Бази от данни

Обектно-ориентиран модел на данни (ODL)

Обзор на обектно-ориентираните концепции

- Моделът „Същност-връзки“ и релационния модел са само два от моделите посредством, които може да се моделират данни в базите от данни.
- Друг такъв модел, който същевременно е и от значение за съвременните СУБД е обектно-ориентирания модел на данни.
- Ще представим чистия обектно-ориентиран модел на данни наречен ODL (Object Definition Language).
- Освен него е известен и още един обектно-ориентиран модел, а имено обектно-релационния модел на данни.
- Обектно-релационния модел на данни е част от SQL стандарта и разширява релационния модел, като включва някои по-общи обектно-ориентирани концепции.
- Повечето от съвременните СУБД се базират имено на този разширен релационен модел.

Обзор на обектно-ориентираните концепции

- Обектно-ориентираното програмиране (ООП) набира широка популярност с развитието на езиците за програмиране C++ и Java и най-вече с миграцията на наследения софтуер на повечето организации именно към тези два езика.
- Основната парадигма на ООП е:
  - Мощна система за типизация – ясно разграничени типове.
  - Класове – които за разлика от конвенционалните типове могат да включват и методи (функции приложими към обектите, принадлежащи на класа)
  - Идентификация на обектите – всеки обект има уникална идентификация, независимо от неговата стойност.
  - Наследяване - организирането на класовете в йерархия от класове, където всеки клас от йерархията наследява свойствата на предхождащите го.

Система за типизация

- ООП предлага на потребителя богата съвкупност от типове, като се започне от елементарните типове като integer, real, boolean и т.н.
- Също така ООП предлага и средства за създаване на нови типове – конструктори.
- Конструкторите са приложими както към простите типове, така и към новосъздадените типове следствие на прилагане на конструктора
- Обикновено се поддържат следните конструктори:
  - Конструктор за структури от записи
  - Конструктор за тип набор (колекции)
  - Конструктор за тип референция

Конструктор за структури от записи

- Ако T1, T2, …, Tn са списък от типове, а f1, f2, …fn е съответстващият им списък от имена на полета, то можем да конструираме тип запис състоящ се от n-компонента, на който i-тият компонент е от тип Ti с име на поле fi.
- Структурите от записи са добре познатите ни структури в езика C++

```
struct struct1 {
    T1 f1;
    T2 f2;
} s;
```

Конструктор за тип набор (колекции)

- Нека T е тип. По дадения тип T се конструира нов тип, който е набор от този тип.
- Различните езици за програмиране използват различни набори, но най-често срещаните набори са масиви, списъци и множества.
- Например ако T е тип integer (цяло число), то чрез конструктора за тип набор може да създадем масив от цели числа.

- Например: integer A[100];

Конструктор за тип референция

- Нека T е тип. По дадения тип T се конструира нов тип, чиито стойности са указатели (референции) към стойности на типа T.
- В езика C++, референцията е указател към стойност. Самият указател е адрес от виртуалната памет на стойността към която сочи.
- Конструкторите разгледани до сега, могат да се прилагат последователно и така да бъдат получени по-сложни типове. Например масив, чиито елементи са структури. Или структура чиито полета са масиви и т.н.

Класове и обекти

- Един клас се състои от тип и една или повече функции, наречени методи, които могат да се изпълняват върху обектите от този клас.
- Самите обекти са инстанции на класа. Обектите могат да бъдат два вида – неизменяеми и изменяеми.
- Неизменяемите обекти са конкретни стойности от този клас – например $\{1, 2, 5\}$ е неизменяем обект от тип множество от цели числа.
- Изменяемите обекти са променливи от тип този клас и тяхното съдържание може да бъде променяно.

Идентификация на обектите

- Всеки обект има идентификатор OID (object identifier)
- Не е възможно два различни обекта да имат едно и също OID, нито пък един обект да има две различни OID.
- За разлика от модела „Същност-връзки“, където всяка същност трябва да има ключ, който е уникален - при ОО модел може да има два различни обекта с еднакво съдържание и те ще бъдат напълно различни обекти (ще се различават по OID)

Методи

- Към всеки клас има асоциирани функции, наречени методи.
- Всеки метод има поне един аргумент, който е обект от класа.
- Методът може да има и други аргументи – обекти от други класове или от същия клас
- Например, клас множество от цели числа може да съдържа методи за сумиране на елементите на множеството, за обединението на две множества или да върнем boolean стойност в зависимост от това дали множеството е празно или не, както и други методи.

Класове - абстракция

- В някои ситуации, класовете се разглеждат като абстрактни типове от данни.
- В смисъл че, те капсулират или ограничават достъпа до обектите на класа, така че само методите дефинирани за класа да могат да модифицират обектите на този клас директно.
- Капсулацията ни гарантира, че обектите на класа могат да бъдат изменяни само по начин, предвиден от създателя на класа.
- Поради тези причини, капсулацията е разглеждана като ключово средство към създаването на надежден софтуер.

Йерархии от класове

- Възможно е да декларираме един клас като подклас на друг.
- Например, може да дефинираме един клас C да бъде подклас на друг клас D. Тогава казваме че C наследява D (свойствата на класа D, включително и типа на D и всички функции (методи) дефинирани в D).
- Класът C може да има и допълнителни свойства. Например нови методи на мястото на методи от класа D или нови методи в добавка на тези от класа D.
- Например ако типа на D е тип структура, може да добавим нови полета към този тип. Така само обектите от класа C ще имат такива полета.

Пример за клас

- Клас Account

```
class Account = {
  accountNo :  integer;
  balance : real;
  owner : REF Customer;
}
```

- Типът на Account е структура от три полета – номер на сметка (цяло число), баланс по сметката (реално число) и собственик на сметката, който е референция (указател) към обект от тип Customer (в случая не е дефиниран)
- Може да дефинираме методи за класа Account, например:
  - deposit(a :Account, m : real) – метод за внасяне на пари по сметка
  - withdraw(a :Account, m : real) – метод за теглене на пари по сметка
- Също така можем да дефинираме подклас на Account – например клас TimeDeposite, който ще има допълнително поле dueDate – дата на падеж на депозита.

Въведение в ODL

- ODL е стандартизиран език за моделиране на данните на една база от данни в термините на ОО концепция.
- Той е разширение на IDL (Interface Description Language), който е част от CORBA (архитектура за разпределени изчисления).
- При ООП светът е съставен от обекти
- Погледнато концептуално обектите са аналогични на същностите от модела „Същност – връзки“, въпреки че с тях могат да се асоциират и методи
- Обектите трябва да притежават OID, за да могат да бъдат идентифицирани уникално
- При организацията на информацията за базата от данни обектите с подобни свойства се групират в класове

Въведение в ODL

- В контекста на ODL групирането на обектите се извършва по следните два критерия:
  - Концепциите от реалния свят, които се представят чрез обектите трябва да са подобни – например, клиентите на дадена банка могат да бъдат групирани в един клас.
  - Свойствата на обектите трябва да са едни и същи. Много често обектите се разглеждат като записи, съставени от полета. Във всяко поле се записва стойност или референция към друг обект
- В ODL свойствата на обектите са три вида:
  - Атрибути – те представят стойности, свързани с обекта;
  - Връзки – те свързват обекта с един или няколко обекта от други класове
  - Методи – това са функции, които могат да се изпълняват върху обекта.

Декларация на клас в ODL

- Декларирането на клас в ODL, става със следния синтаксис:

```
class име_на_клас{
   списък_от_свойства
};
```

- Свойствата са атрибути, връзки методи, записани в произволен ред.
- За разделител в списъка се използва ;

Атрибути в ODL

- Най-простите свойства са атрибутите. Те описват някакъв аспект на обекта, като асоциират с него стойности от фиксиран тип.

- За разлика от модела „Същност-връзки“ , не е задължително стойностите на атрибутите да са атомарни.

- Например клас, който описва хора може да съдържа атрибута дата на раждане, който е от тип структура с три полета – ден, месец година.

Пример за клас със свойства атрибути

```
class Movie{
  attribute string title;
  attribute integer year;
  attribute integer length;
  attribute enum Film {color, black} type;
};

class Star{
  attribute string name;
  attribute struct Addr{string street,
                        string city} address;
};
```

Обяснение на примера

- Чрез ключовата дума enum в ODL се дефинира изброен тип – задават се явно допустимите стойности в този тип.

- Чрез ключовата дума struct в ODL се дефинира структура.

- Изброеният тип в класа Movie и структурата в класа Star имат имена, въпреки че това изглежда излишно. Това позволява тези типове да се използват извън класа чрез операцията :: за разрешаване на достъп.

Връзки в ODL

- Да предположим, че към класа Movies искаме да добавим свойство, което да отразява връзката между Movie и Star, т.е. множество от актьорите които участват в конкретния филм
- Това става като в декларацията на класа Movie добавим следния ред:

```
relationship set<Star> stars_in;
```

- Така всеки обект от класа Movies ще има референция към множество от обекти – актьори. Това множество се казва stars_in, а самата дума relationship специфицира че това е референция към множество от обекти от класа Stars

- В ODL множество от обекти се дефинира с ключовата дума `set<име_на_клас>`

Инверсни връзки

- Както по даден филм разбираме, кои са актьорите които участват в съответния филм, така може да поискаме по даден актьор да разберем филмите, в които е участвал актьора.

- Аналогично на Movie, трябва да добавим следния ред в декларацията на класа Star

```
relationship set<Movie> stared_in;
```

- Този ред и подобният му в класа Movie, показват един важен аспект от връзката между двата класа Movie и Stars

Инверсни връзки

- Логично е да се очаква, че ако един актьор е в множеството stars_in за класа Movies, то филмите в които е участвал този актьор ще бъдат в множеството stared_in за класа Star

- Такива връзки се наричат инверсни и се обозначават с ключовата дума inverse

- Тази ключова дума се поставя в декларацията и на двете връзки и в тази на Movie и в тази на Star, като изрично се упоменава и името на връзката от другия клас.

```
relationship set<Movie> stared_in
inverse Movie :: stars_in;
```

Пример за клас с връзки

```
class Movie{
  attribute string title;
  attribute integer year;
  attribute integer length;
  attribute enum Film {color, black} type;
  relationship set<Star> stars_in inverse Star :: stared_in;
};

class Star{
  attribute string name;
  attribute struct Addr{string street, string city} address;
  relationship set<Movie> stared_in
                    inverse Movie :: stars_in;

};
```

Множественост на връзките

- В разгледаните до момента модели връзките могат да бъдат :
  - Много – много
  - Много – един
  - Един – един
- Езикът ODL не е изключение. Нека C и D са два класа
  - Казваме че, имаме връзка много-много между класовете C и D, ако в класа C типа на връзката е Set<D>, а в класа D типа на връзката е Set<C>;
  - Казваме че, имаме връзка много-един между класовете C и D, ако в класа C типа на връзката е само D, а в класа D типа на връзката е Set<C>
  - Казваме че, имаме връзка един-един между класовете C и D, ако в класа C типа на връзката е само D, и в класа D типа на връзката е само C

Методи в ODL

- Третият вид свойства на ODL класовете са методите
- Подобно на другите ОО езици, метода е парче изпълним код, който може да бъде приложен към обектите на класа.
- В ODL може да декларираме имената на методите свързани с този клас и входни/изходните параметри на метода. Тази декларация се нарича сигнатура (името с входно/изходните параметри)
- Самият код на метода ще бъде написан на host-език (C++, Java) и този код не е част от езика ODL.
- Самата декларация на метода се записва при декларацията на другите свойства на класа (атрибутите и връзките).
- Както е нормално за ОО езици, всеки метод е свързан с клас и методите се извикват от обект на класа. Ето защо обекта е скрит аргумент за метода

Методи в ODL

- Синтаксисът на декларацията на метода е подобна на тази при ОО езици за програмирани с две важни допълнения:
  - Параметрите на метода се специфицират да са in, out или inout. т.е. специфицира се дали параметъра е само входен, само изходен или входен и изходен. Последните два типа out и inout, могат да бъдат променяни от метода, докато първият тип in са само за четене и немогат да бъдат променяни. В резултат на това out и inout параметрите се предават като референции, докато in параметрите като стойности. Методът може да връща и стойност, което е и друг начин за връщане на резултат.

  - Методите могат да предизвикват изключения (exceptions). Това е специален механизъм на действие, който се задейства когато се случи грешка (не обичайна ситуация) при изпълнението на метода. Самите изключения ще бъдат прихванати и обработени от метод, който извиква дефинирания от нас метод. Пример за изключение е делението на 0. В ODL ако метода хвърля изключение това се указва с ключовата дума raises.

Методи в ODL - Пример

```
class Movie {
  attribute string title;
  attribute integer year;
  attribute integer length;
  attribute enumeration (color, black) type;
  relationship set<Star> stars_in
                    inverse Star :: stared_in;
 float lengthInHours () raises (noLengthFound);
};
```

Типовете в ODL

- ODL като обектно-ориентиран предлага мощна система за типизация.

- Системата за типизация включва базови типове и рекурсивни правила, с които могат да бъдат изградени по-сложни типове от базовите.

- Типовете в ODL могат да бъдат:
  - Атомарни типове: integer, float, character, string, boolean и enumeration
  - Съставни типове: конструират се от атомарните типове посредством конструктори

Съставни типове

- Съставни типове: конструират се от атомарните типове посредством следните конструктори:
  - Множество (Set). Ако T е тип, тогава Set<T> е множество с елементи от тип T;
  - Контейнер (Bag). Ако T е тип, тогава Bag<T> е мулт-множество с елементи от тип T. Мулти-множествата позволяват елементите да се повтарят.
  - Списъци (List). Ако T е тип, тогава List<T> е краен списък от нула или повече елементи от тип T;
  - Масив (Array). Ако T е тип, а i е цяло число, тогава Array<T,i> е масив от i на брой елементи от тип T;
  - Речник (Dictionary). Ако T и S са типове, тогава Dictionary<T, S> е множество от наредени двойки, такива че първият елементи на двойката е от тип T, а вторият елемент от двойката е от тип S; Смисълът на такава двойка от елементи е че по дадена стойност (ключ) от тип T се съпоставя стойност от тип S.
  - Структури (Stuct). Ако T1, T2, .., Tn са типове, а F1, F2, …, Fn са имена на полета, тогава Struct N {T1 F1, …., Tn Fn} е структура с име N от елементи, където i-тото поле е от тип Ti и е с име Fi
- .

Съставни типове

- Първите пет типа – set, bag, list, array и dictionary се наричат collection типове.

- Има ясно дефинирани правила, кои типове могат да се използват като типове на атрибути и кои като типове на връзките в един клас:
  - Типът на свойството връзка в един клас може да бъде или тип клас или тип collection приложен към клас.
  - Типът на свойството атрибут в един клас може да бъде или атомарен тип или тип конструиран, чрез конструктор от горе-изброените.

Пример

- Възможни типове на атрибути са:
  - Integer
  - Struct N{string f1, integer f2}
  - List <ral>
  - Array <Struct N{string f1, integer f2}, 10}
- Неправилни типове за връзки са:
  - Struct N{Movies f1, Stars f2} – Типовете за връзки не могат да включват структури
  - integer - типовете за връзки не могат да включват атомарни типове
  - Set <Array<Stars, 10>> - типовете за връзки не могат да включват влагания на колекции от типове

Предимства на ODL

- Има много предимства на ODL, чрез които можем да изразим нещата, които с модела „Същност-връзки“ не можем.
- В ODL се поддържат само бинарни връзки. За да представим е бинарна връзка в ОДЛ трябва да разбием тази връзка на бинарни връзки, аналогично на модела „Същност-връзки“
- Например, нека R е небинарна връзка между класовете C1, C2, …., Cn. За да представим R образуваме нов клас C и връзки много към един от C към Ci. Всеки обект t от класа C е конкретна връзка между обектите от C1, C2, .., Cn с които е свързан t

Пример

- Като пример ще представим връзката Contracts между филмите, звездите и студията. За целта създаваме клас Contracts със следната декларация:

```
class Contract{
  attribute integer salary;
  relationship Movie m inverse Movie :: contractsFor;
  relationship Star s inverse Star :: contractsFor;
  relationship Studio st inverse Studio :: contractsFor;
};
```

- Използваме един атрибут salary – това е атрибутът на връзката
- В класовете Movies, Stars и Studio трябва да добавим следните декларации:
  - relationship Set <Contracts> contractsFor inverse Contracts :: m;
  - relationship Set <Contracts> contractsFor inverse Contracts :: s;
  - relationship Set <Contracts> contractsFor inverse Contracts :: st;

Наследяване и подкласове в ODL

- Подобно на модела „Същност - връзки“, където могат да се строят isa-йерархии, в ODL се допуска един клас C да наследи друг клас D. За целта в декларацията на класа C след неговото име се записва ключовата дума extends, последвана от името на класа D
- Например Cartoons са филми, които са анимации. Те реално наследяват Movie като включват допълнителна връзка към множеството от актьори, които озвучават филма.
- Аналогично Murder са филми, в които има убийства. Те също наследяват Movie като включват допълнителен атрибут за оръжието което се използва във филма.

Наследяване и подкласове в ODL - Пример

```
class Cartoons extends Movie{
        relationship Set<Star> voices inverse Star ::
voiceOf;
};

class Murders extends Movie{
        attribute string weapon;
};
```

- Естествено, в класа Star трябва да добавим следната декларация:

```
relationship Set<Cartoon>
voiceOf inverse Cartoon :: voices;
```

- Всички свойства на суперкласа се наследяват от подкласа – неговите атрибути, връзки и методи.

Множествено наследяване в ODL

- Понякога се налага един клас да наследи два или повече класа.
- Например, може да има филми, които са едновременно анимационни и в които да има убийства.
- В модела „Същност-връзки“, тези филми ще присъстват едновременно и в релацията Cartoons и в релацията Murders
- При обектно-ориентирания подход обаче, основен принцип е всеки обект да принадлежи на точно определен клас, т.е. няма как един обект да принадлежи и на класа Cartoons и на класа Murders;
- За да се разреши тази ситуация, всъщност ни трябва нов клас, който да наследява и Cartoons и Murders. Така получаваме множествено наследяване.

Множествено наследяване в ODL

- При множественото наследяван в ODL, ключовата дума extends е последвана от имената на наследствените класове, разделени с :
- За примера с анимациите, в които има и убийства декларацията за класа е следната:

```
class CartoonMurder extends Cartoon : Murder;
```

- Така дефинираният клас няма собствени свойства, а само наследени.

Множествено наследяване в ODL

- Когато един клас C наследява няколко класа, възниква опасност от дублиране на имената на свойствата
- Два или повече от суперкласовете на C може да имат свойства с еднакви имена.
- Например да предположим, че класът Movie има подкласове Romance и Court. Нека класът Romance има атрибут ending и класът Court също да има такъв атрибут.
- Ако създадем нов подклас RomanceCourt, които наследява двата класа, то атрибута ending които се дублира в двата супер класа не е ясно как точно да се интерпретира в новия подклас.

Множествено наследяване в ODL

- Решението на подобен конфликт не е част от стандарта ODL

- Някои от подходите за решаване на проблема са следните:
  - Забранява се множественото наследяване – това е твърде ограничаващ подход
  - Изрично се указва, коя от декларациите на дублираните свойства да се използва
  - Преименува се някое от дублираните свойства – например в класа Court може да преименуваме ending на verdict

Разширения на класове

- Множеството от обектите на даден клас се нарича разширение на класа
- Подобно на релационния модел , където се прави разлика между схемата на една релация и нейният екземпляр, в ODL се прави разлика между декларацията на един клас и неговото разширение
- Името на разширението на един клас са задава в скоби след името на класа, предшествано от ключовата дума extent
- Общоприетата конвенция е имената на класа и на неговото разширение да са едни и същи, но името на класа да е в единствено число, а името на разширението да е в множествено число

Разширения на класове - Пример

- Например, за класа Movie може да се зададе име на разширението по следния начин:

```
Class Movie (extent Movies){
 attribute string title;
……
};
```

- Заявките към една база от данни, описана в ODL се извършват посредством имената на разширенията на класовете, а не посредством техните собствени имена

Разширения на класове

- Клас без зададено разширение наричаме интерфейс.
- С такъв клас не се асоциират обекти
- Както вече споменахме, интерфейсите се използват при организиране на множествено наследяване в ODL
- Те са полезни когато няколко класа трябва да имат различни разширения, но един и същи свойства.
- В такъв случай е достатъчно да се създаде един интерфейс I и всеки от класовете да наследи I.

Ключове в ODL

- За разлика от модела „Същност-връзки“ и релационния модел, в ODL ключовете не са задължителни.
- Причината е, че обектите се идентифицират уникално посредством своите OID
- Напълно е възможно в ODL два обекта от един и същи клас да притежават еднакви свойства – системата ще ги различава по техните OID
- Въпреки това, в ODL могат да се дефинират ключове. Един или повече от атрибутите на даден клас могат да се декларират като ключ на класа с помощта на ключовата дума key или keys, последвана от заградени в скоби списък от имената на атрибутите, разделени със запетаи.

Ключове в ODL - Пример

```
class Movie (extent Movies key(title, year))
{ attribute string title;
…
};
class Star(extent Stars key name)
{ attribute string name;
……
};
```

- Възможно е да се зададе повече от един ключ за даден клас

```
class Employee(extent Employees key empID,
ssNo)
{ attribute string name;
……
};
```

Ключове в ODL

- В последния пример, ODL интерпретира двата атрибута като ключове по отделно, ако искаме да ги приема като един трябва да сложим скоби

- ODL стандарта позволява свойства, различни от атрибутите да се появяват в ключа. Не е забранено метод или връзка да дефинира ключ.

- Например може да декларираме метод за ключ, като имаме предвид , че за различните обекти на класа метода ще върне различни стойности

Преобразуване на ODL към релационен модел

- Ако моделът „Същност-връзки“ се използва за да направим дизайн на една база от данни и след това да го преобразуваме към релационен модел, то ODL езика е насочен повече към реализацията на една база от данни от колкото към нейния дизайн.

- Въпреки това ODL, като всеки един обектно-ориентиран език може да бъде използван за дизайн на базата от данни и едва след това този модел да бъде сведен до релационен, но при това преобразуване са възможни възникването на проблеми.

Проблеми при преобразуване от ODL към релационен

- Проблемите, който възникват при преобразуването от ODL към релационен модел са:
  - Множествата от същности трябва да имат ключ, но за ODL това не е задължително (защото всеки обект си има OID). Затова в някои ситуации трябва да въведем нов атрибут, който да служи като ключ на преобразуваната релация.
  - Докато при E/R модела, атрибутите са атомарни, както и при релацията, то за ODL няма такова ограничение. Преобразуването на атрибути, които са от тип колекция в релацията не е тривиално и често води до не-нормализирана база от данни.
  - В ODL, може да специфицираме методи при дизайна на базата от данни. Няма прост начин на преобразуване на метод към релационна схема и това е и третия проблем, който може да възникне при преобразуване на ODL към релационен модел. За сега, ще разглеждаме преобразуване само на такива ODL дизайни на класове, които не съдържат методи.

Преобразуване на ODL атрибути

- Преобразуването става по следния начин:
  - Съпоставяме релация на всеки клас, а за всяко свойство на класа съпоставяме атрибут на релацията. В сила са следните ограничения:
    - Всички свойства на класа са атрибути (без методи и връзки)
    - Всички свойства на атрибутите са атомарни
- Например:

```
class Movie (extent Movies){
  attribute string title;
  attribute integer year;
  attribute integer length;
  attribute enum film{color, black} filmType;
};
```

Преобразуване на ODL атрибути

- За горе-описания клас, дефинираме релация със същото име, като името на класа – Movies
- Релацията ще има 4 атрибута, по един за всеки от атрибутите на класа.
- Имената на атрибутите на релацията могат да бъдат същите като имената на атрибутите на класа.

```
Movies(title, year, length, filmType)
```

- За всеки обект от разширението Movies има един кортеж в релацията Movies. Кортежите имат компоненти за всеки един от четирите атрибутите и стойност на всеки компонент съответстваща на стойностите на атрибутите на обектите от разширението на класа.

Преобразуване на неатомарни ODL атрибути

- За съжаление, дори и всички свойства на класа да са атрибути, може да имаме проблеми при преобразуването от ODL Към релационен модел

- Причината е, че атрибутите в ODL могат да бъдат съставни – като структури, множества, мултимножества, списъци и др.

- От друга страна основен принцип в релационния модел е че атрибутите в една релация трябва да бъдат атомарни (като числа, символ, низове и т.н.)

Преобразуване на ODL атрибути - структура

- Най-лесно от съставните атрибути в ODL се преобразува структурата от записи, чиито полета са атомарни

- Преобразуването на структура става, като разширяваме дефиницията на релацията, като добавяме по един атрибут в релацията за всеки атрибут на структурата

- Ако се случи да има две структури, чиито атрибути да имат едни и същи имена, преименуваме за да избегнем конфликта с имената

Преобразуване на ODL атрибути - структура

- Например

```
class  Star (extent Stars){
  attribute string name;
  attribute struct Addr { string street,
string city} address;
};
```

Се преобразува до:

```
Stars(name, street, city);
```

Преобразуване на ODL множествени атрибути

- Освен структура, съставните атрибути на един клас могат да бъдат: Set, Bag, List, Array и Dictionary
- Един подход за представянето на множеството от стойности за един атрибут A е да се направи по един кортеж за всяка стойност. Този кортеж ще съдържа подходящи стойности за всички атрибути на релацията и стойностите от множеството
- Например:

```
class Star(extent Stars){
  attribute String name;
  attribute set <Struct Addr {string street,
string city}> add;
};
```

Преобразуване на ODL множествени атрибути

- Да предположим, че за класа Star е възможно един актьор да има повече от един адрес

```
class Star(extent Stars){
  attribute String name;
  attribute set <Struct Addr {string street, string city}>
add;
};
```

- Тогава релацията, ще изглежда по следния начин

```
Stars(name, street, city)
```

- Ако един актьор има повече от един адрес, тогава в релацията ще бъдат добавени толкова кортежа, колкото адреса има съответния актьор
- Това води до повтаряне на името на актьора за всеки кортеж, което от своя страна за съжаление води до ненормализирана релация.

Преобразуване на ODL множествени атрибути

- Да предположим, че за добавим и атрибут birthdate Към дефиницията на класа Star
- Тогава релацията, ще изглежда по следния начин

```
Stars(name, street, city, birthdate)
```

- Ако един актьор има повече от един адрес, тогава в релацията ще бъдат добавени толкова кортежа, колкото адреса има съответния актьор
- Като името на актьора и рождената му дата ще се повтарят за всеки кортеж – което е излишество
- Ако пък актьор няма нито един адрес, тогава изобщо няма да имаме информация за актьора – което е аномалия при изтриване
- Въпреки че атрибута name е ключ за Stars, необходимостта да имаме няколко кортежа за един актьор, за да представим всичките му адреси, означава че name не е ключ за релацията Stars.
- Така ключът на релацията става {name, street, city}.
- Тогава ФЗ: name->birthdate, нарушава НФБК. Този факт обяснява и защо възникват горе-споменатите аномалии.

Преобразуване на ODL множествени атрибути

- Има няколко подхода как да преобразуваме стойностите на множество от един клас към релация.
- Първият подход е:
  - Поставяме всички атрибути, които са от множеството в схемата на релацията
  - Нормализираме (при преобразуването е възможно нарушаване на НФБК и на 4НФ)
- Вторият подход е:
  - Да разделим всяко множество от стойности на връзки между множеството и обектите на класа. Все едно имаме връзка много-много между обектите на класа и стойностите на множеството

Представяне на типове конструирани с конструктори

- Освен структури и множества в ODL, класа може да съдържа и атрибути от тип списък, масив или речник
- За да представим мултимножество, в което един обект се среща n-пъти в мултимножеството не можем просто да вмъкнем в релацията, n-еднакви кортежа (в релационния модел това не е позволено)
- Вместо това може да добавим към релационната схема друг атрибут – count, които ще показва колко пъти кортежа се среща в мултимножеството
- Списък от адреси може да се представи с нов атрибут – position, показваща позицията в списъка

Представяне на типове конструирани с конструктори

- Масив с фиксирана дължина от адреси може да се представи, чрез атрибутите за всяка позиция в масива. Например, ако масива е от два елемента слагаме два атрибута.

- Речникът може да се представи чрез множество, но с атрибути за ключовата стойност и за range стойността. Например, ако вместо адреса на актьор искаме да запазим за всеки актьор речник, който да ни дава титуляря на ипотеката за всяка от техните къщи, тогава речникът ще има като ключова стойност адрес, а като range стойности имената на банките

Представяне на типове конструирани с конструктори

- В един клас може да имаме атрибути, които да са от различен тип и създадени с различен тип конструктори.
- Също така може да имаме влагане на конструктори
- Ако типа е кои и да е от гореизброените приложен към структура (без речник), може да приложим горната техника, ако структурата е от атомарни стойности и тогава да заменим единичните атрибути, представяйки атомарните стойности, чрез няколко атрибута - по един за всяко поле на структурата.
- В общи линии изводите са, че ако използвате ODL за дизайн с цел преобразуване към релационен модел е добре да се спазват горните ограничения за съставност на атрибутите

Представяне на ODL връзки

- В дефинициите на ODL класовете, може да има и връзки.
- Подобно на E/R модела можем да създадем за всяка връзка нова релация, която свързва ключовете от двата свързани класа.
- В ODL обаче връзките са инверсни, по тази причина трябва да създадем една релация за всяка двойка
- Например:

```
class Movie (extent Movies key(title, year)){
…
Relationship Studio ownedby inverse Studio:owns;
}
class Studio(extent Studios key name){
…
Relationship Set <Movie> owns inverse Movie:ownedBy;
}
```

Представяне на ODL връзки

- Може да създадем релация за двойката връзки owns и ownedBy.
- Релацията трябва да има име, което в случая може да бъде произволно, например StudioOf като име на релацията.
- Схемата за StudioOf има атрибути – ключа на Movie и ключа на Studio, т.е. Title, year и studioname
- Когато релацията е много-един, отново може да приложим правилото за оптимизиране и да комбинираме релацията за връзката с релацията от към страната на „едно“

Какво се прави когато няма ключ?

- След като ключовете са опционални в ODL, можем да изпаднем в ситуация, че наличните ни алтернативи, не могат уникално да идентифицират обектите на класа C

- Това може да е проблем, ако C участва в една или повече връзки.

- В тези случаи е препоръчително да се добави нов атрибут, който да идентифицира обекта и в релационния модел