкций языка программирования в коды целевого компьютера
- Но коль скоро имеется четкая методика преобразования концептуальной схемы БД в реляционную схему, то почему бы не выполнить программную реализацию соответствующего «компилятора» и не включить ее в состав системы проектирования баз данных?
- Эта идея, естественно, показалась разумной производителям CASE-средств проектирования БД

Семантические модели данных (7)

- Подавляющее большинство подобных систем, представленных на рынке, обеспечивает автоматизированное преобразование
  - диаграммных концептуальных схем баз данных, представленных в той или иной семантической модели данных,
  - в реляционные схемы, специфицированные чаще всего на языке SQL
- Может возникнуть вопрос, почему в предыдущем предложении говорится про «автоматизированное», а не про «автоматическое» преобразование?
- Все дело в том, что в типичной схеме SQL-ориентированной БД могут содержаться определения многих объектов
  - ограничений целостности общего вида,
  - триггеров и
  - хранимых процедур,

  которые невозможно сгенерировать автоматически на основе концептуальной схемы
- Поэтому на завершающем этапе проектирования реляционной схемы снова требуется ручная работа проектировщика

Семантические модели данных (8)

- Еще раз обратим внимание на то, какой ход рассуждений привел нас к выводу о возможности автоматизации процесса преобразования концептуальной схемы БД в реляционную схему
- Если создатели семантической модели данных предоставляют методику преобразования концептуальных схем в реляционные схемы, то почему бы не реализовать программу, которая производит те же преобразования, следуя той же методике?
- Зададимся теперь другим, но, по существу, схожим вопросом
- Если создатели семантической модели данных предоставляют язык (например, диаграммный), используя который проектировщики БД на основе исходной информации о предметной области могут сформировать концептуальную схему БД, то
  - почему бы не реализовать программу, которая сама генерирует концептуальную схему БД в соответствующей семантической модели, используя исходную информацию о предметной области?

Семантические модели данных (9)

- Хотя неизвестны коммерческие CASE-средства проектирования БД, поддерживающие такой подход, экспериментальные системы успешно существовали
- Они представляли собой интегрированные системы проектирования с автоматизированным созданием концептуальной схемы на основе интервью с экспертами предметной области и последующим преобразованием концептуальной схемы в реляционную схему
- Как правило, CASE-средства, автоматизирующие преобразование концептуальной схемы БД в реляционную, производят реляционную схему базы данных в третьей нормальной форме
- Нормализация более высокого уровня усложняет программную реализацию и редко требуется на практике

Семантическая модель Entity-Relationship (1)

- На использовании разных вариантов ER-модели основано большинство современных подходов к проектированию баз данных
- Модель была предложена Питером Ченом (Peter Chen) в 1976 г.

  http://old.osp.ru/dbms/1995/03/271.htm
- Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов
- Простота и наглядность представления концептуальных схем баз данных в ER-модели привели к ее широкому распространению в CASE-системах, поддерживающих автоматизированное проектирование баз данных
- Среди множества разновидностей ER-моделей одна из наиболее популярных и развитых применяется в CASE-системе компании Oracle
- Рассмотрим упрощенный вариант этой диаграммной модели, достаточный для понимания основных особенностей проектирования реляционных баз данных с использованием ER-моделей

Семантическая модель Entity-Relationship (2)

Основные понятия ER-модели (1)

- Основными понятиями ER-модели являются *сущность*, *связь* и *атрибут*
- *Сущность* – это реальный или представляемый объект, информация о котором должна сохраняться и быть доступной
- Это «определение» позаимствовано из одного из ранних вариантов документации CASE-системы Oracle
- Понятно, что на самом деле мы имеем дело с тавтологией, поскольку, во-первых, пытаемся определить термин *сущность* через не определенный термин *объект*, а во-вторых, попытки определения термина *объект* настолько же безнадежны
- Обычно авторы пытаются оправдываться тем, что в подобном контексте имеется в виду «житейское», а не сколько-нибудь формализованное понятие *объекта*
- Конечно, от этого не становится легче, поскольку понятие *сущности* должно пониматься в достаточно точном смысле
- Но эта тавтология традиционна для области семантического моделирования. В этой области стремятся максимально избегать формальностей

Семантическая модель Entity-Relationship (3)

Основные понятия ER-модели (2)

- В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности
- При этом имя сущности – это имя типа, а не некоторого конкретного экземпляра этого типа
- Было бы правильнее всегда использовать термины *тип сущности* и *экземпляр типа сущности*, но во избежание многословности в тех случаях, где это не приводит к двусмысленности, мы будем использовать термин сущность в значении типа сущности
- Для большей выразительности и лучшего понимания имя сущности может сопровождаться примерами конкретных экземпляров этого типа

Семантическая модель Entity-Relationship (4)
Основные понятия ER-модели (3)

- На рисунке изображена сущность АЭРОПОРТ с примерными экземплярами «Шереметьево» и «Хитроу»
- Эта примитивная диаграмма, тем не менее, несет важную информацию

- Во-первых, она показывает, что в базе данных будут содержаться однотипные структуры данных (экземпляры сущности), описывающие аэропорты
- Во-вторых, поскольку в жизни существует несколько точек зрения на аэропорты, приведенные примеры аэропортов позволяют несколько сузить допустимый набор точек зрения
- В нашем случае приведены примеры международных аэропортов, так что, скорее всего, имеется точка зрения пассажира или пилота международных авиарейсов

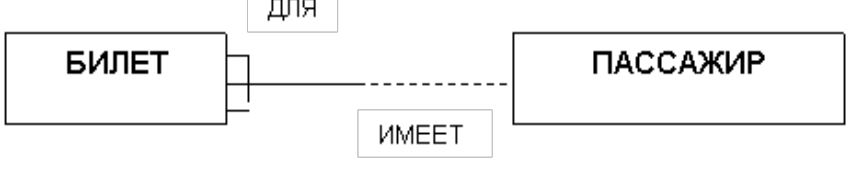
Семантическая модель Entity-Relationship (5)
Основные понятия ER-модели (4)

- При определении типа сущности необходимо гарантировать, что каждый экземпляр типа сущности может быть отличим от любого другого экземпляра того же типа сущности
  - Это требование в некотором роде аналогично требованию отсутствия кортежей-дубликатов в реляционных таблицах
- *Связь* – это графически изображаемая ассоциация, устанавливаемая между двумя типами сущностей
- Как и сущность, связь – это типовое понятие, все экземпляры обоих связываемых типов сущностей подчиняются устанавливаемым правилам связывания
- Поэтому правильнее говорить о типе связи, устанавливаемой между типами сущности, и об экземплярах типа связи, устанавливаемых между экземплярами типа сущности
- Тем не менее, как и в случае типа сущности, для краткости мы будем часто использовать термин *связь* в значении *типа связи*

Семантическая модель Entity-Relationship (6)
Основные понятия ER-модели (5)

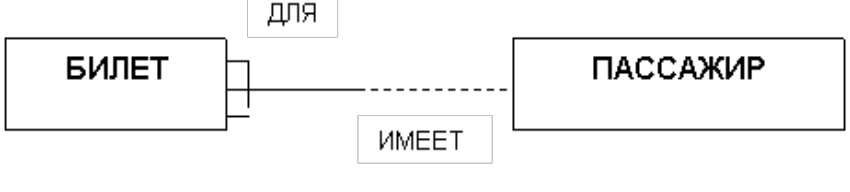
- В обсуждаемом здесь варианте ER-модели связи всегда являются бинарными, то есть соединяющими два типа сущности, и они могут существовать между двумя разными типами сущностей или между типом сущности и им же самим (рекурсивная связь)
- В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указываются
  - имя конца связи,
  - степень конца связи (сколько экземпляров данного типа сущности должно присутствовать в каждом экземпляре данного типа связи),
  - обязательность связи (т. е. любой ли экземпляр данного типа сущности должен участвовать в некотором экземпляре данного типа связи)
- Заметим, что в некоторых вариантах ER-модели конец связи называют ролью связи в данной сущности
- Тогда можно говорить об *имени роли*, *степени роли* и *обязательности роли* связи в данной сущности

Семантическая модель Entity-Relationship (7)
Основные понятия ER-модели (6)



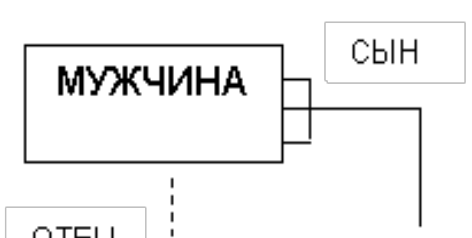
- Связь представляется в виде ненаправленной линии, соединяющей две сущности или ведущей от сущности к ней же самой
- В месте «стыковки» связи с сущностью используются:
  - ✓ трехточечный вход в прямоугольник сущности, если для этой сущности в связи могут (или должны) использоваться много (*many*) экземпляров сущности;
  - ✓ одноточечный вход, если в связи может (или должен) участвовать только один (*one*) экземпляр сущности

Семантическая модель Entity-Relationship (8)
Основные понятия ER-модели (7)



- Обязательный конец связи изображается сплошной линией, а необязательный – прерывистой линией
- Связь между сущностями БИЛЕТ и ПАССАЖИР, связывает билеты и пассажиров
- Конец связи с именем «для» позволяет связывать с одним пассажиром более одного билета, причем каждый билет должен быть связан с каким-либо пассажиром
- Конец связи с именем «имеет» показывает, что каждый билет может принадлежать только одному пассажиру, причем пассажир не обязан иметь хотя бы один билет
- Лаконичная устная трактовка изображенной диаграммы состоит в следующем:
  - ✓ каждый БИЛЕТ предназначен для одного и только одного ПАССАЖИРА;
  - ✓ каждый ПАССАЖИР может иметь один или более БИЛЕТОВ

Семантическая модель Entity-Relationship (9)
Основные понятия ER-модели (8)



- На рисунке изображена рекурсивная связь, связывающая сущность МУЖЧИНА с ней же самой
- Конец связи с именем «сын» определяет тот факт, что несколько мужчин могут быть сыновьями одного отца
- Конец связи с именем «отец» означает, что не у каждого мужчины должны быть сыновья
- Лаконичная устная трактовка изображенной диаграммы состоит в следующем:

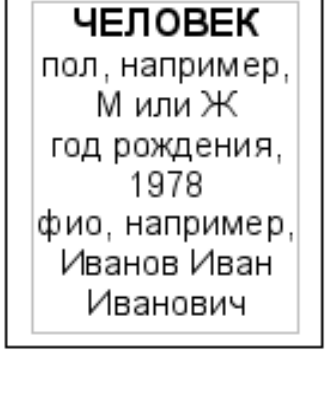
✓ каждый МУЖЧИНА является сыном одного и только одного МУЖЧИНЫ;

✓ каждый МУЖЧИНА может являться отцом одного или более МУЖЧИН

Семантическая модель Entity-Relationship (10)

Основные понятия ER-модели (9)

- *Атрибутом* сущности является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности
- Имена атрибутов заносятся в прямоугольник, изображающий сущность, под именем сущности и изображаются малыми буквами, возможно, с примерами



- Некоторые атрибуты могут помечаться как *необязательные*
- Значения таких атрибутов не обязаны присутствовать во всех экземплярах данного типа сущности
- Атрибуты типа сущности в ER-модели похожи на атрибуты отношения в реляционной модели данных
- Введение именованных атрибутов вводит некоторую типовую структуру данных, имя которой совпадает с именем типа сущности в случае ER-модели или с именем переменной отношения в случае реляционной модели
- Этой типовой структуре должны следовать все экземпляры типа сущности или все кортежи отношения

Семантическая модель Entity-Relationship (11)

Основные понятия ER-модели (10)

- Но имеется и важное отличие
- В реляционной модели данных атрибут определяется как упорядоченная пара <имя_атрибута, имя_домена> (или <имя_атрибута, имя_базового_типа_данных>, если понятие домена не поддерживается)
- Заголовок отношения, определяемый как множество таких пар, представляет собой полный аналог структурного типа данных в языках программирования
- При определении атрибутов типа сущности в ER-модели указание домена атрибута не является обязательным, хотя это и возможно
- Обсудим, чем вызвана эта возможность «ослабленного» определения атрибутов

Семантическая модель Entity-Relationship (12)

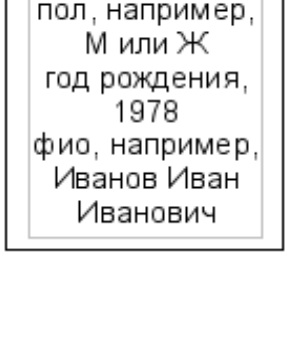
Основные понятия ER-модели (11)

- Семантические модели данных используются для построения концептуальных схем БД, и эти схемы преобразуются в реляционные схемы БД, которые поддерживаются той или иной СУБД
- Несмотря на то, что в настоящее время типовые возможности РСУБД в основном стандартизованы (на основе стандарта языка SQL), детали базового набора типов данных и средств определения доменов в разных системах могут различаться
- Поскольку производители CASE-средств проектирования реляционных БД стремятся не связывать обеспечиваемые ими возможности семантического моделирования с конкретной реализацией СУБД, они стимулируют откладывание строгого определения типов атрибутов до стадии полного определения реляционной схемы

Семантическая модель Entity-Relationship (13)

Основные понятия ER-модели (12)

- Кроме того, при определении заголовка отношения допускается использование имен атрибутов, совпадающих с именами своих доменов
  - это два разных пространства имен, и наличие одинаковых имен у атрибутов и доменов не вызывает коллизий
- Поэтому при определении атрибутов типов сущности можно так подбирать их имена, что они в дальнейшем будут подсказывать, какие домены у этих атрибутов имеются в виду
- Пониманию предполагаемой сути доменов способствует и возможность указания примеров значений атрибутов
- Например, на рисунке имеется атрибут год рождения, в качестве примерного значения которого указано «1978»
- Это подсказывает, что в реляционной схеме при определении соответствующего атрибута наиболее естественным базовым типом данных будет темпоральный тип «ДАТА», значения которого задают дату с точностью до года



Семантическая модель Entity-Relationship (14)

Основные понятия ER-модели (13)

- При определении типа сущности необходимо гарантировать, что каждый экземпляр сущности является отличимым от любого другого экземпляра той же сущности
- Поскольку сущность является абстракцией реального или представляемого объекта внешнего мира, это требование нужно иметь в виду уже при выборе кандидата в типы сущности.
- Например, предположим, что проектируется база данных для поддержки работы книжного склада
- На складе могут храниться произвольные части тиража любого издания любой книги
- Может ли в этом случае индивидуальная книга являться прообразом типа сущности?

Семантическая модель Entity-Relationship (15)

Основные понятия ER-модели (14)

- Утверждается, что нет, поскольку отсутствует возможность различения книг одного издания
- Для книжного склада прообразом типа сущности будет набор одноименных книг одного автора, вышедших в одном издании
- Одним из атрибутов этого типа сущности будет число книг в наборе
- Но когда книга поступает в библиотеку и ей присваивается уникальный библиотечный номер, она становится разумным прообразом типа сущности
- Плохо устроены библиотеки, в которых не различаются индивидуальные книги
  - даже одноименные книги одного автора, вышедшие в одном издании

Семантическая модель Entity-Relationship (16)

Уникальные идентификаторы типов сущности (1)

- Но при проектировании базы данных мало того, чтобы проектировщик убедился в правильном выборе типов сущности, гарантирующем различие экземпляров каждого типа сущности
- Необходимо сообщить системе автоматизации проектирования БД, каким образом будут различаться эти экземпляры, т. е. сообщить, как конструируются уникальные идентификаторы экземпляров каждого типа сущности
- В ER-модели у экземпляра типа сущности не может быть назначаемого пользователем имени или назначаемого системой внешнего уникального идентификатора

Семантическая модель Entity-Relationship (17)

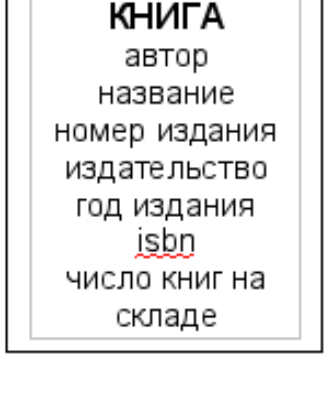
- Уникальные идентификаторы типов сущности (2)
  - Экземпляр типа сущности может идентифицироваться только своими индивидуальными характеристиками, а они представляются
    - значениями атрибутов и экземплярами типов связи, связывающими данный экземпляр типа сущности с экземплярами других типов сущности или этого же типа сущности
  - Поэтому уникальным идентификатором сущности может быть
    - атрибут,
    - комбинация атрибутов,
    - связь,
    - комбинация связей или комбинация связей и атрибутов,

    уникально отличающая любой экземпляр сущности от других экземпляров сущности того же типа

Семантическая модель Entity-Relationship (18)

Уникальные идентификаторы типов сущност

- На рисунке показан тип сущности КНИГА, пригодный для использования в базе данных книжного склада
- При издании любой книги в приличном издательстве ей присваивается уникальный номер – ISBN



- Понятно, что значение атрибута isbn будет уникально идентифицировать партию книг на складе
- Кроме того, конечно, в качестве уникального идентификатора годится и комбинация атрибутов
<автор, название, номер издания, издательство, год издания>

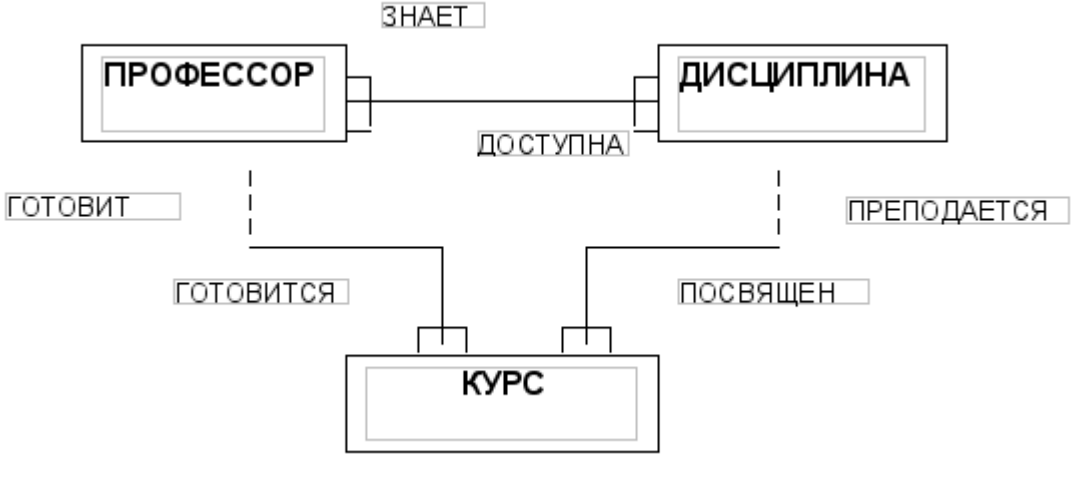
Семантическая модель Entity-Relationship (19)

Уникал[illegible]ры типов су[illegible] (4)

- Здесь диаграмма включает два связанных типа сущности
- У каждого обычного взрослого человека имеется один и только один паспорт
  - ✓ мы не берем в расчет особый случай, когда у одного человека имеется несколько паспортов, хотя это не изменило бы ситуацию, и каждый паспорт может принадлежать только одному взрослому человеку
  - ✓ некоторые уже готовые паспорта могут быть еще никому не выданы)
- Тогда связь человека с его паспортом (конец связи ИМЕЕТ) уникально идентифицирует взрослого человека, т. е., грубо говоря, паспорт определяет взрослого человека
- Поскольку могут существовать паспорта, еще не выданные человеку, эта связь не является уникальным идентификатором сущности ПАСПОРТ

Семантическая модель Entity-Relationship (20)

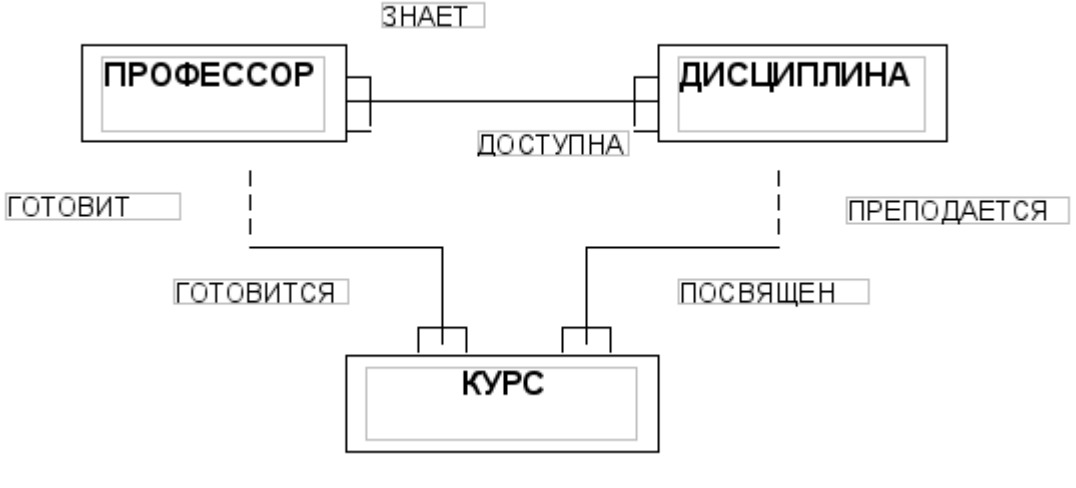
Уникал 5)



- На диаграмме имеются три связанных типа сущности
- Профессора обладают знаниями в нескольких учебных дисциплинах
- Преподавание каждой дисциплины доступно нескольким профессорам
- Другими словами, между сущностями ПРОФЕССОР и ДИСЦИПЛИНА определена связь «многие ко многим»

Семантическая модель Entity-Relationship (21)

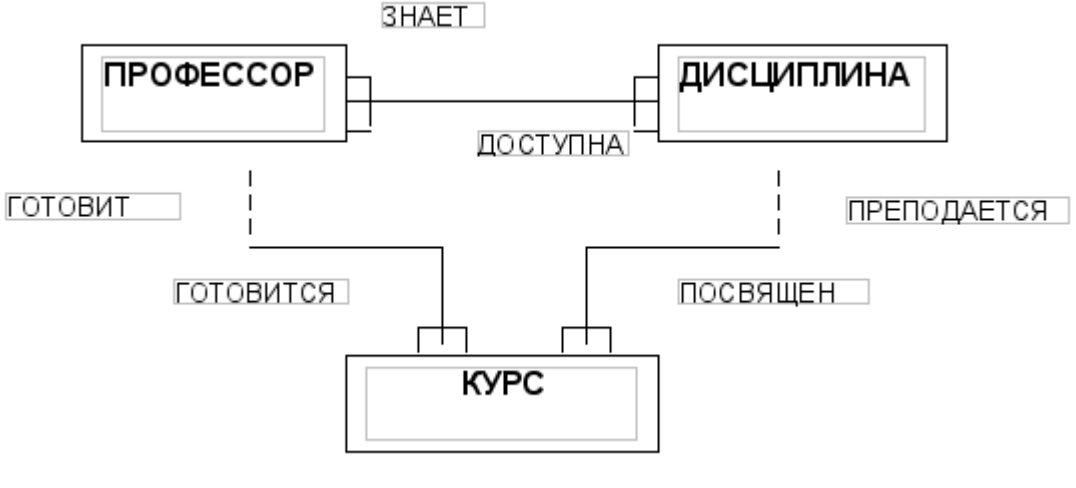
Уникали 6)



- Каждый профессор может готовить курсы по любой доступной ему дисциплине
- Каждой дисциплине может быть посвящено несколько учебных курсов
- Но каждый профессор может готовить только один курс по любой доступной ему дисциплине, и каждый курс может быть посвящен только одной дисциплине

Семантическая модель Entity-Relationship (22)

Уникал 7)

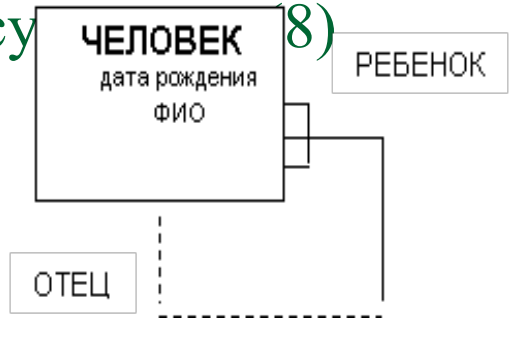


- Тем самым, каждый экземпляр типа сущности КУРС уникально идентифицируется экземпляром сущности ПРОФЕССОР и экземпляром сущности ДИСЦИПЛИНА,
- ✓ т.е. парой связей с именами концов ГОТОВИТСЯ и ПОСВЯЩЕН на стороне сущности КУРС.
- Заметим, что сущности ПРОФЕССОР и ДИСЦИПЛИНА связями не идентифицируются

Семантическая модель Entity-Relationship (23)

Уникальные идентификаторы типов сущностей (8)

- Наконец, пример типа сущности, уникальный идентификатор которого является комбинацией атрибутов и связей
- Это несколько уточненный вариант сущности с рекурсивной связью, показанной раньше



- У каждого человека могут быть дети, и у каждого человека имеется отец
- Тогда, если предположить, что близнецам, появившимся на свет одновременно, не дают одинаковых имен, то уникальным идентификатором типа сущности ЧЕЛОВЕК может быть комбинация атрибутов <дата рождения, ФИО> и связь с именем конца РЕБЕНОК

Семантическая модель Entity-Relationship (24)

Нормальные формы ER-диаграмм (1)

- Как и в случае схем реляционных баз данных, для ER-диаграмм вводится понятие нормальных форм, причем их смысл очень близко соответствует смыслу нормальных форм отношений
- Заметим, что определения нормальных форм ER-диаграмм делают более понятным смысл нормализации схем отношений
- Мы приведем только очень краткие и неформальные определения трех первых нормальных форм
- Конечно, можно было бы ввести дальнейшие нормальные формы ER-диаграмм, аналогичные нормальной форме Бойса-Кодда, 4NF и 5NF, но на практике к такой нормализации обычно не прибегают, а общие идеи должны быть понятны и так

Семантическая модель Entity-Relationship (25)

Нормал

- В первой нормальной форме ER-диаграммы устраняются атрибуты, содержащие множественные значения, т. е. производится выявление неявных сущностей, «замаскированных» под атрибуты
- На рисунке показана диаграмма, в которой тип сущности АЭРОДРОМ не удовлетворяет требованию первой нормальной формы
- Здесь несущественны атрибуты сущности АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ, но сущность АЭРОДРОМ помимо атрибутов, отражающих собственные характеристики аэродромов (длина взлетно-посадочной полосы, число ангаров и т.д.) содержит атрибут, множественное значение которого характеризует самолеты, приписанные к этому аэродрому

Семантическая модель Entity-Relationship (26)

Нормал

- Очевидно, что самолеты нуждаются в ремонте, т.е. должны обслуживаться некоторым авиаремонтным предприятием
- Но поскольку самолеты являются частью сущности АЭРОДРОМ, единственным способом фиксации этого факта на диаграмме является проведение связи «многие ко многим» между типами сущности АЭРОДРОМ и АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ
- Таким образом выражается то соображение, что для ремонта разных самолетов, приписанных к одному аэродрому, могут использоваться разные авиаремонтные предприятия, и каждое авиаремонтное предприятие может обслуживать несколько аэродромов

Семантическая модель Entity-Relationship (27)

Нормал

- Чем плоха эта ситуация?
- Прежде всего, тем, что скрывается тот факт, что авиаремонтное предприятие ремонтирует самолеты, а не аэродромы
- Связь же между типами сущности АЭРОДРОМ и АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ на самом деле означает, что любой аэродром из группы аэродромов обслуживается любым авиаремонтным предприятием из группы таких предприятий
- Проблема состоит именно в том, что значением атрибута «самолеты» является множество экземпляров типа сущности САМОЛЕТ, и этот тип сущности сам обладает атрибутами и связями

Семантическая модель Entity-Relationship (28)

Нормал

- Ситуацию исправляет ER-диаграмма, показанная здесь
- Мы выделили тип сущности САМОЛЕТ
- Связь между сущностями АЭРОПОРТ и САМОЛЕТ показывает, что к одному аэродрому приписывается несколько самолетов
- Связь между сущностями САМОЛЕТ и АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ означает, что каждый самолет из группы самолетов обслуживается любым транспортным предприятием из некоторой группы таких предприятий
- Эта ER-диаграмма находится в первой нормальной форме и, как мы видим, правильнее отображает реальную ситуацию

Семантическая модель Entity-Relationship (29)

Нормальные 1)

- Во второй нормальной форме устраняются атрибуты, зависящие только от части уникального идентификатора. Эта часть уникального идентификатора определяет отдельную сущность
- На рисунке показана диаграмма, на которой тип сущности ЭЛЕМЕНТ РАСПИСАНИЯ не удовлетворяет требованиям второй нормальной формы
- На этой диаграмме у сущности ЭЛЕМЕНТ РАСПИСАНИЯ имеются следующие свойства
- Элементы расписания предназначены для сохранения данных о рейсах самолетов, вылетающих в течение дня
- Некоторыми важными характеристиками рейса являются номер рейса, аэропорт вылета, аэропорт назначения, дата и время вылета, бортовой номер самолета, тип самолета

Семантическая модель Entity-Relationship (30)

Нормальные 2)

- Если говорить про российские авиационные компании, то
  - ✓ у каждого рейса имеется заранее приписанный ему номер (уникальный среди всех других имеющихся номеров рейсов),
  - ✓ не все рейсы совершаются каждый день, поэтому характеристикой конкретного рейса является дата и время его совершения,
  - ✓ бортовой номер самолета определяется парой <номер рейса, дата-время вылета>
- Имеется связь «многие к одному» между сущностями ЭЛЕМЕНТ РАСПИСАНИЯ и ГОРОД (каждый день много рейсов прибывает в один и тот же город)
- Экземпляры типа сущности ГОРОД характеризуют город, в который прибывает данный рейс

Семантическая модель Entity-Relationship (31)

Нормальные 3)

- Уникальным идентификатором типа сущности ЭЛЕМЕНТ РАСПИСАНИЯ является пара атрибутов <номер рейса, дата-время вылета>
- Если вернуться к терминам функциональных зависимостей, то между атрибутами этой сущности имеются следующие FD:
  - ✓ {номер рейса, дата-время вылета} → бортовой номер самолета;
  - ✓ номер рейса → аэропорт вылета;
  - ✓ номер рейса → аэропорт назначения;
  - ✓ бортовой номер самолета → тип самолета

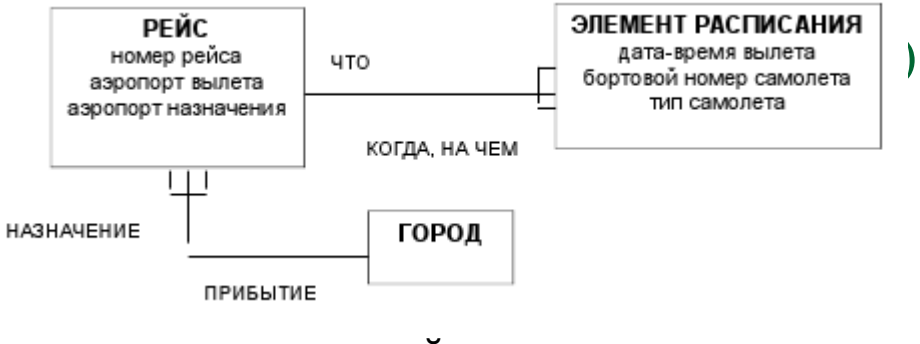
Семантическая модель Entity-Relationship (32)

Нормальные 4)

- Кроме того, очевидно, что каждый экземпляр связи с сущностью ГОРОД также определяется значением атрибута номер рейса
- Налицо нарушение требования второй нормальной формы
- Мы получаем не только избыточное хранение значений атрибутов аэропорт вылета и аэропорт назначения в каждом экземпляре типа сущности ЭЛЕМЕНТ РАСПИСАНИЯ с одним и тем же значением номера рейса
- Искажается и затемняется смысл связи с сущностью ГОРОД
- Можно подумать, что в разные дни один и тот же рейс прибывает в разные города

Семантическая модель Entity-Relationship (33)

Нормальны



- Здесь показан нормализованный вариант диаграммы, в котором все сущности находятся во второй нормальной форме
- Теперь имеются три типа сущности:
  - ✓ РЕЙС с атрибутами номер рейса, дата-время вылета, аэропорт назначения;
  - ✓ ЭЛЕМЕНТ РАСПИСАНИЯ с атрибутами дата-время вылета, бортовой номер самолета, тип самолета;
  - ✓ ГОРОД

Семантическая модель Entity-Relationship (34)

Нормальны

- Уникальным идентификатором сущности РЕЙС является атрибут номер рейса; уникальный идентификатор ЭЛЕМЕНТ РАСПИСАНИЯ состоит из атрибута дата-время вылета и конца связи КОГДА, НА ЧЕМ
- Ни в одном типе сущности больше нет атрибутов, определяемых частью уникального идентификатора
- Свойства второй нормальной формы удовлетворяются, и мы имеем более качественную диаграмму

Семантическая модель Entity-Relationship (35)

Нормальны

)

- В третьей нормальной форме устраняются атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор. Эти атрибуты являются основой отдельной сущности.
- Взглянем еще раз на тип сущности ЭЛЕМЕНТ РАСПИСАНИЯ
- Конечно, каждый день каждый рейс выполняется только одним самолетом, поэтому бортовой номер самолета полностью зависит от уникального идентификатора
- Но бортовой номер является уникальной характеристикой каждого самолета, и от этой характеристики зависят все остальные характеристики, в частности тип самолета

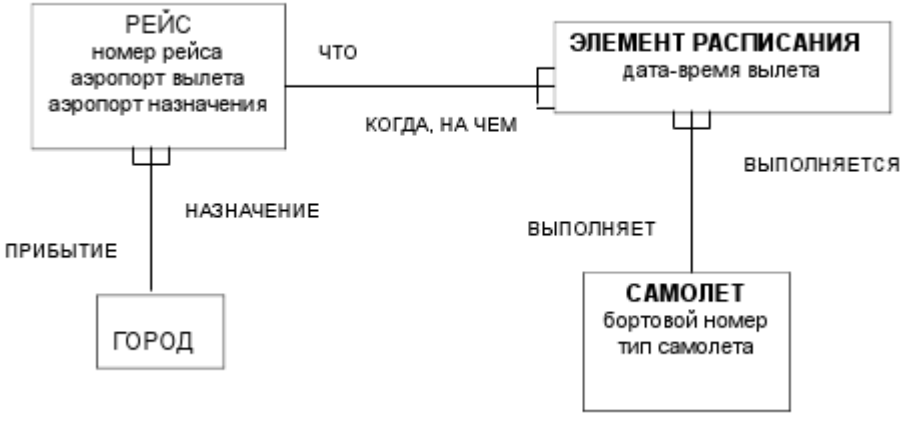
Семантическая модель Entity-Relationship (36)

Нормальны

- Другими словами, между уникальным идентификатором и другими атрибутами типа сущности ЭЛЕМЕНТ РАСПИСАНИЯ имеются следующие функциональные зависимости:
  - ✓ {КОГДА, НА ЧЕМ, дата-время вылета} → бортовой номер самолета;
  - ✓ {КОГДА, НА ЧЕМ, дата-время вылета} → тип самолета;
  - ✓ бортовой номер самолета → тип самолета

Семантическая модель Entity-Relationship (37)

Нормальные



- Как видно, имеется транзитивная FD {КОГДА, НА ЧЕМ, дата-время вылета} $\rightarrow$ тип самолета, и наличие этой FD вызывает нарушение требования третьей нормальной формы
- На самом деле, тип сущности ЭЛЕМЕНТ РАСПИСАНИЯ включает в себя (по крайней мере, частично) тип сущности САМОЛЕТ
- Это вызывает избыточность хранения и затуманивает смысл диаграммы
- Здесь показан нормализованный вариант диаграммы, в котором все типы сущности находятся в третьей нормальной форме

Семантическая модель Entity-Relationship (37)

Более сложные элементы ER-модели (1)

- До сих пор мы рассматривали только самые основные и наиболее очевидные понятия ER-модели данных
- К числу некоторых более сложных элементов модели относятся следующие.
- *Подтипы и супертипы сущностей*. Подобно тому, как это делается в языках программирования с развитыми типовыми системами (например, в языках объектно-ориентированного программирования), в ER-модели поддерживается возможность наследования типа сущности от одного супертипа сущности
  - Механизм наследования в ER-модели обладает несколькими особенностями: в частности, интересные нюансы связаны с необходимостью графического изображения этого механизма (более подробно механизм наследования рассматривается далее)

Семантическая модель Entity-Relationship (38)

Более сложные элементы ER-модели (2)

- *Уточняемые степени связи*. Иногда бывает полезно определить возможное количество экземпляров сущности, участвующих в данной связи (примером может служить то ограничение, что служащему разрешается участвовать не более чем в трех проектах одновременно)
  - Для выражения этого семантического ограничения разрешается указывать на конце связи ее максимально допустимую или обязательную степень.
- *Взаимно исключающие связи*. Для заданного типа сущности можно определить такой набор типов связи с другими типами сущности, что для каждого экземпляра заданного типа сущности может (если набор связей является необязательным) или должен (если набор связей обязателен) существовать экземпляр только одной связи из этого набора

Семантическая модель Entity-Relationship (39)

Более сложные элементы ER-модели (3)

- *Каскадные удаления экземпляров сущностей*. Некоторые связи бывают настолько сильными (конечно, в случае связи «один ко многим»), что при удалении опорного экземпляра сущности (соответствующего концу связи «один») нужно удалить и все экземпляры сущности, соответствующие концу связи «многие»
  - Соответствующее требование каскадного удаления можно специфицировать при определении связи.
- *Домены*. Как и в случае реляционной модели данных, в некоторых случаях полезна возможность определения потенциально допустимого множества значений атрибута сущности (домена)

Семантическая модель Entity-Relationship (40)

Более сложные элементы ER-модели (4)

- Эти и другие усложненные элементы ER-модели делают ее более мощной, но одновременно несколько затрудняют ее использование
- Конечно, при реальном применении ER-диаграмм для проектирования баз данных необходимо ознакомиться со всеми возможностями
- Далее мы подробнее обсудим суть механизма наследования в ER-модели, а также приведем пример типа сущности с взаимно исключающими связями

Семантическая модель Entity-Relationship (41)

Более сложные элементы ER-модели (5) Наследование (1)

- Тип сущности может быть расщеплен на два или более взаимно исключающих подтипов, каждый из которых включает общие атрибуты и/или связи
- Эти общие атрибуты и/или связи явно определяются один раз на более высоком уровне
- В подтипах могут определяться собственные атрибуты и/или связи. В принципе, подтипизация может продолжаться на более низких уровнях, но опыт использования ER-модели при проектировании баз данных показывает, что в большинстве случаев оказывается достаточно двух-трех уровней

Семантическая модель Entity-Relationship (42)

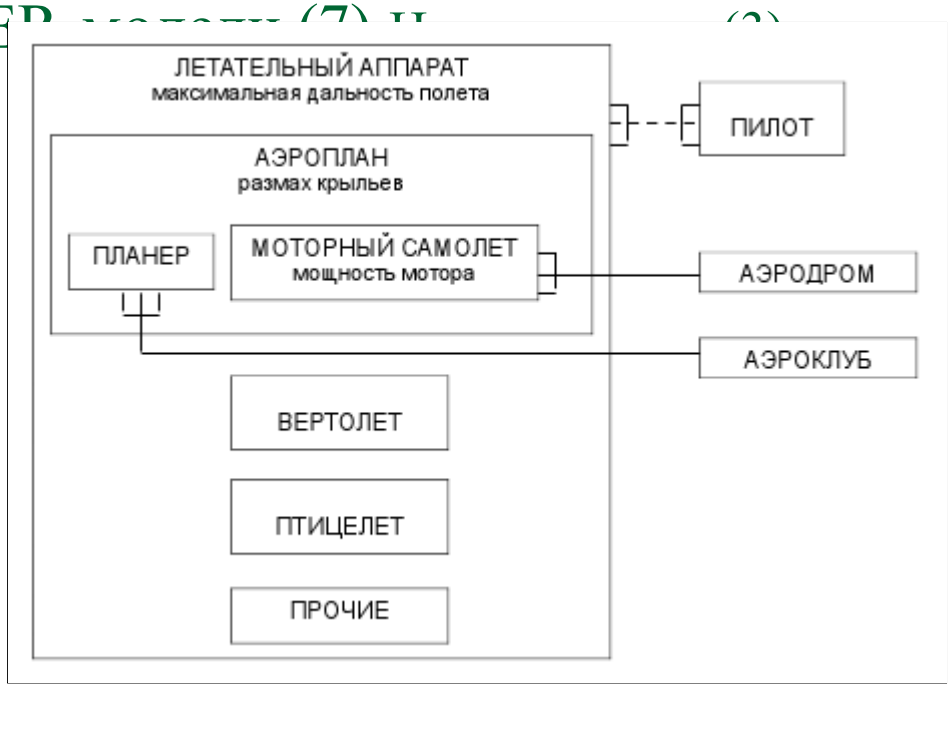
Более сложные элементы ER-модели (6) Наследование (2)

- Особенности механизма наследования в ER-модели определяются следующими правилами. Если у типа сущности *A* имеются подтипы *B1*, *B2*,…, *Bn*, то:
  - любой экземпляр типа сущности B_1, B_2,…, B_n является экземпляром типа сущности *A* (включение);
  - если *a* является экземпляром типа сущности *A*, то *a* является экземпляром некоторого подтипа сущности B_i (i = 1, 2, …, n) (отсутствие собственных экземпляров у супертипа сущности);
  - ни для каких подтипов B_i и B_j (i, j = 1, 2, …, n) не существует экземпляра, типом которого одновременно являются типы сущности B_i и B_j (разъединенность подтипов)

Семантическая модель Entity-Relationship (43)

Более сложные элементы ER-модели (7) И[illegible] (3)

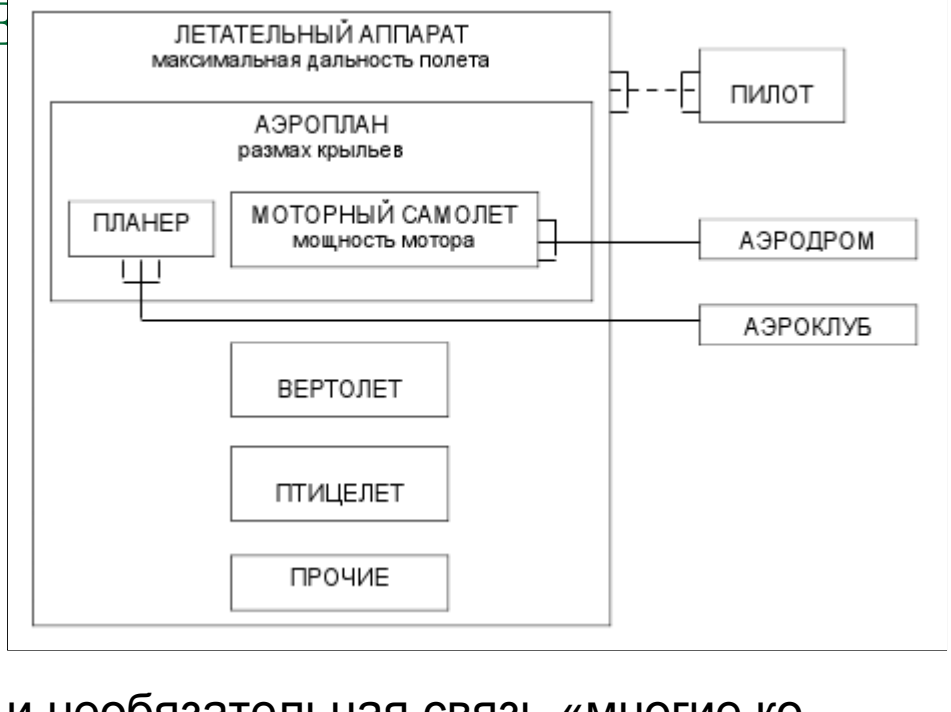
- Тип сущности, на основе которого определяются подтипы, называется супертипом
- Объединение множества экземпляров подтипов должно образовывать полное множество экземпляров супертипа, т.е. любой экземпляр супертипа должен относиться к некоторому подтипу
- Иногда для обеспечения такой полноты приходится определять дополнительный подтип ПРОЧИЕ



Семантическая модель Entity-Relationship (44)

Более сложные элементы E

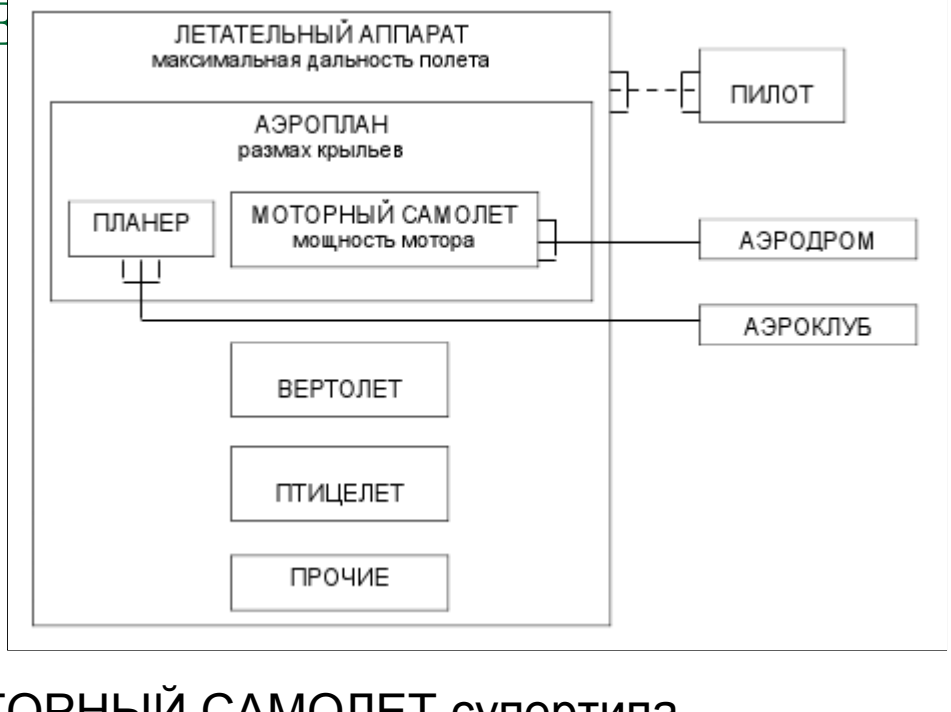
- Здесь показан пример супертипа ЛЕТАТЕЛЬНЫЙ АППАРАТ и его подтипов АЭРОПЛАН, ВЕРТОЛЕТ, ПТИЦЕЛЕТ и ПРОЧИЕ
- У подтипа АЭРОПЛАН имеются два собственных подтипа – ПЛАНЕР и МОТОРНЫЙ САМОЛЕТ
- Для супертипа сущности ЛЕТАТЕЛЬНЫЙ АППАРАТ определен атрибут максимальная дальность полета и необязательная связь «многие ко многим» с типом сущности ПИЛОТ
- ✓ Эти атрибут и связь наследуется всеми подтипами этого супертипа сущности.



Семантическая модель Entity-Relationship (45)

Более сложные элементы E

- У непосредственного подтипа сущности АЭРОПЛАН определяется один дополнительный атрибут, так что в совокупности у данного типа сущности имеются два атрибута максимальная дальность полета и размах крыльев и одна унаследованная связь с типом сущности ПИЛОТ

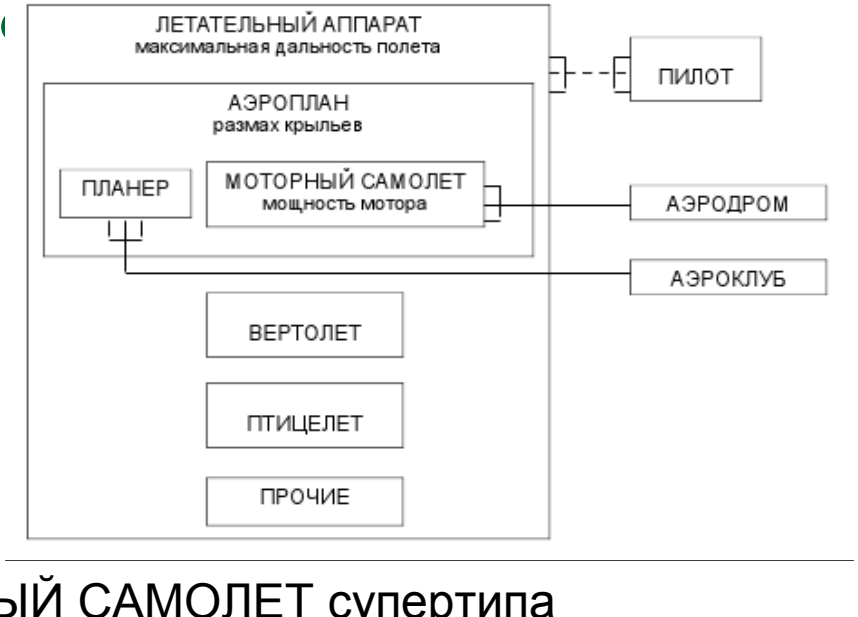


- У подтипа второго уровня МОТОРНЫЙ САМОЛЕТ супертипа АЭРОПЛАН определяется один дополнительный атрибут мощность мотора и одна дополнительная (обязательная) связь с типом сущности АЭРОДРОМ

Семантическая модель Entity-Relationship (46)

Более сложные элементы ER-м

- Тем самым, у типа сущности МОТОРНЫЙ САМОЛЕТ имеются три атрибута: два унаследованных – максимальная дальность полета и размах крыльев и один собственный – мощность мотора, а также две связи: одна унаследованная – с типом сущности ПИЛОТ и одна собственная – с типом сущности АЭРОДРОМ



- У подтипа второго уровня МОТОРНЫЙ САМОЛЕТ супертипа АЭРОПЛАН определяется один дополнительный атрибут мощность мотора и одна дополнительная (обязательная) связь с типом сущности АЭРОДРОМ
- Понятно, что для типа сущности ПРОЧИЕ, бессмысленно определять собственные атрибуты и связи, так что свойства этого типа будут совпадать со свойствами его супертипа

Семантическая модель Entity-Relationship (46)

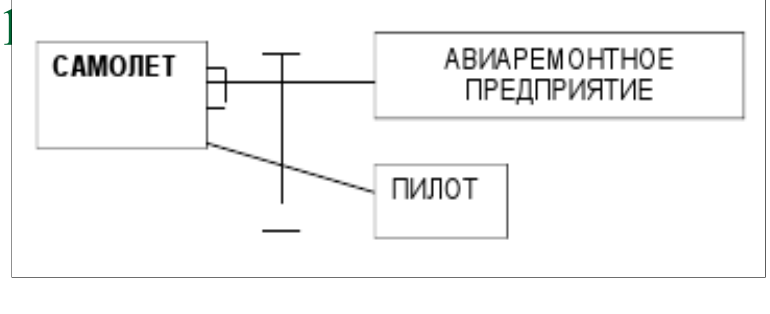
Более сложные элементы ER-модели (10) Наследование (7)

- Как же следует понимать эту диаграмму?
- Если начинать от супертипа, то диаграмма изображает ЛЕТАТЕЛЬНЫЙ АППАРАТ, который должен быть АЭРОПЛАНОМ, ВЕРТОЛЕТОМ, ПТИЦЕЛЕТОМ или ДРУГИМ ЛЕТАТЕЛЬНЫМ АППАРАТОМ
- Если начинать от подтипа (например, сущности ВЕРТОЛЕТ), то это ВЕРТОЛЕТ, который относится к типу ЛЕТАТЕЛЬНОГО АППАРАТА
- Если начинать от подтипа, который является одновременно супертипом, то это АЭРОПЛАН, который относится к типу ЛЕТАТЕЛЬНОГО АППАРАТА и должен быть ПЛАНЕРОМ или МОТОРНЫМ САМОЛЕТОМ
- В механизме наследования ER-модели допускается наличие двух или более разбиений сущности на подтипы
  - Например, тип сущности ЧЕЛОВЕК может быть расщеплен на подтипы по профессиональному признаку (ПРОГРАММИСТ, ДОЯРКА и т. д.), а может быть расщеплен и по половому признаку (МУЖЧИНА, ЖЕНЩИНА)

Семантическая модель Entity-Relationship (46)

Более сложные элементы ER-модели (1

- Здесь показан пример диаграммы из двух сущностей с взаимно исключающими
- Самолет может находиться в рабочем состоянии, и тогда у него имеется один и только один пилот

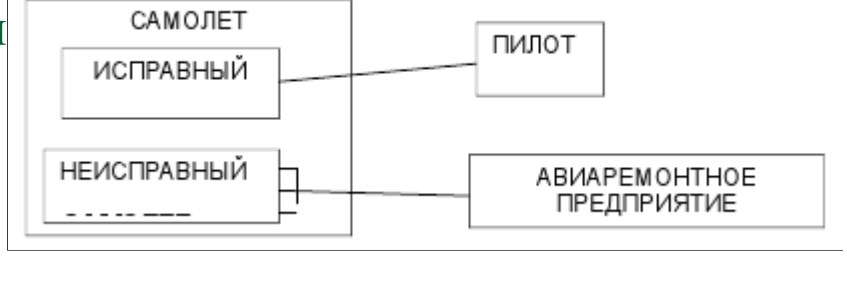


- Или же самолет может быть неисправным, и тогда он находится на ремонте на некотором авиаремонтном предприятии (каждое предприятие может производить ремонт нескольких самолетов)
- В данном случае для каждого экземпляра типа сущности САМОЛЕТ должен существовать экземпляр одной из указанных связей
- Для экземпляров типа сущности САМОЛЕТ, соответствующих исправным самолетам, должен существовать экземпляр связи «один к одному» с экземпляром типа сущности ПИЛОТ, а экземпляры, соответствующие неисправным самолетам, должны участвовать в экземпляре типа связи «многие ко одному» с экземпляром типа сущности АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ

Семантическая модель Entity-Relationship (47)

Более сложные элементы ER-модели

- Диаграмма с взаимно исключающими связями может быть преобразована к диаграмме без взаимно исключающих связей путем введения подтипов



- Поскольку любой самолет может быть либо исправным, либо неисправным, можно корректным образом ввести два подтипа супертипа САМОЛЕТ – ИСПРАВНЫЙ САМОЛЕТ и НЕИСПРАВНЫЙ САМОЛЕТ
- На уровне супертипа сущности связи не определяются
- Для подтипа ИСПРАВНЫЙ САМОЛЕТ определяется обязательная связь «один к одному» с типом сущности ПИЛОТ, а для подтипа НЕИСПРАВНЫЙ САМОЛЕТ определяется обязательная связь «многие к одному» с типом сущности АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ

Семантическая модель Entity-Relationship (48)

Получение реляционной схемы из ER-диаграммы (1)

- Опишем типовую многошаговую процедуру преобразования ER-диаграммы в реляционную (более точно, в SQL-ориентированную) схему базы данных
- Следует заметить, что предполагается использование «традиционных» средств определения данных SQL, не включающих возможности определения структурных типов данных с поддержкой механизма наследования типов и типизированных таблиц
- Отсутствует общепризнанная методология проектирования SQL-ориентированных баз данных, в которых используются «объектные» расширения SQL

Семантическая модель Entity-Relationship (49)

- Каждый *простой тип сущности* превращается в таблицу. (Простым типом сущности называется тип сущности, не являющийся подтипом и не имеющий подтипов.)
  - *Имя сущности* становится именем таблицы
  - *Экземплярам* типа сущности соответствуют *строки* соответствующей таблицы
- Каждый *атрибут* становится *столбцом* таблицы с тем же именем; может выбираться более точный формат представления данных
  - Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, – не могут

Семантическая модель Entity-Relationship (50)

- *Компоненты уникального идентификатора сущности* превращаются в *первичный ключ таблицы*
  - Если имеется несколько возможных уникальных идентификаторов, для первичного ключа выбирается наиболее характерный уникальный идентификатор
  - Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно, и в общем случае может привести к зацикливанию)
  - Для именования этих столбцов используются имена концов связей и/или имена парных типов сущностей

Семантическая модель Entity-Relationship (51)

Получение реляционной схемы из ER-диаграммы (4) Базовые правила (3)

- Связи «многие к одному» (и «один к одному») становятся внешними ключами, т. е. образуется копия уникального идентификатора сущности на конце связи «один», и соответствующие столбцы составляют внешний ключ таблицы, соответствующей типу сущности на конце связи «многие»
  - Необязательные связи соответствуют столбцам внешнего ключа, допускающим наличие неопределенных значений; обязательные связи – столбцам, не допускающим неопределенных значений.
  - Если между двумя типами сущности A и B имеется связь «один к одному», то соответствующий внешний ключ по желанию проектировщика может быть объявлен как в таблице A, так и в таблице B
  - Чтобы отразить в определении таблицы ограничение, которое заключается в том, что степень конца связи должна равняться единице, соответствующий (возможно, составной) столбец должен быть дополнительно специфицирован как возможный ключ таблицы

Семантическая модель Entity-Relationship (52)

Получение реляционной схемы из ER-диаграммы (5) Базовые приемы (4)

- Для поддержки связи «многие ко многим» между типами сущности *A* и *B* создается дополнительная таблица *AB* с двумя столбцами, один из которых содержит уникальные идентификаторы экземпляров сущности *A*, а другой – уникальные идентификаторы экземпляров сущности *B*
  - Обозначим через *УИД(c)* уникальный идентификатор экземпляра некоторого типа сущности *C*
  - Тогда, если в экземпляре связи «многие ко многим» участвуют экземпляры a_1, a_2, …, a_n типа сущности *A* и экземпляры b_1, b_2, …, b_m типа сущности *B*, то в таблице *AB* должны присутствовать все строки вида <УИД(*ai*), УИД(*bj*)> для $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$
  - Понятно, что, используя таблицы *A*, *B* и *AB*, с помощью стандартных реляционных операций можно найти все пары экземпляров типов сущности, участвующих в данной связи

Семантическая модель Entity-Relationship (53)

Получение реляционной схемы из ER-диаграммы (6) Супертипы и подтипы

- (1) Если в концептуальной схеме (ER-диаграмме) присутствуют подтипы сущностей, то возможны два способа их представления в реляционной схеме:
  - собрать все подтипы в одной таблице;
  - для каждого подтипа образовать отдельную таблицу
- При применении первого способа таблица создается для максимального супертипа (типа сущности, не являющегося подтипом), а для подтипов определяются представления
- Таблица содержит столбцы, соответствующие каждому атрибуту (и связям) каждого подтипа
  - В таблицу добавляется, по крайней мере, один столбец, содержащий «код типа»; он становится частью первичного ключа
  - Для каждой строки таблицы значение этого столбца определяется конкретный тип сущности, экземпляру которого соответствует строка
  - Столбцы этой строки, которые соответствуют атрибутам и связям, отсутствующим в данном типе сущности, должны содержать неопределенные значения

Семантическая модель Entity-Relationship (54)

Получение реляционной схемы из ER-диаграммы (7) Супертипы и подтипы

(2) При использовании второго метода для каждого подтипа первого уровня (непосредственного подтипа максимального супертипа) создается отдельная таблица

- Для более глубоких уровней наследования применяется первый метод
- Супертип воссоздается с помощью объединения проекций таблиц, соответствующих подтипам, на заголовок таблицы супертипа (т.е. из всех таблиц подтипов выбираются общие столбцы – столбцы супертипа)

Семантическая модель Entity-Relationship (55)

Получение реляционной схемы из ER-диаграммы (8) Супертипы и подтипы

- У каждого способа есть свои достоинства и (3) недостатки
- К достоинствам первого способа (общая таблица для супертипа и всех его подтипов) можно отнести следующее:
  - соответствие логике супертипов и подтипов: поскольку любой экземпляр любого подтипа является экземпляром супертипа, логично хранить вместе все строки, соответствующие экземплярам супертипа;
  - обеспечение простого доступа к экземплярам супертипа и не слишком сложный доступ к экземплярам подтипов;
  - возможность обойтись небольшим числом таблиц

Семантическая модель Entity-Relationship (56)

Получение реляционной схемы из ER-диаграммы (9) Супертипы и подтипы

- Недостатки первого метода:

(4)

- приложения, работающие с одной таблицей супертипа, должны содержать дополнительный программный код для работы с разными наборами столбцов (в зависимости от значения столбца «кода типа») и разными ограничениями целостности (в зависимости от особенностей связей, определенных для подтипа);
- общая для всех подтипов таблица потенциально может стать узким местом при многопользовательском доступе по причине возможности блокировки таблицы целиком;
- потенциально в общей таблице будет содержаться много неопределенных значений, что может привести к непроизводительному расходу внешней памяти

Семантическая модель Entity-Relationship (57)

Получение реляционной схемы из ER-диаграммы (10) Супертипы и подтипы

- Достоинства второго метода состоят в следующем:

(5)

  - действуют более понятные правила работы с подтипами (каждому подтипу соответствует одноименная таблица);
  - упрощается логика приложений; каждая программа работает только с нужной таблицей

- Недостатки второго метода:
  - в общем случае требуется слишком много отдельных таблиц;
  - работа с экземплярами супертипа на основе представления, объединяющего таблицы супертипов, может оказаться недостаточно эффективной;
  - поскольку множество экземпляров супертипа является объединением множеств экземпляров подтипов, не все РСУБД могут обеспечить выполнение операций модификации экземпляров супертипа

Семантическая модель Entity-Relationship (58)

Получение реляционной схемы из ER-диаграммы (11) Взаимноисключающие связи (1)

- Как отмечалось ранее, ER-диаграмму с взаимно исключающими связями можно преобразовать к диаграмме с подтипами исходного типа сущности
- Однако можно построить схему SQL-ориентированной базы данных и без такого преобразования.
- Существуют два способа формирования схемы реляционной БД при наличии взаимно исключающих связей (имеются в виду связи «один ко многим», причем конец связи «многие» находится на стороне сущности, для которой связи являются взаимно исключающими):
  - определение таблицы с одним столбцом для представления всех взаимно исключающих связей, т.е. общее хранение внешних ключей;
  - определение таблицы, в которой каждой взаимно исключающей связи соответствует отдельный столбец, т.е. раздельное хранение внешних ключей

Семантическая модель Entity-Relationship (59)

- Понятно, что если имеются взаимно исключающие связи упомянутой категории, то в таблице, соответствующей сущности, для которой связи являются взаимно исключающими, необходимо хранить внешние ключи
- Если внешние ключи всех потенциально связанных таблиц имеют общий формат, то можно применить первый способ, т. е. создать два столбца, содержащие идентификатор связи и уникальный идентификатор соответствующей сущности (второй столбец может быть составным)
  - Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения.
- Если результирующие внешние ключи не относятся к одному домену, то приходится прибегать к использованию второго способа, т. е. создавать для каждой связи, покрываемой дугой исключения, явные столбцы внешних ключей;
  - каждый из этих столбцов может содержать неопределенные значения

Семантическая модель Entity-Relationship (60)

- Преимущество первого подхода состоит в том, что в таблице, соответствующей сущности с взаимно исключающими связями, появляется всего два дополнительных столбца
- Очевидным недостатком является усложнение выполнения операции соединения: чтобы воспользоваться для соединения одной из альтернативных связей, нужно сначала произвести ограничение таблицы в соответствии с нужным значением столбца, содержащего идентификаторы связей.
- При использовании второго подхода соединения являются явными (и естественными)
- Недостаток состоит в том, что требуется иметь столько столбцов, сколько имеется альтернативных связей
- Кроме того, в каждом из таких столбцов будет содержаться много неопределенных значений, хранение которых может привести к излишнему расходу внешней памяти

Семантическая модель Entity-Relationship (61)

- На этом мы заканчиваем краткую экскурсию в семантическое моделирование с использованием ER-диаграмм
- Основной целью этого раздела было ознакомление с семантическими моделями данных на примере упрощенного варианта ER-модели
- Представленный вариант ER-модели, с одной стороны, является достаточно развитым, чтобы можно было почувствовать общую специфику семантических моделей данных, а с другой стороны, не перегружен деталями и излишними понятиями, затрудняющими общее понимание подхода
- С практической точки зрения наибольшую пользу могут принести рассмотренные приемы перехода от ER-диаграмм к схеме реляционной базы данных
- Особенно могут пригодиться рекомендации по представлению в реляционной схеме связей «многие ко многим», подтипов и супертипов сущности и взаимно исключающих связей

Диаграммы классов языка UML (1)

- Теперь мы обсудим основные понятия диаграмм классов языка UML и возможности применения этой диаграммной модели для проектирования реляционных баз данных
- Кроме того, будет кратко рассмотрен язык объектных ограничений OCL и приведены примеры формулировок на языке OCL ограничений целостности в терминах концептуальной схемы базы данных
- Языку объектно-ориентированного моделирования UML (Unified Modeling Language) посвящено великое множество книг, многие из которых переведены на русский язык (а некоторые и написаны российскими авторами)
- Язык UML разработан и развивается консорциумом OMG (Object Management Group) и имеет много общего с объектными моделями, на которых основана технология распределенных объектных систем CORBA, и объектной моделью ODMG (Object Data Management Group)

Диаграммы классов языка UML (2)

- UML позволяет моделировать разные виды систем: чисто программные, чисто аппаратные, программно-аппаратные, смешанные, явно включающие деятельность людей и т. д.
- Но, помимо прочего, язык UML активно применяется для проектирования реляционных БД
- Для этого используется небольшая часть языка (диаграммы классов), да и то не в полном объеме
- С точки зрения проектирования реляционных БД модельные возможности не слишком отличаются от возможностей ER-диаграмм
- Но все же мы кратко опишем диаграммы классов UML, поскольку их использование при проектировании реляционных БД позволяет оставаться в общем контексте UML и применять другие виды диаграмм для проектирования приложений баз данных

Диаграммы классов языка UML (3)

Основные понятия диаграмм классов UML (1)

- Диаграммой классов в терминологии UML называется диаграмма, на которой показан набор классов (и некоторых других сущностей, не имеющих явного отношения к проектированию БД), а также связей между этими классами
- Кроме того, диаграмма классов может включать комментарии и ограничения
- Здесь следует сделать два замечания
  - Во-первых, в этом разделе термин *сущность* используется настолько же неформально, как в предыдущем разделе использовался термин *объект*
  - UML претендует на обеспечение более точного и формального понятия *объекта* (UML обычно называют языком о*бъектно-ориентированного моделирования*)
  - В спецификации языка UML даже присутствует определение понятия объекта средствами самого UML

Диаграммы классов языка UML (4)

Основные понятия диаграмм классов UML (2)

- Однако, несмотря на эти попытки, понятие *объекта* в UML остается таким же нечетким, как и понятие *сущности* в ER-модели
- По-прежнему приходится опираться в основном на интуицию и здравый смысл

- Во-вторых, в UML, как и в модели ER-диаграмм, для родового обозначения связей используется термин *relationship*
  - Во многих переводах книг про UML на русский язык вместо термина *связь* применяется термин *отношение*
  - Как и в предыдущем разделе, мы используем термин *связь*
- Для диаграмм классов UML могут задаваться ограничения на естественном языке или же на языке объектных ограничений OCL (Object Constraints Language)
- Язык OCL является частью общей спецификации UML, но, в отличие от других частей языка, имеет не графическую, а линейную нотацию
  - Более подробно язык OCL обсуждается в конце лекции

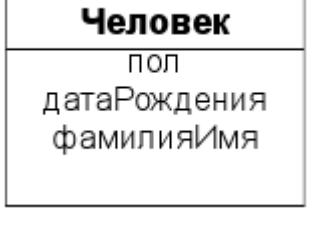
Диаграммы классов языка UML (5)

Основные понятия диаграмм классов UML (3) Классы, атрибуты, операции (1)

- *Классом* называется именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой
- Графически класс изображается в виде прямоугольника
- У каждого класса должно быть имя (текстовая строка), уникально отличающее его от всех других классов
- При формировании имен классов в UML допускается использование произвольной комбинации букв, цифр и даже знаков препинания
- Однако на практике рекомендуется использовать в качестве имен классов короткие и осмысленные прилагательные и существительные, каждое из которых начинается с заглавной буквы
- Примеры описания классов показаны на рисунке

Диаграммы классов языка UML (6)

Основные понятия диаграмм классов UML (4) Классы, атрибуты, операции (2)



- *Атрибутом класса* называется именованное *свойство* класса, описывающее множество значений, которые могут принимать экземпляры этого свойства
- Класс может иметь любое число атрибутов (в частности, не иметь ни одного атрибута)
- Свойство, выражаемое атрибутом, является свойством моделируемой сущности, общим для всех объектов данного класса
  - ✓ Так что атрибут является абстракцией состояния объекта
- Имена атрибутов представляются в разделе класса, расположенном под именем класса
- На практике для имен атрибутов рекомендуется использовать короткие прилагательные и существительные, отражающие смысл соответствующего свойства класса
- Первое слово в имени атрибута рекомендуется писать с прописной буквы, а все остальные слова – с заглавной
- Пример описания класса с указанными атрибутами показан на рисунке

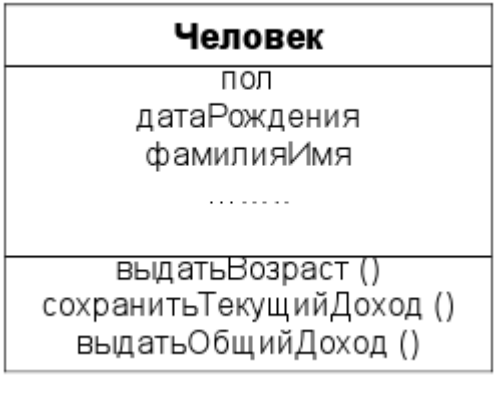
Диаграммы классов языка UML (7)

Основные понятия диаграмм классов UML (5) Классы, атрибуты, операции (3)

- *Операцией* класса называется именованная услуга, которую можно запросить у любого объекта этого класса
- Операция – это абстракция того, что можно делать с объектом
- Класс может содержать любое число операций (в частности, не содержать ни одной операции)
- Набор операций класса является общим для всех объектов данного класса.
- Операции класса определяются в разделе, расположенном ниже раздела с атрибутами
- Можно ограничиться только указанием имен операций, оставив детальную спецификацию выполнения операций на более поздние этапы моделирования
- Для именования операций рекомендуется использовать глагольные формы, соответствующие ожидаемому поведению объектов данного класса
- Описание операции может также содержать ее *сигнатуру*, т. е. имена и типы всех параметров, а если операция является функцией, то и тип ее значения

Диаграммы классов языка UML (8)

Основные понятия диаграмм классов UML (6) Классы, атрибуты, операции (4)



- Для класса Человек определены три операции:
  - ✓ выдатьВозраст,
  - ✓ сохранитьТекущийДоход,
  - ✓ выдатьОбщийДоход
- В операции выдатьВозраст используются значение атрибута датаРождения и значение текущей даты
- Операция сохранитьТекущийДоход позволяет зафиксировать в состоянии объекта сумму и дату поступления дохода данного человека
- Операция выдатьОбщийДоход выдает суммарный доход данного человека за указанное время
- Заметим, что состояние объекта меняется при выполнении только второй операции
- Результаты первой и третьей операций формируются на основе текущего состояния объекта

Диаграммы классов языка UML (9)

Основные понятия диаграмм классов UML (7) Категории связей. Связь-зависимость (1)

- В диаграмме классов могут участвовать связи трех разных категорий:
  - *зависимость* (dependency),
  - *обобщение* (generalization) и
  - *ассоциация* (association)
- При проектировании реляционных БД наиболее важны вторая и третья категории связей, поэтому о связях-зависимостях будет сказано только самое основное.
- *Зависимостью* называют связь по применению, когда изменение в спецификации одного класса может повлиять на поведение другого класса, использующего первый класс
- Чаще всего зависимости применяют в диаграммах классов, чтобы отразить в сигнатуре операции одного класса тот факт, что параметром этой операции могут быть объекты другого класса
- Понятно, что если интерфейс второго класса изменяется, это влияет на поведение объектов первого класса

Диаграммы классов языка UML (10)

Основные понятия диаграмм классов U

(2)

- Пример диаграммы классов со связью-зависимостью
- В классе РасписаниеЗанятий определены две операции с очевидной семантикой, параметрами которых являются объекты класса Курс

- При изменении интерфейса класса Курс изменится поведение объектов класса РасписаниеЗанятий.
- Зависимость показывается прерывистой линией со стрелкой, направленной к классу, от которого имеется зависимость
- Связи-зависимости существенны для объектно-ориентированных систем (в том числе и для ООБД)
- При проектировании традиционных РБД непонятно, как использовать информацию о наличии связей-зависимостей между классами

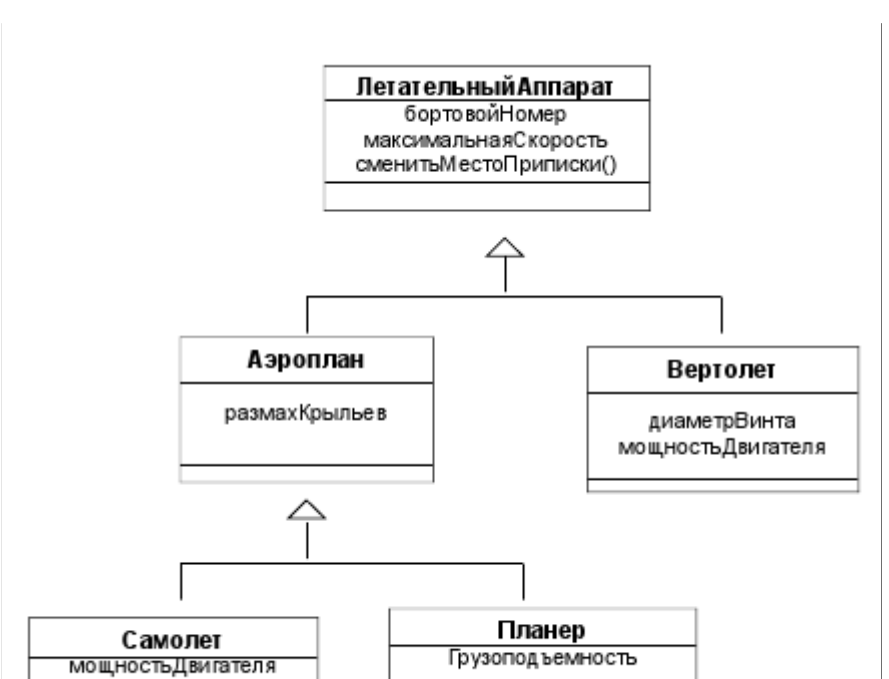
Диаграммы классов языка UML (10)

- *Связью-обобщением* называется связь между общей сущностью, называемой *суперклассом*, или *родителем*, и более специализированной разновидностью этой сущности, называемой *подклассом*, или *потомком*
- Обобщения иногда называют связями «*is a*», имея в виду, что класс-потомок является частным случаем класса-предка
- Класс-потомок наследует все атрибуты и операции класса-предка, но в нем могут быть определены дополнительные атрибуты и операции.
- Объекты класса-потомка могут использоваться везде, где могут использоваться объекты класса-предка
- Это свойство называют полиморфизмом по включению, имея в виду, что объекты потомка можно считать включаемыми во множество объектов класса-предка
- Графически обобщения изображаются в виде сплошной линии с большой незакрашенной стрелкой, направленной к суперклассу

Диаграммы классов языка UML (11)

Основные понятия диаграмм класс

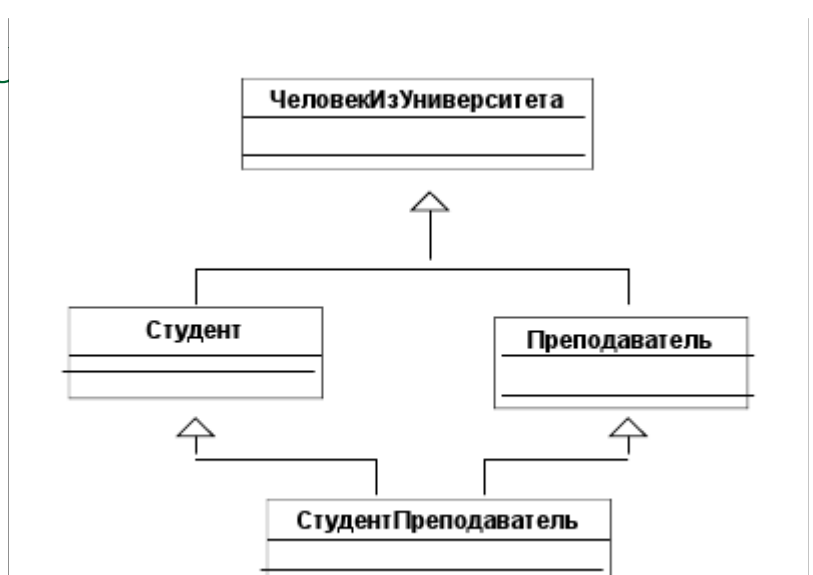
- Здесь показан пример иерархии одиночного наследования: у каждого подкласса имеется только один суперкласс.
- В отличие от механизма наследования типов сущностей ER-модели здесь отсутствует класс ПРОЧИЕ, т.е. в классе ЛетательныйАппарат могут присутствовать «собственные» объекты, не относящиеся ни к классу Аэроплан, ни к классу Вертолет



- Одиночное наследование является достаточным в большинстве случаев применения связи-обобщения

Диаграммы классов языка UML (12)

- Однако в UML допускается и множественное наследование, когда один подкласс определяется на основе нескольких суперклассов
- В качестве одного из разумных (не слишком распространенных) примеров рассмотрим данную диаграмму классов
- На этой диаграмме классы Студент и Преподаватель порождены из одного суперкласса ЧеловекИзУниверситета

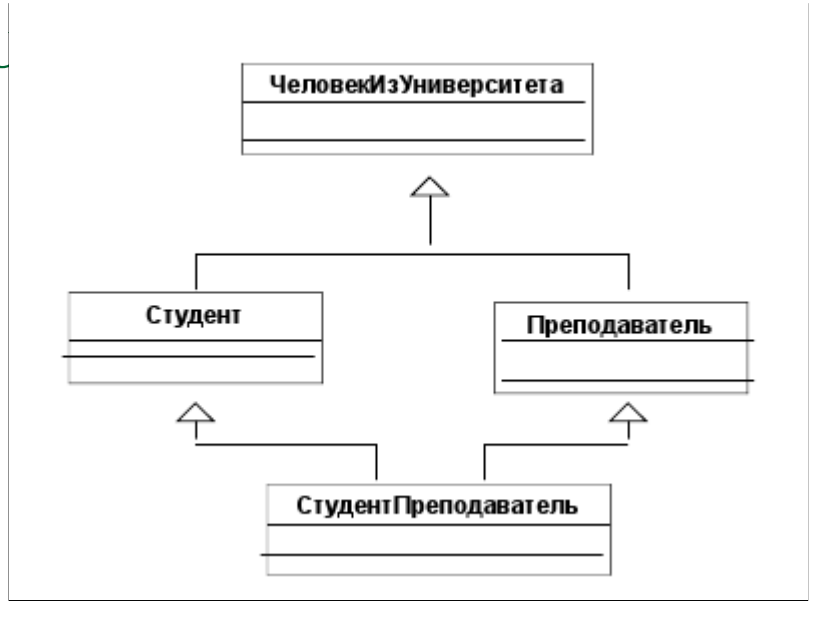


- Вообще говоря, к классу Студент относятся те объекты класса ЧеловекИзУниверситета, которые соответствуют студентам, а к классу Преподаватель – объекты класса ЧеловекИзУниверситета, соответствующие преподавателям

Диаграммы классов языка UML (13)

Основные понятия диаграмм классов U

- Но, как это часто случается, многие студенты уже в студенческие годы начинают преподавать, так что могут существовать такие два объекта классов Студент и Преподаватель, которым соответствует один объект класса ЧеловекИзУниверситета
- Итак, среди объектов класса Студент могут быть преподаватели, а некоторые преподаватели могут быть студентами

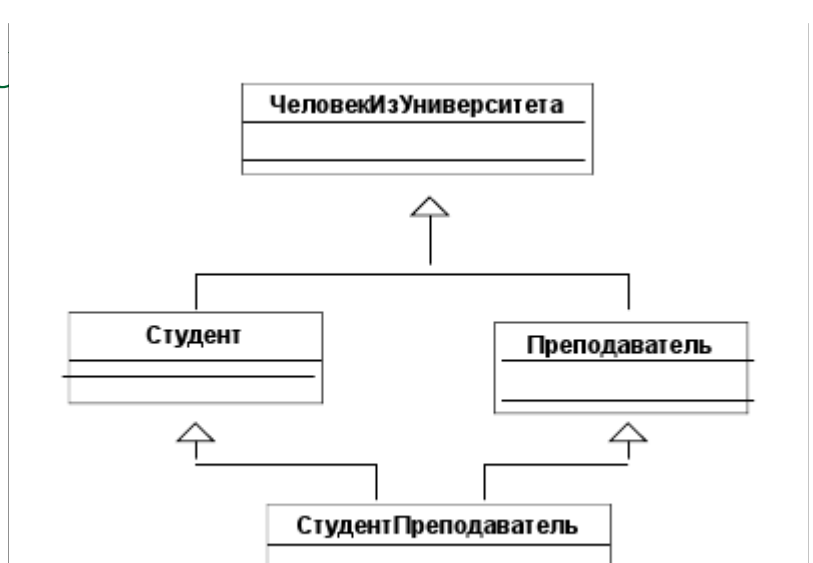


- Тогда мы можем определить класс СтудентПреподаватель путем множественного наследования от суперклассов Студент и Преподаватель

Диаграммы классов языка UML (14)

- Так что полиморфизм по включению продолжает работать
- Заметим, что для этого пришлось отказаться от еще одного свойства механизма наследования ER-модели – отсутствие общих экземпляров у подтипов одного типа сущности
- В данном случае множественное наследование возможно именно потому, что в классы Студент и Преподаватель входят разные объекты, которым соответствует один и тот же объект суперкласса

Основные понятия диаграмм классов U



Диаграммы классов языка UML (15)

- Следует также отметить, что множественное наследование, помимо того, что не слишком часто требуется на практике, порождает ряд проблем, из которых одной из наиболее известных является проблема именования атрибутов и операций в подклассе, полученном путем множественного наследования
- Например, предположим, что при образовании подклассов Студент и Преподаватель в них обоих был определен атрибут с именем номерКомнаты
- Очень вероятно, что для объектов класса Студент значениями этого атрибута будут номера комнат в студенческом общежитии, а для объектов класса Преподаватель – номера служебных кабинетов
- Как быть с объектами класса СтудентПреподаватель, для которых существенны оба одноименных атрибута (у студента-преподавателя могут иметься и комната в общежитии, и служебный кабинет)?

Диаграммы классов языка UML (16)

- Основные конструкции диаграмм классов UML (14) | Связь «обобщение» (7)
- На практике применяется одно из следующих решений:
  - запретить образование подкласса СтудентПреподаватель, пока в одном из суперклассов не будет произведено переименование атрибута номерКомнаты;
  - наследовать это свойство только от одного из суперклассов, так что, например, значением атрибута номерКомнаты у объектов класса СтудентПреподаватель всегда будут номера служебных кабинетов;
  - унаследовать в подклассе оба свойства, но автоматически переименовать оба атрибута, чтобы прояснить их смысл; назвать их, например, номерКомнатыСтудента и номерКомнатыПреподавателя

Диаграммы классов языка UML (17)

- Ни одно из решений не является полностью удовлетворительным
  - Первое решение требует возврата к ранее определенному классу, имена атрибутов и операций которого, возможно, уже используются в приложениях
  - Второе решение нарушает логику наследования, не давая возможности на уровне подкласса использовать все свойства суперклассов
  - Наконец, третье решение заставляет использовать длинные имена атрибутов и операций, которые могут стать недопустимо длинными, если процесс множественного наследования будет продолжаться от полученного подкласса
- Но, конечно, сложность проблемы именования атрибутов и операций несопоставимо меньше сложности реализации множественного наследования в реляционных БД
- Поэтому при использовании UML для проектирования реляционных БД нужно очень осторожно использовать наследование классов вообще и стараться избегать множественного наследования

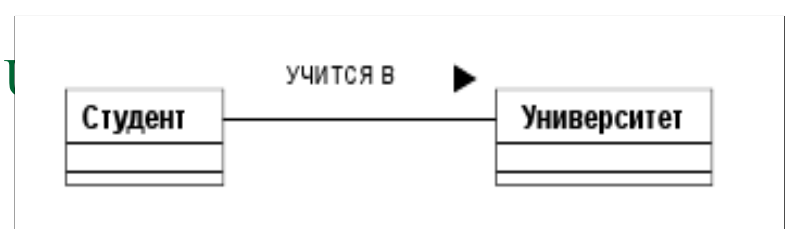
Диаграммы классов языка UML (18)

- Ассоциацией называется структурная связь, показывающая, что объекты одного класса некоторым образом связаны с объектами другого или того же самого класса
- Допускается, чтобы оба конца ассоциации относились к одному классу
- В ассоциации могут связываться два класса, и тогда она называется *бинарной*
- Допускается создание ассоциаций, связывающих сразу *n* классов (они называются *n-арными* ассоциациями)
  - мы ограничимся обсуждением бинарных ассоциаций
- Графически ассоциация изображается в виде линии, соединяющей класс сам с собой или с другими классами.
- С понятием ассоциации связаны четыре важных дополнительных понятия: *имя*, *роль*, *кратность* и *агрегация*

Диаграммы классов языка UML (19)

Основные понятия диаграмм классов U

- Ассоциации может быть присвоено *имя*, характеризующее природу связи

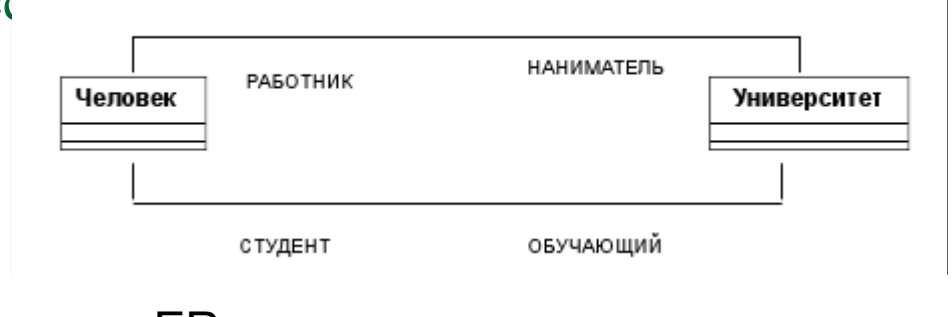


- Смысл имени уточняется с помощью черного треугольника, который располагается над линией связи справа или слева от имени ассоциации
- Этот треугольник указывает направление чтения имя связи
- Здесь показан пример именованной ассоциации
- Треугольник показывает, что именованная ассоциация должна читаться как «Студент учится в Университете»

Диаграммы классов языка UML (20)

Основные понятия диаграмм класс

- Другим способом именования ассоциации является указание *роли* каждого класса, участвующего в этой ассоциации



- Роль класса, как и имя конца связи в ER-модели, задается именем, помещаемым под линией ассоциации ближе к данному классу
- На рисунке показаны две ассоциации между классами Человек и Университет, в которых эти классы играют разные роли
- Как мы видим, объекты класса Человек могут выступать в роли РАБОТНИКОВ при участии в ассоциации, в которой объекты класса Университет играют роль НАНИМАТЕЛЯ
- В другой ассоциации объекты класса Человек играют роль СТУДЕНТА, а объекты класса УНИВЕРСИТЕТ – роль ОБУЧАЮЩЕГО

Диаграммы классов языка UML (21)

- В общем случае, для ассоциации могут задаваться и ее собственное имя, и имена ролей классов
- Это связано с тем, что класс может играть одну и ту же роль в разных ассоциациях, так что в общем случае пара имен ролей классов не идентифицирует ассоциацию
- С другой стороны, в простых случаях, когда между двумя классами определяется только одна ассоциация, можно вообще не связывать с ней дополнительные имена.
- *Кратностью* (multiplicity) роли ассоциации называется характеристика, указывающая, сколько объектов класса с данной ролью может или должно участвовать в каждом экземпляре ассоциации
- В терминологии UML экземпляр ассоциации называется *соединением* – link, но мы не будем здесь использовать этот термин, чтобы не создавать путаницу
  - трудно одновременно говорить про *связи*, *ассоциации* и *соединения*, имея в виду разные понятия

Диаграммы классов языка UML (22)

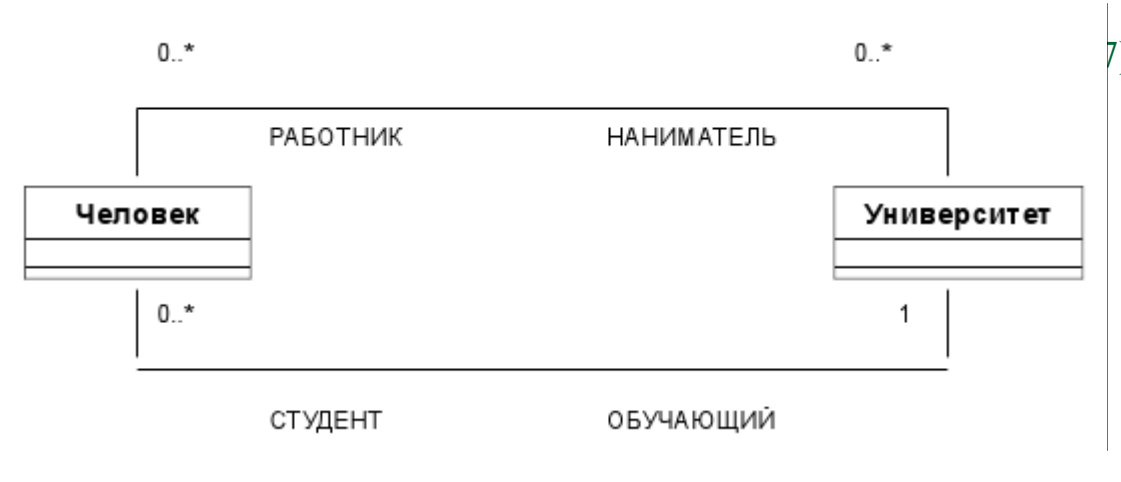
- Наиболее распространенным способом задания кратности роли ассоциации является указание конкретного числа или диапазона
- Например, указание «1» говорит о том, что каждый объект класса с данной ролью должен участвовать в некотором экземпляре данной ассоциации, причем в каждом экземпляре ассоциации может участвовать ровно один объект класса с данной ролью
- Указание диапазона «0..1» говорит о том, что не все объекты класса с данной ролью обязаны участвовать в каком-либо экземпляре данной ассоциации, но в каждом экземпляре ассоциации может участвовать только один объект

Диаграммы классов языка UML (23)

- Аналогично, указание диапазона «1..*» говорит о том, что все объекты класса с данной ролью должны участвовать в некотором экземпляре данной ассоциации, и в каждом экземпляре ассоциации должен участвовать хотя бы один объект (верхняя граница не задана)
- Толкование диапазона «0..*» является очевидным расширением случая «0..1»
- В более сложных (но крайне редко встречающихся на практике) случаях определения кратности можно использовать списки диапазонов
- Например, список «2, 4..6, 8..*» говорит о том, что все объекты класса с указанной ролью должны участвовать в некотором экземпляре данной ассоциации, и в каждом экземпляре ассоциации должны участвовать два, от четырех до шести или более семи объектов класса с данной ролью

Диаграммы классов языка UML (24)

Основные пон



- На диаграмме показано, что произвольное (может быть, нулевое) число людей являются сотрудниками произвольного числа университетов
- Каждый университет обучает произвольное (может быть, нулевое) число студентов, но каждый студент может быть студентом только одного университета

Диаграммы классов языка UML (25)

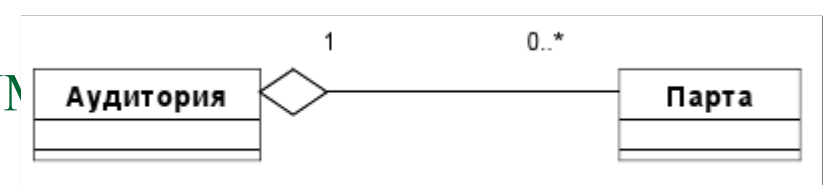
Основные понятия диаграмм классов UML (23) – связь ассоциация (1)

- Обычная ассоциация между двумя классами характеризует связь между равноправными сущностями: оба класса находятся на одном концептуальном уровне
- Но иногда в диаграмме классов требуется отразить тот факт, что ассоциация между двумя классами имеет специальный вид «часть-целое»
- В этом случае класс «целое» имеет более высокий концептуальный уровень, чем класс «часть»
- Ассоциация такого рода называется *агрегатной*
- Графически агрегатные ассоциации изображаются в виде простой ассоциации с незакрашенным ромбом на стороне класса-«целого»

Диаграммы классов языка UML (26)

Основные понятия диаграмм классов UML

- Объектами класса Аудитория являются студенческие аудитории, в которых проходят занятия



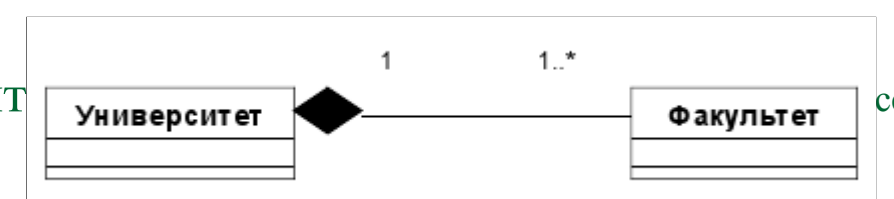
- В каждой аудитории должны быть установлены парты
- Поэтому в некотором смысле класс Парта является «частью» класса Аудитория
- Мы умышленно сделали роль класса Парта в этой ассоциации необязательной, поскольку могут существовать аудитории без парт (например, класс для занятий танцами) и некоторые парты могут находиться на складе
- Обратите внимание, что, хотя аудитории, не оснащенные партами, как правило, непригодны для занятий, объекты классов Аудитория и Парта существуют независимо
- Если некоторая аудитория ликвидируется, то находящиеся в ней парты не уничтожаются, а переносятся на склад

Диаграммы классов языка UML (27)

Основные понятия диаграмм классов UML (25) | Связь «часть-целое» (26)

- Бывают случаи, когда связь «части» и «целого» настолько сильна, что уничтожение «целого» приводит к уничтожению всех его «частей»
- Агрегатные ассоциации, обладающие таким свойством, называются *композитными*, или просто *композициями*
- При наличии композиции объект-часть может быть частью только одного объекта-целого (композита)
- При обычной агрегатной ассоциации «часть» может одновременно принадлежать нескольким «целым»
- Графически композиция изображается в виде простой ассоциации, дополненной закрашенным ромбом со стороны «целого»

Диаграммы классов языка UML (28)

Основные понят 

ссоциации (11)

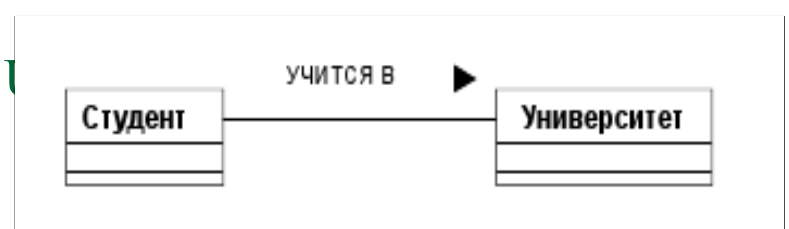
- Пример композитной агрегатной ассоциации
- Любой факультет является частью одного университета, и ликвидация университета приводит к ликвидации всех существующих в нем факультетов
  - ✓ хотя во время существования университета отдельные факультеты могут ликвидироваться и создаваться

Диаграммы классов языка UML (29)

- Заметим, что в контексте проектирования реляционных БД агрегатные и в особенности композитные ассоциации влияют только на способ поддержки ссылочной целостности
- В частности, композитная связь является явным указанием на то, что ссылочная целостность между «целым» и «частями» должна поддерживаться путем каскадного удаления частей при удалении целого

Диаграммы классов языка UML (30)

Основные понятия диаграмм классов U

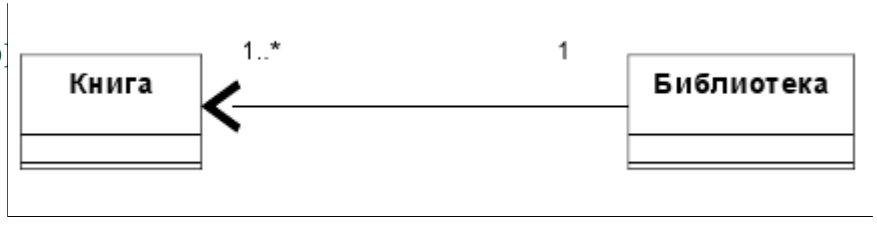


- При наличии простой ассоциации между двумя классами предполагается возможность *навигации* между объектами, входящими в один экземпляр ассоциации
- Если известен конкретный объект-студент, то должна обеспечиваться возможность узнать соответствующий объект-университет
- Если известен конкретный объект-университет, то должна обеспечиваться возможность узнать все соответствующие объекты-студенты
- Другими словами, если не оговорено иное, то навигация по ассоциации может проводиться в обоих направлениях
- Однако бывают случаи, когда желательно ограничить направление навигации для некоторых ассоциаций
- В этом случае на линии ассоциации ставится стрелка, указывающая направление навигации

Диаграммы классов языка UML (31)

- Пример диаграммы классов с однонаправленной навигацией

Основные понятия диаграмм классо

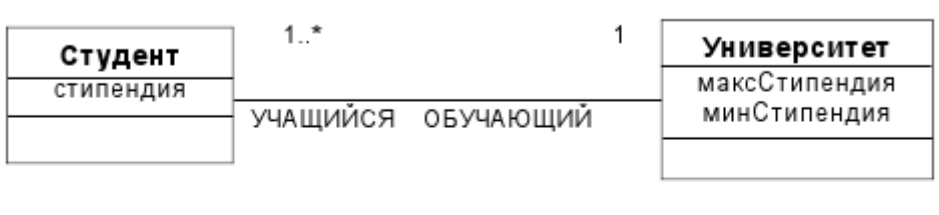


- В библиотеке должно содержаться некоторое количество книг, и каждая книга должна принадлежать некоторой библиотеке
- С точки зрения библиотечного хозяйства разумно иметь возможность найти книгу в библиотеке, т. е. произвести навигацию от объекта-библиотеки к связанным с ним объектам-книгам
- Однако вряд ли потребуется по данному экземпляру книги узнать, в какой библиотеке она находится

Диаграммы классов языка UML (32)

Ограничения целостности

- В диаграммах классов могут указываться ограничения целостности, которые должны поддерживаться в проектируемой базе данных

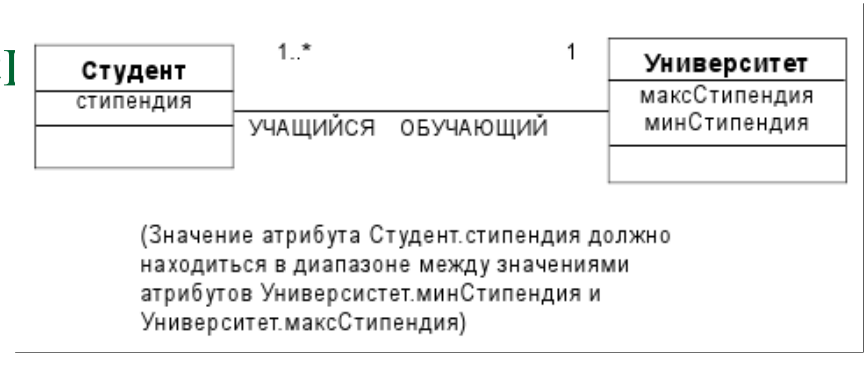


- В UML допускаются два способа определения ограничений: на естественном языке и на языке OCL
- В данном случае накладывается ограничение на состояние объектов классов Студент и Университет, входящих в один экземпляр ассоциации
- Объект класса Студент может входить в экземпляр ассоциации с объектом класса Университет только при условии, что размер стипендии данного студента находится в диапазоне, допустимом в данном университете

Диаграммы классов языка UML (33)

Ограничения целостности и язык OCl

- Более точный и лаконичный способ формулировки ограничений обеспечивает язык OCL (Object Constraints Language)



- Вот, например, формулировка на языке OCL ограничения, показанного на рисунке:

```
context Студент inv:
self.стипендия ≥ self.обучающий.минСтипендия
and
self.стипендия ≤ self.обучающий.максСтипендия
```

Диаграммы классов языка UML (34)

- Хотя язык OCL формально считается частью UML, он специфицирован в отдельном документе, в котором присутствуют ссылки на другие части спецификации UML, а также вводятся собственные понятия и определения
- Из языка UML в OCL заимствованы следующие понятия:
  - класс, атрибут, операция;
  - объект (экземпляр класса);
  - связь-ассоциация;
  - тип данных (включая набор предопределенных типов Boolean, Integer, Real и String);
  - значение (экземпляр типа данных)

Диаграммы классов языка UML (35)

- Для понимания языка OCL существенны определяемые в UML традиционные для объектных моделей данных различия между объектом некоторого класса и значением некоторого типа:
  - Объект обладает уникальным идентификатором и может сравниваться с другими объектами только по значению идентификатора
    - Следствием этого является возможность определения операций над множествами объектов в терминах их идентификаторов
    - Следует заметить, что ни в спецификации UML, ни в описании какой-либо другой объектной модели никогда прямо не говорится, что в операциях над множествами объектов в действительности участвуют идентификаторы объектов
    - Но другого понимания не существует

Диаграммы классов языка UML (36)

- Объект может быть ассоциирован через бинарную связь с другими объектами, что позволяет определить в OCL операцию перехода от данного объекта к связанным с ним объектам
  - Заметим, что хотя в UML допускаются *n*-арные связи, в OCL речь идет только об уже привычном для нас бинарном варианте
- В то же время значение является «чистым значением» в том смысле, что:
  - при сравнении двух значений проверяются сами эти значения;
  - значения не могут участвовать в связях, поскольку понятие связи определено только для объектов классов

Диаграммы классов языка UML (37)

- В дополнение к *скалярным* типам данных, заимствованным из UML, в OCL предопределены *структурные* типы, которые являются разновидностями типов коллекций (*collection*):
  - математическое множество (set),
    - неупорядоченная коллекция, не содержащая одинаковых элементов;
  - *мультимножество* (bag),
    - неупорядоченная коллекция, которая может содержать повторяющиеся элементы-дубликаты;
  - *последовательность* (sequence),
    - упорядоченная коллекция, которая может содержать элементы-дубликаты

Диаграммы классов языка UML (38)

- В дополнение к *скалярным* типам данных, заимствованным из UML, в OCL предопределены *структурные* типы, которые являются разновидностями типов коллекций (*collection*):
  - математическое множество (set),
    - неупорядоченная коллекция, не содержащая одинаковых элементов;
  - *мультимножество* (bag),
    - неупорядоченная коллекция, которая может содержать повторяющиеся элементы-дубликаты;
  - *последовательность* (sequence),
    - упорядоченная коллекция, которая может содержать элементы-дубликаты
- Элементами каждого из трех типов коллекций могут быть объекты или значения одного класса или одного типа соответственно

Диаграммы классов языка UML (39)

- Язык OCL предназначен, главным образом, для определения ограничений целостности данных, соответствующих модели, которая представлена в терминах диаграммы классов UML
- OCL может применяться для определения
  - ограничений, описывающих пред- и постусловия операций классов, и
  - ограничений, представляющих собой инварианты классов
- С точки зрения определения ограничений целостности баз данных более важны средства определения инвариантов классов

Диаграммы классов языка UML (40)

- Под инвариантом класса в OCL понимается условие, которому должны удовлетворять все объекты данного класса
- Если говорить более точно, инвариант класса – это логическое выражение, при вычислении которого для любого объекта данного класса в течение всего времени существования этого объекта получается булевское значение *true*
- При определении инварианта требуется указать имя класса и выражение, определяющее инвариант указанного класса

Диаграммы классов языка UML (41)

- Синтаксически это выглядит следующим образом:

```
context <class_name> inv:
<OCL-выражение>
```

- <class-name> является именем класса, для которого определяется инвариант
- inv – ключевое слово, говорящее о том, что определяется именно инвариант, а не ограничение другого вида
- context – ключевое слово, которое говорит о том, что контекстом следующего после двоеточия OCL-выражения являются объекты класса <class-name>,
  - т. е. OCL-выражение должно принимать значение *true* для всех объектов этого класса
- OCL является типизированным языком, поэтому у каждого выражения имеется некоторый тип (тип значения, получаемого при вычислении выражения)
  - OCL-выражение в инварианте класса должно быть логического типа

Диаграммы классов языка UML (42)

- В общем случае OCL-выражение в определении инварианта основывается на композиции операций, которым посвящена большая часть определения языка
- В спецификации языка эти операции условно разделены на следующие группы:
  - операции над значениями предопределенных в UML (скалярных) типов данных;
  - операции над объектами;
  - операции над множествами;
  - операции над мультимножествами;
  - операции над последовательностями
- Последовательно обсудим эти группы операций

Диаграммы классов языка UML (43)

Ограничения целостности на языке OCL (12). Инварианты (4)

- Синтаксически это выглядит следующим образом:

```
context <class_name> inv:
<OCL-выражение>
```

- <class-name> является именем класса, для которого определяется инвариант
- inv – ключевое слово, говорящее о том, что определяется именно инвариант, а не ограничение другого вида
- context – ключевое слово, которое говорит о том, что контекстом следующего после двоеточия OCL-выражения являются объекты класса <class-name>,
  - т. е. OCL-выражение должно принимать значение *true* для всех объектов этого класса

Диаграммы классов языка UML (44)

Ограничения целостности и язык OCL (

- **Операции над значениями предопределенных типов данных**
- В OCL поддерживаются скалярные типы данных Boolean, Integer, Real и String
- Операция xor – это стандартное «исключающее или»
- Она принимает два параметра булевского типа и вырабатывает значение *true*, если значением одного и только одного параметра является *true*
- В противном случае операция вырабатывает значение *false*
- Операция implies – это импликация
- Она принимает два параметра булевского типа и вырабатывает значение *true*, если значением первого параметра является *false*, или если значениями обоих параметров является *true*
- В противном случае операция вырабатывает значение *false*

Предопределенный скалярный тип	Список операций
Boolean	and, or, xor, not, implies, if-then-else
Integer	*, +, -, /, abs(), операции сравнения
Real	*, +, -, /, floor(), операции сравнения
String	concat(), size(), substring()

Диаграммы классов языка UML (45)

- Ограничения целостности и язык OCL (
- Операция floor вырабатывает наибольшее значение целого типа, меньшее или равное значению параметра операции
- Операция concat конкатенирует две строки-аргументы
- size выдает целое значение, равное длине строки-аргумента

Предопределенный скалярный тип	Список операций
Boolean	and, or, xor, not, implies, if-then-else
Integer	*, +, -, /, abs(), операции сравнения
Real	*, +, -, /, floor(), операции сравнения
String	concat(), size(), substring()

- substring выдает подстроку строки-аргумента с заданными начальной позицией и длиной

Диаграммы классов языка UML (46)

- **Операции над объектами**
- В OCL определены три операции над объектами:
  - получение значения атрибута;
  - переход по экземпляру ассоциации,
  - вызов операции класса (последняя операция для целей проектирования «традиционных» реляционных БД несущественна)
- Для обозначения всех трех этих операций используется «точечная нотация»

Диаграммы классов языка UML (47)

- Результатом выражения вида

  ```
  объект>.<имя атрибута>
  ```

  является текущее значение атрибута с именем имя атрибута, если объект имеет такой атрибут
- Если атрибут типизирован именем некоторого класса, то результатом вызова операции является некоторый объект этого класса (объектный идентификатор), к которому также применимы операции над объектами.
- В противном случае использование подобного выражения приводит к возникновению ошибки типа

Диаграммы классов языка UML (48)

- Ограничения целостности. Язык OCL (17) Выражения (48)
- Результатом применения к объекту операции перехода по экземпляру связи-ассоциации является коллекция, содержащая все объекты, которые ассоциированы с данным объектом через указываемый экземпляр ассоциации
- Этот экземпляр ассоциации идентифицируется именем роли, противоположной по отношению к данному объекту
- Таким образом, синтаксис выражения перехода по соединению следующий:

```
<объект>.<имя роли, противоположенной
по отношению к объекту>
```

Диаграммы классов языка UML (49)

Ограничения целостности на языке OCL (18) Инварианты (9)

- **Операции над множествами, мультимножествами и последовательностями**
- В OCL поддерживается обширный набор операций над значениями коллекционных типов данных
- Обсудим только те из них, которые являются уместными в контексте данного раздела
- Синтаксически вызовы операций над коллекциями записываются в нотации, аналогичной точечной, но вместо точки используется стрелка (→)
- Общий синтаксис применения операции к коллекции выглядит следующим образом:

```
<коллекция> → <имя операции> (<список
фактических параметров>)
```

Диаграммы классов языка UML (50)

Операция select

- В OCL определены три одноименных операции select, которые обрабатывают заданное множество, мультимножество или последовательность на основе заданного логического выражения над элементами коллекции
- Результатом каждой операции является новое множество, мультимножество или последовательность, соответственно, из тех элементов входной коллекции, для которых результатом вычисления логического выражения является *true*

Диаграммы классов языка UML (51)

Ограничения целостности и язык OCL (20) Инвариант класса (11)

Операция collect

- В OCL определены три операции collect, параметрами которых являются множество, мультимножество или последовательность и некоторое выражение над элементами соответствующей коллекции
- Результатом является мультимножество для операций collect, определенных над множествами и мультимножествами, и последовательность для операции collect, определенной над последовательностью
- При этом результирующая коллекция соответствующего типа (коллекция значений или объектов) состоит из результатов применения выражения к каждому элементу входной коллекции

Диаграммы классов языка UML (52)

- Операция collect используется, главным образом, в тех случаях, когда от заданной коллекции объектов требуется перейти к некоторой другой коллекции объектов, которые ассоциированы с объектами исходной коллекции через некоторый экземпляр ассоциации
- В этом случае выражение над элементом исходной коллекции основывается на операции перехода по экземпляру ассоциации

Диаграммы классов языка UML (53)

- *Операции exists, forAll, count*

- В OCL определены три одноименных операции exists над множеством, мультимножеством и последовательностью; дополнительным параметром этих операций является логическое выражение
  - В результате каждой из этих операций выдается *true* в том и только в том случае, когда хотя бы для одного элемента входной коллекции значением логического выражения является *true*
  - В противном случае результатом операции является *false*
- Операции forAll отличаются от операций exist тем, что в результате каждой из них выдается *true* в том и только в том случае, когда для всех элементов входной коллекции результатом вычисления логического выражения является *true*
- В противном случае результатом операции будет *false*
  - Операция count применяется к коллекции и выдает число содержащихся в ней элементов

Диаграммы классов языка UML (54)

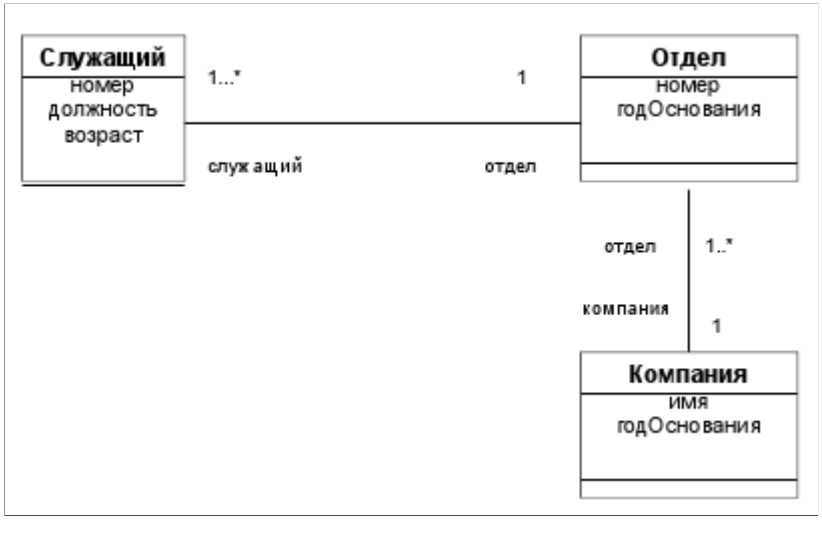
- *Операции union, intersect, symmetricDifference*
- Параметрами двуместных операций union (объединение), intersect (пересечение), symmetricDifference (симметричное вычитание) являются две коллекции, причем в OCL операции определены почти для всех возможных комбинаций типов коллекции
- Не будем рассматривать все определения этих операций и кратко упомянем только две из них
- Результатом операции union, определенной над множеством и мультимножеством, является мультимножество, т. е. из результата объединения таких двух коллекций дубликаты не исключаются
- Результатом же операции union, определенной над двумя множествами, является множество, т. е. в этом случае возможные дубликаты должны быть исключены

Диаграммы классов языка UML (55)

- **Примеры инвариантов**

Определить ограничение «возраст служащих должен быть больше 18 и меньше 100 лет»

```
context Служащий inv:
self.возраст >18 and
self.возраст < 100
```

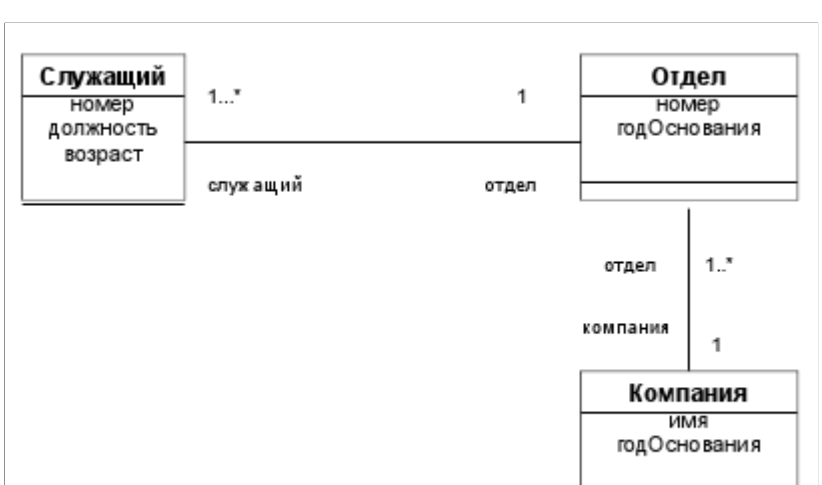


- Условие инварианта накладывает ограничение на значения атрибута возраст класса Служащий
- self обозначает текущий объект класса-контекста инварианта
- При проверке данного условия будут перебираться существующие объекты класса Служащий, и для каждого объекта будет проверяться, что значения атрибута возраст находятся в пределах заданного диапазона
- Ограничение удовлетворяется, если условное выражение принимает значение *true* для каждого объекта класса-контекста

Диаграммы классов языка UML (56)

Ограничения целостности и язык OCL

Выразить на языке OCL ограничение, в соответствии с которым в отделах с номерами больше 5 должны работать сотрудники старше 30 лет



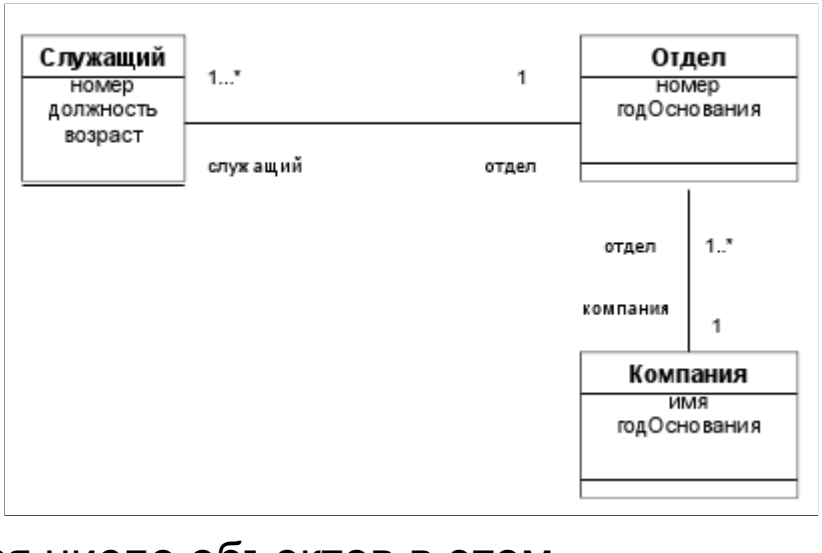

```
context Отдел inv:
self.номер > 5 or
self.служащий → select (возраст
≤ 30) → count () = 0
```

- Условное выражение инварианта будет вычисляться для каждого объекта класса Отдел
- Подвыражение справа от операции or вычисляется слева направо
- Сначала вычисляется подвыражение self.служащий, значением которого является множество объектов, соответствующих служащим, которые работают в текущем отделе
- Далее к этому множеству применяется операция select (возраст > 30), в результате которой вырабатывается множество объектов, соответствующих служащим текущего отдела, возраст которых превышает 30 лет

Диаграммы классов языка UML (57)

Ограничения целостности и язык OCL

Выразить на языке OCL ограничение, в соответствии с которым в отделах с номерами больше 5 должны работать сотрудники старше 30 лет



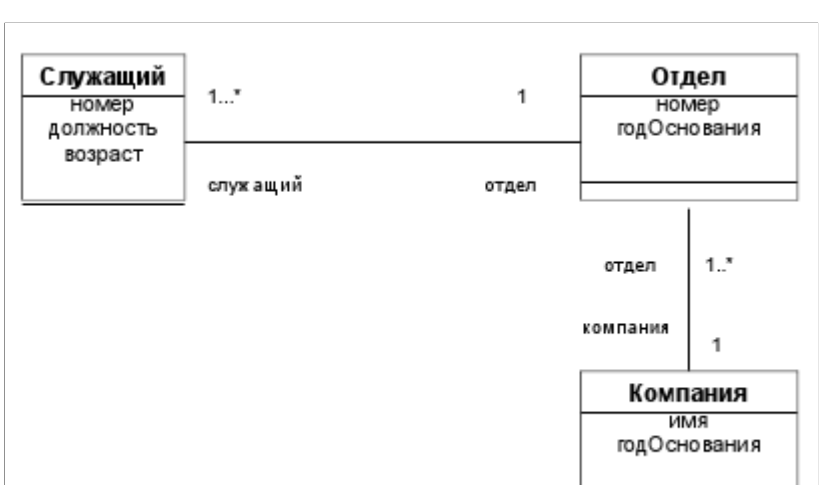

```
context Отдел inv:
self.номер ≤ 5 or
self.служащий → select (возраст
≤ 30) → count () = 0
```

- Значением операции count () является число объектов в этом множестве
- Все выражение принимает значение *true*, если последняя операция сравнения «=0» вырабатывает значение *true*, т. е. если в текущем отделе нет сотрудников младше 31 года
- Ограничение в целом удовлетворяется только в том случае, если значением условия инварианта является *true* для каждого отдела

Диаграммы классов языка UML (58)

- Тот же инвариант можно сформулировать в контексте класса Служащий:

```
context Служащий inv:
self.возраст > 30 or
self.отдел.номер ≤ 5
```



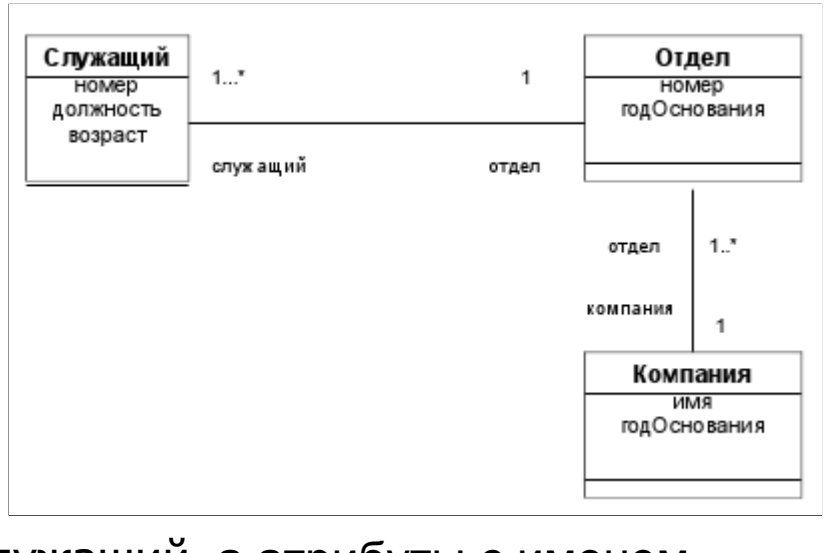
- Здесь следует обратить внимание на подвыражение self.отдел.номер ≤ 5
- Поскольку отдел – это имя роли ассоциации, значением подвыражения self.отдел является коллекция (множество)
- Но кратность роли отдел равна единице, т. е. каждому объекту служащего соответствует в точности один объект отдела
- Поэтому в OCL допускается сокращенная запись операции self.отдел.номер, значением которой является номер отдела текущего служащего

Диаграммы классов языка UML (59)

- Определить ограничение, что у каждого отдела должен иметься менеджер, и любой отдел должен быть основан не раньше соответствующей компании

Ограничения целостности на языке OCL

```
context Отдел inv:
self.служащий → exists (должность
= “manager”) and
self.компания.годОснования ≥
self.годОснования
```



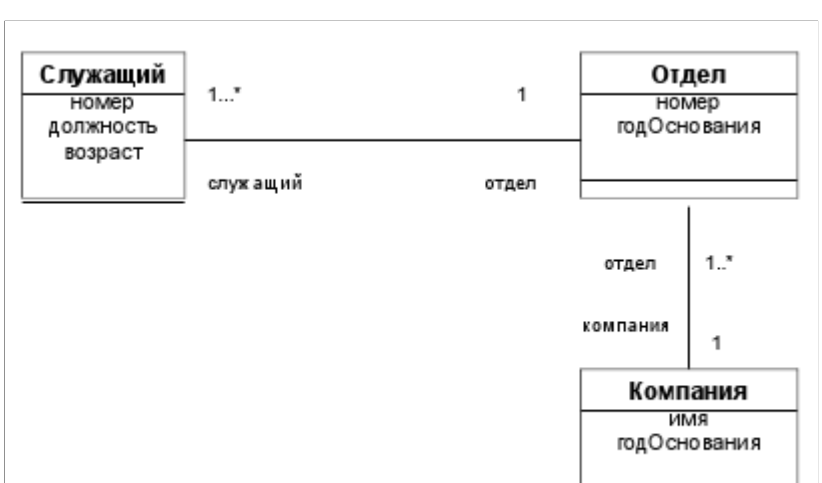
- Здесь должность – атрибут класса Служащий, а атрибуты с именем годОснования имеются и у класса Отдел, и у класса Компания
- В условном выражении этого инварианта подвыражение self.служащий → exists (должность = “manager”) эквивалентно выражению self.служащий → select (должность = “manager”) → count () > 1

Диаграммы классов языка UML (60)

Ограничения целостности на языке OCL

- Определить ограничение, что у каждого отдела должен иметься менеджер, и любой отдел должен быть основан не раньше соответствующей компании

```
context Отдел inv:
self.служащий → exists (должность
= “manager”) and
self.компания.годОснования ≥
self.годОснования
```

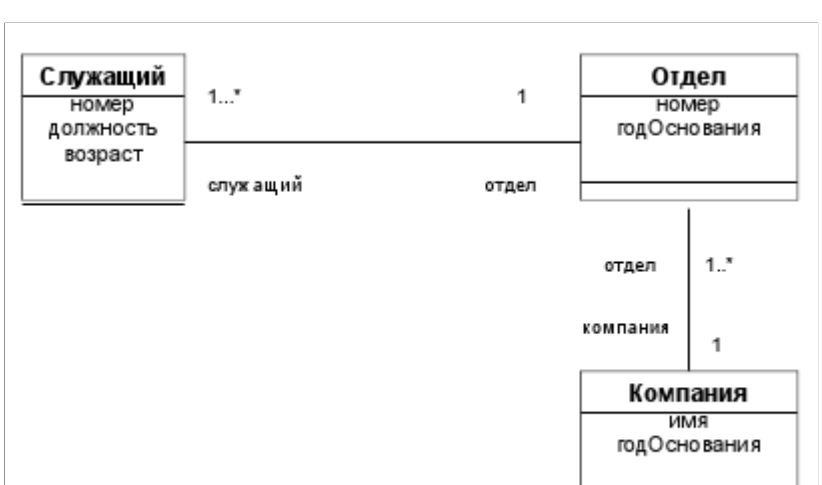


- Если бы в ограничении мы потребовали, чтобы у каждого отдела был только один менеджер, то следовало бы написать ... count () = 1, и это было бы не эквивалентно варианту с exists
- Обратите внимание, что в этом случае снова законным является подвыражение self.компания.годОснования, поскольку кратность роли компания в ассоциации классов Отдел и Компания равна единице

Диаграммы классов языка UML (61)

Ограничения целостности и язык OCL

- Условие четвертого инварианта ограничивает максимально возможное количество сотрудников компании числом 1000




```
context Компания inv:
self.отдел → collect (служащие) →
count () < 1000
```

- Здесь полезно обратить внимание на использование операции collect
- Проследим за вычислением условного выражения
- В нашем случае в классе Компания всего один объект, и он сразу становится текущим
- В результате выполнения операции self.отдел будет получено множество объектов, соответствующих всем отделам компании
- При выполнении операции collect (служащие) для каждого объекта-отдела по экземпляру ассоциации с объектами класса Служащие будет образовано множество объектов-служащих данного отдела, а в результате будет образовано множество объектов, соответствующих всем служащим всех отделов компании, т.е. всем служащим компании

Диаграммы классов языка UML (62)

- Плюсы и минусы использования языка OCL при проектировании реляционных баз данных
- Язык OCL позволяет формально и однозначно (без двусмысленностей, свойственных естественным языкам) определять ограничения целостности БД в терминах ее концептуальной схемы
- Скорее всего, наличие подобной проектной документации будет полезным для сопровождения БД, даже если придется преобразовывать инварианты OCL в ограничения целостности SQL вручную

Диаграммы классов языка UML (63)

Ограничения целостности и язык OCL (32) Инвариант класса (23)

- К отрицательным сторонам использования OCL относится, прежде всего, сложность языка и неочевидность некоторых его конструкций
- Кроме того, строгость синтаксиса и линейная форма языка в некотором роде противоречат наглядности и интуитивной ясности диаграммной части UML
- Да, в инвариантах OCL используются те же понятия и имена, что и в соответствующей диаграмме классов, но используются совсем в другой манере

Диаграммы классов языка UML (64)

Ограничения целостности и язык OCL (23) Инвариант класса (24)

- Трудно доказать или опровергнуть как предположение, что на языке OCL можно выразить любое ограничение целостности, которое можно определить средствами SQL, так и утверждение, что на языке OCL нельзя выразить такой инвариант, для которого окажется невозможным сформулировать эквивалентное ограничение целостности на языке SQL
- Неизвестны работы, в которых бы сравнивалась выразительная мощность этих языков в связи с ограничениями целостности реляционных БД

Диаграммы классов языка UML (65)

Получение схем реляционной базы данных из диаграмм классов UML

(1)

- Если не обращать внимания на различия в терминологии, то здесь выполняются практически те же шаги, что и в случае преобразования в схему реляционной БД ER-диаграммы
- Поэтому ограничимся только некоторыми рекомендациями, специфичными для диаграмм классов.
- Рекомендация 1. Прежде чем определять в классах операции, подумайте, что вы будете делать с этими определениями в среде целевой РСУБД
  - Если в этой среде поддерживаются хранимые процедуры, то, возможно, некоторые операции могут быть реализованы именно с помощью такого механизма
  - Но если в среде РСУБД поддерживается механизм определяемых пользователями функций, возможно, он окажется более подходящим

Диаграммы классов языка UML (66)

Получение схемы реляционной базы данных из диаграммы классов UML (2)

- Рекомендация 2. Помните, что сравнительно эффективно в РСУБД реализуются только ассоциации видов «один ко многим» и «многие ко многим»
  - Если в созданной диаграмме классов имеются ассоциации «один к одному», следует задуматься о целесообразности такого проектного решения
  - Реализация в среде РСУБД ассоциаций с точно заданными кратностями ролей возможна, но требует определения дополнительных триггеров, выполнение которых понизит эффективность

Диаграммы классов языка UML (67)

- Рекомендация 3. В спецификации UML говорится о том, что, определяя однонаправленные связи, вы можете способствовать эффективности доступа к некоторым объектам
  - Для технологии реляционных баз данных поддержка такого объявления вызовет дополнительные накладные расходы и тем самым снизит эффективность
- Рекомендация 5. Не злоупотребляйте возможностями OCL
- Диаграммы классов UML – это мощный инструмент для создания концептуальных схем баз данных, но, как известно, все хорошо в меру

Диаграммы классов языка UML (68)

Заключение (1)

- Нельзя сказать, что проектирование баз данных на основе семантических моделей в любом случае ускоряет и/или упрощает процесс проектирования
  - Все зависит от сложности предметной области, квалификации проектировщика и качества вспомогательных программных средств
  - Но так или иначе этап диаграммного моделирования обеспечивает следующие преимущества:
- На раннем этапе проектирования до привязки к конкретной РСУБД проектировщик может обнаружить и исправить логические недочеты проекта, руководствуясь наглядным графическим представлением концептуальной схемы

Диаграммы классов языка UML (69)

Заключение (2)

- Нельзя сказать, что проектирование баз данных на основе семантических моделей в любом случае ускоряет и/или упрощает процесс проектирования
- Все зависит от сложности предметной области, квалификации проектировщика и качества вспомогательных программных средств
- Но так или иначе этап диаграммного моделирования обеспечивает следующие преимущества:
  - На раннем этапе проектирования до привязки к конкретной РСУБД проектировщик может обнаружить и исправить логические недочеты проекта, руководствуясь наглядным графическим представлением концептуальной схемы

Диаграммы классов языка UML (70)

Заключение (3)

- Окончательный вид концептуальной схемы, полученной непосредственно перед переходом к формированию реляционной схемы, а может быть, и промежуточной версии концептуальной схемы, должен стать частью документации целевой реляционной БД
  - Наличие этой документации очень полезно для сопровождения и, в особенности, для изменения схемы БД в связи с изменившимися требованиями.
- При использовании CASE-средств концептуальное моделирование БД может стать частью всего процесса проектирования целевой информационной системы, что должно способствовать правильной структуризации процесса, эффективности и повышению качества проекта в целом

Диаграммы классов языка UML (71)

Заключение (4)

- В контексте проектирования реляционных БД структурные методы проектирования, основанные на использовании ER-диаграмм, и объектно-ориентированные методы, основанные на использовании языка UML, различаются, главным образом, лишь терминологией
- ER-модель концептуально проще UML, в ней меньше понятий, терминов, вариантов применения
- И это понятно, поскольку разные варианты ER-моделей разрабатывались именно для поддержки проектирования реляционных БД, и ER-модели почти не содержат возможностей, выходящих за пределы реальных потребностей проектировщика реляционной БД

Диаграммы классов языка UML (72)

Заключение (3)

- Язык UML принадлежит объектному миру
- Этот мир гораздо сложнее (если угодно, непонятнее, запутаннее) реляционного мира
- Поскольку UML может использоваться для унифицированного объектно-ориентированного моделирования всего чего угодно, в этом языке содержится масса различных понятий, терминов и вариантов использования, избыточных с точки зрения проектирования реляционных БД
- Если вычленить из общего механизма диаграмм классов то, что действительно требуется для проектирования реляционных БД, то мы получим в точности ER-диаграммы с другой нотацией и терминологией

Диаграммы классов языка UML (73)

Заключение (6)

- Поэтому выбор конкретной концептуальной модели – это вопрос вкуса и сложившихся обстоятельств
- Понятно, что если в организации уже имеется сложившаяся инфраструктура проектирования приложений, то разумно продолжать ею пользоваться до тех пор, пока это не станет тормозом
- При построении же новой инфраструктуры стратегические соображения высшего руководства компании имеют больший вес, чем предпочтения технических специалистов, хотя эти предпочтения тоже обязательно должны учитываться