Sistemi Informativi:

Progetto della base dati

argomenti

- progetto della Base Dati
- Progetto per database ad oggetti
- Progetto per database ad oggetti/relazionale
- Progetto per database relazionale

Gestione dei dati

- alcuni oggetti del modello devono essere persistenti
- modi di realizzazione
  - Struttura dati
    - dati volatili (runtime in memoria, serializzabili tuttavia su disco)
  - Files
    - economici, semplici, permanenti. veloci
    - Basso livello (Read, Write)
    - le applicazioni devono fornire operazioni al livello di astrazione richiesto
  - Database
    - potenti, facili da portare
    - supportano concorrenza, transazioni ..

File o Database?

- quando un file?
  - dati voluminosi (bit maps)?
  - molti dati grezzi (core dump, tracce di eventi)?
  - conservazione per brevi periodi?
  - bassa densita' di informazioni (archivi, tracce)?
- quando e che database?
  - accesso a gran livello di dettaglio da molti utenti?
  - dati su molte piattaforme (sistemi eterogenei)?
  - accesso da parte di molte applicazioni?
  - occorre molta infrastruttura di gestione dati?

Database Management System

- contiene meccanismi per descrivere dati, gestire la memorizzazione persistente, fornire backup
- accesso concorrente
- contiene informazioni sui dati (“meta-dati”), I cosiddetti “schemi”

Proprieta' DBMS

- Grande
- Persistente
- Condivisibile, multi-utente
- Recovery
- Consistenza
- Sicurezza
- Estensibilita'

requisiti database

- spazio su disco
  - un DB richiede il triplo di spazio dei dati veri e propri
- tempo di risposta
  - I moderni DB distribuiti sono limitati dall I/O (trasferimento dati). Il tempo di risposta dipende anche dalla CPU,e dai lock
- modalita' di lock
  - *pessimistico:* Lock prima dell'accesso e rilascio dopo il termini dell'accesso
  - *ottimistico:* lettura e scrittura libere. Al termine si verifica: se si sono verificati conflitti, il lavoro e' perso
- gestione
  - DB grandi richiedono personale specializzato per security, performance, backup, …

gestione dei dati

- dati distribuiti?
- DB estensibile?
- frequenza di accesso al DB?
- frequenza di interrogazioni? nel caso peggiore?
- dimensioni delle richieste?
- occorre archiviare i dati (dati non piu' attuali)?
- trasparenza della locazione dati?
- occorre una sola interfaccia per accedere ai dati?
- formato delle query?
- data warehouse?
- relazionale o O-O?

Esempi importanti di DB

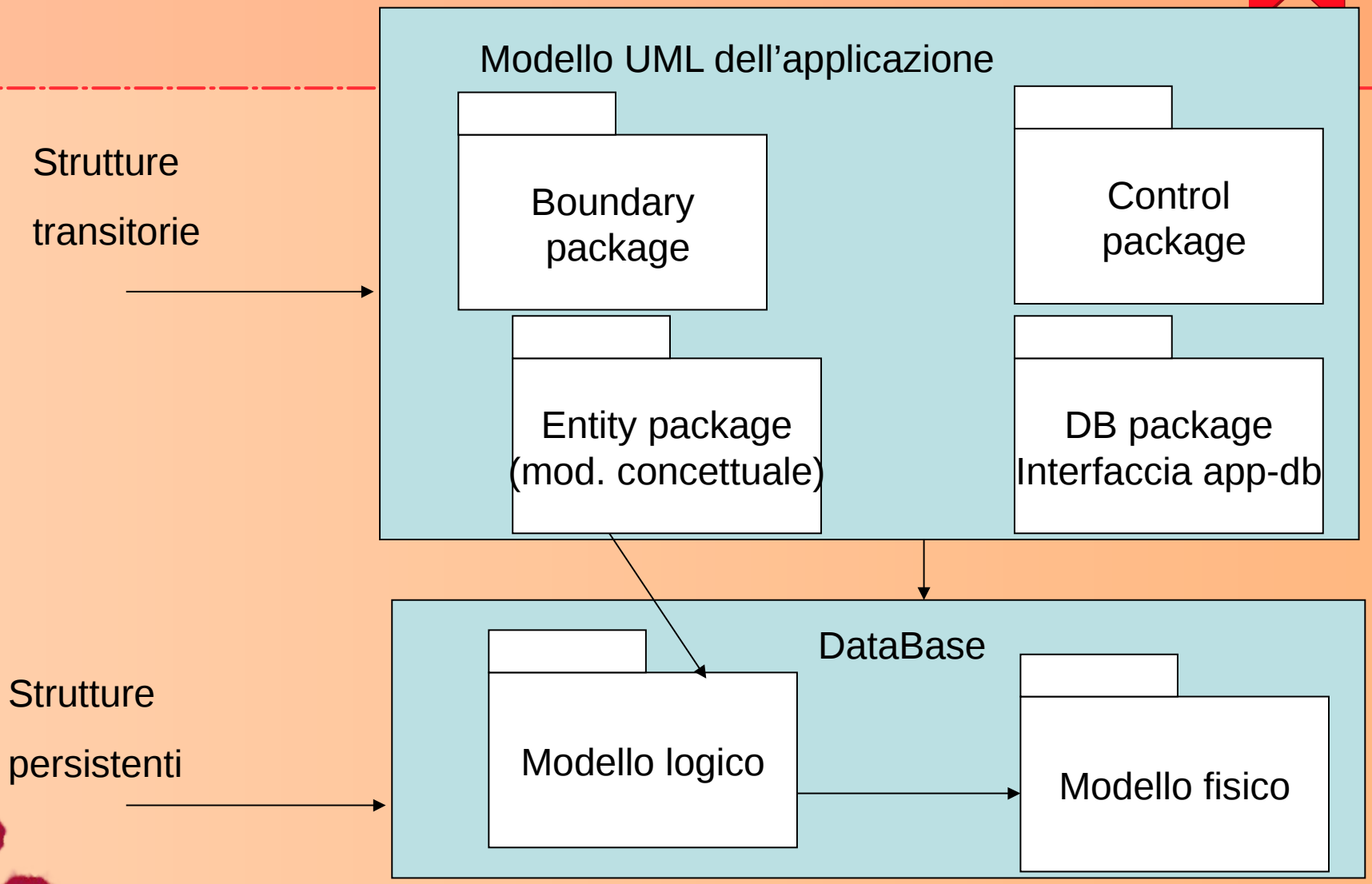
- Oggetti
  - ObjectStore Versant
- Oggetti / Relazionale
  - Oracle8 UniSQL
- Relazionale
  - Oracle sybase DB2 informix Access
- Vecchi modelli
  - Gerarchico (IMS)
  - Rete (IDMS)
  - Adabas
- File normale/archivio ad accesso diretto (hash table) Google ha BigTable
- Vasta documentazione online e tutorial per tutti i tipi di DB

Le prestazioni scendono piu' si sale di astrazione

Modello dei dati

- Si cerca di fornire modelli a diversi livelli di astrazione per nascondere o mostrare i dettagli
- **Modello concettuale**
  - Modello sintetico che concerne una singola applicazione
- **Modello logico**
  - Tiene conto di tutte le applicazioni presenti e future che accedono a quei dati
- **Modello fisico**
  - Descrive i meccanismi di memorizzazione dei dati sui supporti fisici

Modello persistente ed applicazione



Rappresentazione di oggetti in un DB

- Nella maggior parte dei DB in uso corrente le strutture dati del DB sono diverse (piu' semplici) da quelle degli oggetti
- Il DB deve essere progettato per fornire un accesso ottimale a tutte le applicazioni che ne faranno uso
  - Massimo Comune Divisore
- Tutto sarebbe risolto se non ci fosse un grosso problema di efficienza per i DB di grandi dimensioni
- Traducendo un modello UML ad un DB, l'enfasi e'posta sulla modellazione dei dati; poco rilevante supportare vincoli di visibilita'delle operazioni associate alle classi. Il DB deve in generale fare poche assunzioni sul modo di accesso

Database Object-Oriented

- Supporta tutti I concetti di modellazione O-O
  - Classi, Attributi, Metodi, Associazioni, Eredita'
- Mappare un modello ad oggetti a un O-O DB
  - Determinare gli oggetti persistenti.
  - Consueta analisi e progetto deli oggetti
  - Creare indici con specifici attributi per accesso efficiente
  - Mapping allo specifico prodotto:
    - In diversi framework classi ed associazioni sono implementate preparando specifiche dichiarazioni nel linguaggio opportuno per esse
    - Differenze da un sql a nseconda del fornitore

Il modello di un Object DB

- Standard del Object Database Management Group (ODMG)
- Un ODB fa apparire oggetti del DB come oggetti del linguaggio
- Interfaccia Object Storage - implementa il DB package
- ODBMS – il sistema di gestione di un ODB fornisce in genere un linguaggio Object SQL (OSQL) che permette di navigare un ODB come se fosse relazionale

operazioni in un ODB

- Un ODB fornisce un insieme di operazioni predefinite per la manipolazione di tipi di base e collezioni:
  - Su data, tempo, intervallo
  - Unione ed intersezione di insiemi, presenza di un elemento in una collezione etc

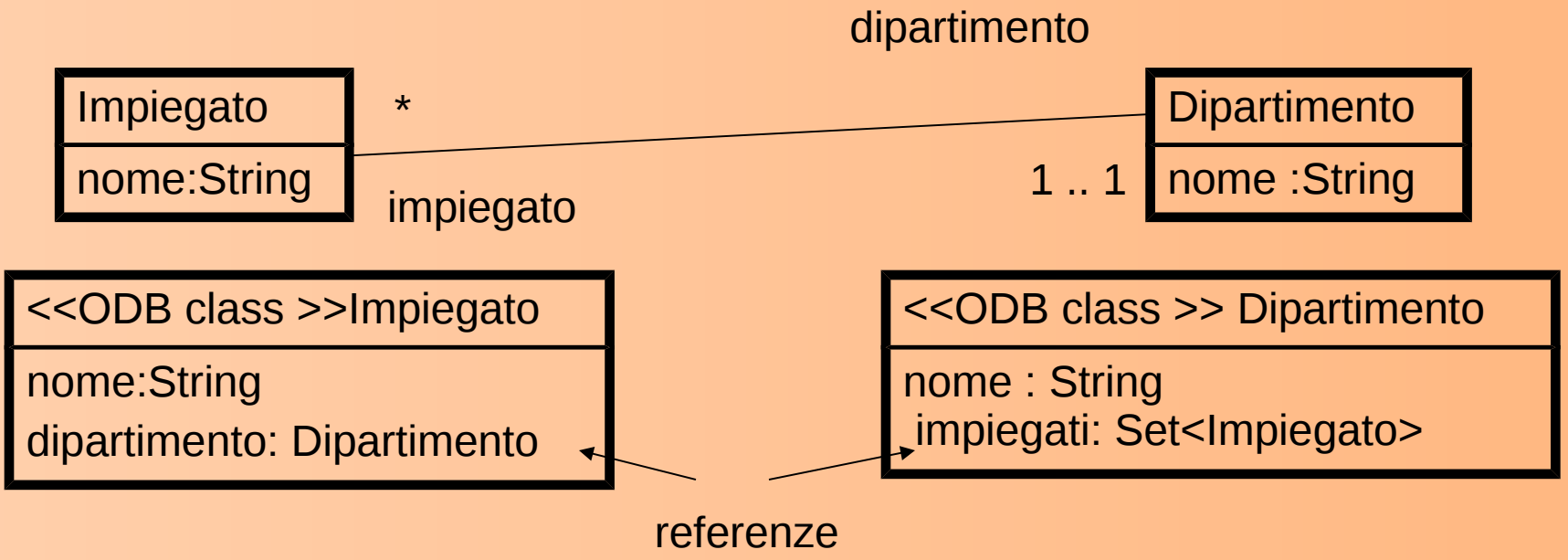
Supporto alle relazioni in un ODB

- Associazioni
  - Implementate tramite collezioni, Set<> e List<>
  - L'aggettivo **inverse** assicura integrita' referenziale
- Generalizzazioni
- Aggregazioni
- Chiavi per accesso tramite indici
  - Semplici e composte per accesso diretto per la individuazione di un oggetto
  - Diverse dall OID che identifica l'oggetto

Da UML a ODB

- In UML si aggiungono stereotipi dell' ODB
- Si traducono le classi del package **Entity** (oggetti persistenti)
- Si modella lo stato istantaneo (attributi) degli oggetti, mentre il comportamento e' definito nel progetto architetturale

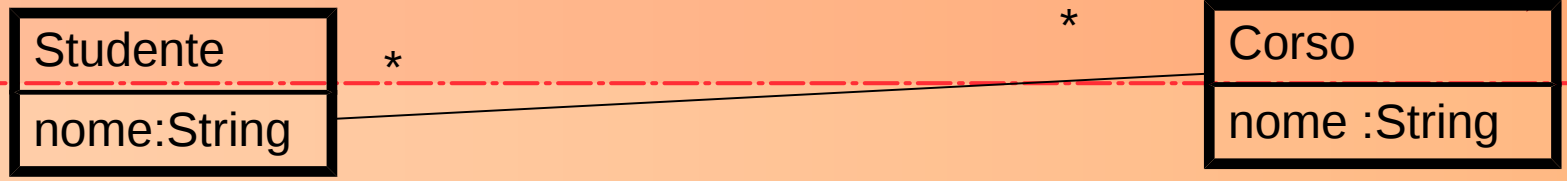
Associazioni da UML a ODB



Se una relazione ha cardinalita' max = 1 l'associazione si mappa con un attributo

Se cardinalita' max > 1 si usa Set (non ordinato) o List (ordinato)

Associazioni in un ODB



<<ODB>>Studente
<<attribute>>nome:String <<relationship>>frequenta: Set<Corso>

<<ODB>> Corso
<<attribute>>nome:String <<relationship>>alunni:List<<Studente>>

La creazione delle tabelle e' automatica

Class Studente {

Attribute string nome;

Relationship Set<Corso> frequenta

Inverse Corso::alunni; }

Class Corso {

Attribute string nome;

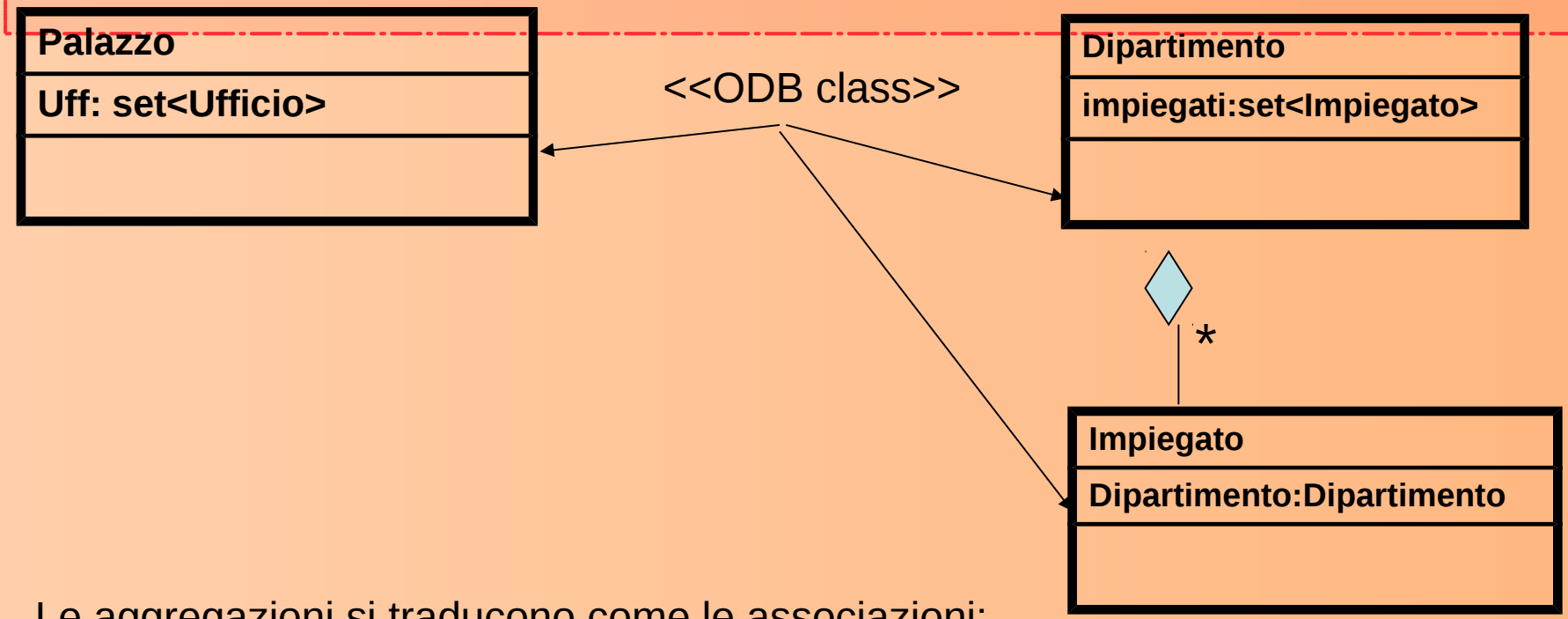
Relationship List<Studente> frequenta

Inverse Studente::frequenta; }

Mapping di aggregazioni

- Le aggregazioni per referenza si traducono come le associazioni:
- Quelle per valore diventano attributi composti **annidati** nell'oggetto di base.
- La semantica esatta va verificata proceduralmente

Mapping di aggregazioni



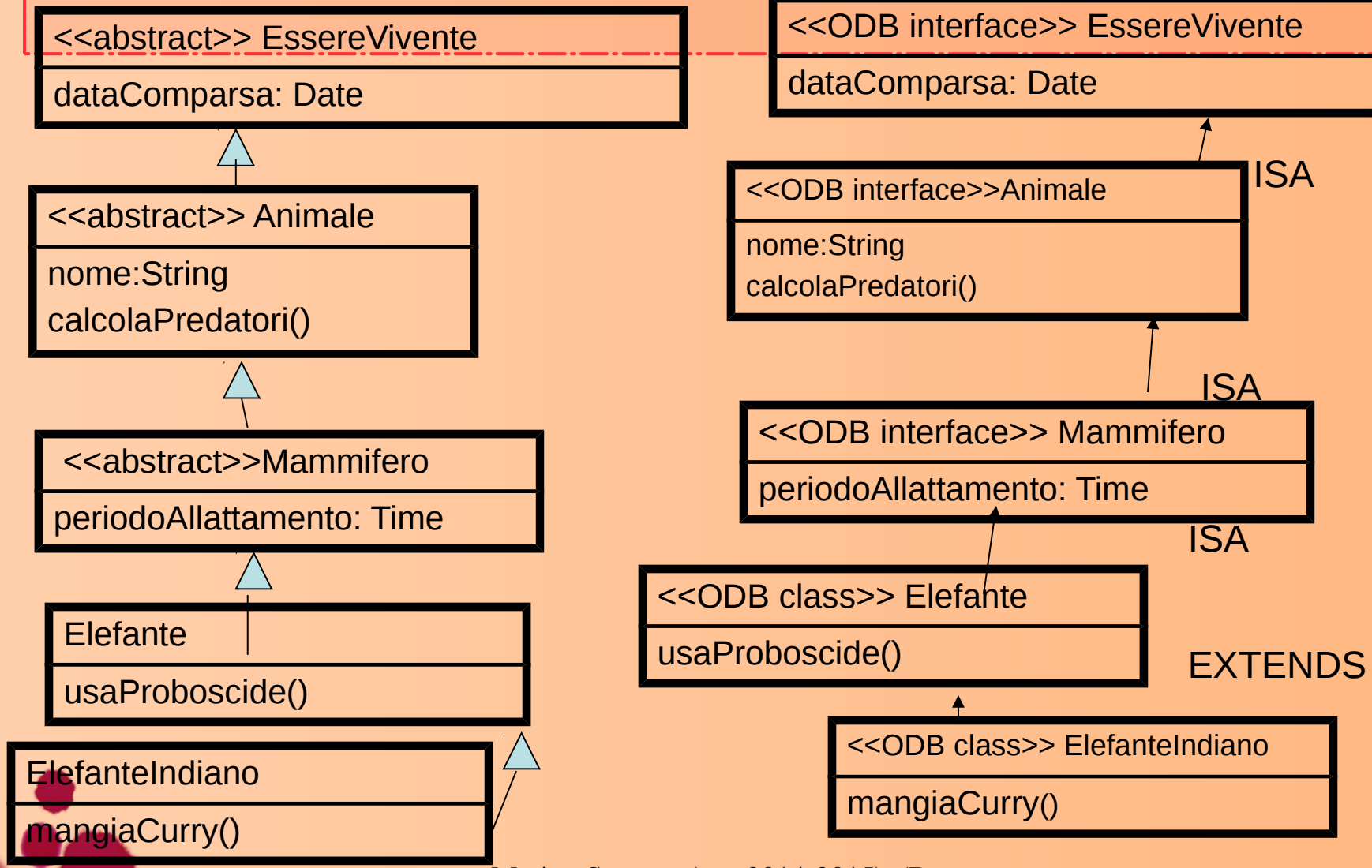
Le aggregazioni si traducono come le associazioni:

Che siano per valore o referenza deve essere realizzato proceduralmente: db neutrale

Generalizzazioni da UML a ODB

- **ISA** equivalente ad ereditare interfaccia (specifica del comportamento)
  - puo' essere multipla
  - non permessa tra classi
- **EXTENDS** -> ereditare implementazione (anche lo stato)
  - singola
  - Le interfacce non possono ereditare da classi; da quando entra una classe nella catena, tutti i discendenti sono classi
- Lo standard ODMG
  - Parola riservata (keyword) **interface** definisce una classe astratta
  - La parola **class** per classi istanziabili

ISA ed EXTENDS



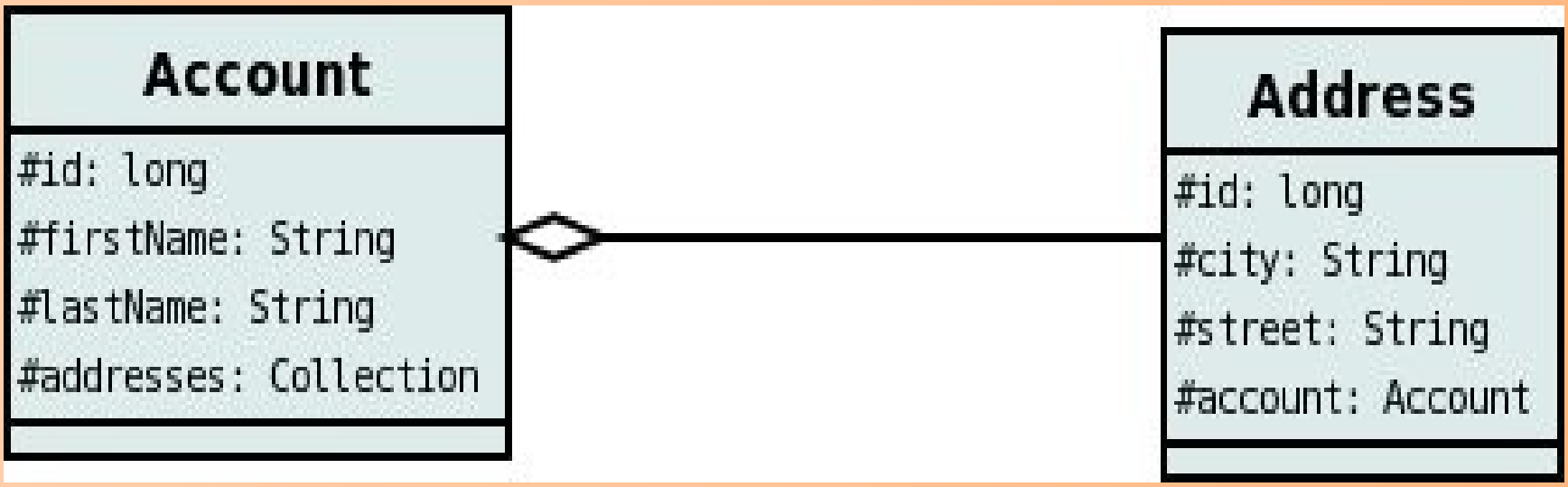
Modello ORDB

- Standard ANSI e ISO 1999
- In grado di gestire sia strutture relazionali che strutture ad oggetti
- I tipi di base possono essere composti in strutture arbitrariamente complesse
- Un ibrido di cui non parliamo in dettaglio

Mapping oggetti-relazionale tramite annotazioni o xml config file

- Esempio progetto eclipse
- Ogni classe ha un file associato che specifica come sara' scritta nelle tabelle relazionali
- Le proprieta' sono molte e spesso non semplici da usare
- Si cerca ovviamente la massima efficienza

Mapping oggetti-relazionale tramite annotazioni o xml config file



```
<package name="com.mydomain">
    <class name="Account">
        <field name="id" primary-key="true">
            <column name="ACCOUNT_ID"/>
        </field>
        ...
        <field name="addresses" mapped-by="account">
            <collection element-type="com.mydomain.Address"/>
            <join/>
        </field>
    </class>

    <class name="Address">
        <field name="id" primary-key="true">
            <column name="ADDRESS_ID"/>
        </field>
        ...
        <field name="account"/>
    </class>
</package>
```

Uso di annotazioni

```
public class Account
{
    ...

    @Persistent(mappedBy="account")
    @Join
    Collection<Address> addresses;
}

public class Address
{
    ...
}
```

Mapping oggetti-relazionale tramite annotazioni

- http://www.datanucleus.org/products/accessplatform_2_2/guides/jpa/tutorial.html
-
- Effetti delle annotazioni
  - detachable
  - default-fetch-group
- Attributo deve essere acceduto per essere caricato
  - Tutto poco automatico sembra creare quasi tanti problemi quanto ne risolve

Mapping diretto tramite api del linguaggio (python)

```
db.define_table('person',
                Field('name'))
db.define_table('thing',
                Field('name'))
db.define_table('ownership',
                Field('person', 'reference person'),
               Field('thing', 'reference thing'))
```

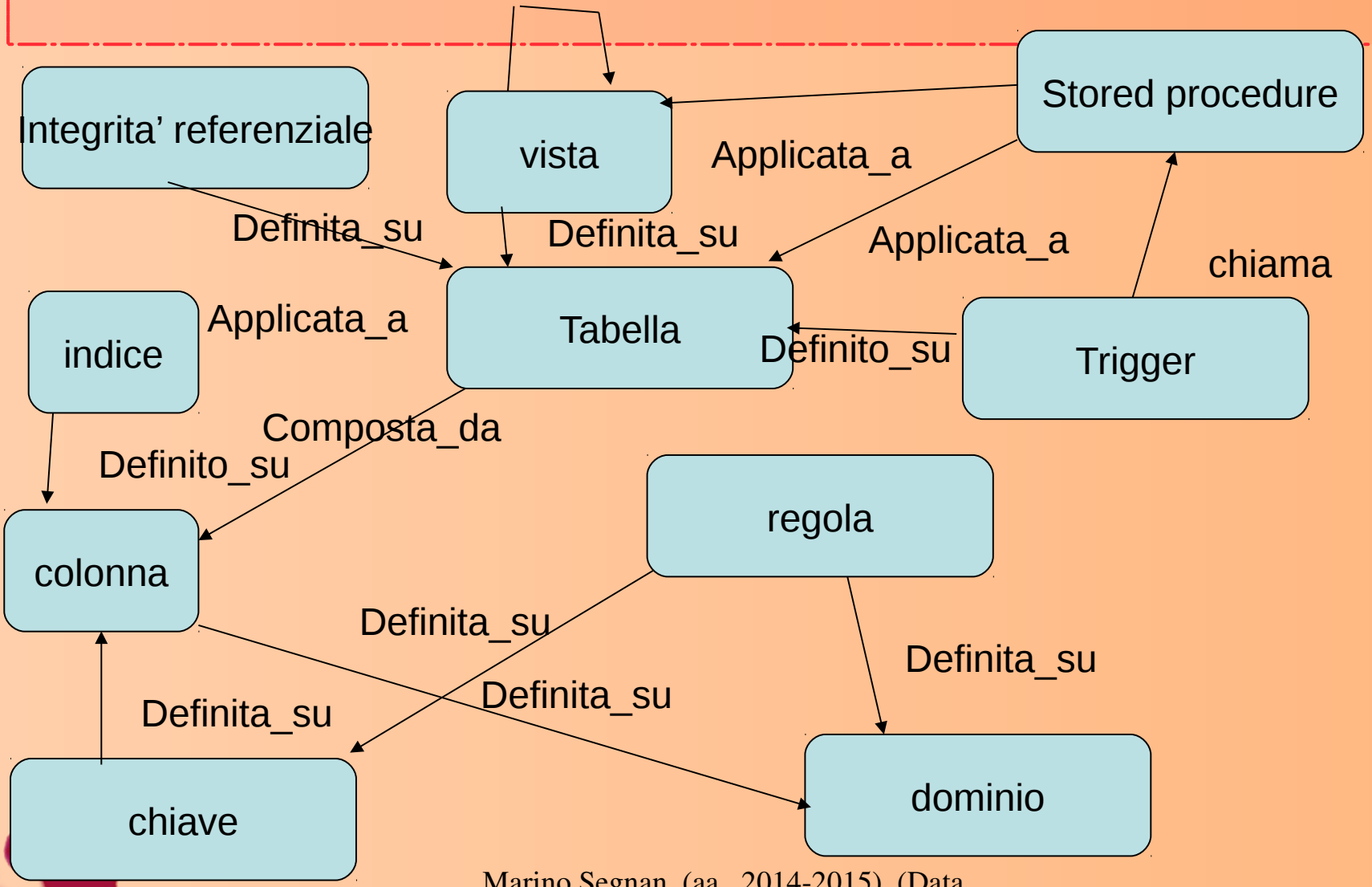
DB relazionale

- basato su algebra relazionale (insiemistica)
- Evita duplicazione dei dati
- dati come tabelle bidimensionali. con colonne fisse e righe arbitrarie distinte
  - chiave primaria: combinazione di attributi che identifica univocamente una riga (non esistono due righe uguali!) . ogni tabella ha solo una di esse
  - chiave esterna: referenza ad una chiave primaria in un'altra tabella
- SQL e' lo standard di manipolazione

Modello RDB

- Standard ANSI ISO 1992
- Sostituiva modello gerarchico
- Sara’ sostituito da ORDB (quando ?)

Strumenti di modellazione in RDB

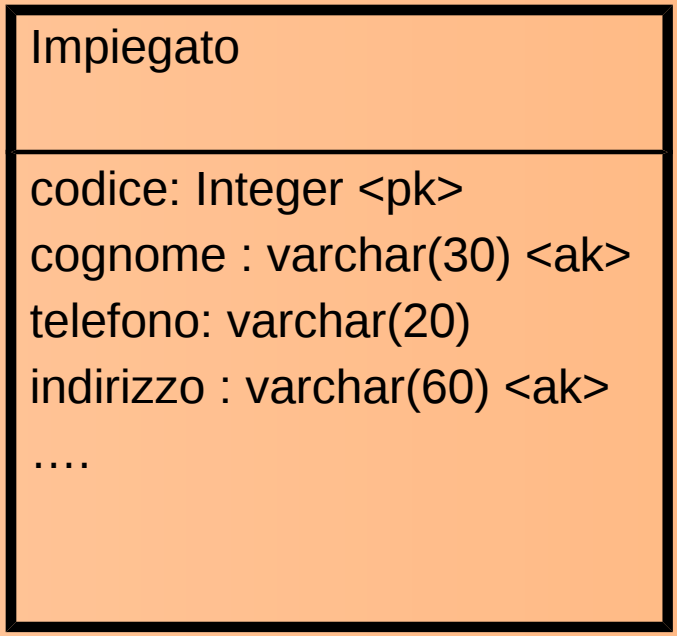


Domini, colonne e regole in RDB

- Dominio - insieme valori possibili per una colonna
- Le colonne memorizzano **solo** tipi elementari
  - Le regole impongono vincoli procedurali sui valori memorizzabili:
    - Es. la data deve essere gg/mm/aaaa
    - Se la data non e' immessa, assumi oggi

tabella relazionale RDB

- Numero fisso di colonne
- Corrisponde ad un insieme matematico: non ci possono essere due righe identiche
  - Ogni riga e' identificata dai valori di un insieme dei suoi attributi: **chiave candidata**
  - una **chiave candidata** viene eletta **chiave primaria**
  - **Spesso si aggiunge**



Integrita’ referenziale

- Occorre verificare quando si modifica una tabella che non vi siano righe di altre tabelle che si riferivano ai valori cancellati
- Se ad esempio avessi due tabelle con treni e passeggeri, potrei imporre che un treno puo’ essere cancellato solo se non ci sono viaggiatori prenotati

Relational DB in pratica

- Un insieme di tabelle rettangolari (righe e colonne) di tipi elementari
- In una tabella, ogni riga ha le stesse colonne
- Ogni riga e' diversa, per evitare duplicazioni inutili
- Ogni riga e' identificata da un insieme unico di valori, una o piu' colonne, detta Primary Key (PK)
- Ogni colonna puo' essere usata per creare associazioni temporanee tra tabelle diverse
- Il linguaggio SQL permette di creare e combinare in mille modi diversi i dati delle varie tabelle.

RDB in pratica

- Class Cliente { nome:String, prodotto: String}
- Class Prodotto { nome:String,costo: Currency}

mario	telefono
cesare	bici
mario	tv
antonio	

telefono	100
bici	150
tv	300
giacca	100

Colonne simili su cui fare “JOIN”

Quanto ha speso mario? C'e' qualche prodotto invenduto?..

RDB in pratica

- Abbiamo concettualmente due tipi di dati:
  - Dati elementari (nome , cognome...)
  - Dati che sono creati per fare da “ponte”, relazioni tra tabelle, anche N..N
- Si creano inoltre degli indici sui dati per avere accesso efficiente ai dati piu' usati

Soci di un club e corsi di ginnastica

mario	10-10-99	00001
gino	8-8-88	00002
dino	7-7-77	00003
pino	6-6-66	00004

pilates	12-13	001
nuoto	14-15	002
yoga	16-17	003

00001	001
00001	002
00001	003
00002	001
00002	003
00003	002

RDB in pratica

- Nella slide precedente, avremmo potuto anche creare la associazione usando direttamente nomi sport e persone, ma:
- E' prudente (e praticamente sempre necessario) creare, per evitare potenziali duplicati, una chiave primaria aggiuntiva rispetto ai dati di base, che fara' il ruolo di identificatore della riga in tutte le possibili relazioni del DB

da UML a RDB

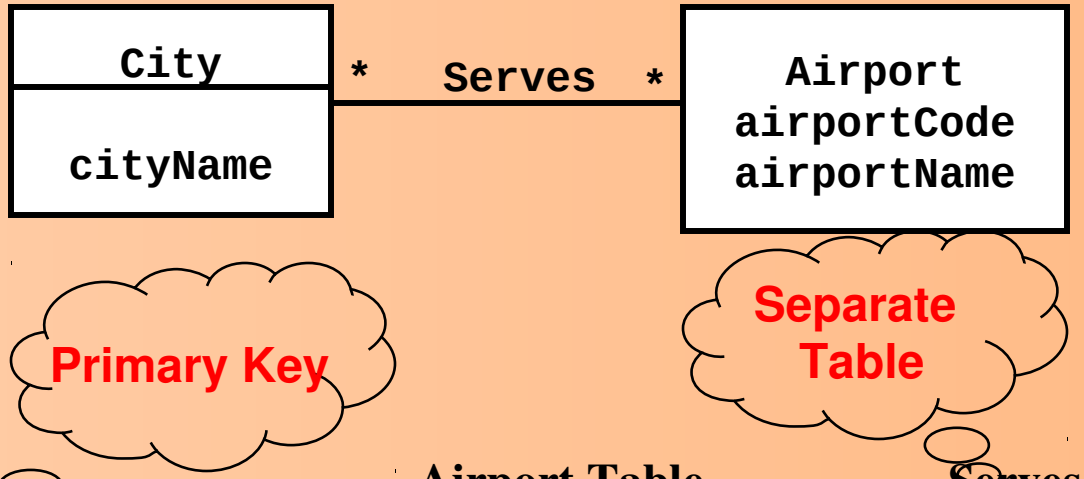
- In un RDB non abbiamo referenze, ne' OID.
- Tutte le relazioni che si possono modellare in UML, lo sono in RDB esclusivamente tramite relazioni tra valori delle colonne delle tabelle che fungono da **chiave**
- Due tabelle qualunque possono essere messe in relazione purche' esista in entrambe almeno una colonna definita sullo stesso dominio.
- Modello elementare ma che e' molto **flessibile** quando si devono fare correlazioni tra dati che non sono state previste in anticipo (o quasi).

Mappare oggetti su un DB relazionale

- In generale I modelli UML si possono mappare su DB relazionali come segue:
  - semplificazione perche' tutto va solo rappresentato in termini di tabelle
  - ogni classe e' una tabella
  - ogni attributo e' una colonna
  - una istanza di una classe e' una riga
  - *Una associazione molti a molti e' mappata su una sua tabella association,*
  - una associazione uno-a–molti e' mappata come una chiave esterna dal lato "molti"
  - I metodi non sono rappresentati

Oggetti trasformati in tabella

associazione molti-a -molti



City Table

cityName
Houston
Albany
Munich
Hamburg

Airport Table

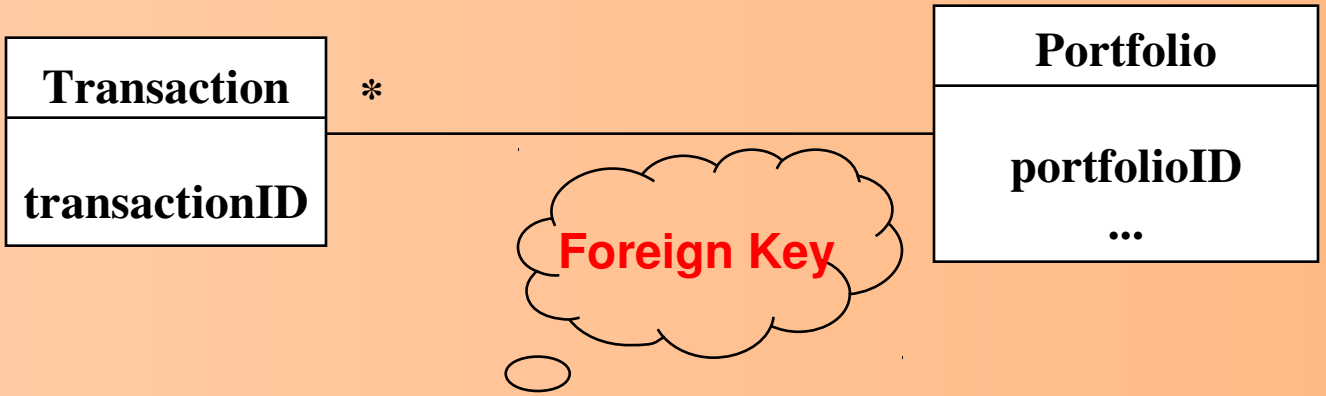
airportCode	airportName
IAH	Intercontinental
HOU	Hobby
ALB	Albany County
MUC	Munich Airport
HAM	Hamburg Airport

Serves Table

cityName	airportCode
Houston	IAH
Houston	HOU
Albany	ALB
Munich	MUC
Hamburg	HAM

Oggetti trasformati in tabella

associazione uno a molti e viceversa: chiavi



Transaction Table

transactionID	portfolioID

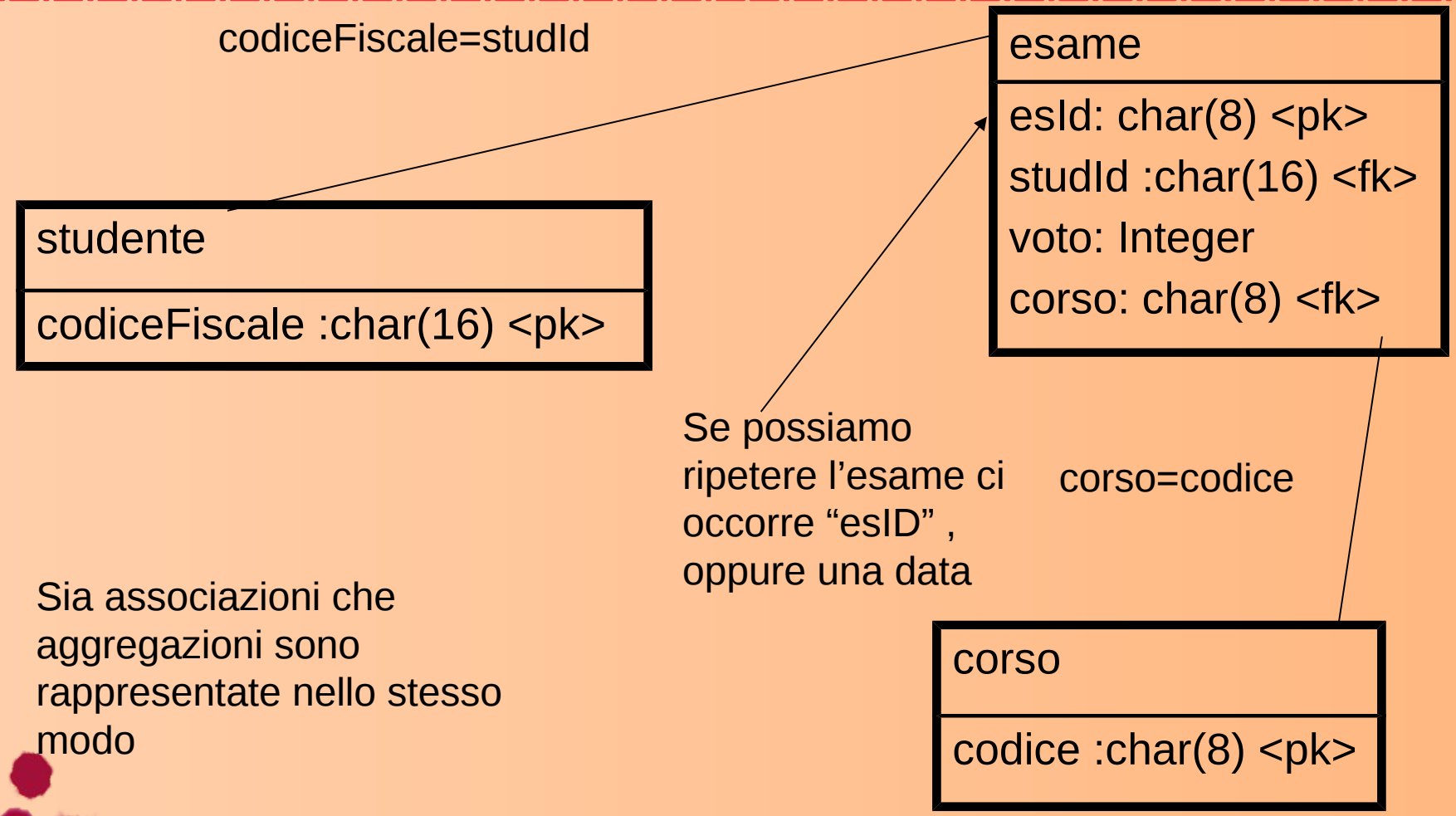
Portfolio Table

portfolioID	...

Associazioni ed aggregazioni da UML a RDB

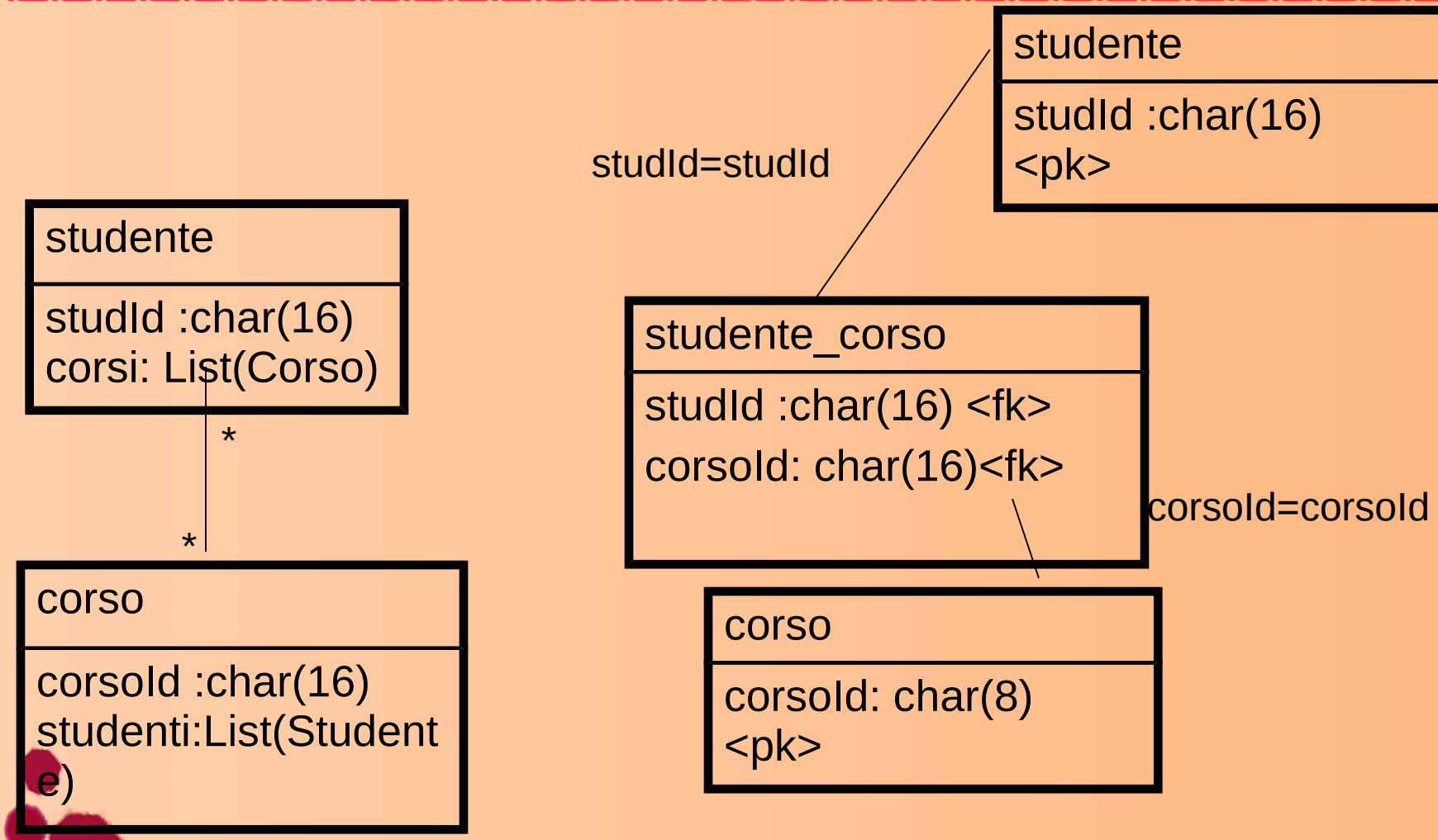
- creare tabelle che memorizzano in ogni colonna una chiave del componente dell'aggregato ;
  - Relazione con la semantica di **member**
- In tutti i casi l'operazione di JOIN ricostruisce l'oggetto desiderato
- Proprieta' rispettate tramite procedure

da UML a RDB

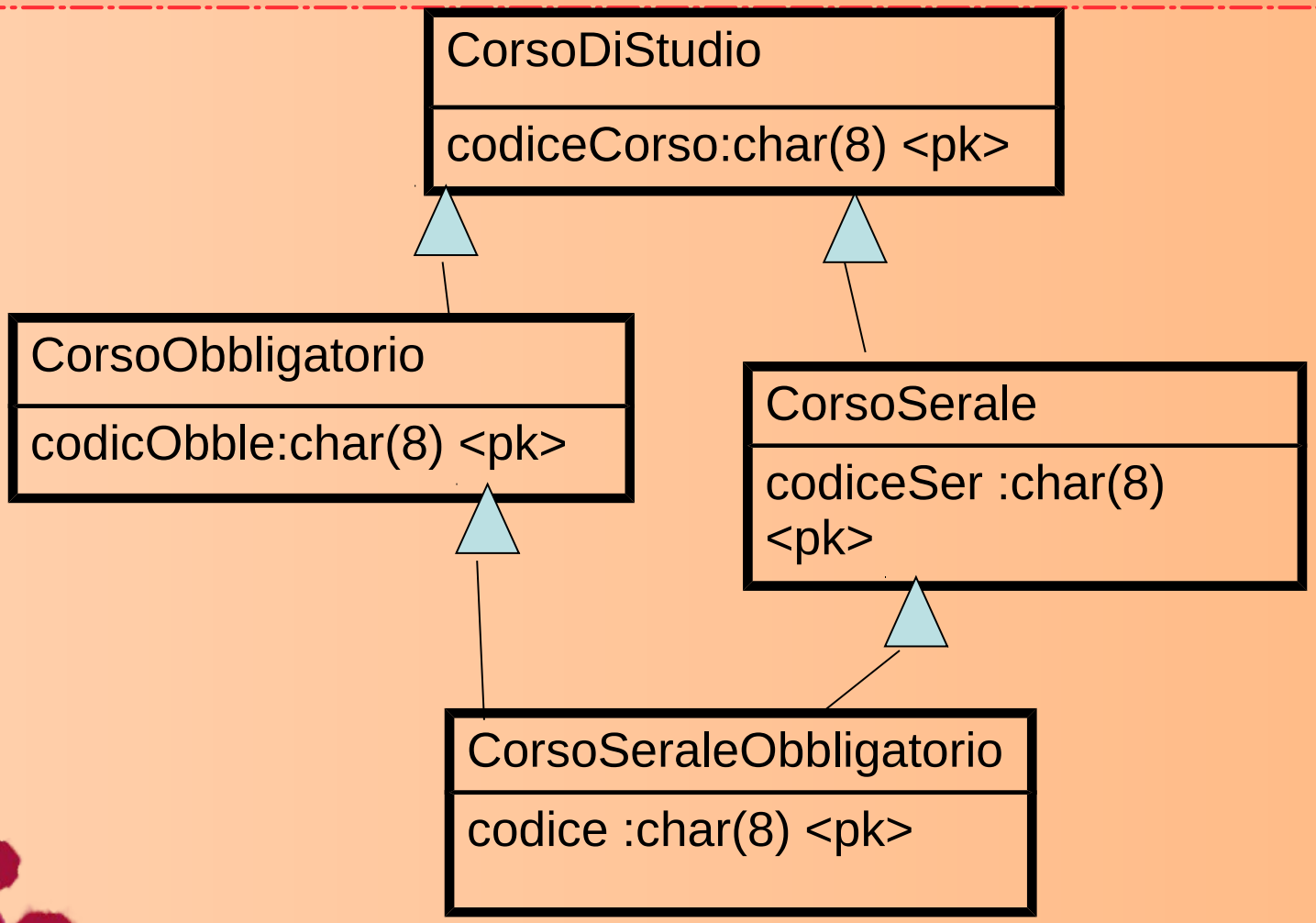


Sia associazioni che aggregazioni sono rappresentate nello stesso modo

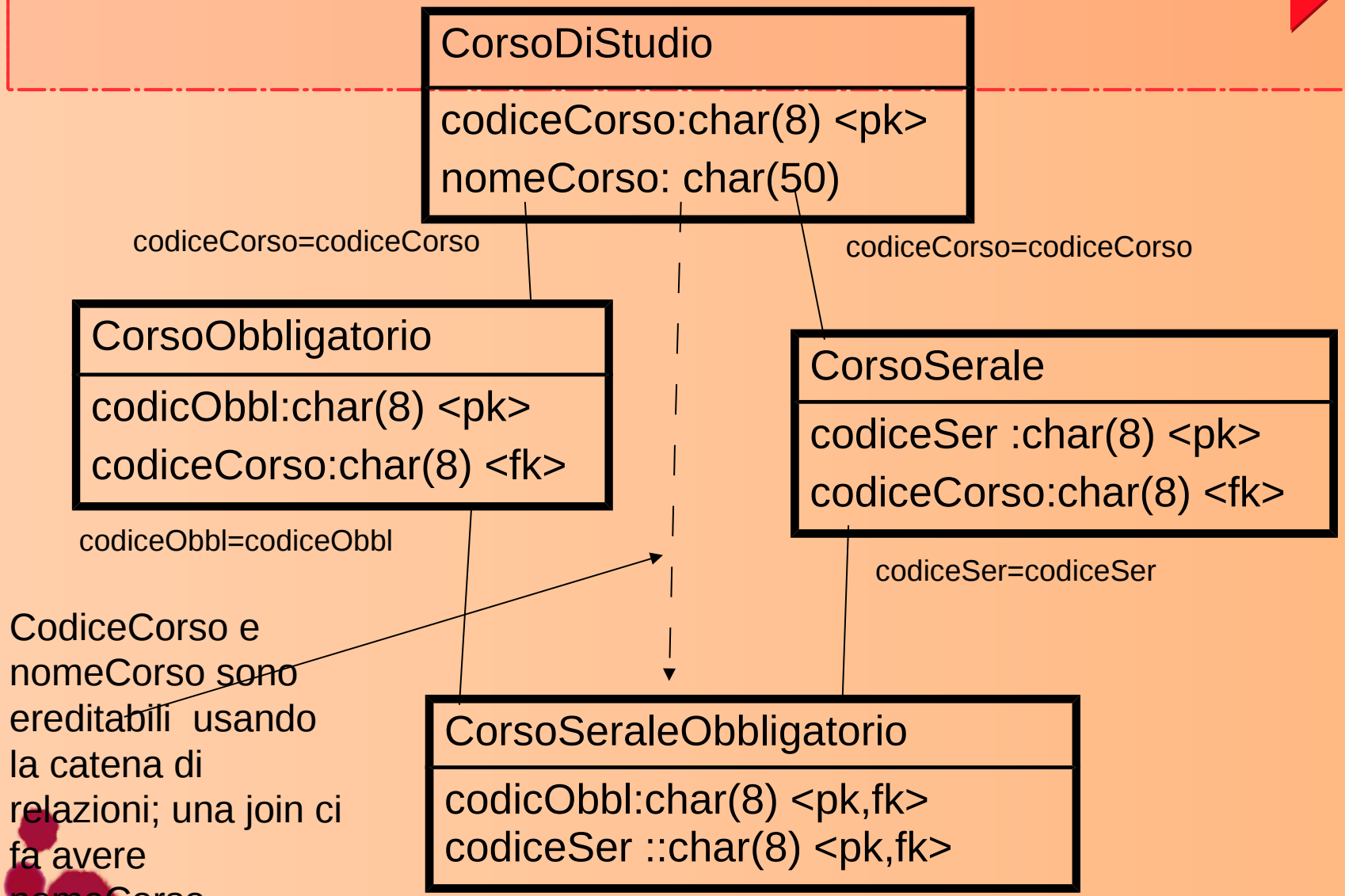
Associazioni multiple da UML a RDB



Generalizzazioni da UML a RDB



Generalizzazioni in RDB 1:1



CodiceCorso e nomeCorso sono ereditabili usando la catena di relazioni; una join ci fa avere nomeCorso ovunque

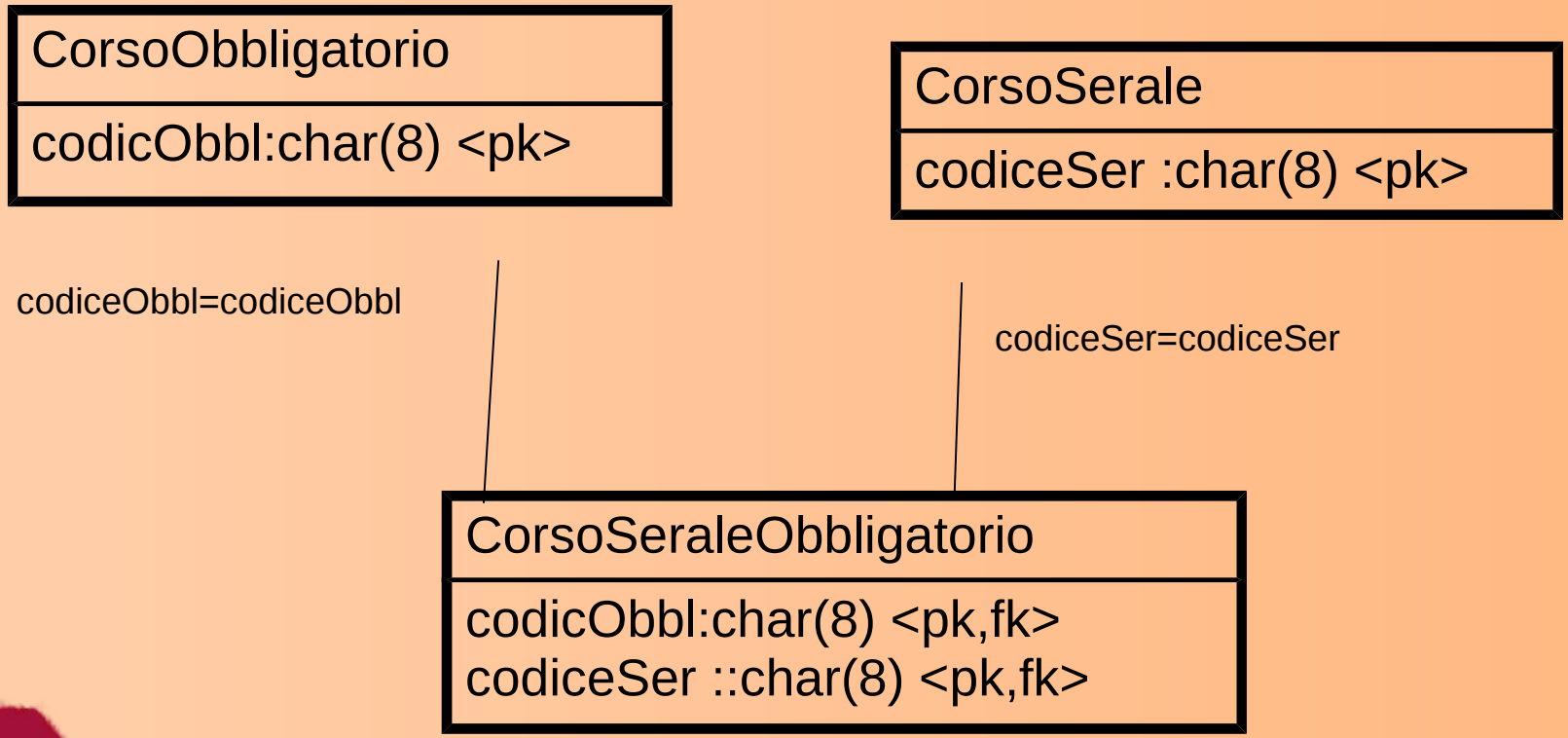
Generalizzazioni in RDB: una classe unica

CorsoDiStudio
codiceCorso:char(8) <pk> nomeCorso: char(50) serale: char (1) obbligatorio: char (1)

Deve poter contenere tutti gli attributi possibili; simile ad una UNION in “C”. ci sara' un campo (attributo) “discriminatore”

Generalizzazioni in RDB

Ogni classe concreta definita da una tabella



Generalizzazioni in RDB

Ogni classe concreta disgiunta definita da una tabella

CorsoObbligatorio
codicObbl:char(8) <pk> serale: char(1)

CorsoSerale
codiceSer :char(8) <pk> obbligatorio: char(1)

RDB: se non abbiamo

- Gerarchie molto articolate
- Esigenze di risparmio spazio
- Allora:
- Usiamo una tabella per ogni classe

Google datastore (non RDB)

- Gerarchico
- Key-value
- Attributi diversi possibili per entita' dello stesso tipo
- Scalabile
- Strong/weak consistency
- No Joins
- http://developers.google.com/appengine/docs/python/datastore/

Google datastore

- Schema-less (entita' di tipo uguale ma con attributi diversi)
- Scalabile, tempo query proporzionale al risultato e non ai dati
- Query consistente su una entita'
- Query eventualmente consistente su entita' separate.

```
import datetime
from google.appengine.ext import db
from google.appengine.api import users


class Employee(db.Model):
  name = db.StringProperty(required=True)
  role = db.StringProperty(required=True,
                           choices=set(["executive", "manager", "producer"]))
  hire_date = db.DateProperty()
  new_hire_training_completed = db.BooleanProperty(indexed=False)
  email = db.StringProperty()


e = Employee(name="John",
             role="manager",
             email=users.get_current_user().email())
e.hire_date = datetime.datetime.now().date()
e.put()
```

Sommario 1

- Tre modelli del DB: concettuale, logico, fisico
- Un modello ad oggetti in UML viene tradotto in modello logico del DB
- La traduzione e’ piu’ laboriosa passando da ODB a RDB:
  - Nel caso RDB in particolare il modellatore ha diverse opzioni di scelta della corrispondenza tra oggetti ed entita’ del RDB

Sommario 2

- Un puro ODB non risolve tutto:
- Tutte le app devono essere Object Oriented oppure occorre interfaccia;
- Strumento in piu' da apprendere
- Query difficili se impreviste
- Ci sono tanti DB preesistenti da usare,
  - Il mapping Object-Relational puo' essere complicato e non esonera dall' apprendimento del modello relazionale
  - Tanto vale scrivere direttamente l'interfaccia Oggetto → db relazionale