Azzolini Riccardo 2020-03-10

Object-relational data model – Ereditarietà

1 Ereditarietà tra tipi

Nel modello dei dati object-relational, l'ereditarietà riguarda sia i tipi che le tabelle. Nell'ambito dei tipi, essa permette di definire dei **sottotipi**, che, come in Java, ereditano gli attributi dei loro supertipi.

Ad esempio, dato

```
CREATE TYPE Person AS (
  name Name,
  address Address,
  birth DATE
)
NOT FINAL;
```

per definire un tipo `Student` come sottotipo di `Person` si usa la parola chiave `UNDER` (analoga all'`extends` di Java):

```
CREATE TYPE Student UNDER Person AS (
  id NUMERIC,
  degree VARCHAR(50)
)
FINAL;
```

Le istanze di `Student` avranno allora tutti gli attributi di `Person`, ma in più anche `id` e `degree`.

Nota: In questo esempio, `Student` è stato dichiarato `FINAL`, quindi non potrà avere ulteriori sottotipi. Ciò non è obbligatorio: dichiarando anch'esso `NOT FINAL`, come `Person`, si potrebbe creare una gerarchia a più livelli.

1.1 Tipi non istanziabili

Il comando `CREATE TYPE` ha una clausola opzionale che specifica se il tipo definito potrà essere istanziato:

- Con `NOT INSTANTIABLE`, il tipo non può essere direttamente istanziato, ma è comunque possibile creare istanze dei suoi eventuali sottotipi (analogamente a una classe astratta in Java).

  ```
  CREATE TYPE Person AS (
    name Name,
    address Address,
    birth DATE
  )
  NOT INSTANTIABLE
  NOT FINAL;
  ```

  Ciò ha senso solo per tipi `NOT FINAL`: un tipo `FINAL` e `NOT INSTANTIABLE` sarebbe effettivamente inutilizzabile.

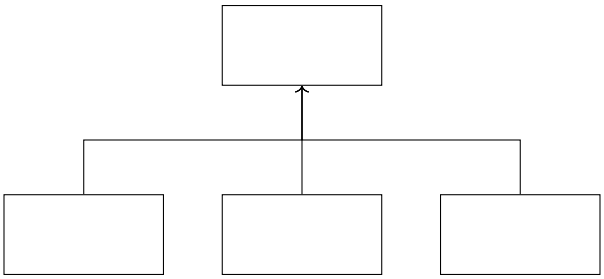
- Con `INSTANTIABLE`, che è l'opzione di default, è possibile creare istanze del tipo (che è quindi analogo a una classe concreta).

  ```
  CREATE TYPE Student UNDER Person AS (
    id NUMERIC,
    degree VARCHAR(50)
  )
  INSTANTIABLE
  FINAL;
  ```

1.2 Ereditarietà singola

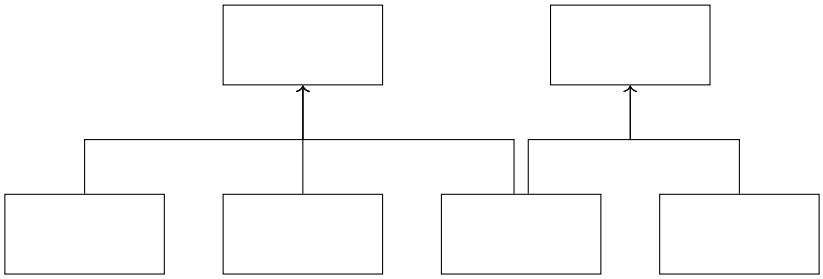
Un tipo SQL può avere un numero qualsiasi di sottotipi (a meno che non sia `FINAL`), ma *al più un supertipo*: non è supportata l'*ereditarietà multipla*.[1]

Di conseguenza, è possibile realizzare solo gerarchie ad albero,



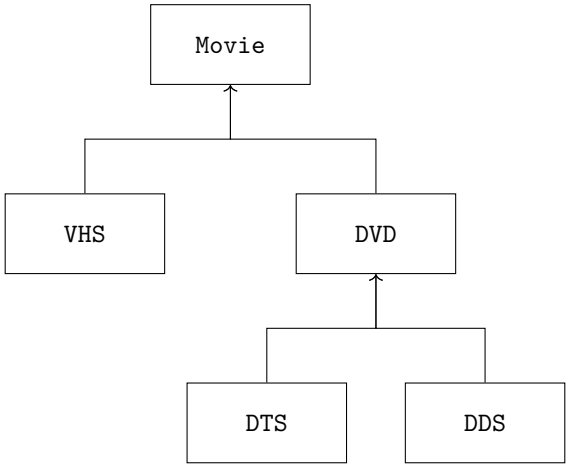
e non gerarchie come la seguente, che sarebbe comunque valida a livello concettuale:

[1] Ciò è analogo alle classi (ma non alle interfacce) Java, con la differenza che un tipo SQL può non avere alcun supertipo, mentre in Java tutte le classi ereditano (direttamente o indirettamente) dalla superclasse `Object`.



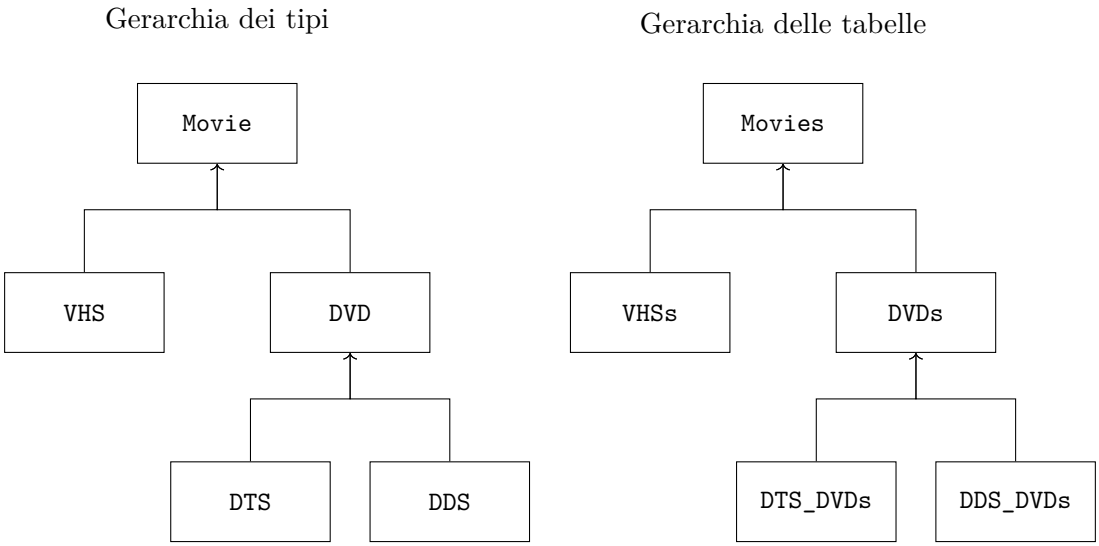
2 Ereditarietà tra tabelle

Data una gerarchia di tipi, come ad esempio

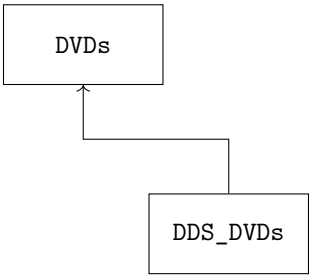



```
CREATE TYPE Movie AS (
  title VARCHAR(100),
  description VARCHAR(500),
  runs INTEGER -- durata in minuti
)
INSTANTIABLE
NOT FINAL;

CREATE TYPE DVD UNDER Movie AS (
  stock_number INTEGER,
  price DECIMAL(5, 2)
)
INSTANTIABLE
NOT FINAL;

-- Analogamente per i tipi VHS, DTS e DDS
```

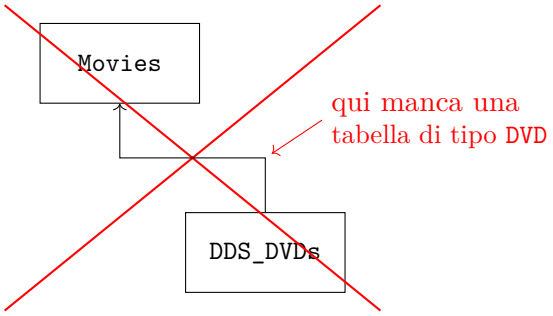
si può realizzare una gerarchia analoga tra le *typed table* che ne contengono le istanze:



È anche ammesso creare tabelle corrispondenti solo a una parte della gerarchia dei tipi,[2]



purché non si "saltino" livelli della gerarchia. Di conseguenza, la seguente gerarchia di tabelle non è ammessa:

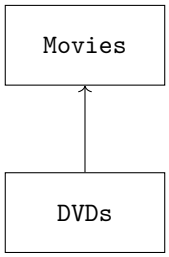


[2]In generale, non è necessario che un database contenga almeno una tabella per ogni tipo dichiarato.

Spetta al programmatore SQL creare gerarchie di tabelle che siano "corrette" rispetto alle gerarchie dei tipi corrispondenti, e, per schemi complessi, ciò può diventare piuttosto difficile.

2.1 Sintassi

Per definire, ad esempio, la gerarchia di tabelle



si usano i seguenti comandi:

```
CREATE TABLE Movies OF Movie (
  REF IS movie_oid SYSTEM GENERATED
);

CREATE TABLE DVDs OF DVD UNDER Movies;
```

- Con la parola chiave `REF`, si definisce nella tabella `Movies` un attributo aggiuntivo (oltre a quelli del tipo `Movie`), che funge da identificatore univoco per le tuple. Questo è necessario per poter distinguere eventuali istanze di `Movie` i cui attributi contengono valori uguali:[3] siccome non è possibile definire una chiave per un tipo, l'identificazione deve essere gestita nelle typed table associate a esso.

  In questo esempio, l'attributo `REF` è chiamato `movie_oid` (dove `oid` è un'abbreviazione di "object identifier"), e i suoi valori sono assegnati automaticamente in modo univoco dal DBMS (`SYSTEM GENERATED`), probabilmente mediante l'uso di un contatore.

- Per indicare che `DVDs` è una sottotabella di `Movies`, si usa la parola chiave `UNDER` (come nella definizione dei sottotipi).

- `DVDs` eredita l'attributo identificatore di `Movies`, quindi non è necessario (né ammesso) specificarne uno anche qui.

[3] La possibilità di distinguere istanze di un tipo contenenti valori uguali è una delle caratteristiche tipiche del modello object-oriented (ad esempio, in Java, ciò avviene automaticamente, sfruttando come identificatore di ciascun oggetto il suo riferimento). Poiché SQL classico non prevede tale concetto, nella sua estensione object-relational esso viene appunto implementato mediante l'aggiunta di un attributo speciale alla typed table.

È anche possibile creare typed table che impongono dei vincoli sugli attributi, mediante la sintassi `WITH OPTIONS`; tali vincoli vengono ereditati da eventuali sottotabelle. Ad esempio:

```
CREATE TABLE ShortMovies OF Movie (
  REF IS movie_oid SYSTEM GENERATED,
  runs WITH OPTIONS CHECK (runs < 90)
);

CREATE TABLE ShortDVDs OF DVD UNDER ShortMovies (
  price WITH OPTIONS CHECK (price < 1.99)
);
```

- `ShortMovies` può contenere solo istanze di `Movie` che hanno una durata inferiore a 90 minuti.
- `ShortDVDs` può contenere solo istanze di `DVD` con durata inferiore a 90 minuti (vincolo ereditato da `ShortMovies`) e prezzo minore di 1.99.

2.2 Interrogazioni

Quando si esegue un'interrogazione su una supertabella, vengono automaticamente incluse anche le tuple delle sottotabelle, cioè le istanze dei sottotipi. Questo meccanismo può essere motivato osservando che, a livello teorico, tutte le istanze di un sottotipo sono anche istanze del suo supertipo.

Ad esempio, la query

```
SELECT title, runs
FROM ShortMovies
WHERE runs < 60;
```

considera sia le tuple di `ShortMovies` che quelle di `ShortDVDs`, cioè sia le istanze di `Movie` che quelle di `DVD`, perché tutte le istanze di `DVD` sono anche istanze di `Movie`.

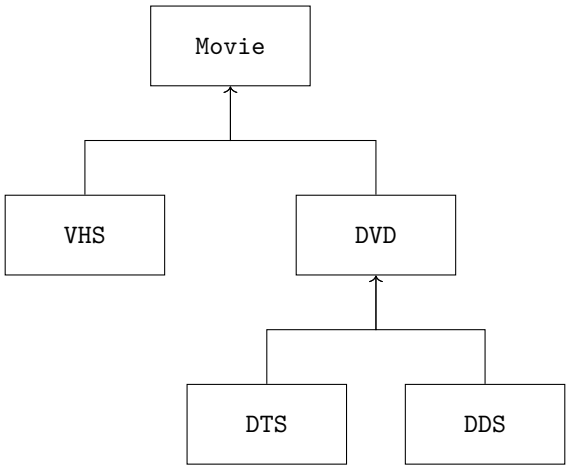
Se, invece, si vogliono ottenere solo le tuple della specifica tabella indicata nel `FROM`, è necessario usare la parola chiave `ONLY`:

```
SELECT title, runs
FROM ONLY(ShortMovies)
WHERE runs < 60;
```

3 Interazione tra gerarchia dei tipi e gerarchia delle tabelle

L'interazione tra le typed table che formano una gerarchia, contenenti istanze di tipi a loro volta messi in gerarchia, è un aspetto che richiede molta attenzione, e può essere risolto in modi diversi.

Dal punto di vista dei tipi (cioè quello object-oriented), ogni volta che si crea un'istanza di un sottotipo, si sta anche creando un'istanza del suo supertipo (perché, come già detto, un'istanza di un sottotipo appartiene anche al supertipo). Ad esempio, data la gerarchia dei tipi



se si crea un'istanza di `DDS`, se ne stanno creando anche una di `DVD` e una di `Movie`.

Invece, dal punto di vista delle tabelle (cioè quello relazionale), l'interazione tra i livelli della gerarchia, e, più in generale, l'organizzazione dei dati rappresentati, possono essere gestite in vari modi, chiamati *modelli di consistenza dei dati tra object-oriented data model e relational data model*:

- **duplicate row model**;
- **single table model**;
- **union model**.

3.1 Duplicate row model

- Ogni riga nella supertabella ha *al più* una riga corrispondente in ciascuna sottotabella, ma potrebbe non averne nessuna, se rappresenta un'istanza del supertipo che non è istanza di alcun sottotipo.
- Viceversa, ogni riga in una sottotabella ha *esattamente* una riga corrispondente in ogni supertabella (risalendo la gerarchia fino alla radice).

In altre parole, ogni oggetto è presente come riga nelle tabelle corrispondenti a tutti i tipi di cui è istanza. Ciò significa che un'interrogazione sulla supertabella restituisce naturalmente anche le istanze dei sottotipi, ma, in compenso, le operazioni di aggiornamento devono essere effettuate tenendo conto della gerarchia:

- se si inserisce una tupla in una sottotabella, bisogna aggiungere anche nelle supertabelle delle tuple corrispondenti (contenenti solo gli attributi che la sottotabella eredita);
- se si cancella una tupla nella supertabella, è necessario cancellare a cascata anche le eventuali tuple corrispondenti nelle sottotabelle;
- le modifiche ai valori degli attributi ereditati devono essere effettuate sia sulla supertabella che sulle sottotabelle.

Tale gestione è molto simile a un vincolo di integrità referenziale, e se ne occupa il DBMS, se implementa questo modello; il programmatore deve solo sapere che gli aggiornamenti effettuati su una tabella possono avere "pesanti" ripercussioni anche sulle altre tabelle della gerarchia.

3.2 Single table model

Tutta la gerarchia è materializzata in un'unica tabella, contenente gli attributi di tutti i tipi coinvolti, più un attributo aggiuntivo (chiamato ad esempio "hierarchy") che indica il tipo dell'oggetto rappresentato da ciascuna riga.

All'interno della singola tabella, gli attributi appartenenti ai sottotipi di cui una riga determinata non è istanza assumono valori nulli.

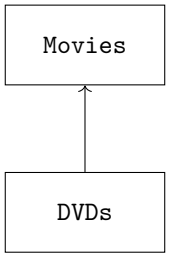
Ad esempio, considerando sempre `Movie` e i suoi sottotipi, se si creassero tabelle corrispondenti a ciascuno dei tipi di questa gerarchia si otterrebbe una struttura del genere:

`title`	`description`	`runs`	`stock_number`	…	*hierarchy*
…	…	…	`NULL`	…	`Movie`
…	…	…	83125	…	`DVD`
…	…	…	19846	…	`DDS`
…	…	…	`NULL`	…	`VHS`
…	…	…	…	…	…

3.3 Union model

Ciascuna tabella della gerarchia contiene *solo* le istanze dello specifico tipo corrispondente (e non quelle dei suoi sottotipi).

Ad esempio, se si creasse la gerarchia di tabelle



`Movies` conterrebbe solo le istanze del tipo `Movie` che non sono anche istanze di `DVD`, mentre i `DVD` sarebbero contenuti solo nella tabella `DVDs`.

Allora, però, siccome un'interrogazione deve restituire anche le istanze dei sottotipi (a meno che non si usi `ONLY`), il DBMS la deve automaticamente riscrivere come l'unione di interrogazioni sulla supertabella e sulle varie sottotabelle. Per esempio, considerando ancora la gerarchia di tabelle raffigurata sopra, l'interrogazione

```
SELECT title, runs
FROM Movies
WHERE runs < 60;
```

viene trasformata in:

```
SELECT title, runs
FROM Movies
WHERE runs < 60
UNION
SELECT title, runs
FROM DVDs
WHERE runs < 60;
```

3.4 Confronto tra i modelli di consistenza

Il problema dell'interazione tra la gerarchia dei tipi e quella delle tabelle non ha una soluzione ideale: ciascuno di questi modelli ha dei pro e dei contro.

- Il *duplicate row model* permette di eseguire facilmente le interrogazioni, ma rende invece più complessi (e costosi, in termini di tempo) gli aggiornamenti, e introduce una notevole ridondanza dei dati (perché tutti gli attributi ereditati sono duplicati tra la supertabella e la sottotabella).
- Con il *single table model* sono semplici sia le interrogazioni che gli aggiornamenti, e non si hanno dati ridondanti, ma solitamente c'è comunque un notevole spreco di spazio, perché possono essere presenti tantissimi valori nulli.

- Lo *union model* evita sia la ridondanza dei dati che i valori nulli, e non richiede particolari accorgimenti per gli aggiornamenti, ma rende invece necessaria la riscrittura delle query, che risultano quindi meno efficienti. Questo è forse il modello più "pulito", dal punto di vista concettuale.

4 Uso delle gerarchie in pratica

Nella pratica, la possibilità di modellare direttamente nello schema del database le gerarchie (individuate in fase di progettazione concettuale) non viene molto sfruttata, per vari motivi:

- Le idiosincrasie tra il modello relazionale e quello object-oriented rendono difficile la gestione delle gerarchie di tabelle (come illustrato in precedenza), introducendo anche delle inefficienze in termini di spazio e/o tempo.
- Spesso, si ha la necessità di effettuare la migrazione a un DBMS diverso. Ciò tende a essere già problematico di per sé, anche limitandosi alle funzionalità relazionali di base, ma con l'uso delle estensioni object-oriented diventa ancora più difficile (perché ci possono essere molte differenze tra le varie implementazioni, come ad esempio modelli di consistenza diversi).