PROGETTAZIONE LOGICA

Descrizione

Descrive la situazione di una base di dati attraverso un modello **logico**. È **specifica per ogni** tipo di **DB** (SQL o noSQL).

È stato inventato da **Codd** e si basa sul concetto matematico di “**relazione**”, rappresentabili per mezzo di **tabelle**.

Prodotto cartesiano

Dati D1, D2… Dn insiemi anche non distinti, il **prodotto cartesiano** di D1 \* D2 … **Dn** è **l’insieme di tutte le n-uple ordinate** (d1, d2… **dn**) **tali che**: d1 є D1, d2 є D2 … e **dn є Dn**.

|  |  |  |
| --- | --- | --- |
| Poniamo il caso: | D1 | D2 |
| D1 = { a, b }  D2 = { x, y, z }  Prodotto cartesiano = D1 \* D2  Quindi --> | a | x |
| a | y |
| a | z |
| b | x |
| b | y |
| b | z |

**Relazione**

La **relazione** è il costrutto di base per la descrizione dei dati.

(Dati una sequenza di insiemi D1, D2, … Dn) una **relazione** R è un **sottoinsieme del prodotto cartesiano** D1 \* D2 … Dn.

(Quindi un insieme di n-uple ordinate v1, v2, … vn con vi appartenente a Di). (non è meglio d1 … ?)

Una relazione modella un’entità, mentre ogni tupla è un’istanza dell’entità stessa.

Una relazione R c D1 \* D2, esempio: (**Sottoinsieme** perché va considerata la realtà d’interesse)

|  |  |
| --- | --- |
| D1 = { a, b } | Prodotto cartesiano: { ax, ay, az, bx, by, bz } |
| D2 = { x, y, z } | Relazione: { ax, ay, az } (esempio, elementi che contengono “a”) |

In una relazione quindi:

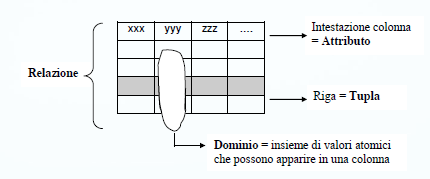
* **Non** è **definito** alcun **ordinamento** tra le **n-uple**,
* Le **n-uple** sono **distinte** tra loro (con la **PK**).

Domini (**Insiemi D** di una relazione)

Sono **l’insieme di tutti i possibili valori che un attributo** può assumere (tutti i valori delle celle di una colonna db).

* Ciascuno di essi si rifà ad un tipo di dato elementare (int, char…),
* Ciascuno di essi ha un nome, detto **attributo** che lo distingue dagli altri nella relazione (intestazione/nome colonna).

In tabella quindi: **colonne** (nomi) = **attributi**, **righe** = **tuple**.

**Dominio** == **attributo ER**.

In un **ER** con Utente (id, nome), **nome** è **l’attributo/nome del dominio** dei possibili nomi, e nella tabella potrebbero esserci (**Mario**, **Paolo**, **Gilberto**), ovvero il **dominio**: un **sottoinsieme** **di** **tutti** i **nomi** **possibili**.

Caratteristiche

Grado

Il **grado** di una relazione è il **n° di attributi che compongono quella relazione** (n° di colonne).

Cardinalità

La **cardinalità** di una relazione è il **n° di tuple** (n° di righe).

Schema

Lo **schema** (o **intensione**) di una relazione **R** (A1, A2… An) è costituito dal **nome R** e dagli **attributi Ai**, si indica come:

**<NomeRelazione>(<A1>, …, <An>)** - [Es: Giocatore (Nome, Cognome, Squadra)]

(Giocatore sarà un sottoinsieme di tutti i giocatori del mondo).

Lo **schema** è **statico**, **non cambia nel tempo** (NomeRelazione e i suoi attributi non cambiano).

Lo **schema** del **db** è **l’insieme di tutti gli schemi di relazione all’interno di esso**.

Istanza

Una **istanza** (o **estensione**) di una relazione **R** (A1, A2… An) dello schema di R (A1… An) è un **insieme di tuple** **r(R) = { t1, … tn }** **in un certo istante di tempo** (l’insieme di tutte le tuple/righe di una relazione in un certo istante).

L’**istanza** è **dinamica**, **cambia nel tempo** (i valori delle colonne possono cambiare nel tempo).

Un’occorrenza di database invece è l’insieme di tutte le istanze di tutti gli schemi di relazione (tutte le tabelle).



Rappresentazione di una relazione

Una relazione può essere rappresentata in diversi modi, tramite: **insiemi**, **elenchi** (tipo json) o **tabelle**.

Quindi una **tabella di database** è la **rappresentazione di una relazione** e **non la relazione stessa**.

Tabelle

Nelle tabelle:

* **Colonne** (**intestazione**) = **attributi**, sono **unici**, **identificano un dominio** e il loro **ordine** è **ininfluente**,
* **Righe** (**intere**) = specifiche **tuple/record** **diversi fra loro**.

Una **tabella** **rappresenta** una **relazione** **se**:

* I **valori** di ogni **colonna** sono **omogenei** fra loro (appartengono allo stesso dominio, tipo tutti int…),
* Le **righe** sono **diverse** tra loro (**univoche** considerandone tutti i valori),
* Le **intestazioni** delle colonne (**attributi**) sono **diversi** tra loro.

Inoltre in una tabella che rappresenta una relazione:

* **L’ordinamento** tra le **righe** è **irrilevante**,
* **L’ordinamento** tra le **colonne** è **irrilevante**.

Struttura basata su valori

I db relazionali sono **basati sui valori**, ovvero sul **contenuto delle singole celle**, senza puntatori o stronzate.

Valori NULL

Nei db relazionali c’è il valore **NULL**, indicante che il **dato** in quella cella **manca**, **non c’è** (!= da “” o 0), per vari motivi:

* Valore **sconosciuto**, quando **esiste un valore nel dominio ma non è noto** (Firenze non si sa indirizzo prefettura),
* Valore **inesistente**, **non esiste un valore nel dominio** (Tivoli non è provincia, non ha prefettura),
* Valore **senza informazione**, **non si sa se esiste un valore o meno** (Prato è “nuova”, non so se ce l’ha la prefettura).

I vari **DBMS non distinguono il tipo NULL** (implicitamente è il valore “no info” per tutti).

Primary Key

La **PK** è un **attributo** (**o** un **insieme** di essi) che **identifica in modo univoco una tupla** della relazione (NOT NULL).

I valori NULL (nei campi PK) impediscono di identificare le tuple e di realizzare i riferimenti ad altre relazioni.

Foreign Key

La **FK** è un **attributo** (**o** un **insieme** di essi) che **non è la PK della relazione in cui compare**, ma **di un’altra**, alla quale è legata tramite una **associazione**.

**Vincoli di integrità**

Un DBMS deve prevenire l’immissione di **info non corrette**.

* Ad uno **schema di db** si può associare un insieme di **vincoli d’integrità**,
* Un’**istanza di db** (tutte le istanze delle relazioni del db) che **soddisfa tutti** i **vincoli di** **integrità** definiti è detta **legale**.

Definizione

Insieme di **regole** che devono essere **rispettate** affinché il **db** **non contenga informazioni inconsistenti** durante le **manipolazioni** (inserimenti, modifiche, cancellazioni…).

Vincoli

I vincoli possono essere di 2 tipi:

Espliciti

Quelli che impongono **restrizioni sui valori** assumibili **dagli attributi** (V1: <Entità.Attributo> <espressione>)

Impliciti

Sono definiti **all’interno delle strutture dati** (schema logico), e sono:

* Vincoli **PK** --> **in** un’**entità** **ogni** **istanza** (riga) deve essere **univoca**, quindi i **campi** **PK** devono essere **tutti** **diversi**,
* Vincoli **referenziali** --> date 2 entità **A** e **B** aventi un’**associazione**, **non esiste** (o non è inseribile) un **elemento** in **A** che **non** sia **associato** ad un altro **elemento** in **B** (rappresentato disegnando la relazione nell’ER / associazioni totali).

Tipi

Nel modello logico ci sono **2 tipi di vincoli**:

Intrarelazionali (o interni)

**Definiti all’interno di una singola relazione**. Si suddividono a loro volta in:

* Vincoli su una singola tupla (coinvolgono **1 sola riga**)   
  Forma: Vn° (relazione) … E sono:
  + Su **valori** / di **dominio**: coinvolgono **1 solo attributo** (vincolo su valori che può assumere: V1(E) voto < 10…),
  + Su **+ attributi** (ES: V1(E): data\_start < data\_end)
* Vincoli su + tuple (sono in pratica la **PK** che impedisce tuple uguali)

Interrelazionali (o esterni)

Sono **vincoli** definiti **tra + relazioni**.

Forma: Vn° (<relazione1> … <relazioneN>): <espressione>

ES: V1(Azienda, Dipendente): SE D.livello < 7 ALLORA D.stipendio < A.MediaStipendio;

Integrità referenziale

È un insieme di **regole** che garantisce **l’integrità dei dati nelle relazioni associate tramite** una **FK**.

Quindi **ogni** valore (*not null*) della **FK** deveavereun valore di **PK** **corrispondente** nella tabella associata.

È il vincolo più importante tra i vincoli **interrelazionali**. (Ste regole rendono valide le query e vi eliminano gli errori).

Quindi:

Un **vincolo d’integrità referenziale** (**FK**) tra un attributo **A1** di una relazione **R1** (**associata/secondaria**) e un’altra relazione **R2** (**primaria**) impone ai valori di **A1** in R1 di comparire come valori **PK** in **R2**.

Quindi:

1) **Non si può mettere** un valore nella **FK** della tabella associata **se non è** una (esiste tra i valori della) **PK** della **tabella** **primaria**,

2) **È possibile** immettere **NULL** nella **FK** per dire che le **righe** **non** sono **correlate** (**associazione parziale**),

3) **Non si può eliminare** una **tupla** dalla tabella **primaria** se ha righe **legate** ad essa con la **FK** nella tabella **associata**,

4) **Non si può modificare** la **PK** nella tabella **primaria** se ad essa **corrispondono** **righe** nella tabella **associata**.

ER --> SCHEMA LOGICO

Per farlo bisogno:

* Calcolare il n° di relazioni con: n° relazioni = n° entità – n° associazioni 1:1 + n° associazioni N:N.
* ?

Cioè

* Mappare ogni entità dell’E/R …
* Tradurre ogni associazione in base a molteplicità o cardinalità.

3) per ogni relazione indicare le FK :

Dipendente (…CF\*, ID)

Con CF\* FK sulla PK CF della relazione Persona

4) se nell’E/R ci sono vincoli inter/intra espliciti, riportarli con sintassi: V1 : (<nomerelazione>) <espressione>.

5) Controllare che il numero di relazioni scritte coincida col n° trovato col punto 1.

Vedere da slide 12 gildown.

Regole di derivazione

* Entità --> Relazione | (nome dell’entità --> nome dello schema di relazione),
* Attributo (entità) --> Attributo (relazione),
* PK (entità) --> PK (relazione, cmq sottolineata),
* Associazioni (tra entità) --> si mappano in base a cardinalità/molteplicità.

Mapping delle associazioni

Le associazioni, per passare dall’ER al modello logico, devono essere mappate.

Mapping

Associazione 1:1 totale

Si derivano le 2 relazioni in una unica che contiene gli attributi della 1° + della 2° entità. Viene scelta 1 delle 2 PK.

Associazione 1:1 parziale

Nell’entità con parte dell’associazione totale è messa la PK dell’altra entità che diventa FK.

Associazione 1:N

Vengono aggiunti all’entità con parte della relazione N la FK dell’entità con parte a 1.

Associazione N:N

Si crea una nuova relazione fatta dalle PK delle 2 entità + eventuali attributi dell’associazione.

ALGEBRA RELAZIONALE

Definizioni

Interrogare un db = ottenere info desiderate estraendole da una tabella, una sottotabella o una combinazione di esse.

I linguaggi di interrogazione si basano sull’algebra relazionale.

Il risultato di una query su una relazione è un’altra relazione con grado e cardinalità dipendenti dalle operazioni.

Operazioni: unione, differenza, prodotto, proiezione, restrizione, intersezione, join.

Operazioni

Fondamentali

* Unarie (1 relazione): selezione (), proiezione ()
* Binarie (2 o + relazioni): Unione (U), differenza (-), prodotto cartesiano (X)

Derivate

* Binarie: intersezione (), join (infinito a triangoli).

Compatibilità all’unione

2 relazioni sono compatibili all’unione se:

* Hanno lo stesso grado (n° attributi),
* Ogni attributo nella stessa posizione è dello stesso tipo (stesso dominio)

R1(a11, a12, … a1N) e R2(a21, a22, … a2N) sono compatibili all’unione se, avendo lo stesso grado N, risulta anche vero che dominio(a1i) = dominio(a2i) per ogni 1 <= i <= N.

Unione

Date 2 relazioni R e S compatibili all’unione, il risultato dell’unione è:

(R unione S è l’insieme ( {} ) delle tuple t tali che ( | ) t appartiene (e) a R oppure t appartiene ad S).

Intersezione

Date 2 relazioni R e S, l’intersezione di R e S da la relazione fatta da tutte le tuple presente sia in R sia in S.

Il risultato dell’intersezione è

Grado(RintS) = grado r = grado s

Cardinalità = ?

Differenza

Date 2 relazioni R e S, la differenza di R con S è la relazione data dalla differenza insiemistica R - S.

Il risultato della è

Grado = grado r = grado s

Card() = Card r – n° tuple contenute (oltre che in R) anche in S

Mentre quello della differenza

Grado = grado r = grado s

Card() = Card S – n° tuple contenute (oltre che in S) anche in R

(ES: date le 2 relazioni Clienti(…) e Fornitori(…) visualizzare solo i clienti)

Prodotto cartesiano

Date 2 relazioni R e S, il loro prodotto cartesiano è la relazione data da R \* S.

Il prodotto cartesiano di 2 relazioni è l’insieme di tutte le possibili tuple ottenibili concatenando ogni tupla di R con ogni tupla di S.

Dalle 2 relazioni R e S, rispettivamente di grado g1 e g2 e cardinalità c1 e c2, il prodotto di R e S è la relazione di grado g1 + g2 e cardinalità c1 \* c2 le cui tuple si ottengono concatenando ogni tupla di R con ogni tupla di S.

Siano r = (a1, a2, … an) e S = (b1, b2, … bn) due tuple; la concatenazione di R e S è data da:

R conc S = (a1, a2, … an, b1, b2, … bn2)

Il risultato del prodotto cartesiano è

GILDISS

Proiezione

La proiezione di una relazione R (a1, a2, … an) di grado N su un suo sottoinsieme qualunque di attributi a1, a1, … ak e una relazione di grado K

Il risultato della proiezione è

…

Grado (rel. Risultante) = K

Card (rel. Risultante) <= card(R)

ES

CLIita(id, nome, indirizzo, citta, ordini)

CLIest(id, nome, indirizzo, citta, nazione, ordini)

1) Get nome, indirizzo, citta clienti ita

--> S = (CLIita)

--> SELECT nome, indirizzo, citta FROM CLIita;

2) Get nome, indirizzo, citta tutti clienti

--> S = (CLIita), S1 = (CLIest) --> S U S1

ES2

C08(id, nome, citta, ordini)

C09(“,”,”,”)

1) Nome all cli

--> (C08) U (C09)

2) Nome cli sia in 08 sia in 09

--> (C08) (C08)

Restrizione (sigma)

Data una relazione R (a11, a12, … a1n) e una condizione C, espressa sugli attributi di R, la restrizione di R in base a C è una elazione contenente tutte e solo le uple che soddisfano la condizione data.

Il risultato della restrizione è:

ES: Studente(id, nome, cognome, classe, voto) --> T = (sigma)cognome=”rossi”(studente) --> R = (pi)voto(T)

Grado(rel ris) = grado(orig)

Card(rel ris) <= card(orig)

ES3

View nome e cognome studenti 5DI con voto > 7

T = (sigma)classe=”5DI”ANDvoto>7(studente) | R = (pi)nome,cognome(T)

L = LIBRI (id, autore, titolo)

U = UTENTI (id, nome, cognome)

P = PRESTITI (fkUtente, fkLibro, data)

a) Titoli dei libri presi in prestito il giorno 13/7/2011

a = ()

b) Autori dei libri presi in prestito da Paolo Bianchi

b = ()

c) Codici degli utenti che hanno preso in prestito libri scritti da Camilleri oppure da De Luca

c = ()

d) Titoli dei libri presi in prestito il giorno 13/7/2011 da Paolo Bianchi

d = ()

e) Trovare gli utenti che hanno preso in prestito solo libri di Camilleri

()

e = ()

f) visualizzare gli titolo e autore dei libri presi in prestito nel 2011

f = ()

g) titolo dei libri di Camilleri e di Leopardi presi in prestito da Rossi Mario

()

e = ()