

DEFINIZIONE DI "BASI DI DATI"

Collezione di dati omogenei e strutturati, memorizzati in maniera permanente

- Strutturati = modello di collezione di dati ben definito

DBMS = DataBase Management System

Un DBMS è un sistema per la gestione di una base di dati. I suoi 3 aspetti fondamentali sono:

- Definizione delle strutture = il DBMS deve mettere a disposizione un linguaggio per le definizioni dei dati. - DDL: data definition language
METADATI = definizioni delle strutture dei dati
- Linguaggio per la manipolazione dei dati / DML = data manipulation language

Operazioni fondamentali:

- Insert
- Delete
- Update

N.b. Si possono fare anche sui metadati.

- Linguaggio per il recupero dei dati / DQL = Data Query language. Il dato manifesta la sua esistenza solo se è possibile recuperarlo

Altri aspetti riportati:

- Sicurezza = vi è la necessità di un filtro per l'accesso sui dati
- Performance = soprattutto nell'interrogazione dei dati. Ottimizzazione Query
 - Di solito il Data Query language è di tipo didattico = dice al sistema che tipo di dati vogli, ma non dice alle macchine come fare a recuperarli. È l'ottimizzazione dei dati che si occupa del lavoro "sotterraneo"
- Recovery da fuoco
- Gestione dell'accesso condiviso alle basi di dati. Devo garantire un'accesso ordinario alle informazioni qualora vengano modificate
- Gestione delle transazioni = una sequenza di operazioni elementari sui dati che deve essere fornita in maniera atomica e indirizzabile (o ormai alla fine con successo, o comunque tutto). Es: Montiamo su schede di un auto bancaria ad un altro

all'utente viene mostrato solo il risultato finale. (le operazioni avvengono in isolamento)

Tipologie di DBMS offerto nel corso:

- Relazionali (Modello di confronto di dati, avendo il modello dei dati)
- Transazionali (basi di dati i cui dati cambiano frequentemente)

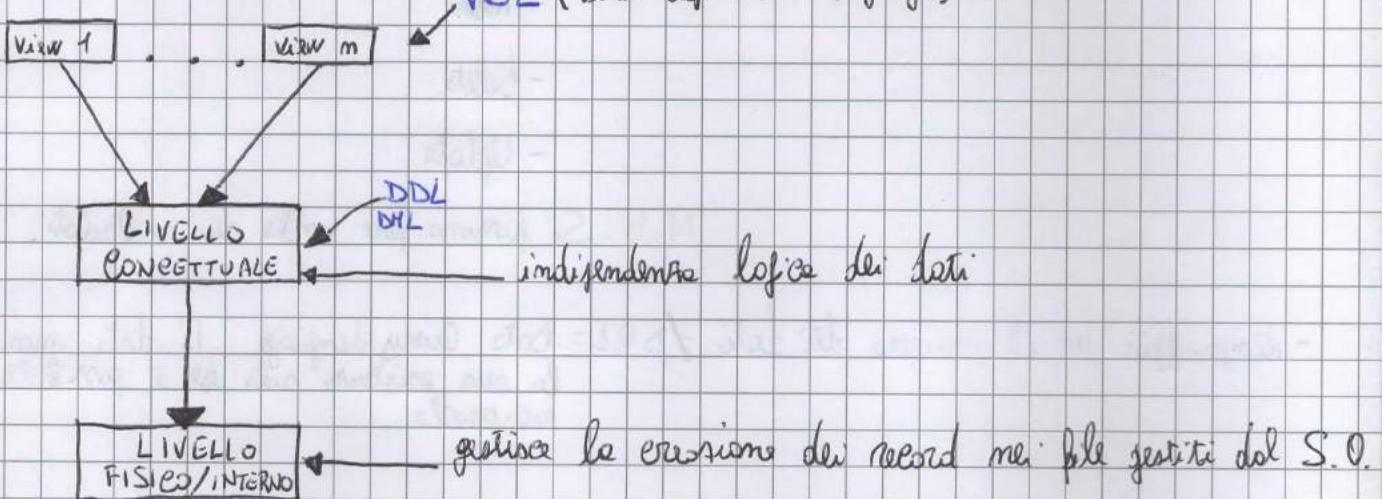
N.b. Esistono DBMS non relazionali e non transazionali.

Un DBMS può essere diviso in livelli d'accesso

LIVELLO ESTERNO VIEW 1 . . . VIEW m (Visione parziale del contenuto) del database

Un DBMS è un blocco unico che può assumere diverse forme a seconda dei privilegi degli utenti

VDL (View definition language)



• Se si vorrà combinare il livello concettuale nella descrizione dei dati, esso non andrà ad impattare necessariamente il livello esterno

ESEMPIO

STUDENTE (Matr, Nome, Cognome, Esame, Data, voto)

infanzia

schema

STUDENTE (Matr, Nome, Cognome)

ESAMI (Esame, Data, voto)

} è una struttura completamente diversa dalla prima
ma può essere "visualizzata" tramite una vista esattamente
uguale al precedente.

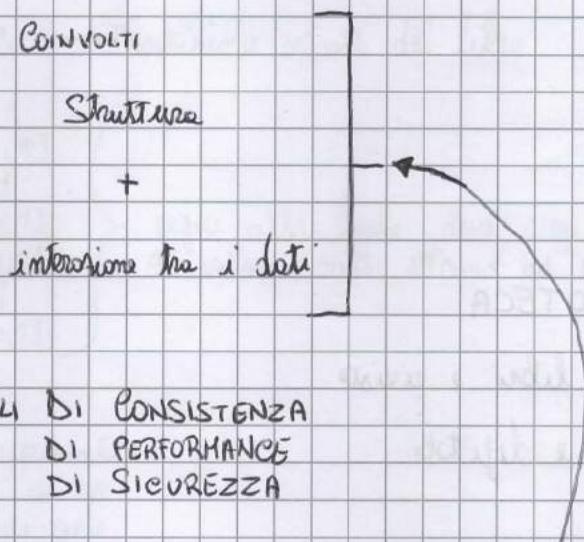
- **INDIPENDENZA FISICA** = le strutture dei dati sono mappate su strutture fisiche, questo garantisce la totale immunità del livello concettuale se sono effettuate modifiche sui file in modo fisico.

→ PROGETTAZIONE CONCETTUALE

- DEL DB

→ FUNZIONALITÀ RICHIESTE

→ DATI COINVOLTI



→ VINCOLI DI CONSISTENZA
DI PERFORMANCE
DI SICUREZZA

INPUT

- descrizione testuale

OUTPUT

- class diagram

- dizionario delle classi

- dizionario delle associazioni

- dizionario dei vincoli (maniera sistematica di rappresentare i vincoli)

→ PROGETTAZIONE LOGICA (Indipendente dal DBMS)

- Modello relazionale dei dati
- Revisione del Class Diagram
- Codifica relazionale del class diagram ristrutturato

DDL DI SQL

→ DEFINIZIONE BD

- Manipolazione dei dati
- Interoperazione dei dati

ESERCIZIO

BIBLIOTECA

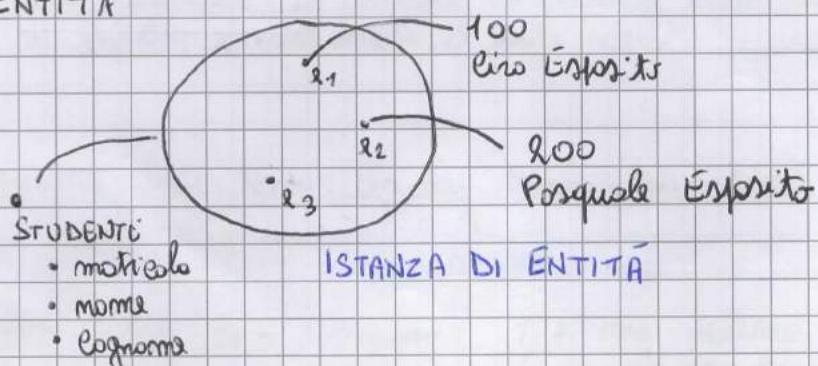
- 1) Gestione dei descrittori bibliografici libri e riviste
- 2) Gestire possedimenti libri, elettori e digitali
- 3) Collocazioni fisiche dei possedimenti
- 4) Profilarione dell'utente con diritti di prestito
- 5) Gestione prestiti, solleiti, danneggiamenti e furti
- 6) Visualizzazioni digitali
- 7) Registrazione degli utenti

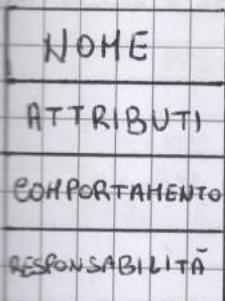
- Cosa vedere?

- I concetti domici

- ENTITÀ è un concetto descrivibile mediante un insieme di attributi e di associazioni ad altre entità

ENTITÀ





← caratteristiche di un'entità

COMPORTAMENTO

← Porte testuale contenente dei dizionari

CLASSE CHE CODIFICA IL PUNTO 1

DESCRITTORE P.L → Descrizione astratta del libro

Titolo	1
Autore	[1, *]
ISBN	1
Cosa Edit	[0, 1]
Edizione	[0, 1]
Anno	1
Numb Pag	[0, 1]

} ogni attributo deve avere una multiplicità che viene scritta affianco ad esso.

Una classe rappresenta una astrazione di un oggetto in questo caso un libro.

MOLTEPLICITÀ

[0, 1] → ATTR. PARZIALE
obbligatorio, max 1

[1, 1] → ATTR. TOTALE
obbligatorio, max 1

[0, *] → multipli, non obbligatorio
[*]

[1, *] → multipli, obbligatorio

Un attributo di una classe, deve essere atomico, ovvero non divisibile. In generale la forma è ~~attingibilmente~~. Nel caso di Autore dipende dalla sua rilevanza. Se è necessario scrivere l'autore in Nome e Cognome, allora si perdebbile l'atomicità [min, max]

CODIFICA PUNTO 2/3

LIBRO

→ Descrizione fisica del libro

Colo - o. Desc.	: STRING
Collaborazione	: STRING
Condizioni	: T- cond
Stato	: T- Note
Precio	: FLOAT

} affianco ad ogni attributo, va dichiarato il tipo

Vedere minimo:

- 0 = l'attributo è facoltativo

- 1 = l'attributo è obbligatorio

Sai che gli attributi **Note** e **Condizioni** sono già noti e in numero finito, possiamo definire una enumerazione da "assegnare" all'attributo come tipo di dato

ENUMERAZIONE

In sintesi, l'enumerazione è l'insieme dei valori possibili di un attributo.
È un vinoce di tipi

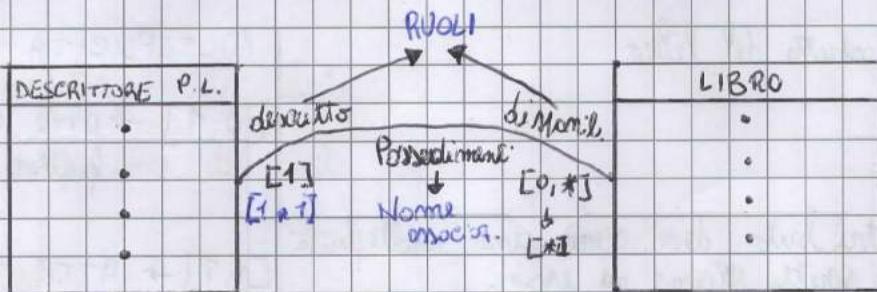
L'enumerazione
T - Note
Disponibile
Pronto
Smorrito

L'enumerazione
T - cond
Intefro
Danneggiato

Nella costruzione di un class diagram, e in generale di un DB, non deve trovarsi solo le entità (classi), ma anche le associazioni fra esse.

CARATTERISTICHE ASSOCIAZIONI

- Nome
- Anelli
- Cardinalità
- Grado dell'associazione = numero di entità coinvolte



Nome associazione = Possedimento

Multiplicità associazione = OBBLIGATORIA = descritto [1] \Rightarrow disponib. [0..*]

\Rightarrow Cardinalità associazione = uno o molti

Possibili cardinalità:

- uno a molti
- uno a uno
- molti a molti

EDIFICIA PUNTO 4 / PROFILAZIONE

UTENTE		PROFILO	
Nome	1	Tipi	
Cognome	1	Max - pref.	
C.F.	1	Ha - prof.	Durata - max
Residenza	1	Professione [1]	
Telefono	[0..1]		
E-mail	[0..1]		

CODIFICA PUNTO 5

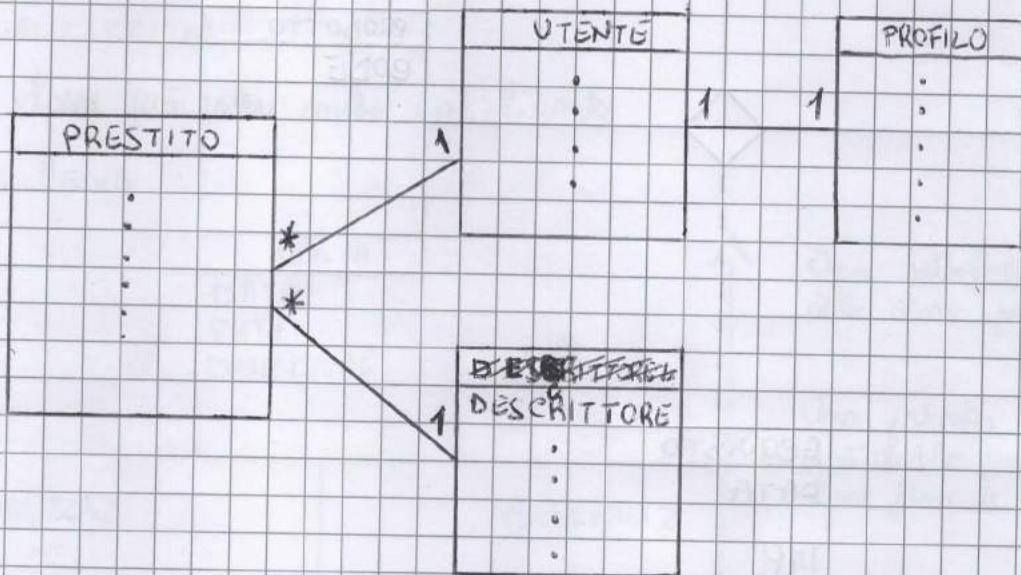
PRESTITO
Data_iniz : DATE
Data_fine : DATE
Stato : t - state2

DATE è un tipo di dato specifico per le date. È una stringa di 10 caratteri "yyyy-mm-dd"

Esempio: "2014-10-03"

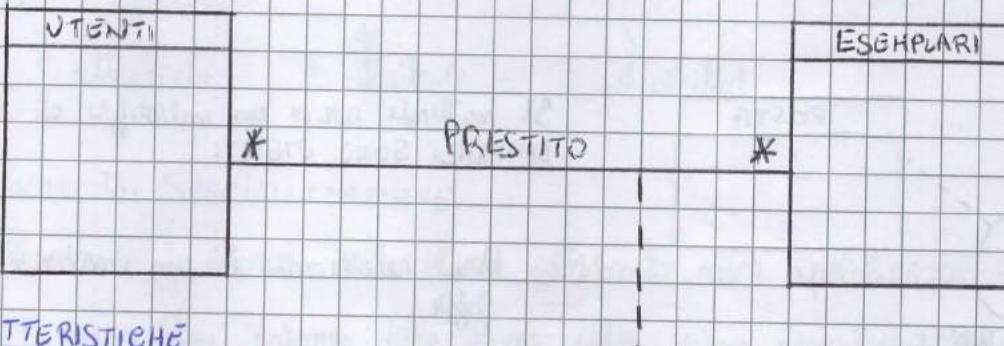
La classe prestito può essere implementata in 2 modi

MODO 1



MODO 2

Il modo 2 prevede l'implementazione delle classi prestito come classi d'associazione

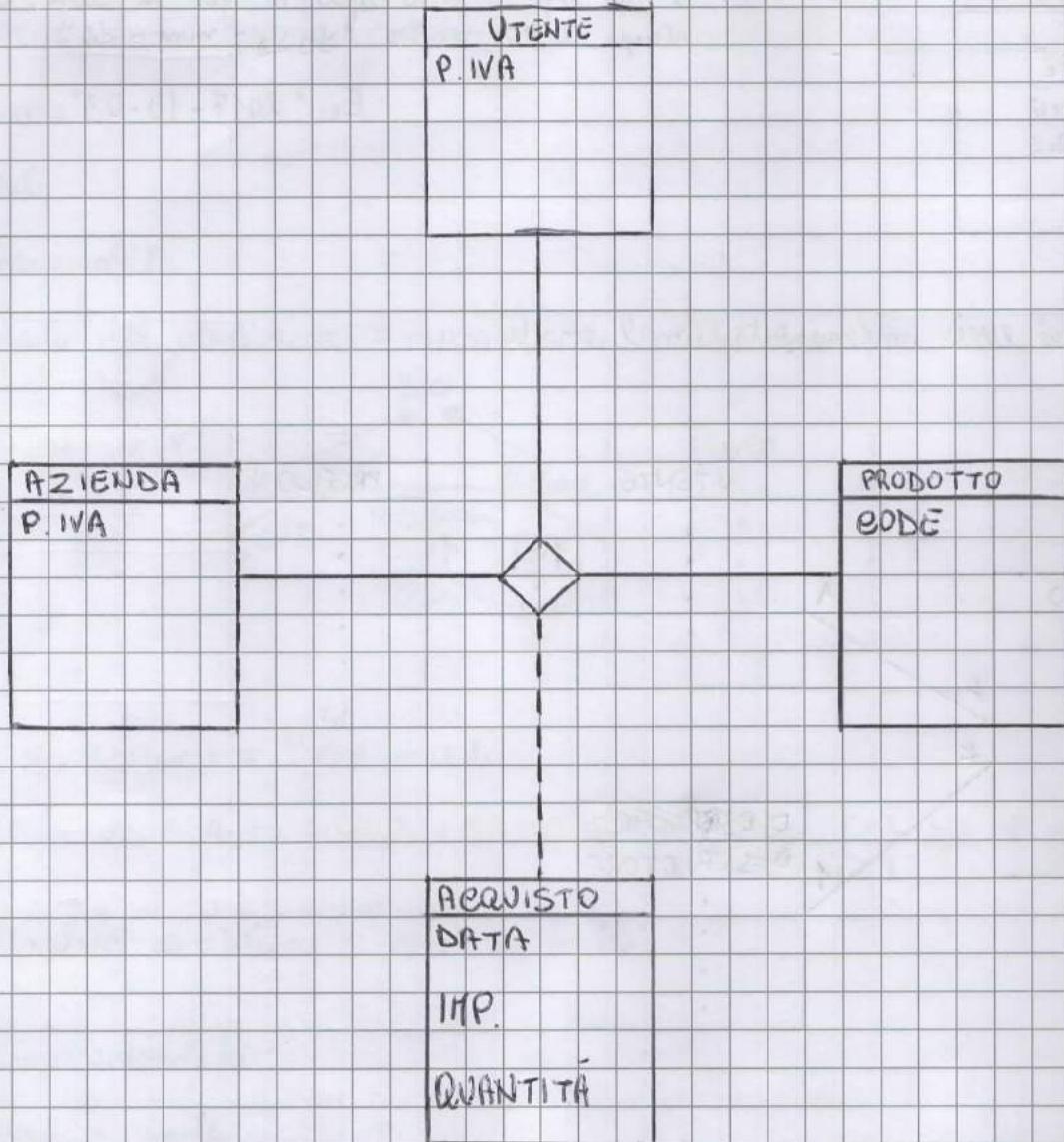


CARATTERISTICHE

- Nome classe classe = nome classe.
- Hanno senso solo per classi molti a molti
-

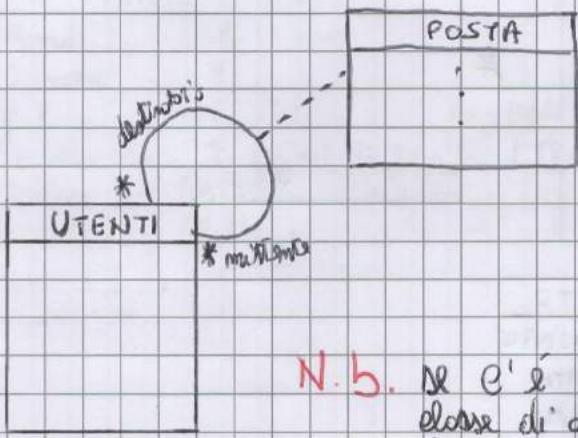
PRESTITO
DATA_INIZIO: DATE
DATA_FINE: DATE
STATO: t - state2

ESEMPIO DI CLASSE D'ASSOCIAZIONE A 3



CLASSE DI ASSOCIAZIONE RICORSIVA

Una classe di associazione può essere anche ricorsiva.

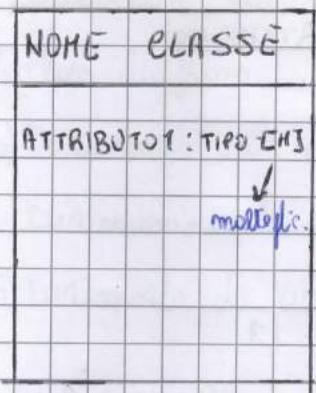


Il mittente scrive un messaggio al destinatario.
ENTRAMBI SONO UTENTI

N.b. Una relazione è un insieme di coppie

N.b. Se c'è l'esigenza di collegare una classe ad una classe di associazione, allora quest'ultima non è una classe di associazione

SCHEMA CLASSE / IN SINTESI



CARATTERISTICHE

- Ogni attributo ha un nome, un tipo e una molteplicità.

SPECIALIZZAZIONI

Una classe può essere anche specializzata.

CLASS MEDIA

MEDIA
DATA
PATH
DIMENSIONE

Ogni sottoclasse eredita gli attributi della classe padre.

SPECIALIZZAZ.

GENERALIZ.

V	VIDEO	FOTO	TESTO
	DURATA	FORMATO	FORMATO
	CODEC	RISOLUZIO.	
	FORMATO		
	RISOLUZIONE		

4 attributi

5 attributi

4 attributi

Il distinzione di una sottoclasse non può fare sì per studenti che per magistri.

TOPOLOGIA DI SPECIALIZZAZIONE

- PARTIAL = l'insieme delle classi può anche non specializzarsi e restare generale
- TOTAL = Ogni insieme delle classi padre viene specializzato sempre (ogni elemento del sette specializza)

VINCOLO DI CONSISTENZA

- OVERLAPPING = un elemento può essere specializzato in più classi di specializzazione
- DISJOINT = un elemento può essere specializzato in una sola classe di specializzazione

AGGREGAZIONE / COMPOSIZIONE



COMPOSTO DI

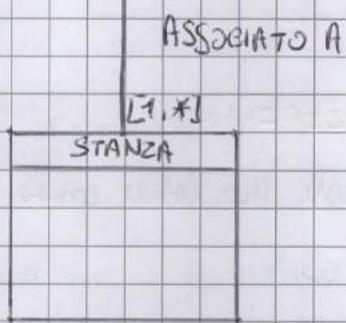


VINCOLI

CONCETTUALMENTE
UGUALE A



1



- Non può appartenere a più di un appartamento (è scritto 1 delle relazioni)
- Una composizione non può vivere senza la classe (comtemtitore)

NON VA USATA A CAZZO!

Un elemento può appartenere a diversi comtemtitori



Questo esempio è una aggregazione poiché le corse sono formate da studenti, ma senza le corse gli studenti esistono comunque



Il class diagram è solo uno dei documenti da produrre per la realizzazione concettuale

1) Descrizione testuale

2) Class diagram

3) Dizionario delle classi

4) Dizionario delle associazioni

5) Dizionario dei vincoli

N.b. devono essere presenti tutti e 4 i documenti (2-5)

Nome classe	Descrizione della classe con testo libero	Attributi	Descrizione di ciascun attributo

Nome associazione	Descrizione generale	Classi coinvolte con i loro ruoli e cardinalità	Descrizione

ESEMPIO

CLASSE

MEDIA
DATA_PATH
DIMENSIONE
DATA_POST
*
HA
PUBBLICAZIONE
1 IN
POST
.
.
.

Dizionario delle classi (MEDIA)

Nome classe	Descrizione	Attributi	Descrizione attributi
MEDIA	Describe il contenuto multi-pubblicato in un post	DATA_PATH DIMENSIONE DATA_POST	indirizzo relativo al docum nel filesystem. Dimensione in Byte del docum. Data pubblicazione post Kilo = kilometer

DIZIONARIO DELLE ASSOCIAZIONI

Nome associazione	Descrizione generale	Classi coinvolte con i loro ruoli e cardinalità	Descrizione
Pubblicazione	posto a media pubblicazione post	MEDIA, HA *	media contiene post
		POST, IN 1	post contiene il media

DIZIONARIO DEI VINCOLI

È una restrizione che i dati di un database deve rispettare per potersi dire correttamente

STUDENTE
EF.
MATRICOLA
NOME
COGNOME

ESAHE
TITOLO
VOTO
LODE
DATA

- IN STUDENTE

- Ci deve essere corrispondenza tra EF e dati omografici

Quella unità è di tipo antro - relazionale poiché resto all'interno delle stesse domande

- IN ESAME

vincolo di dominio

~~vincolo di mple~~ - da lode può assumere solo valori S/N. Se la lode è S allora il voto è 30

~~inter-relazionale~~

- il titolo dell'esame deve essere ~~titolo~~ ^{presente} tra gli insegnamenti attivi

~~vincolo~~
di dominio

- Il voto è compreso tra 0 e 30

- Uno studente non può avere più di un voto di uno stesso esame superato

FASE DI PROGETTAZIONE LOGICA

PROGETTAZIONE CONCETTUALE

MODELLO DI DATI RELAZIONALE (non è l'unica)

PROGETTAZIONE LOGICA

Modello relazionale

- Il concetto è quello delle relazioni (nel senso matematico)

SCHEMA RELAZIONALE

È definito da:

- Un nome / R
- Una sequenza di nomi di attributi / A₁, ..., A_m
- Per ogni attributo un dominio dei valori / dom(A_i)

ESEMPIO

DATA (YYYY, MM, DD)

dom(YYYY) = {0, ..., 2500}

dom(MM) = {1, ..., 12} N.b. potremmo anche mettere i nomi

dom(DD) = {1, ..., 31}

Uno schema relazionale è simile ad una delle classi diagram

Dato uno schema relazionale

$R(A_1, \dots, A_m)$ con dominio $\text{dom}(A_i)$

Una relazione R è un sottoinsieme qualunque del prodotto cartesiano

RELAZIONE PIÙ GRANDE = $R \subseteq \text{dom}(A_i)$

$r_1 = (2017, 10, 12)$ è una relazione contenente solo quell'elemento
 $(2017, 10, 12)$ è un elemento

RELAZIONE PIÙ PICCOLA = $R = \emptyset$

DATA

yyyy	mm	dd
2017	10	12
2017	09	31
2017	02	31

relazione = una riga

(riga)

\rightarrow finché non introduce vincoli di consistenza, questa tabella va bene.

La cardinalità della relazione è il prodotto tra i domini

1 - Una relazione è UN INSIEME (gli insiemi non hanno molte proprietà degli elementi) (doppioni)
(le relazioni non sono ordinate)

2 - I valori dei domini non hanno struttura

PERSONA (NOME, Residenza)

{(Ciro Esposito, Via Roma 15)}

È una struttura
vole come nome
(non posso scrivere
un record di record)
ogni valore ha
struttura

- Per ogni dominio esiste un valore speciale = NULL / un valore niente, vi è anche del valore null la parola

ESEMPIO

gggg	HH	DD
2017	09	NULL

possibile fatto, perché non conoscere il giorno

IDENTIFICATIVI = Superchiave

$$R(A_1, \dots, A_m)$$

Superchiave = un sottoinsieme (B_1, \dots, B_k) degli attributi (A_1, \dots, A_m) tale che associaendo i valori B_1, \dots, B_k identifica un valore univoco agli elementi in ogni possibile relazione.

N.b. Per ogni relazione non esistono due elementi distinti della relazione che abbiano su B_1, \dots, B_k gli stessi valori

STUDENTE (CF, MATR, COGNOME, NOME, DATA)

- Superchiave $\{CF, MATR\}$

* $\{CF\}$ }
* $\{MATR\}$ } MINIMALI

Qualsiasi insieme di attributi che ~~contenga~~ ^{contenga} matricola o CF è una superchiave

N.b. Esiste sempre una superchiave, nel peggio dei casi è l'insieme A_1, \dots, A_m di tutti gli attributi

Una superchiave permette di individuare un elemento in maniera univoca.

CHIAVE CANDIDATA

^(super)
È una chiave minimale

Una superchiave $\tilde{\tau}$ minimale se $\forall 1 \leq i \leq k, \{B_1, \dots, B_k\} \setminus \{B_{i+1}\}$ non è più una superchiave

Una chiave primaria è una chiave candidata scelta come rappresentativo
^{(idem), colto}

N.b. Una chiave primaria è una superchiave minima

ESEMPIO

ESAMI (CF, TITOLO, DATA, VOTO, LODE)

CHIAVE CANDIDATA

$\{TITOLO, CF\}$

CHIAVE ESTERNA

Sottoinsieme di attributi A_1, \dots, A_m che è possibile mettere in corrispondenza biunivoca con una chiave primaria in un altro schema (o nello stesso)

N.B. SERVE A CODIFICARE LE ASSOCIAZIONI

DA CLASS DIAGRAM A SCHEMA RELAZIONALE

CLASS DIAGRAM



COLLEZIONI DI SCHÉMI
RELAZIONALI

ESEMPIO

AUTORE	SCRITTURA	LIBRO
ORCID	scritto da	ISBN
Nome	1-*	titolo
Cognome	1-*	scris. Cosa-ed Amm.

Pk = ORCID

Pk = ISBN

AUTORE (ORCID, Nome, Cognome) LIBRO (ISBN, Titolo, Cosa-ed Amm.) SCHEMI RELAZIONALI

SCRITTURA (ORCID, ISBN) →



Cosa generale

nuova relazione

relazionale

(molti & molti)

PROGETTAZIONE
LOGICA

Lo schema relazionale serve anche a caratterizzare le associazioni

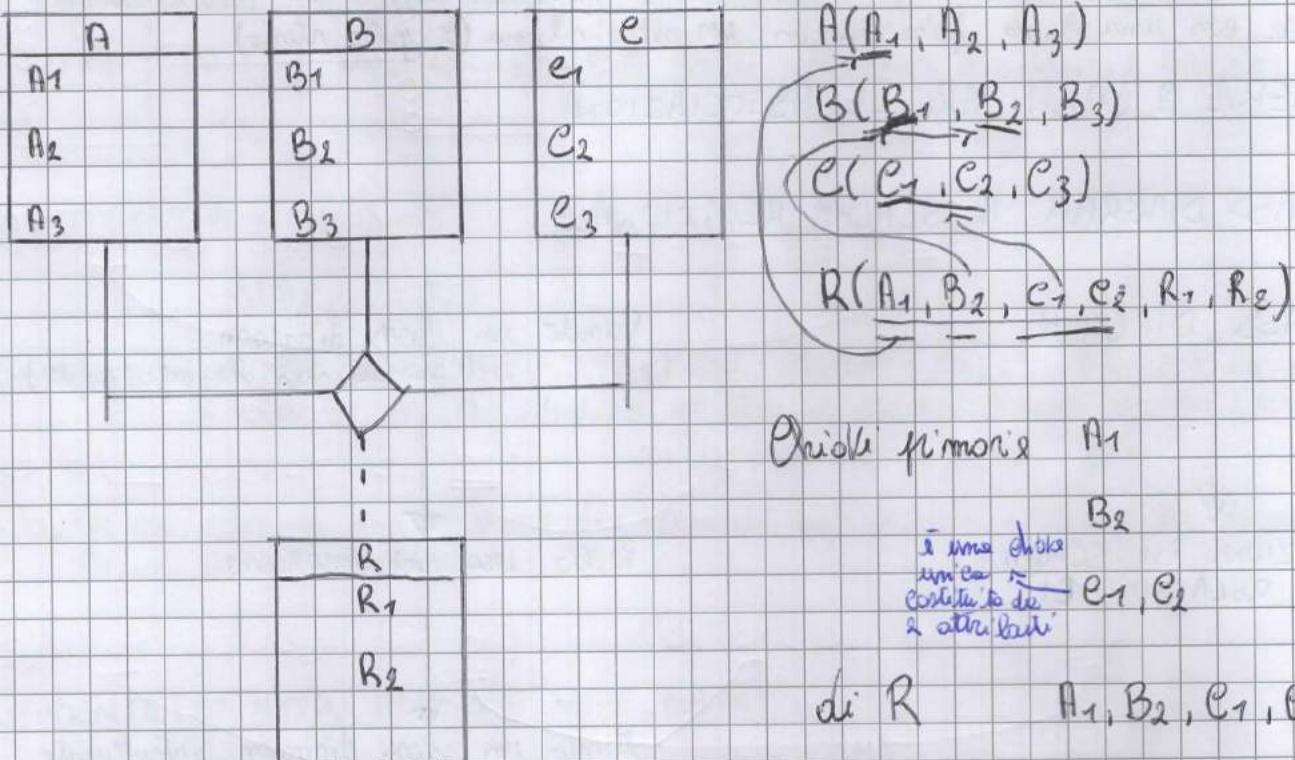
N.B. Se tutta la classe "funzione" de elice, il valore delle elice perde di utilità
le chiavi esterne possono stare in corrispondenza SOLO con **una** chiave primaria

Le 2 funzioni principali delle chiavi primarie

1- Imporre un vincolo di unicità (vincolo identificativo)

2- Usate per costruire le associazioni

La chiesa primaria va individuata nello spazio del paese



ESODIFICA ASSOCIAZIONI 1 - a - MOLTI

Corrispondenza tra P_K e F_K

- Stesso numero di attributi
 - Gli attributi in corrispondenza debbono avere lo stesso dominio
 - Non è obbligatorio che abbiano lo stesso nome

N.b. In alcuni casi si può evitare di introdurre un nuovo schema relazionale per una associazione (uno a uno, uno e molti)

ENTITÀ

FORTE ↵

STUDENTE
EF
MATRICOLA
NOME
EGONOHÉ

ESAMI
TUTTO
DATA
VOTO
LODE

Uno studente fa molti esami.
Uno esame è fatto da un
studente.

ENTITÀ DEBOLE (anche utilizzando solo i suoi attributi, non sono identificare una chiave)

ENTITÁ DEBOLE

- Per trarre una classe bisogna considerare anche i ruoli oltre gli attributi.

$P_K(D)$, TITOLO

STUDENTE (CF, MATRICOLA, NOME, COGNOME)

PK

ESAME (TITOLO, DATA, VOTO, LODE)

← non permette di avere
un identificativo

Se si desidera impostare un vincolo di
unicità, si deve individuare una chiave
primaria.

ESAME (TITOLO, DATA, VOTO, LODE, STUDENTE)

FK

in questo caso studente
si riferisce alla chiave primaria
EF

da chiave esterna va messa nella classe che partecipa con "1"

ESEMPIO

A	B
A ₁	*
A ₂	*
A ₃	*
A ₄	*

1

- * B₁
- * B₂
- B₃
- B₄

A(A₁, A₂, A₃, A₄)

B(B₁, B₂, B₃, B₄, A₁, A₂)

FK

Non è necessario introdurre uno schema relazionale per l'associazione

QUINDI, IN GENERALE

Per le associazioni 1-a molti, introdurre la chiave esterna nello schema per la classe che
partecipa con 1

N.b.

A	B
A ₁	*
A ₂	*
A ₃	*
A ₄	*

A(A₁, A₂, A₃, A₄, B₁)

B(B₁, B₂, B₃, B₄, A₁)

É SBAGLIATA

Non codifica una associazione
molti a molti

CODIFICA ASSOCIAZIONI 1-a-1

Ci possono essere fondamentalmente 3 casi:

A	B
a_1	b_1
a_2	b_2
a_3	b_3
a_4	b_4

1) $A(\underline{a_1}, a_2, a_3, a_4)$ $B(b_1, \underline{b_2}, b_3, b_4, \underline{b_1})$

2) $A(\underline{a_1}, a_2, a_3, a_4, \underline{\underline{b_1}})$ $B(\underline{b_1}, b_2, b_3, b_4)$

3) $A(\underline{a_1}, a_2, a_3, a_4, \underline{\underline{b_1}})$ $B(\underline{b_1}, b_2, b_3, b_4, \underline{\underline{a_1}})$

~~gomecio de A o B~~
~~gomecio de B o A~~

ESEMPIO

A		B
a_1		b_1
a_2	$0..1$	b_2
a_3		b_3
a_4		b_4

In questo caso la migliore da usare è la 2

$A(\underline{a_1}, a_2, a_3, a_4, \underline{\underline{b_1}})$ $B(\underline{b_1}, b_2, b_3, b_4)$

poiché si ha sempre corrispondenza in B, quindi
l'insieme A ha sempre corrispondente

N.b. Nel caso in cui la cardinalità dell'associazione si invertisse, si userebbe il mette

$1 / 1 \quad \underline{0..1}$

da giunzione si estende a livello computazionale / giunzione = struttura dati generale, l'assoc
è rappresentata da una relazione
relazionale.

ATTRIBUTO STRUTTURATO

STUDENTE
Motr
EF
Nome
Cognome
Residenza

*

STRUTTURATO

- Via
- EPP
- N° civico
- Città

3 possibilità di ristrutturazione

1- Se non ho bisogno di accedere ai campi della struttura → "eliminare le strutture" / Codificare la struttura →
 → Es: una strip di caratteri.

STUDENTE
Motr
EF
Nome
Cognome
Via
EPP
N° civico
Città

3- Emplazzando l'attributo residenza

RISIDENZA
Via
EPP
N° civico
Città

REVISIONE CLASS DIAGRAM

- Eliminazione attributi strutturati
- Eliminazione attributi con valori multipli
- Eliminazione gerarchie

ELIMINAZIONE ATTRIBUTI CON VALORI MULTIPLI

1- Molti attributi codificata in una strippe / Es: una strippe con i nomi separati da un ; Atenei; eri; ecc

PRO = semplicità

CONTRO = non è direttamente supportata la ricerca diretta "per autore"

- Molta dipendenza dalla codifica e possibili errori

2- Classe esterna + associazione

LIBRO	AUTORE
ISBN	*
Titolo	* Nome Cognome

3 - Duplica i campi in negozio delle moltiplicità

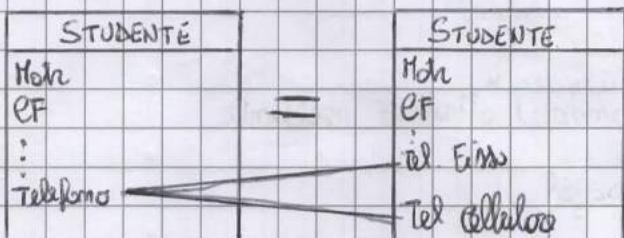
PROBLEMI = 1) Se gli autori sono ≥ 6 , risulta incompleto

	LIBRO
ISBN	
TITOLO	
AUTORE1	
⋮	
AUTORE5	

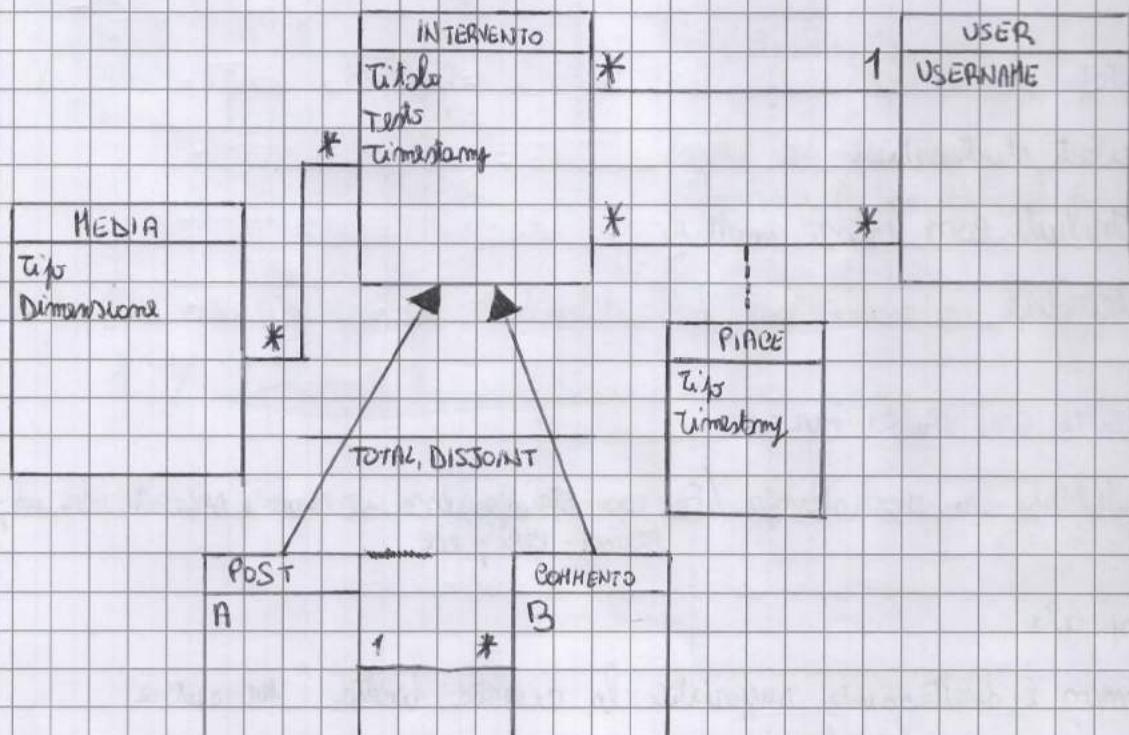
2) Potrebbero essere molte latenze innestate. Il problema non è lo spazio, ma il trasferimento da memoria secondaria a memoria centrale. Se trasportasse un libro vuoto, dura poco tempo.
Per Es: molti libri con autri < 5

3) Per le ricerche per autore, devono essere in tutti i campi. (B&B)

Situazione in cui il punto 3 è ottimale



DISTRUZIONE DELLE GERARCHIE

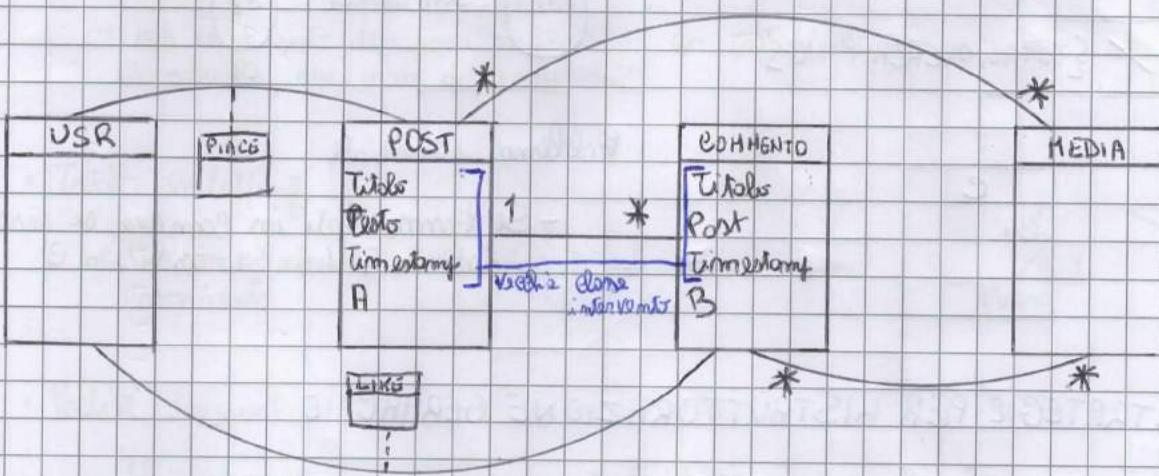


OBIETTIVO = produrre un altro diagramma equivalente = stessa implementazione / stessa funzionalità.

Pi sono 3 metodi:

METODO 1

• Scrivere la gerarchia sulle specializzate / SOLO SE TOTALE



VANTAGGIO

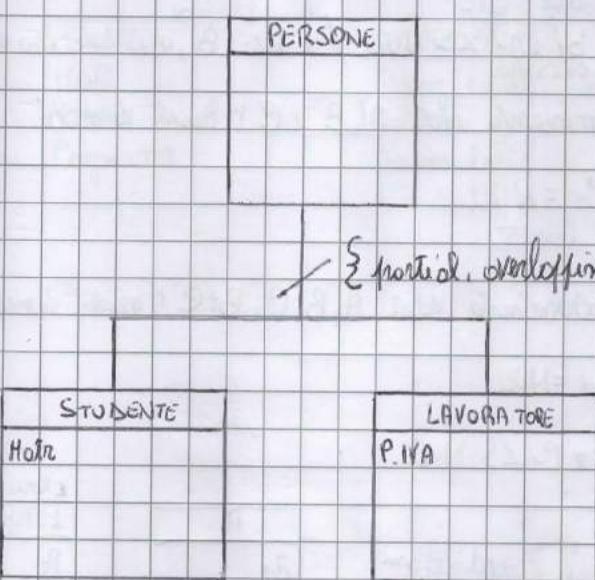
- È vantaggioso per le associazioni sulle classi specializzate

SVANTAGGIO

- Per associazioni su classi generali

N.b. in un class-diagram, due classi di associazione non possono avere lo stesso nome

METODO 1 NEI CASI NON TOTAL/DISJOINT



✗ {partial, overlapping?}

SCIACCIAMENTO



NON VA BENE

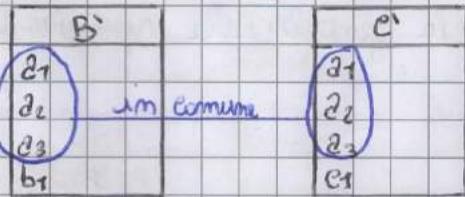
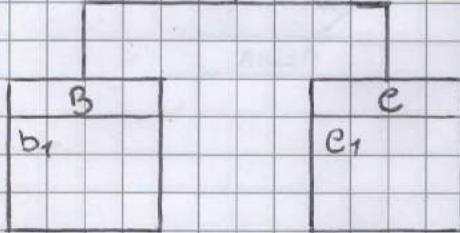
Problema 1

- lo sciacciamento non va bene poiché se non ha specializzazioni non saprà dove andare / perdite di informazioni

SCHIAVIAMENTO

A
a ₁
a ₂
a ₃

+ {TOTAL, OVERLAPPING}



Problema potenziale

- Se hanno dati in comune, si modifichino i dati in B' solo per chi sia in C'

SINTESI STRATEGIE PER RISTRUTTURAZIONE GERARCHIE

Schiaviamento generale nelle specifiche

- Total, disjoint OK.
- Total, overlapping Problema potenziale = VINCOLI / gestione delle gerarchie dei duplicati
- Partial, disjoint NO!
- Partial, overlapping NO!

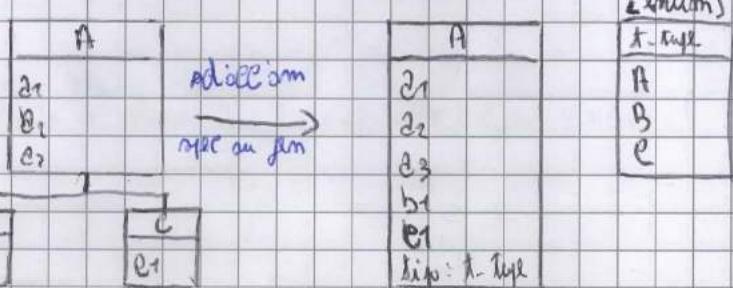
- Schiaviamento delle specifiche nelle generali / valore discriminante = enum. nelle classi

- Total, disjoint = attributi discriminanti valori B e C
- Total, overlapping = attributi del discriminante di valori: B, C, B+C + nuovi vincoli
 - Nuovi vincoli = Se tipo = B+C, b₁ & c₁ > NULL / Se tipo = B, c₁ = NULL & b₁ <= NULL
Se tipo = C, b₁ = NULL & c₁ > NULL
- Partial disjoint = attributi del discriminante valori: A, B & C + nuovi vincoli
- Nuovi vincoli = Se tipo = A, b₁ & c₁ = NULL
Se tipo = B+C, b₁ & c₁ > NULL
- Partial, overlapping = attributi del discriminante valori: A, B, C, B+C + nuovi vincoli
 - Nuovi vincoli = Se tipo = A, b₁ & c₁ = NULL

Se tipo = B+C, b₁ & c₁ > NULL

Vincoli base

- Se tipo = B, b₁ > NULL & c₁ = NULL
- Se tipo = C, c₁ > NULL & b₁ = NULL



- Non rimuovere classi, rimuovere le associazioni e trasformarle in associazioni

N.b. non crea duplicazioni:

- Total, disjoint / bisogna aggiungere dei vincoli

- Un intervento deve essere associato ad un post o ad un commento, ma non ad entrambi

- Total, overlapping

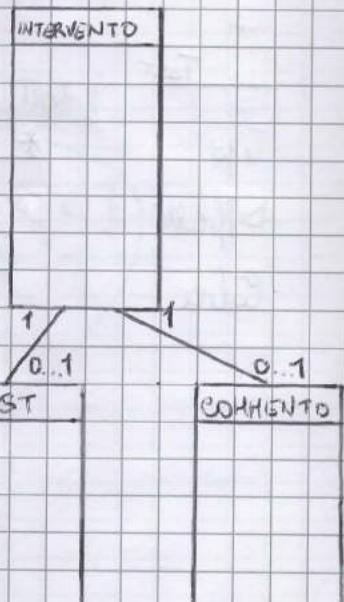
- Un intervento deve essere associato ad un post o un commento

- Partial, disjoint

- Un intervento non può essere contemporaneamente commento e post

- Partial, overlapping

- NESSUN VINCOLO



REVISIONE / EXTRA

- Introduzione di chiavi sintetiche = quando faccio riferimento all'oggetto

- Esame delle associazioni 1 a 1

STUDENTI
Motr.
Nome
Cognome

CURRICULUM
Num. esami
media
num. lodi

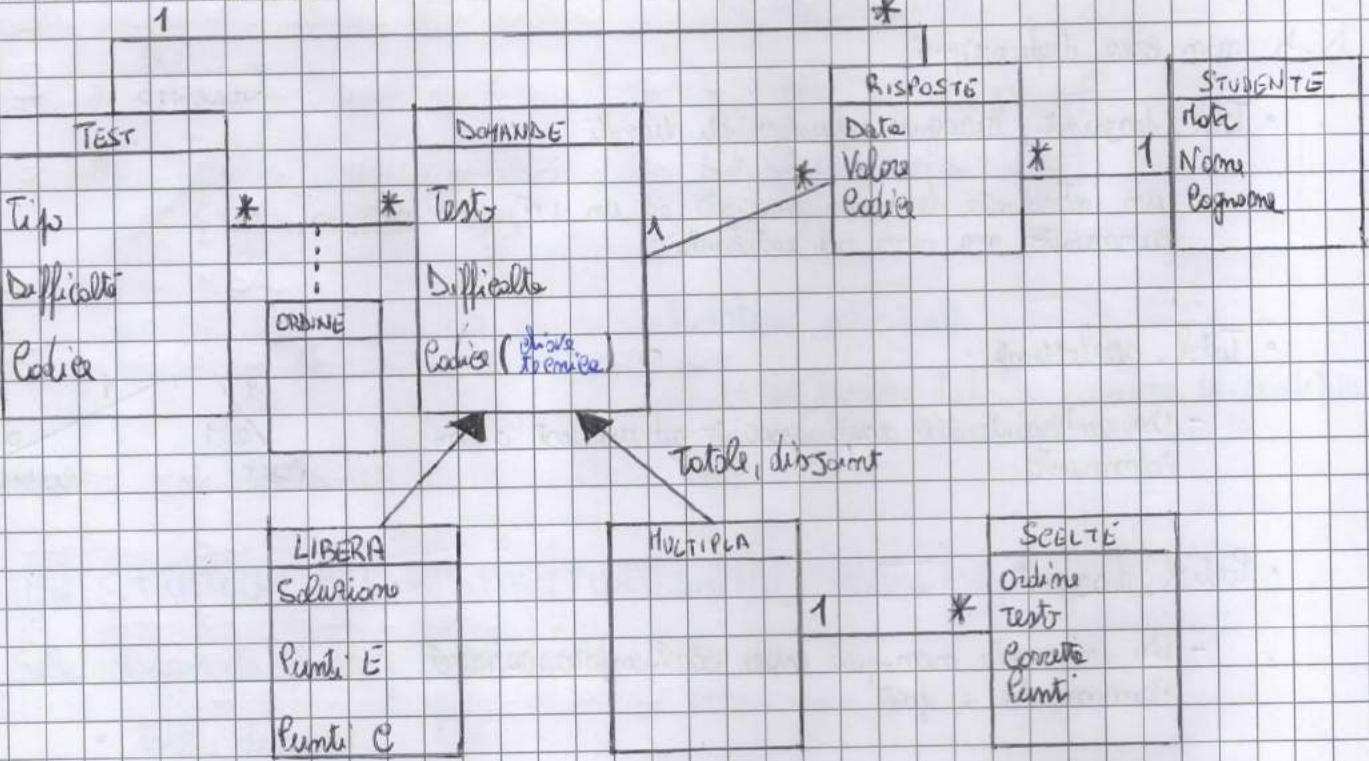
STUDENTI
Motr
Nome
Cognome
Num. esami
media
num. lodi



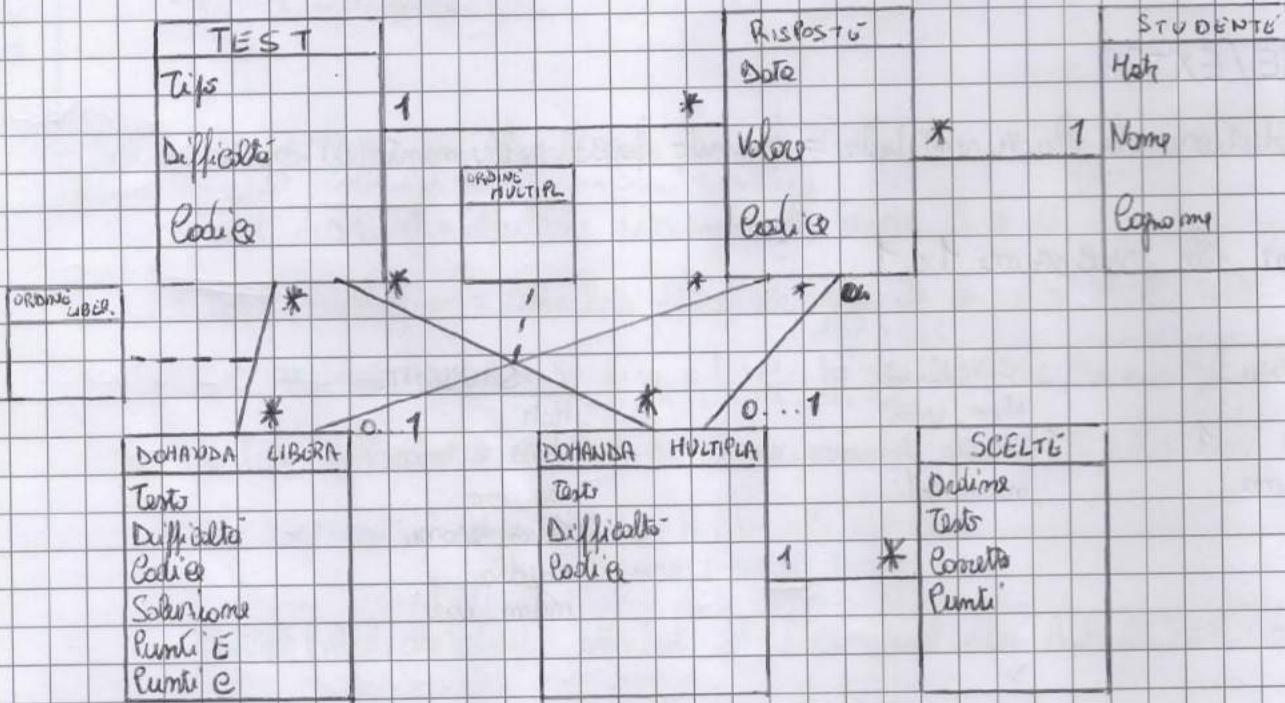
Qual è più
adeguata tra le 2?

Battito di testa = tipologie e frequentate d'intervistazione

ESEMPIO RISTRUTTURAZIONE



RISTRUTTURAZIONE



Per le ristrutturazioni delle classi domande con le relative specificazioni, si è usato il metodo di richeggiamento delle generali sulle specificate.

N.b. **Testo im domanda** potrebbe essere una chiave, ma poiché potrebbe essere molto lungo va a le misure di una chiave primaria.

In risposta → chiave secondaria (**Codice**)

SCHEMA RELAZIONALE

STUDENTE { Matricola, Nome, Cognome }

RISPOSTE { Data, Valore, Codice, Codice - M, Codice - L, Codice - Test }

DOMANDA LIBERA { Testo, Difficoltà, Codice, Soluzione, Punti E, Punti e }

DOMANDA MULTIPLA { Testo, difficoltà, Codice - M }

SCELTE { Ordine, Testo, Corretto, Punti, Codice - M }

TEST { Tipo, Difficoltà, Codice }

ORDINE LIBERA { Codice - Test, Codice - L, Ordine }

ORDINE MULTIPLA { Codice - Test, Codice - M, Ordine }

VINCOLO RISPOSTA

- Codice - M o Codice - L possono essere NULL, ma mai entrambi
- Codice - M o Codice - L possono avere valore, ma mai entrambi

Nel caso si forse utilizzata la tecnica dello raddrizzamento delle specifiche sulle generali:

DOMANDE		SCELTI
TEST		
	Testo	Ordine
	Difficoltà	Testo
	Codice	Corretto
*	Tipo : T-dom 1	Punti E
	Soluzione	
	Punti E	
	Punti e	

Codifica / test senza classi di associazione

TEST { Codice - Test, Tipo, Difficoltà, Codice - domanda }

ERRATO

DOMANDA { Codice - domanda, ..., codice - test }

Codifica per associazioni 1 e 1

VINCOLI

- Se tipo = multipla, allora esistono istanze di ~~scelta~~ SCELTE

- ② Se tipo ≠ multiplo, NON esistono informazioni di SCELTE
- ③ Se tipo = Libero, soluzione ≠ NULL, Punti E ≠ NULL, Punti C ≠ NULL
- ④ Se tipo ≠ Libero, soluzione = NULL, Punti E = NULL, Punti C = NULL

IMPLEMENTAZIONE IN SQL

- linguaggio per definire, implementare, manutenere

DEFINIZIONE DEI METADATI

CREAZIONE TABELLA

```
CREATE TABLE < nome >
(
    < nome - attributo > < tipo > [< valore - di - default >] [< vincoli >],
    .
    .
    .
    < nome - attributo > . . . .
    Altri - vincoli
)
```

ESEMPIO

STUDENTE (CF, Matricola, Nome, Cognome, Data -N)

ESAME (Matricola, Corso, Voto, Data, Lode)

CREATE DATABASE < nome >

```
CREATE TABLE Studenti
(
    Matricola CHAR(9) PRIMARY KEY, → questo vincolo va bene solo se la PK è
    Nome VARCHAR(25) NOT NULL, formato da un solo num.
    Cognome VARCHAR(25) NOTNULL,
    Data_N DATE NOT NULL,
    CONSTRAINT PK_1 PRIMARY KEY (Matricola) -> [ENABLE]
    CONSTRAINT UNQ UNIQUE (CF) -> [DISABLE]
```

In più oltre a ENABLE / DISABLE con una chiave vincolata sul nome PK_1

N.B. il nome tabella deve essere unico

[PRIMARY KEY e UNIQUE sono dei tipi interrelazionale] Costo medi -olti di gestione

N.b. Per primary key SQL costruisce direttamente un indice / Coste il 20% del DB
Gli indici vengono aggiornati automaticamente

Tipi di dati di base

CHAR (m) = Stringa di caratteri di lunghezza m / se la stringa eccede, viene troncata
se infine viene completata con blank

VARCHAR (m) = Stringa di dimensione variabile / la gestione è complessa, m < il limit sup
int
~~smallint~~ = integer

~~smallint~~ = small integer

DECIMAL (i, s) = i = precisione / s = mantissa } scritte con una linea
NUMERIC (i, s) = } entrambi valori decimali

REAL

DOUBLE

DATE = rappresentazione di CHAR/10 caratteri ("YYYY-MM-DD")

SINTASSI INSERT = DATE "2017-11-15"

TIME = tipo speciale per la gestione del tempo / "HH:MM:SS" / m è la precisione

SINTASSI INSERT = TIME "11:17:25"

TIMESTAMP = DATE + TIME

INTERVAL = espriime un intervallo di tempo

N.b. Non ha senso le somme di due tempi, ha senso le somme di un tempo + un intervallo.

CREAZIONE NUOVO DOMINIO (enumerazione)

CREATE DOMAIN

SESSO AS CHAR (1)

ALTERNATIVO

CHECK SESSO = 'M'

✓ OR SESSO = 'F'

restrizioni
sul dominio

CHECK SESSO IN

('M', 'F') → cose sensibili

ESERCIZIO RISANAZIONE SQL STUDENTI - ESAHI

CREATE TABLE Studenti:

(

Nome VARCHAR(20) NOTNULL,

Cognome VARCHAR(20) NOT NULL,

CF CHAR(16) UNIQUE,

Matri CHAR(10) PRIMARY KEY,

Data_N DATE NOT NULL,

Residenza VARCHAR(30)

)

CREATE TABLE Esami:

(

Motricità CHAR(10),

essendo chiave esterna
dove c'è tra le altre tipi

Esame VARCHAR(15) NOT NULL,

Data DATE NOT NULL,

Voto SMALLINT NOT NULL, CHECK Voto BETWEEN 18 AND 30,

Lode CHAR(1) DEFAULT 'N' CHECK Lode IN ('N', 'S') → vincolo di dominio

CONSTRAINT UNI UNIQUE (motricità, esame) → le coppie deve essere uniche

CONSTRAINT D1 CHECK Voto ≥ 18 AND ≤ 30 | Possibilità 2

Vincolo
impossibile. ← CONSTRAINT E1 CHECK Lode = 'N' OR Voto = '30' → Non è un vincolo di dominio, è
un vincolo di esclusione

CONSTRAINT FKI FOREIGN KEY (motricità)

corrispondente
della chiave REFERENCES Studenti (Matri) → integrità
referenziale

ON DELETE NO ACTION

- CASCADE

elimina/affanna in cascata per
dalle i vincoli violati

ON UPDATE CASCADE

- NO ACTION

non compie nessuna azione sulle
referenze

ENABLE → vale per il futuro

- SET DEFAULT

Vai a settare a
definiti / designa
delle impostazioni di
defautl

di volta
si una volta
coppia

DISABLE

VALIDATE → verifica che il simbolo volte
della stessa entità/relazione, ma non
il simbolo

NO VALIDATE

non guarda il
simbolo/bloccante

- SET NULL

l'oggetto viene messo a Null

Integrità referenziale = non posso inserire un esame per uno studente non esistente. I^lo blocca delle chiavi esterne compone tra i valori delle chiavi primarie nelle tabelle riferite.

N.b. Se scrivo unico viene a matricola ed esame, significa che referentemente sono unici

N.b. Per un'elemento dentro dell'esame, il tipo è intervallo

CONSTRAINT & norme CHECK espressione logica / ^{SOLAMENTE} intervallistica

INSERIMENTO

INSERT INTO

Esami VALUES

('N86002277', 'BD', DATE '2017-10-31', '29', 'N')

ESPRESSIONI BOOLEANE

attributo di valore

$OP = \{ =$

attributo di attributo

$\neq \rightarrow$ diverso

attributo IS NULL

$\begin{cases} < \\ <= \\ > \\ >= \end{cases}$ solo per domini uguali

attributo IS NOT NULL

$e_1 \text{ OR } e_2$

$e_1 \text{ AND } e_2$

$c_1 \quad c_2 \quad e_1 \text{ OR } e_2$

		$c_1 \text{ AND } c_2$
c_1	c_2	
T	T	T
T	F	F
F	T	F
F	F	F

c_1	c_2	$e_1 \text{ OR } e_2$
T	T	T
T	F	T
F	T	T
F	F	F

c_1	c_2	$e_1 \text{ OR } e_2$	$\text{NOT } e$	$\text{NOT } e$
UNDEF	T	UNDEF	T	T
T	UNDEF	UNDEF	T	T
UNDEF	F	F	UNDEF	F
F	UNDEF	F	UNDEF	UNDEF
UNDEF	UNDEF	UNDEF	UNDEF	UNDEF

VINCOLI INTRARELAZIONALI

I vincoli intrarelazionali comprendono una sola riga (elementi della stessa tabella)

- NOT NULL = indica che il valore NULL non è ammesso come valore dell'attributo

- UNIQUE = Si impone ad un attributo o un insieme di attributi, e impone che i valori dell'attributo siano una superchiave, ovvero che righe differenti della tabella non possano avere gli stessi valori

- PRIMARY KEY = può essere specificata una sola PK per ogni tabella (un attributo o un insieme di attributi)

VINCOLI INTERRELAZIONALI

I vincoli interrelazionali più diffusi sono i vincoli di integrità referenziale. Questi vincoli eretano legame tra i valori di un attributo della tabella su cui è definito e i valori di un attributo di un'altra tabella. Il vincolo può essere definito in due modi:

- REFERENCES = si specifica la tabella esterna e l'attr. della tab. a cui fa riferimento
- FOREIGN KEY = è un vincolo referenziale tra 2 o più tabella

SINTASSI REFERENCES

REFERENCES < nome_tabella > (< nome_attributo_della_tab > ^{esterna})

SINTASSI FK

FOREIGN KEY < attributo₁, ..., attributo_n >

REFERENCES < nome_tab > (attributo₁, ..., attributo_n)

Le operazioni sulla tabella esterna possono introdurre delle violazioni dei vincoli.

- UPDATE

- DELETE

Le politiche di reazione a queste operazioni sono:

- CASCADE

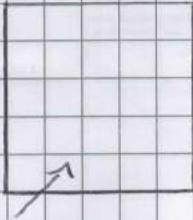
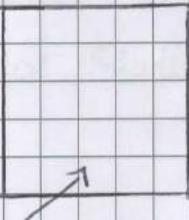
- NO ACTION

- SET NULL

- SET DEFAULT

DATI

METADATI



INSERT / INSERISCI
DELETE / ELIMINA
UPDATE / AGGIORNA

CREATE / INSERISCI
DROP / ELIMINA
ALTER / AGGIORNA

CREAZIONE

CREATE DOMAIN Voto_ty

AS INTEGER CHECK

Voto BETWEEN 1 AND 30

ELIMINAZIONE

DROP DOMAIN Voto_ty

UPDATE

AGGIUNTO



Studenti (Matr, Nome, Cognome, DataN, Residenza)

ALTER TABLE Studenti

ADD COLUMN Residenza VARCHAR(30) [NOT NULL] → imposta un valore obbligatorio dei dati precedenti

↑ aggiunta colonna

ALTER TABLE Studenti

DROP COLUMN Residenza

↑ eliminazione colonna

ALTER TABLE Studenti

ADD CONSTRAINT... → aggiunge vincolo

DROP CONSTRAINT <nome_constraint> → eliminazione vincolo

ALTER CONSTRAINT SET <nome_constraint> ENABLE/DISABLE → modifica vincolo

ALTER TABLE Exam

ALTER COLUMN last

SET DEFAULT 'N'

QUERY LANGUAGE

Relazionale
pure

DBMS -
Relazionale



Algebra
relazionale



SQL

Nel relazionale pure non possono essere ripetizioni
e ordinamento. Nei DBMS si

Il dominio su cui si lavora è quello delle relazioni

STUDENTE (Matr, Nome, Cognome, DataN)

Matr	Nome	Cognome	DataN	Nome
100	Ciro	Esposito	1997-1-1	→ R
200	Annie	Esposito	1997-2-1	relazione
300	François	Coppola	1996-3-1	
400	Walter	Goliazzo	1996-11-28	

Proiezione

perde il
nome delle
relazioni

$\Pi_{\text{Nome}, \text{Cognome}} (R) \longrightarrow ? (\text{Nome}, \text{Cognome})$

✓
è una nuova
relazione di 2
attributi, formata
da una di 4

Ciro	Esposito
Annie	Esposito
François	Coppola
Walter	Goliazzo

$\Pi_{\text{Cognome}} (R)$

← Nel relazionale non
possono essere ripetizioni

Esposito
Coppola
Goliazzo

N.b. Non comincia n

IN SQL

$\pi_{\text{Nome}, \text{Cognome}}$

```
SELECT S.Nome, S.Cognome
FROM Studenti AS S
```

\Rightarrow

$\pi_{\text{Nome}, \text{Cognome}} (\pi)$

(π) alias

L'alfabeto relazionale è il linguaggio su cui lavorano i DBMS

N.b.

```
SELECT S.Cognome
FROM Studenti AS S
```

Esposito
Esposito
Coppola
Bolliero

In SQL ci possono essere le ripetizioni

Per ottenere lo stesso risultato del relazionale pur

↑

```
SELECT [DISTINCT] S.Cognome
FROM Studenti AS S
```

Esposito
Coppola
Bolliero

GENERAL SYNTAX

se non metto nullo
è sostituito ALL

```
SELECT [ALL | DISTINCT] alias. nomeColonna
FROM nomeTabelle AS alias
```

RINOMINA

d'operazioni di rinomina deve solo la classe relazionale

P(π)
nomedatab (attributi)

Esempio

$P_{STI}(\text{Cognome}, \text{Nome})$ ($\pi_{\text{Cognome}, \text{Nome}} (\pi)$)

Serve per prendere da

?($\text{Cognome}, \text{Nome}$)

a

$ST_i(\text{Cognome}, \text{Nome})$

IN SQL

...

FROM .. AS .. / AS è nome rinomina

1- Serve per combinare monne.

2- Serve per avere duplicati

3- Per operazioni di Conteggio \rightarrow assegnare un nome a risultati di operazioni di aggregati.

Rinnomina attributi

$P_{ST1}(\text{Cognome-studenti}, \text{nome-studenti}) (\Pi_{\text{Cognome, nome}}(\pi))$

IN SQL

```
SELECT DISTINCT ST1.Cognome AS Cognome-studente, ST1.Nome AS Nome-studente  
FROM Studenti AS ST1
```

Rinnomina simboli attributi

P (π)
Pilotreale \leftarrow Non

equivisamente a

$P(\pi)$
 $\text{Studente}(\text{Pilotreale, Nome, Cognome, DataN})$

SELEZIONE

$G(\pi)$
(e)

Il risultato è
un sottoinsieme
di π e ha lo
stesso schema

E' una condizione composta da operatori
binari (AND, OR, NOT) e delle condizioni si base
sulle forme

- attributo op valore
- attributo op attributo
- attributo IS NULL

ESEMPIO SU π

$G(\pi)$

$\text{DataN} > "1996-12-31"$ \rightarrow Nel caso di $\text{DataN} = \text{NULL}$ si ha un valore
indefinito.



G esclude sia falso che indefinito.

IN SQL

```
SELECT DISTINCT *  
FROM Studenti AS S  
WHERE S.DataN > DATE '1996-12-31'
```

proiezione di
tutti gli
attributi

- π
- argomento
- Gondola

- Il risultato è un sottoinsieme di R e ha lo stesso schema
- Il sottoinsieme contiene gli elementi per cui C è VERO ed esclude gli elementi per cui C è FALSE o UNDEFINED

N.b. de interrogazioni su SQL chiammo queste forme $\rightarrow \Pi(G_e(\text{tabelle}))$

di quelle tabelle trae le righe che hanno quelle caratteristiche

SQL è di tipo dichiarativo.

d' utilità vere e propria delle norme è per le operazioni funzionali

③ SELECT DISTINCT T.A₁...T.A_X

① FROM tabella AS T

② WHERE condizione

↓
il formato è rigido

im
d'altra relaz.

$\Pi_{A_1 \dots A_X}$

(G
condizione)

(T)

3

2

1

il formato
è flessibile

N.b. DISTINCT ha un costo formidabile

PRODOTTO CARTESIANO

$R_1 \times R_2 = \text{ogni riga di } R_1 \text{ combinata con ogni riga di } R_2$

Esempio

MATR	Nome	Cognome	(R ₁)
100	Elio	Esposito	4
200	Maria	Rossi	
300	Walter	Baldassarri	

È messo in
corrispondenza
e riunione in Matrice

Matr.	Cognome	Voto	Lode	(R ₂)
300	Algebra	24	'N'	1
300	Proj I	24	'N'	2
200	Proj I	28		3

$R_1 \times R_2$

MATR	Nome	Cognome	Materie	Lotto	Voto	Dode
100	Elio	Esposito	300	Algebra	24	'N'
100	Elio	Esposito	300	Proj I	24	'N'
100	Elio	Esposito	200	Proj I	28	
200	Maria	Rossi	300	Algebra	24	'N'
200	Maria	Rossi	300	Proj I	24	'N'
200	Maria	Rossi	200	Proj I	28	
300	Walter	Baldassarri	300	Algebra	24	'N'
300	Walter	Baldassarri	300	Proj I	24	'N'
300	Walter	Baldassarri	200	Proj I	28	

Ha costituito una nuova relazione.

Il problema in questi sono sommate le righe in cui ad uno studente vengono associati esami di altri studenti

SOLUZIONE

una select con condizione
 \exists matr = matrile ($R_1 \times R_2$)

da giustificare la richiesta di uguaglianza tra PK & FK

IN SQL

SELECT S. matr, S. Nome, S. Cognome, E. corso, E. voto (per le proiezioni dopo la giunzione)
FROM Studenti AS S,
Exami AS E

WHERE S. matr = E. matricola (condizione di giunzione)

PRODOTTO CON GIUNZIONE

$\pi_1 \bowtie \pi_2$
condizione

N.b. Se usiamo il simbolo \bowtie nel 99,9% stiamo sbagliando

Il prodotto con giunzione non è altro che un prodotto cartesiano con condizione di giunzione

ESEMPIO DI PRIMA

$\pi_1 \bowtie \pi_2$
matr=matriede



MATR	Nome	Cognome	Corso	...
200	Maria	Rossi	Prog I	...
300	Walter	Gigliani	Algebra	...
300	Walter	Gigliani	Prog I	...

IN SQL

SELECT *

FROM Studenti AS S JOIN

Exami AS E ON S. MATR = E. matricola

DIFERENZE CON LA SOLUZIONE PRECEDENTE

Nel primo caso è una selezione di un prodotto cartesiano
Nel secondo caso è una giunzione esplicita } STESSO OUTPUT

GIUNZIONE NATURALE

$\pi_1 \bowtie \pi_2$ da condizione è implicita / gli attributi di nome uguale, devono avere valori uguali (se ci sono). Se non ci sono è un prodotto cartesiano normale

ESEMPIO DI PRIHA

$\pi_1 \bowtie p(\pi_2) \rightarrow$ il campo MATR non viene duplicato = da tabella con 6 colonne invece di 7
 MATR \leftarrow Matricola

N.b.

$\pi_1 \bowtie \pi_1$ = Essendo tutti gli attributi uguali, prendendo l'ugualianza nella riga = π_1

	A	B		A	B			$S_1 \bowtie S_2$
S_1	a	a		a	a			
	a	b		c	a			
	b	a		c	b			
	b	b		b	b			

S_2	A	B
	a	a
	c	a
	c	b
	b	b

A B

\rightarrow

a	a
b	b

da congiunzione matriciale di rel sullo stesso schema è l'intersezione

IN SQL

~~SELECT~~

FROM S_1 AS S_1 NATURAL JOIN

S_2 AS S_2

OPERAZIONI INSIEMISTICHE / Solo se nomi compatibili = stesso numero di colonne e stessa dominio

$$S_1 \cap S_2 = \begin{array}{|c|c|} \hline A & B \\ \hline a & a \\ b & b \\ \hline \end{array}$$

	A	B	C	D
S_1	a	a	a	a
	a	b	a	a
	b	a	e	a
	b	b	b	b

$$S_1 \cup S_2 = \begin{array}{|c|c|} \hline A & B \\ \hline a & a \\ a & b \\ b & a \\ b & b \\ a & c \\ c & a \\ \hline \end{array}$$

$$S_1 \setminus S_2 = \begin{array}{|c|c|} \hline a & b \\ \hline b & a \\ \hline \end{array}$$

gli elementi di S_1 non presenti in S_2

N.b. la sottrazione non implica l'ugualianza del nome delle colonne, solo lo stesso numero

l'unione non produce duplicati.

ESEMPIO IN ALGEBRA RELAZIONALE

- 1) Nome e Cognome degli studenti che hanno fatto esami
- 2) Nome e Cognome degli studenti che NON hanno fatto esami

Risoluzione

- 1) Studenti \bowtie esami (stesso campo matricola) $\Rightarrow \Pi_{\text{Nome}, \text{Cognome}}(\text{Studenti} \bowtie \text{esami})$
- 2) $\Pi_{\text{Nome}, \text{Cognome}}(\text{Studenti}) \setminus \Pi_{\text{Nome}, \text{Cognome}}(\text{Studenti} \bowtie \text{esami})$

N.b. Studenti $\setminus \Pi_{\text{Nome}, \text{Cognome}}(\text{Studenti} \bowtie \text{esami})$ non hanno le stesse strutture quindi è ERRORE

ESERCIZI SVOLTI

- 1) Nome e Cognome degli studenti che hanno preso domande un 30
- 2) Nome e Cognome degli studenti che hanno preso solo 30

Risoluzione

$$1) \Pi_{\text{Nome}, \text{Cognome}}(\text{Studenti} \bowtie \delta_{\text{Voto}=30}(\text{Esami}))$$

$$2) NO \leftarrow \Pi_{\text{matr.}}(\text{Studenti} \bowtie \delta_{\text{Voto}=30}(\text{Esami}))$$

i no

$$SI \leftarrow \Pi_{\text{matr.}}(\text{Studenti}) \setminus NO$$

matricole degli studenti con tutti 30

$$\Pi_{\text{Nome}, \text{Cognome}}(\text{Studenti} \bowtie SI)$$

proiezione di nome e cognome dei 30

ESERCIZI PARTITO

PARTITE (C-Stadio, Data, C-Sq₁, C-Sq₂, C-Sc₁, C-Sc₂)

STADIO (C-Stadio, Città, Capienza, Nome)

SQUADRA (C-Sq, Nome, Città)

- 1) Nome delle squadre mai battute
- 2) Nome delle squadre che non hanno mai vinto al San Paolo
- 3) Squadre che hanno sempre vinto in casa

1) $\Pi_{e-sq_2} (\delta_{E-CAL2>E-ON1} \wedge \text{DATA} > "2019-03-11") \bowtie_{e-sq_1 = e-sq_2} \Pi_{c-stadio} (\exists_{NOME = "SAN PAOLO"} (\text{STADIO}))$

$\Pi_{NOME} (\text{SQUADRA} \bowtie_{e-sq_1 = e-sq_2} (\text{MAI PERSO}))$

- Calcolare NO₁ e NO₂

$$NO_1 \cup NO_2 \rightarrow NO$$

$$\Pi_{e-sq} (\text{SQUADRA} \setminus NO) \rightarrow SI$$

$\Pi_{NOME} (\text{SQUADRA} \bowtie SI)$

ESEMPIO ALGEBRA RELAZIONALE

STUDENTI (Matr, Nome, Cognome)

CORSI (Codice, Anno, CFU, SSD)

ESAMI (Matr, Codice, Data, Voto, Dode)

1) Nome e Cognome degli studenti che hanno sostenuto tutti gli esami del primo anno.

$\text{CORSI I} \leftarrow \exists_{\text{ANNO}=1} (\text{CORSI})$

Situazione ideale

$I \leftarrow \Pi_{\text{Matr}} (\text{STUDENTI}) \times \Pi_{\text{Codice}} (\text{CORSI I})$

Situazione reale

$R \leftarrow \Pi_{\text{Matr}, \text{Codice}} (\text{ESAMI})$

Studenti con almeno un esame non dato

$I \setminus R$

N.B. Se fosse la posizione sulle matricole, ottieni gli studenti a cui manca almeno un esame del I anno



$\text{NO} \leftarrow \Pi_{\text{Matr}} (I \setminus R)$

$\Pi_{\text{Matr}} (\text{STUDENTI}) \setminus \text{NO}$

STUDENTI

CORSI I

ESAMI

IDEALE

100	.	.
200	.	.
300	.	.

P1
ADE
:
:

100	P1
100	ADE
200	P1
300	ADE

Matr Codice_Corso

100	P1
100	ADE
200	P1
200	ADE
300	P1
300	ADE

$\Pi_{\text{Matr}} (\text{STUDENTI}) \times \Pi_{\text{Codice}} (\text{CORSI I})$

GIUNZIONE ESTERNA

L'operatore di JOIN permette di correlare dati contenuti in relazioni diverse, confrontando i valori contenuti in esse e utilizzando la caratteristica fondamentale del modello, quella di avere le righe su Valori. La caratteristica dell'operatore di "trasformare" le tuple di una relazione senza rapporto nell'altra è utile in molti casi ma potenzialmente pericolosa in altri, in quanto può portare a perdere informazioni rilevanti. Per risolvere questa situazione, vi è una variazione dell'operazione di JOIN chiamata OUTER JOIN. Ne esistono 3 tipi:

- LEFT OUTER JOIN = la tabella di sinistra viene completata dalla destra
- RIGHT OUTER JOIN = la tabella di destra viene completata dalla sinistra
- FULL OUTER JOIN = vengono complete entrambe

ESEMPIO

Studenti che non hanno fatto nessun esame

$\Pi_{\text{Matr}} (\delta_{\text{Corso}=\text{Null}} (\text{STUDENTI} \bowtie \text{ESAMI}))$

N.b. Ho NULL solo se non si aggiunge nessuna riga

DIFFERENZE TRA JOIN E OUTER JOIN

STUDENTI	ESAMI	JOIN	OUTER JOIN
100	100 ProfI 200 ADE	100 200	100 ProfI 100 ADE 200 ADE 300 Null
200			
300	100 ADE		

IN SQL

```
SELECT S.Matr ] 3
FROM STUDENTI AS S RIGHT OUTER
JOIN ESAMI AS E ON S.Matr=E.Matr ] 1
WHERE E.Corso IS NULL ] 2
```

N.b. le operazioni insiemistiche vengono anche per SQL

Esempio / Squadre che hanno vinto in casa e fuori

SELECT DISTINCT P. Squadra1

FROM PARTITE AS P (vole solo per l'intersezione corrente, faranno prima di unione)

WHERE P. Goal1 > P. Goal2 AND P. Data >= "2017-08-15"

UNION DISTINCT/ALL

SELECT DISTINCT P. Squadra2

FROM PARTITE AS P

WHERE P. Goal2 > P. Goal1 AND P. Data >= "2017-08-15"

Esempio di unione di due intersezioni

Altro l'unione vi sono pure:

- INTERSECT (intersezione)
- EXCEPT (sottrazione) → Sempre DISTINCT o ALL

OPERAZIONE DI CONTEGGIO

1- RAGGRUPPAMENTO = individuare un insieme di attributi secondo un criterio

N.b. posso raggruppare anche l'intero insieme

2- CONTEGGIO

N.b. Se facciamo nel raggruppamento, sbagliamo poi nel conteggio

IN ALGEBRA RELAZIONALE

Criterio raggruppamento

Attributo \sum attributo 1 of, attributo 2 of
operazione su attributo 1

OP =

- COUNT = Conta all'interno del gruppo quanti elementi sono diversi da NULL
Nessuna restrizione sul dominio

- AVG = media

Necessita dominio di tipo numerico

- MIN, MAX = fornisce min e max

Necessita dominio ordinato

- SUM = fa la somma

Necessita dominio di tipo numerico

Esempio

Solo queste colonne hanno nome

matr Σ

queste colonne non hanno il nome

Corsi COUNT, Voto AVG, Lode COUNT (τ)

tabelle

4 colonne

(1)

	MATR	ESAMI	VOTO	LODE
1	100	BD1	30	SI
	100	BD2	24	N
2	200	BD1	30	NULL
	200	ALG	20	NULL
	300	ALG	20	NULL
	300	BD1	24	NULL

(2)

MATR
100
200
300
400

N.b. Il problema delle colonne senza nome, si risolve con l'operazione di rimozione

P matr, m° esami, media, lode (matr Σ) corsi COUNT, voto AVG, lode COUNT (τ)

MATR	N° ESAMI	MEDIA	LODI
100	2	24	1
200	2	25	0
300	2	22	0

Problema: Mamba la matricola 400, poiché ha effettuato 0 esami

Soluzione: matr È corso COUNT, voto AVG, lode COUNT (SARÀ)

N.b. bisogna vedere la completezza

MATR	N° ESAMI	MEDIA	LODI
100	2	24	1
200	2	25	0
300	2	22	0
400	0	0	0

ESERCIZIO

Calcolare per numero esami e media per ogni corso ed ogni anno

in SQL
entro le date

ESAMI, YEAR(DATA) È corso COUNT, Voto AVG (π)

N.b. GRANDEZZA TABELLA = m. Corsi x m. anni

ESERCIZIO 2

Trovare numero lodi nel 2016

È esami COUNT $\left(\begin{array}{l} \text{YEAR(DATA)} = "2016" \text{ AND} \\ \text{LODI} = "SI" \end{array} \right)$

Il risultato è sempre una
relazione, mai un risultato

simples

N.b. In questo caso non serve nessun criterio di raggruppamento

ESERCIZIO 3

Studente che nel 2016 ha ottenuto media MAX

ESAMI 2016 \leftarrow $\sigma_{YEAR(DATA) = '2016'}$ (π)

Media $\leftarrow p_{Matri, Media} \left(\text{Matri} \in \text{Voto AVG(ESAMI 2016)} \right)$

N.b. MAX(AVG) È SBAGLIATO

Media_Max $\leftarrow P_{MediaMax} (\text{E media MAX(Media)}) \rightarrow 1 \text{ riga e 1 colonna}$

Media_Studente $\leftarrow (Matri \in \text{media MAX(Media)})$

$\pi_{\text{Nome}, \text{Cognome}, \text{Matri}} (\text{Media_Studente} \bowtie_{\frac{\text{Media-Max}}{\text{Media} - \text{MediaMax}}} \text{Studente})$

CONTEGGIO IN SQL

```
SELECT S.Matri, COUNT(Corso) AS N_Esami, AVG(E.Voto) AS Media
```

```
FROM Studenti AS S LEFT OUTER
```

```
JOIN Esami AS E ON S.Matri = E.Matri
```

```
WHERE YEAR(E.DATA) = '2016'
```

```
GROUP BY S.Matri
```

```
HAVING COUNT(Corso) > 5 / filtro sui raggruppamenti
```

N.b. gli unici attributi che devono figurare in SELECT sono quelli in GROUP BY,
gli altri solo nelle operazioni.

ESEMPIO

Select S.nome, S.cognome, COUNT(Corso), AVG(E.Voto)

Matri nome
raggruppati

E SBAGLIATA

N.b. COUNT(*) Conta tutti, senza preoccuparsi di NULL o altro

SINTASSI SQL

Calcolo delle
espressioni

optional

⑥ SELECT expr1 [AS n₁], expr2 [AS n₂]

costruzione
delle tabelle

① FROM Tab₁ AS A₁ JOIN Tab₂ AS A₂ ON C₁ JOIN Tab₃
AS A₃ ON C₂ ...

filtraggio

R ← di R ② WHERE (condizione sulle righe di R)
(facoltativo)

G ← Partizionamento
di R ③ GROUP BY (lista di raggruppamento)
(gli elementi

sono i gruppi
non più le
righe di R)

G ← filtriaggio
gruppi ④ HAVING (Condizioni gruppi)
di G (Prende in considerazione determinati gruppi)
(facoltativo)

ordinamento
di G ⑤ ORDER BY attrib₁ [ASC | DESC]
(facoltativo)

default

N.b. Nel caso di ordinamento per cognome, se due persone hanno lo stesso
cognome, inserendo un order by su un secondo attributo (nome), considera il nome
come secondo elemento di ordinamento

Tabelle Virtuale = risultato di una interrogazione. Non è salvato in modo permanente

ESEMPIO SQL

Studenti che iniziano per C con media ≥ 24

⑥ SELECT S.Matricola, S.Nome, S.Cognome, AVG(E.Voto) AS media30,
(AVG(E.Voto)/30) * 110 AS Media110

① FROM Studenti AS S LEFT OUTER JOIN Esami AS E
ON E.Matn = S.Matn

Studenti il
cui cognome ② WHERE S.Cognome LIKE 'C%'
inizia per C

③ GROUP BY S.Matn, S.Nome, S.Cognome
↓
Sola le matricole
contenute i gruppi

↓
Saranno solo per la
select

④ HAVING AVG(E.Voto) ≥ 24

⑤ ORDER BY S.Cognome, S.Nome, S.Matricola DESC
ASCENDENT DESCENDENT

- LIKE

È una operazione di confronto

TIPI

CHAR / VARCHAR

SYNTAX

attributo LIKE 'stringa caratteri + mittacaratteri'

- % = sequenza arbitraria di caratteri / stringa su caratteri
generici di lunghezza
ogni volta

- _ = 1 carattere generico

ESEMPI

- Inizia per C e finisce per I = 'C%I'
- Inizia per C, in mezzo c'è la I, finisce per O = 'C%I%O'
- Inizia per C ed è lungo 4 caratteri = 'C ---'

ALTERNATIVA
→ 'C%' AND LENGTH(S.Cognome) = 4

ESEMPIO ESERCIZIO

Decorraziona di una parola in un testo

POST (Cod, Testo, Data)

SELECT

FROM Post AS P

WHERE P.Texto LIKE '%Algiers %'

Case Sensitive

SOTTOINTERROGAZIONE

Studenti che non hanno fatto esami

SELECT S.Nome, S.Cognome, S.Matricola

FROM Studenti AS S

]- Non ci sono duplicati

EXCEPT

DISTINCT (ORA NON PIÙ)

SELECT ✓ S.Nome, S.Cognome, S.Matricola

FROM Studenti AS S JOIN Esami AS E

ON S.Matricola = E.Matricola

]- Contiene duplicati

STessa SOLUZIONE / VARIANTE DELL'ESERCIZIO

SELECT S.Matricola, S.Cognome, S.Nome

FROM STUDENTI AS S LEFT OUTER JOIN ESAMI AS E

ON S.Matricola = E.Matricola

WHERE E.Corso = NULL

VARIANTE 2

SELECT S.Nome, S.Cognome, S.Matricola
FROM Studenti AS S

WHERE S.Matricola NOT IN (SELECT E.Matricola

FROM Esami AS E)

→ Sottointerrogazione correlata

Io che non fa uso di
nessun nome logico
dell'elenco.

SOTTOINTERROGAZIONE CORRELATA

SELECT S.Nome, S.Cognome, S.Matricola

FROM Studenti AS S

WHERE NOT EXISTS (SELECT *

FROM Esami AS E

WHERE E.Matricola = S.Matricola)

IN / NOT IN

(attr₁, attr₂, ..., attr_k) [NOT] IN (SELECT B₁, B₂, ..., B_k
FROM)

IN è vero quando i valori (attr₁, ..., attr_k) sono inclusi in (B₁, ..., B_k)

EXISTS / NOT EXISTS

[NOT] EXISTS (SELECT *

FROM

WHERE)

Exists vero quando la select ha almeno un elemento selezionato $\neq \emptyset$



simbolo di connivenza

=, <, >

Valore = (SELECT

il risultato deve
essere un valore

(restituisce una sola riga)

ESEMPIO

5 = SELECT COUNT()

:
)

Valore < [ANY, ALL] (SELECT A
↓
dimeno uno tutti
:
)

IN GENERALE

Valore op (SELECT

:
)

VINCOLI INTER-RELAZIONALI

Copie di grafi G_1 e G_2 , tale che G_2 sottografo di G_1 . Gli archi di G_2 sono contenuti in quelli di G_1 e i nodi di G_2 sono contenuti in G_1

GRAFO (CodT, Radice)

NODE (CodN, label)

ARC (CodA, CodC, Sig, Tig)

COHPN (CodT, CodN)

COMP A (CodT, CodA)

SELECT G_1 .CodG, G_2 .CodG

FROM GRAFO AS G_1 , GRAFO AS G_2

WHERE NOT EXISTS (SELECT *

FROM COHPN AS N₂

WHERE N₂.CodT = G_2 .CodT AND

N₂.CodN NOT IN (SELECT N.CodN

FROM COHPN AS N₁

WHERE N₁.CodT = G_1 .CodT)

N.b. I vincoli inserimento-eliminazione, in Oracle, non sono presenti in forma base. Possono essere implementati con i trigger

ASSEZIONE

CREATE ASSERTION nome

CHECK espressione booleana

ESEMPIO 1

CREATE ASSERTION nome

CHECK NOT EXISTS (SELECT

insieme delle
violenzioni delle
proprietà.

ESEMPIO 2

CREATE ASSERTION Albero

CHECK NOT EXISTS (SELECT *

FROM COMPN AS N JOIN

GRAFO AS G ON N.CodT = G.CodT

WHERE N.CodN <> G.Radic AND

1 <> (SELECT COUNT(*)

FROM ARC AS A

WHERE A.CodT = N.CodT

AND A.Trg = N.CodN)

Ogni nodo che non sia la radice, ha un unico padre

VIOLAZIONI

- Un nodo \neq radice, non ha un padre

- Un nodo ha più di un padre. Se un nodo appartiene al Grafo, anche i nodi appartenenti al Grafo

ESEMPIO 3

CREATE ASSERTION P₁

CHECK NOT EXISTS (SELECT *

FROM Are AS A JOIN COMPA AS AC

WHERE A.Sig NOT IN (SELECT C.CodN

FROM ROMP AS C

WHERE C.CodT = A.CodT)

VIEW

TABLE → memorizzato

VIEW → Vincoli e non memorizzato

Tabelle
Virtuali

MATERIALIZED VIEW

MATERIALIZZATO
IL CALCOLO

SELECT OK

UPDATE ... NO

SELECT OK
UPDATE, INSERT, DELETE SI

SELECT OK
UPDATE, INSERT, DELETE NO

Nei tipi trasversi
è vincolato.

Le view sono tabella virtuali il cui contenuto dipende dal contenuto delle altre tabelle di una base di dati. Le view vengono definite in SQL associando un nome e una lista di attributi al risultato dell'esecuzione di una interrogazione.

N.b. Nell'interrogazione che definisce le view, possono comparire anche altre view

IN SQL

CREATE VIEW nome (A₁, ..., A_k) AS

SELECT (Exp₁, ..., Exp_k)

CONTEGGIO (CodT, N_Nodi, N_Arci, MaxNodi, MaxArci)

↓
ContN

→ Non posso fare tutte insieme
o prime nodi o prime archi

ContA

CREATE VIEW NodiView (Cod_T, N_nodi, MaxNodi)

AS

SELECT e.CodT, COUNT(*), MAX(N.nodi)
FROM CompyN AS e JOIN Nodi AS N ON e.CodT = N.CodT
GROUP BY e.CodT

CREATE VIEW ArchiView (Cod_T, N_Archi, MaxNodi)

AS

SELECT e.CodT, COUNT(*), MAX(A.nodi)
FROM COMPA AS e JOIN ARCHI AS A ON A.CodT = e.CodT
~~WHERE~~
GROUP BY e.CodT

CREATE VIEW CONTEGGIO (CodT, N_nodi, N_Archi, MaxNodi, MaxArci)

AS

SELECT N.CodT, N.N_nodi, A.N_Archi, N.MaxNodi, A.MaxArci
FROM NodiView AS N JOIN ArchiView AS A ON e.CodT = A.CodT

N.b. Anche nelle FROM posso usare le sottointerrogazioni

MANIPOLAZIONE DEI DATI

INSERT

Vi sono due forme di inserimento:

- valori costanti
- valori calcolati

INSEGNAMENTO DATI COSTANTI

è opzionale Se manca
e' e' riempito di zero
nell'ordine degli attributi (tutti)

INSERT INTO < nome tabella > < lista attributi >

VALUES (value), (value) ... → Si possono inserire più righe contemporaneamente

↓ ↓
inserimento inserimento

d' inserimento va a buon fine se non produce violazione dei vincoli

N.b. Il vincolo viene considerato nullo nell'inserimento.

ESEMPIO

ESAMI (Matricola, Corso, Data, Voto, Lode)

INSERT INTO ESAMI (Matricola, Corso, Voto, Data) → Posso cominciare ordine di inserimento
VALUES ('N860022-11', 'BD', 24, DATE '2017-11-24'),
(.) → inserimento seconda riga

Nel corso lode viene prima di matricola, ~~matricola~~, e non si vogliono definire gli attributi che inserire:

INSERT INTO ESAMI

VALUES (NULL,)

↓
Per le lode

N.b. Se lode fosse stato l'ultimo attributo, non sarebbe stato necessario specificare il NULL, poiché i campi non "riempiti" vengono automaticamente riempiti con NULL.

VALORE CALCOLATO

UTENTE (CodU, Nome, Cognome, Punti)

CARRELLO (CodC, CodU, Merito, Flag)

COMP_C (CodC, CodA, N_punti)

ORDINE (CodO, CodU, Data, Stato)

COSTORD (CodO, CodA, N_punti, Contro)

ARTICOLI (CodA, Descrizione, Prezzo, Scorte)

INSERT INTO ORDINE (CodO, CodU, Data, Stato)

(
SELECT C.CodC, C.CodU, SYSDATE, 'DA INVIARE'
FROM CARRELLO AS C
WHERE C.Flag = 'TRUE'
data corrente (invia una routine di sistema)

INSERT INTO COMPORDING

SELECT C. CodC, C. CodA, C. Nett, H. prezzo

FROM COMPE AS C JOIN CARRELLO AS A ON C.CodC = A.CodC

JOIN ARTICOLI AS H ON C.CodA = H.CodA

WHERE A.Flag = 'TRUE'

Adesso faccio comodato il carrello

SYNTAX DELETE SQL

OPZIONALE

DELETE FROM <nome tabella> [WHERE condizione]

N.b. Senza condizione cancella tutti gli elementi (NON LA TABELLA)

DELETE FROM COMPE

WHERE C.CodC IN (SELECT R.CodC

FROM CARRELLO AS R

WHERE R.Flag = 'TRUE')

DELETE FROM CARRELLO

WHERE FLAG = 'TRUE'

SYNTAX UPDATE

UPDATE <nome tabella> [AS Alias]

SET < nome attributo > = < espressione >

WHERE condizione

Adesso faccio che restare il magazzino

UPDATE ARTICOLI AS A

SET Scorte = Scorte - (SELECT SUM(N.nett))

FROM ORDINE AS O JOIN COMPORD AS C ON
O.CodO = C.CodO

WHERE O.Stato = 'Da Invio' AND C.CodA = A.CodA

WHERE A.CodA IN (SELECT C.CodA
FROM Ordine AS O JOIN COMPORD AS C ON
O.CodO = C.CodO
WHERE O.Stato = 'Da Invio')

Aggiornamento utenti

UPDATE UTENTE AS U

SET Punti = Punti + (SELECT SUM(C.Punto + A.Punto)
FROM ORDINE AS O JOIN COMPORD AS C ON
O.CodO = C.CodO JOIN ARTICOLO AS A
ON C.CodA = A.CodA

WHERE O.CodU = U.CodU AND O.Stato = 'Da Invio')

WHERE U.CodU IN (SELECT O.CodU

FROM ORDINE AS O

WHERE O.Stato = 'Da Invio')

Aggiornamento ordine

UPDATE ORDINE

SET Stato = 'Completato'

WHERE Stato = 'Da Invio'

TRIGGER

I trigger seguono il paradigma Evento - Condizione - Azione: ogni trigger si attiva quando
occorre una specifica eventi all'interno delle liste di dati; se è soddisfatta una data
condizione, allora il trigger esegue un'azione stabilita.

- Quelli sono le variazioni?

- INSERT, DELETE, UPDATE su una tabella

- Quando azione?

aggiornamento
tabella

aggiornamento
vista

- 'BEFORE', 'AFTER', 'INSTEAD OF'

- Quante volte?
 - FOR EACH ROW

N.b. I trigger riendrono nei metadati

SYNTAX

CREATE TRIGGER < nome trigger >

ESEMPIO

CREATE TRIGGER CarrelloCompleto

AFTER UPDATE ON CARRELLO OF Stato

WHERE OLD.State = 'False' AND NEW.State = 'True'

OPZIONALE

FOR EACH Row

BEGIN

•
•
•
•
•

END

OLD = Solo per Cancellazione e ~~Insert~~^{Update}

NEW = Solo per Inserimento e Update

INSERT INTO ORDINI (CodO, CodU, Data, Stato)

VALUES (NEW.CodO, NEW.CodU, SYSDATE, 'DA INVIARE')

EVENTI

- INSERT
- DELETE
- UPDATE

DML

ESEMPIO 2

LIBRO (ISBN, Titolo, Data)

COPIA (ISBN, Proprietario, Data)

CREATE SEQUENCE Prop

START WITH 0

INCREMENT BY 1

CREATE TRIGGER ChiaveCopia

BEFORE INSERT ON Copia

FOR EACH ROW

BEGIN

NEW. Proprietario := Prop. NEXTVAL

END

CREATE TRIGGER Vnif

BEFORE INSERT ON LIBRO

FOR EACH ROW

BEGIN

NEW. Titolo := UPPER(NEW. Titolo)

END

ESEMPIO 2

A(X)
imposto

CREATE TRIGGER impre

AFTER UPDATE OF X

FOR EACH ROW

WHEN OLD.X < NEW.X

BEGIN

UPDATE A AS I

```
SET X := X+1  
WHEN I.ID = NEW.ID  
END
```

↑
OVERFLOW

N.b. NEW e OLD sono utili solo con FOR EACH ROW

AGGIORNAMENTO VISTE

Le viste sono aggiornabili se

- Costruite con SELECT che non usano funzioni nelle clausole From
- Proiettate i campi chiave
- Non devono contenere raggruppamenti

CREATE VIEW Conteggio (Matricola, N_Esami, Media)

AS SELECT Matr, COUNT(*), AVG(Voto)
FROM Esami

GROUP BY Matr → VIOLA LA 3° PROPRIETÀ

INSERT INTO Conteggio

VALUES('N86100', 3, 24)

→ Non ho info necessarie per trovare "corrispondenze" nelle tab principali

ESEMPIO AGGIORNABILE

STUDENTE (Matr, CF, Residenza, Recupero)

STUDENTE2 (Matr, CF, DataN) → VIEW

- **DELETE** = Posso farlo poiché c'è corrispondenza 1:1 tra view e lista
- **UPDATE** = Stesso motivo, solo che non posso fare update su attributi che "non vedo"
- **INSERT** = Solo se la view include tutti gli attributi NOT NULL

STUDENTE (CF, Matr, Data_N, Residenza)

ESAMI (Matr, Corso, Voto, data)

CREATE VIEW Schifra

AS SELECT *

FROM Studenti AS S LEFT OUTER JOIN ESAMI AS E ON
S.Matr = E.Matr

CREATE TRIGGER ins_nohi

INSTEAD OF INSERT ON Schifra

FOR EACH ROW

BEGIN

IF (NEW.Corso IS NULL)

THEN INSERT INTO STUDENTE

VALUES (NEW.CF, NEW.Matr, NEW.Data_N, NEW.Residenza)

ELSE

INSERT

INTO ESAMI

VALUES (NEW.Matr, NEW.Corso, NEW.Data, NEW.Voto, NEW.data)

END IF

END

CORPO TRIGGER

BEGIN

| DECLARE

| BEGIN

| | EXCEPTION

| END

END

VARIANTE SELECT / RESTITUISCE UNA SOLA RIGA

SELECT expr₁, ..., expr_m INTO Vari₁, ..., Vari_m

FROM . . .

IN PL-SQL

BEGIN

DECLARE

CONT : INTEGER := 0

BEGIN

SELECT COUNT(*) INTO CONT

FROM STUDENTI AS S

WHERE S.Matricola = NEW.Matricola

IF CONT > 0 THEN

INSERT INTO ESAMI

VALUES(. . .)

ELSE

INSERT INTO STUDENTI

VALUES(. . .)

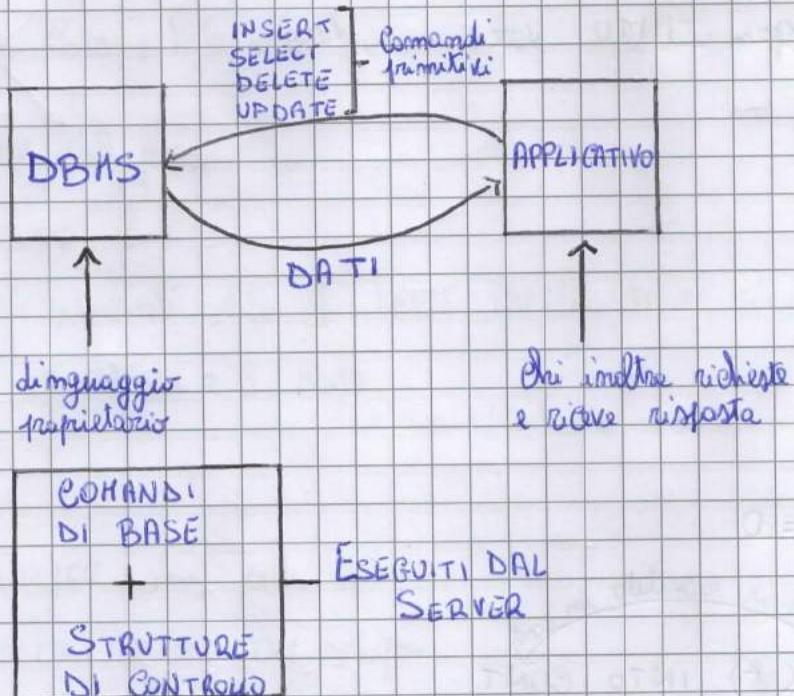
IF NEW.Cogn IS NOT NULL THEN

INSERT . . .

END IF

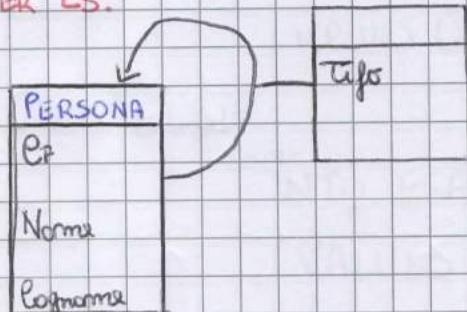
END IF

INTERAZIONE TRA DBMS E APPLICATIVO



N.b. SQL non è computazionalmente completo, ovvero esistono forme di interazione che non possono essere eseguite

Per Es.



PERSONA (PE, Nome, Cognome, Madre, Padre)

- Stored procedure

- Trigger

PL-SQL
Per Oracle

PL-PSQL
Per PostgreSQL

linguaggio
specifco

Vantaggio = Allineamento tra i tipi delle varibili e tipi dati del DB

Svantaggio = Portabilità

INTERFAZIA DEFINITA STANDARD TRA DBMS E AMBIENTE DI PROGRAMMAZIONE

→ DBMS

- Procedure di interfaccia
- API

→ AMBIENTE DI PROGRAMMAZIONE

- È importante che conosce le interfacce

Mod. VECCHIA

EMBEDDED SQL

Quando fai's le chiamate ad SQL, inserisci il codice SQL

(Mescola SQL nel linguaggio di programmazione)

EXEC (INSERT

)

Serve un compilatore / da portabilità
ad hoc.
dipende dall'hardware

VANTAGGIO

- Non deve conoscere le API

Mod. NUOVA

INTERFAZIA STANDARD = portabile per chi aderisce allo standard. Utilizzare i pacchetti standardizzati.

PER JAVA = JDBC



Come rivedo il contenuto nelle variabili?

IN PLSQL

SELECT P₁, ..., P_n INTO (var₁, ..., var_n)
FROM ...

SELECT → PRIMA RIGA
 SELECT e_1, \dots, e_n INTO (v_1, \dots, v_n)
 ↓ PIÙ RIGHE
 Definizione del cursore

CURSORI

BUFFER

RISULTATO
SELECT

Un cursore è uno strumento che permette a un programma di accedere alle righe di una tabella una alla volta. Il cursore viene definito su una generica interrogazione.

CURSOR

DEFINIZIONE CURSORI

DECLARE nomecursor [Insensitive]
[scroll] CURSOR

[WITH HOLD] FOR SELECT . . .

[ORDER BY]

[FOR READ ONLY / FOR UPDATE [OF . . .]]
 ↑
 default!
 il risultato è stato fatto solo per modificare quelle righe.
 ↑
 Nome Colonne

OPZIONI CURSORI

- SCROLL = specifica se si vuole permettere al programma di muoversi liberamente sul risultato dell'interrogazione

- NO SCROLL = ordine strettamente sequenziale / forza liberare il buffer già visitato. Forza rimanere

- INSENSITIVE = Verrà iscritto da tutti quelli che accedono nel DB

OPEN (nome cursor)

esecuzione select + riempimento buffer

→ Al momento dell'esecuzione del comando OPEN, viene eseguita l'interrogazione associata al cursore e il risultato diventa accessibile tramite l'istruzione FETCH

FETCH nomecursor INTO v_1, \dots, v_n

↓
 Prende la riga puntata e trasferisce il contenuto nelle variabili. Aggiorna il puntatore al successivo

- NEXT (implied)
- PRIOR = precedente
- FIRST = la prima riga
- LAST = l'ultima riga
- ABSOLUTE = Va direttamente ad i
- RELATIVE = Allontanamento

CLOSE (nome cursori)]
deslobce buffer] — la terminazione della procedura implica la chiusura. Comunica al sistema che il risultato dell'interrogazione non serve più.

N.b. Nelle dichiarazioni del cursori va dichiarata l'espressione da svilgere.
Per l'apertura si usa OPEN

PL-SQL

- Funzioni o Procedura
- Trigger

CREATE PROCEDURE nome_procedura (nome_parametro [IN|OUT] tipo,
[INOUT] tipo, ...) tipo

CREATE FUNCTION nome_funcionali (nome_parametro [IN|OUT|INOUT] tipo)
RETURN tipo

As . . .

Dichiarazioni

BEGIN

Body

[EXCEPTION]

END

DATABASE PER I METADATI

Esiste una base di dati offerta per i metadati

CATALOGO → Oracle mette a disposizione solo delle viste → prefisso USER → USER CATALOG

USER_TABLES = la maggior parte degli attributi sono a scopo statistico (contemporaneo info).

Il più importante è TABLE_NAME

Es. ATTRIBUTI A SCOPO STATISTICO

NUM_ROWS

BLOCKS

Avg_BLOCK_LEN = Media lunghezza riga
ROW

Definizione delle colonne = USER_TABLE_COLUMNS

	TABLE_NAME = Chiave esterna rispetto a TABLE_NAME
IDENTIFICATORI	COLUMN_NAME
	COLUMN_ID
	POSITION
VINCOLI	DATA_TYPE
	NULLABLE
	NUM_DISTINCT
STATISTICHE	LOW_VALUE
	HIGH_VALUE
	NUM_NULLS

Definizione dei constraint = USER_CONSTRAINTS

CONSTRAINT_NAME
CONSTRAINT_TYPE
TABLE_NAME
SEARCH_CONDITION (ha valore solo se tipo = C)
R_CONSTRAINT_NAME (ha valore solo se tipo = R)

<enum>
CONSTRAINT_TYPE

C / Check
P / Primary key
U / Unique
R / Reference

Chiavi esterne

N.b. Detto che chiave e unique possono essere composti da più attributi, va scritta una nuova tabella

USER_CONS_COLUMNS

CONSTRAINT_NAME
TABLE_NAME
COLUMN_NAME
POSITION

INTERROGAZIONE DEL CATALOGO

Una funzione che prende il nome delle tabelle e restituisce le primary key

CREATE FUNCTION PK (NomeTab USER_TABLES. TABLE_NAME % TYPE) RETURN VARCHAR2 (1000) IS

per dire "stesso tipo" definito nel catalogo

NomeVincolo USER_CONSTRAINTS. CONSTRAINT_NAME % TYPE := "",

NomeAttributo USER_TAB_COLUMNS. COLUMN_NAME % TYPE := "",

Risultato VARCHAR2(1000) := "",

Cont INTEGER := 0,

BEGIN

SELECT COUNT(*) INTO Cont

N.b. Se restituisce un solo valore uno lo SELECT INTO MAI USARE UNA SELECT ISOLATA!

FROM USER_TABLES AS U

WHERE U.TABLE_NAME = NomeTab; → interrogazione di tipo parametrico

IF (Cont = 0) THEN RETURN Risultato

ELSE

SELECT U.CONSTRAINT_NAME INTO NomeVincolo

FROM USER_CONSTRAINTS AS U

WHERE U.CONSTRAINT_NAME = 'P' AND

U.TABLE_NAME = NomeTab

IF (NomeVincolo IS NULL) THEN RETURN Risultato

ELSE

Concatenazione

Risultato := "CONSTRAINT" || NomeVincolo || "Primary Key ("

OPEN (att); → APERTURA CURSOR

LOOP

EXIT WHEN att%NOT FOUND

- FOUND
- COUNT
(numero righe per cui non trovato)

FETCH att INTO NomeAttributo

Risultato := Risultato || NomeAttributo;

END LOOP

N.b. il cursor si ridurrà
mille dichiarazioni variabili

DICHIARAZIONE CURSOR

CURSOR att IS

```
SELECT COLUMN_NAME
```

```
FROM USER_CONS_COLUMNS
```

```
WHERE TABLE_NAME = nomeTab AND CONSTRAINT_NAME = NomeConstraint
```

```
ORDER BY Position;
```

CONTINUO DEL PROGRAMMA

```
CLOSE (att)
```

```
RETURN Risultato || ''
```

```
END IF
```

```
END IF
```

BESTIONE DEL PROBLEMA DELLE VIRGOLE NEL CASO DI ATTRIB. MULTIPLI

```
EXIT ...
```

```
IF att%COUNT > 1 THEN Risultato := Risultato || ',' END IF;
```

```
FETCH ...
```

ITERAZIONI

```
- Loop Exit WHEN att%COUNT > 1
```

- Nel caso debba ricordare sempre tutto il cursor = For I IN att

definita
univocamente

```
FOR I IN att  
LOOP  
:  
END LOOP
```

non ha bisogno
di aprire e chiudere
il cursor, è implicito
di stessa cosa per
FETCH, si automatico
in I

VARIANTE 1

```
For I IN att
```

```
Loop
```

```
IF att%COUNT > 1 THEN Risultato := Risultato || ',' END IF;
```

```
Risultato := Risultato || I.COLUMN_NAME ;
```

```
END Loop
```

```
RETURN Risultato || ''
```

```
END
```

VARIANTE 2

Count := 1

Risultato := "CONSTRAINT" || NomeTablo || "PRIMARY KEY"

FOR I IN (SELECT COLUMN_NAME

FROM USER_CONS_COLUMN

WHERE TABLE_NAME = NomeTablo AND ...)

Loop

IF Count > 1 THEN Risultato := Risultato || ",";

Risultato := Risultato || I.COLUMN_NAME;

Count := Count + 1;

END Loop

ESERCIZIO DA FARE

CREATE FUNCTION Unique (NomeTablo USER_TABLES.TABLE_NAME % TYPE)

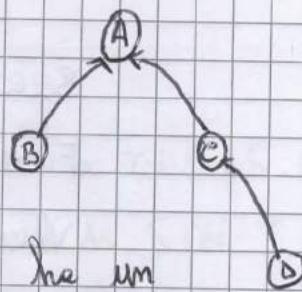
Suggerimento = al posto delle select into che abbiamo usato per trovare le primary, si usano due cursori indipendenti

ESERCIZIO SVOLTO

Una funzione che dato un albero, restituisce il peso del cammino massimo da una foglia alla radice

NODO (CodN, datel, CodT, Radice)

Arco (CodA, datel, CodT, Padre, figlio)



N.b Partendo dalle foglie è più facile poiché ogni nodo ha un solo padre

① Recupero le foglie

CREATE FUNCTION MaxPath (CodTra Nodo, CodT % TYPE) RETURN INTEGER

Risultato INTEGER := 0

↓
continua

CURSORIE foglie IS

SELECT N. CodN, N. label

FROM Nodo AS N

WHERE N. CodT = Padre AND NOT EXISTS (SELECT *

FROM Area AS A

WHERE A. CodT = CodIzq
AND A. Padre = N. CodN)

BEGIN

FOR I IN foglie

Loop

variable := I. label;

Corrente := I. CodN;

Loop

SELECT N. radice INTO rad

FROM Nodo AS N

WHERE N. CodN = Corrente

EXIT WHEN rad

SELECT A. Padre INTO Corrente

FROM Area AS A

WHERE A. figura = Corrente;

SELECT N. label + variable INTO variable

FROM Nodo AS N

WHERE ~~A. figura~~ N. CodN = Corrente;

END LOOP

IF Variable > risultato

THEN

risultato := Variable

END IF

END LOOP

RETURN risultato

Esercizio

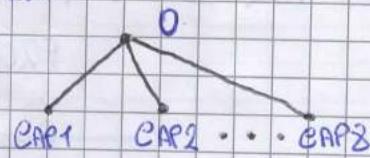
SEZIONE (CodS, Titolo, Posizione Padre)

TMP (Indice, Titolo, Cod)

Idea: Iniziare dalla radice, quindi riempiere la prima riga di TMP con le info della radice

INDICE	TITOLO	RADICE
0	BD	
0.1	Cop I	
0.2	Cop II	
0.3	Cop III	

LIBRO



0 Basi di dati

0.1 (Primo Capitolo) DBHS

0.1.1 . . .

Perro controllo
pertinente alla
radice

CREATE FUNCTION Indice (CodS SEZIONE.CodS % TYPE) RETURN VARCHAR(2000)

Cont1 INTEGER := 1

Cont2 INTEGER := 1

Risultato VARCHAR(2000) := ""

BEGIN

DELETE FROM TMP;

(SELECT S.Posizione, S.Titolo, S.CodS
FROM SEZIONE AS S
WHERE S.Codice)

LOOP

INSERT INTO TMP

(SELECT T.Indice || '.' || S.Posizione, S.Titolo, S.CodS
FROM TMP AS T JOIN SEZIONE AS S ON T.Cod = S.Cod
WHERE S.CodS NOT IN (SELECT *
FROM TMP)) → Controlla che
non ci sia già

SELECT COUNT * INTO Cont2
FROM TMP

EXIT WHEN Cont1 = Cont2

Cont1 := Cont2

END LOOP

FOR I IN (SELECT *

FROM TMP

ORDER BY indice)

Loop

Risultato := Risultato || I . indice || I . Titolo

END Loop

RETURN Risultato

END

Trigger = Chiamato automaticamente da un evento

Procedure = So lo chiamo

} IL CORPO È LO STESSO

ESERCIZIO

USR (Cod , MaxDur, MaxUp, Nome, Stato)

FILE (CodFile, Nome, Tipo)

Blocco (CodFile, CodBlocco, Dim)

Possiede (CodUtente, CodFile, CodBlocco)

DOWN (CodFile, CodBlocco, UsoUp, UsoDown)

Coda (CodU, CodFile, CodBlocco, TimeStamp)

Quando l'USR passa da inattivo ad attivo mette a disposizione i blocchi degli utenti in coda

1- Priorità ad una richiesta precedente

2- Non si devono superare i limiti di download

CREATE TRIGGER attiva

AFTER UPDATE ON USR OF Stato

FOR EACH ROW

WHEN NEW.Stato = 'A' AND OLD.Stato = 'D'

BEGIN

DECLARE

CURSOR candidati IS

SELECT C.CodFile, C.CodBlocco, C.CodU

FROM Coda AS C JOIN Possiede AS P ON C.CodFile = P.CodFile

AND C.CodBlocco = P.CodBlocco

WHERE NEW.Cod = P.CodUtente

ORDER BY C.TimeStamp.

NumU := 0, MaxU := 0;

NumD := 0, MaxD := 0;

H candidati % ROWTYPE → definisce un record i cui campi corrispondono alle colonne per le un cursor o una var. cursor. Ogni campo assume il tipo di dato della corrispondente colonna.

BEGIN

```

SELECT Max_Uf INTO MAX_U
FROM USR AS U
WHERE U.Cod = NEW.Cod

```

SQL DINAMICO

Si intende un comando SQL Runtime.

1- Scrittura del comando (durante la scrittura stessa l'esecuzione dello stesso) ospita in una variabile di tipo stringa.

2 Strade possibili

1- Esecuzione immediata del comando (**EXECUTE IMMEDIATE** Comando)

2- **PREPARE** Comando (Fa es un check sintattico del Codice scritto prima di eseguirlo)

EXECUTE Comando

N.b. Se fallisce, l'esecuzione non è attuabile

Altro vantaggio del **PREPARE**, se il Comando viene eseguito più volte, la preparazione viene fatta solo una volta.

- In un linguaggio proprietario è una possibilità aggiuntiva
- In un linguaggio che s'interfaccia con API, è la norma.

menu a
tavolino

	Norma	Cognome	...	Residenza
	Ciro			Napoli

N.b. Sceglie un elemento della tabella equivalente ad una interrogazione del tipo

SELECT *

FROM Studenti

WHERE Norma = 'Ciro' AND Residenza = 'Napoli'

Ovviamente non si può scrivere in anteprima l'interrogazione per tutto.

Una eccezione possibile potrebbe essere

'Nome @ Ciro, Residenza @ Napoli' → Serve un algoritmo per decifrarlo e poi lo trasforma in interrogazione

Convenzione stabilita dal programmatore

Convenzione Proiezione = attributi separati da virgole / Es: attr1, attr2 ...

CREATE FUNCTION ComandoSQL (tabella VARCHAR, Condizione VARCHAR, Proiezione VARCHAR)

RETURNS VARCHAR

Proiezione attr1, attr2 ...

Condizione = Nomeatt = valore, ...

N.b. Nel caso si restituiscano più elementi, si necessita un cursore, poiché non sono dichiarabili, dichiarare un simbolo a cursore

INSTR (Stringa, pattern, [posizione], [numdiocorrenze])

trovare

SUBSTR (Stringa, posiz, [length])

entrare

a partire dalla posizione

se ascendente prende tutto
e scende prende tutto

Comando VARCHAR := 'Select ' || proiezione || ' FROM ' || tabella || ' WHERE True';

/* DICHIAZIONE REF A CURSOR */

Curs REF CURSOR

↓

tipo esistente
in PL-SQL

Es. INSTR('abbadada', 'a', 1, 3)

Se il pattern è trovato restituisce la posizione iniziale del pattern, se non è trovato, restituisce -1

Nel caso sopra restituisce 4 (settima parola)

Corrente VARCHAR(1000) := Condizione

RIS Curs % ROWTYPE

BEGIN

Loop

EXIT WHEN INSTR(Corrente, '=') < 0

IF INSTR(Corrente, ',') > 0 THEN

Comando := Comando || ' AND ' || SUBSTR(Corrente, 1, INSTR(Corrente, ',') - 1);

Corrente := SUBSTR(Corrente, INSTR(Corrente, ',') + 1);

ELSE

Comando := Comando || ' AND ' || Corrente;

Comando := ''
 END IF
 Open Curs FOR Comando → Solo cursore sequenziali
 |
 |
 Crea il binding
 tra cursore & Comando
 Loop
 EXIT WHEN Curs % NOT FOUND
 FETCH Curs INTO RIS
 Risultato := Risultato || RIS
 END LOOP

CREATE PROCEDURE Creal (Tabelle VARCHAR2, attributi VARCHAR2, nomechiavi VARCHAR2)

Comando := 'ALTER TABLE :1 ADD CONSTRAINT :2 UNIQUE (:3)'
 Significato:
 1 row è
 chiave unica

EXCEPTION METADATI - INCOERENTI

Corrente VARCHAR2 := attributi

att VARCHAR2 := '';

BEGIN

SELECT COUNT(*) INTO Cont

FROM USER_TABLE

WHERE TABLE_NAME = Tabelle;

IF (Cont = 0) THEN RAISE Metadati - Incoerenti;

Loop

EXIT WHEN LENGTH (Corrente) < 1

IF INSTR (Corrente, ',') < 0 THEN

SELECT COUNT(*) INTO Cont

FROM USER_TAB_COLUMNS AS C

WHERE C.TABLE_NAME = Tabelle AND C.COLUMN_NAME = Corrente

IF Cont = 0 RAISE Metadati_Inesistenti

ELSE

SELECT COUNT(*) INTO Cont

FROM USER_TAB_COLUMNS AS C

WHERE C.TABLE_NAME = tabella AND C.COLUMN_NAME = SUBSTR(Parola, 1,

INSTR(Parola, ',')
-1

IF Cont = 0 THEN RAISE Metadati_Inesistenti; ENDIF

Parola := SUBSTR(Parola, 1, INSTR(Parola, ',') + 1);

END LOOP

EXECUTE IMMEDIATE Comando USING Tabella, Nome_Attributo
(ordine posizionale)

Exception

WHEN Metadati_Inesistenti ..

Gestione eccezione

WHEN .. .

..
..
..
..

WHEN .. .

Si possono anche modificare le eccezioni già presenti. Per Es. ZERO_DIVIDE

DATA_NOT_FOUND

ESERCITAZIONE ESAME

FOTO(CodF, CodiceAlbum, Proprietari, Dim, Descr, Path)

TAG(CodF, Parole)

KEY(Parole)

ALBUM(Coda, Nome, Path, Descr, AlbumParole)

THP(Coda, Parole)

CREATE FUNCTION N_Parole(Alb ALBUM.Code % TYPE) RETURN INTEGER

DECLARE

Risultato INTEGER

BEGIN

DELETE FROM THP

INSERT INTO THP (SELECT DISTINCT F.Coda, T.Parole
FROM FOTO AS F NATURAL JOIN TAG AS T
WHERE F.CodiceAlbum = Alb)

LOOP

EXIT WHEN (0 = (SELECT COUNT(*)
FROM ALBUM AS A
WHERE A.Code NOT IN (SELECT THP.Code
FROM THP) AND
A.AlbumParole IS NULL
(SELECT THP.Code
FROM THP))

INSERT INTO THP (SELECT DISTINCT F.Coda, T.Parole
FROM FOTO AS F NATURAL JOIN TAG
WHERE T.Code IN (SELECT Code
FROM ALBUM AS A JOIN THP

(Select TMP Code

FROM TMP)

Select COUNT(DISTINCT ^{Forale} RANDA) INTO Resultat

FROM TMP

RETURN Resultat

END