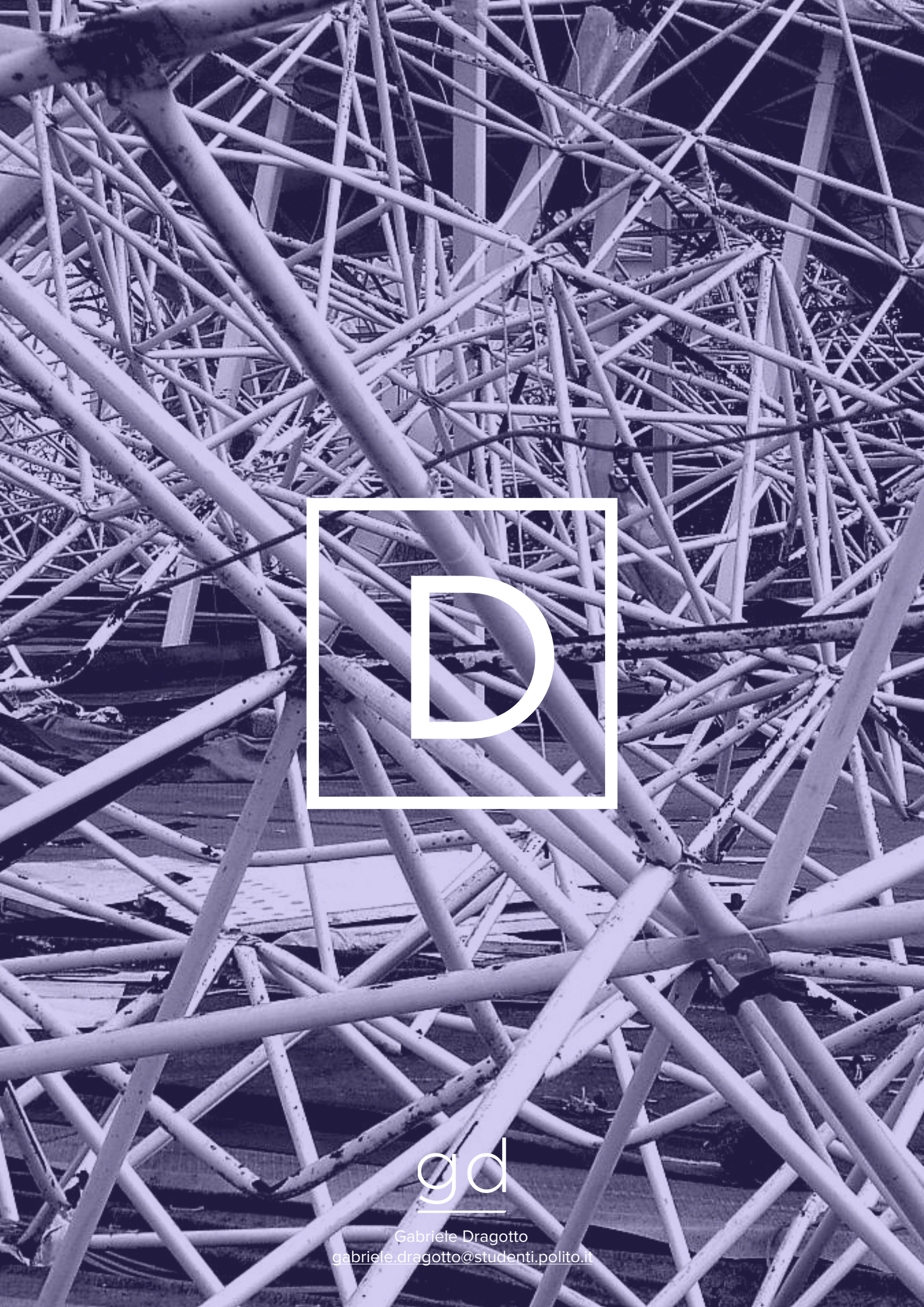


gd

Gabriele Dragotto
gabriele.dragotto@studenti.polito.it

Basi di dati

Introduzione

OBIETTIVI

- **PROGETTARE UNA BASE DATI**
- **INTERROGAZIONE SQL**
- **PROGETTARE UN'APP INTEGRATA**

GESTIONE DELLE INFORMAZIONI

Storicamente vi è sempre stata necessità di gestione delle informazioni, anche **prima dell'avvento dell'informatica.**
L'automazione della gestione dell'informazione tramite calcoli automatici (**o algoritmi**) in grado di elaborare dei **dati numerici** al fine di ottenere delle **informazioni.**

DATO

Valore numerico rappresentante una determinata entità.

La progettazione della struttura dati è molto più importante e duratura della progettazione del **metodo di interpolazione dello stesso.**

INFORMAZIONE

*Interpretazione dei simboli grezzi o **dati** che vengono associati ad una **semantica.***

BASE DATI

Collezione di dati** che rappresenta **informazioni** d'interesse per un determinato sistema IT,. E' gestito da un software **DBMS

DBMS

Un sistema di gestione di basi dati è un sistemi in grado di gestire collezioni di dati avente **3 caratteristiche**:

- **GRANDI**
  la gestione avviene in **memorie secondarie**

- **CONDIVISE**
  tra utenti e applicazioni diverse, in modo **integrato.**
  ACCESSO CONCORRENTE + RIDONDANZA + CONSISTENZA

- **PERSISTENTI**
  il tempo di vita della basi dati **non è condizionato** da quello dell'**applicazione** che la interroga.

Assicurando:

- **AFFIDABILITA'**
  meccanismi di **backup & recovery** e duplicazione.

- **PRIVATEZZA**
  autorizzazione e **privilege separation**

- **EFFICIENZA ED EFFICACIA**
  risolvere il problema con il **minor ammontare di risorse**

RISPETTO AI FS
Vi è un accesso semplificato, e **soprattutto condivisione.**

MODELLI DI DATI

Insieme di concetti utili per **organizzare dati** d'interesse e descriverne la loro **struttura** all'interno dell'elaboratore.

- **TIPI ELEMENTARI** (int, char, long int...)
- **MECCANISMI DI STRUTTURAZIONE** (record, array, rows, matrici)

La **struttura delle entità tabellari è omogenea**, ovvero ogni rows o record contiene gli stessi dati.

MODELLO RELAZIONALE

Modello utile alla definizione delle relazioni tra i dati.
In una base dati sono definiti:

- **SCHEMA**
  descrive la struttura dei dati. è **invariato nel tempo**

- **ISTANZA**
  contenuto della tabella, ovvero il **contenuto** di dati. **varia rapidamente**

TIPI DI MODELLO

- **CONCETTUALE**
  permette di rappresentare i dati in modo **indipendente dal modello logico.** Esso rappresenta unicamente i concetti del problema, ma non scende nel livello logico-relazionale.

- **LOGICO**
  descrive **la struttura dei dati nel DBMS**, tramite le relazioni tra le tabelle stesse.

- **ESTERNO**
  viste di dati per utenti utilizzatori finali

INDIPENDENZA DEI DATI

*I livello di gestione dei dati **prescinde dal livello fisico** di gestione di questi ultimi.*

L'indipendenza avviene grazie alla **struttura di suddivisione in livelli di astrazione.**
Posso modificare il meccanismo di memorizzazione fisica, ma

- **MODELLO LOGICO INALTERATO**
- **RELAZIONI INALTERATE**
- **POSSIBILITA' DI MODIFICA**
  del livello fisico di memorizzazione.

ACCESSO AI DATI

- **USER FRIENDLY INTERFACES**
- **LINGUAGGI TESTUALI INTERATTIVI (SQL)**
- **LINGUAGGI OSPITE**
  php, Java, C++...
- **LINGUAGGI PROPRIETARI**

LINGUAGGIO SQL

- **DDL**
  Il data definition language è necessario per la **definizione degli schemi** e della struttura logica, comprese le autorizzazioni d'accesso.

- **DML**
  Il data manipulation language serve per interagire con i dati

TIPI DI UTENTE

- **DBA ADMINISTRATOR**
  Master della base dati: garantisce **PRESTAZIONI AFFIDABILITA' E AUTORIZZAZIONI.**

- **PROGETTISTI**
  Definiscono e realizzano nuove basi dati, e definiscono le **applicazioni di interpolazione.**

- **UTENTI**
  Utenti finali che interagiscono tramite **transazioni strutturate e applicazioni.**
  Utenti casuali che interagiscono con **interrogazioni customizzate e poco strutturate.**

VANTAGGI E SVANTAGGI

- **MODELLO DATI UNIFICATO**
- **RIDUZIONE RIDONDANZA E INCONSISTENZE**
- **CONTROLLO CENRTALIZZATO**
- **INDIPENDENZA FISICA**

- **COSTI LICENZE**
- **COSTI RISORSE**
- **CONVERSIONE APPLICAZIONI**
- **FORMAZIONE PERSONALE**

Modello relazionale

DEFINIZIONE

Nel 1970 ***E.F. Codd*** propose di elevare il livello di astrazione rispetto ai modelli precedenti, implementando **l'indipendenza dei dati.**
1981 IBM implementa DB2, mentre **Oracle** presenta **SQL Server**

RELAZIONE = TABELLA
ATTRIBUTO = COLONNA
NUPLE = RECORD o RIGA = TUPLA
DOMINIO
Insieme dei valori ammissibili per un dato **attributo.**
CARDINALITA'
Numero di tuple della relazione
GRADO
Numero di attributi della relazione

PROPRIETA' DELLA RELAZIONE

- **NON-ORDINE TUPLE e ATTRIBUTI**
  i record non sono ordinati, così come le colonne
- **TUPLE UNIVOCHE**
  almeno un attributo deve variare

RIFERIMENTO DEI VALORI

Posso costruire legami tra tabelle diversi **basandomi su valori di attribut**i, e non tramite puntatori.

- **INDIPENDENZA LINK**
- **DIREZIONAMENTO LINK**
  Con una chiave primaria posso aggirare il problema.
- **PORTABILITA' DATI**

EX
Riordinando le tuple di una relazione i link sono compromessi. Tramite un valore metto in relazione le tabelle, e posso modificare liberamente le tabelle.
Foreign KEYS

DATI INCOMPLETI

- **VALORI SPECIALI**
  valori fuori dal dominio che indicano l'incompletezza del dato.
  Non sempre esiste un valore **poco significativo —> invento null**
- **VALORE NULL**
  fuori da qualsiasi dominio. rappresenta un valore **ignoto o non definito.**
  non tutti gli attributi possono avere il valore NULL

VINCOLI DI INTEGRITA'

- **VINCOLI INTRA-RELAZIONALI**
  definiti sugli attributi di una sola relazione (unicità, dominio, n-upla)
- **VINCOLI INTER-RELAZIONALI**
  definiti su più relazioni contemporaneamente.

CHIAVE PRIMARIA

Insieme di attributi che consente di indentificare in maniera univoca le tuple. Gode di 3 proprietà fondamentali

- **E' UNA PROPRIETA' DELLO SCHEMA RELAZIONALE**
  UNIVOCITA'
  non esistono 2 tuple aventi gli stessi valori per K
- **MINIMALITA'**
  non esistono sottoinsiemi propri di K univoci, ovvero non posso assumere un sottoinsieme di K come chiave primaria

SUPERCHIAVE: Se una chiave è **univoca ma non minimale**
CHIAVE PRIMARIA: Non può inoltre assumere il valore **NULL**

EX: {Matricola,Nome} è **superchiave ma non chiave primaria**
{Matricola} è una chiave, anche **primaria**
{Nome, Cognome, Codice fiscale} è una chiave ma **non primaria**

VINCOLI DI DOMINIO

Esprime le condizioni sul valore assunto da un singolo attributo di una tupla.

VINCOLI DI TUPLA

Esprime le condizioni sul valore assunto da un singolo attributo di una tupla basandosi su altri attributi

EX: {Costo} = {Prezzo}*{Unità}

VINCOLI D'INTEGRITA' REFERENZIALI

*Date due relazioni **R** (referenziata) e **S** (referenziante) mediante l'insieme **attributi X.** I **valori assunti dall'insieme X di S** possono essere **esclusivamente** valori già assunti dalla **chiave primaria di R.***

REFERENZIANTE: Tabella che punta, con un attributo, ad un'altra tabella.
REFERENZIATA: Tabella con un riferimento da parte di una referenziante.

FOREING KEY: Insieme di attributi X

Algebra Relazionale

DEFINIZIONE

L'albegra relazionale è l'estensione dell'insiemistica *per il mdello relazionale, e definisce un insieme di operatori che agiscono sille relazioni*

- **PROPRIETA' DI CHIUSURA**
  il risultato di qualunque operazione algebrica su relazioni è a sua volta una relazione.

- **UNARI**
- **BINARI**
- **INSIEMISTICI**
- **RELAZIONALI**

- **UNIONE**
- **INTERSEZIONE**
- **DIFFERENZA**
- **PRODOTTO CARTESIANO**
- **SELEZIONE**
- **PROIEZIONE**
- **JOIN**
- **DIVISIONE**

SELEZIONE

*L'operatore di selezione estrae un sottoinsieme di righe, o **orizzontale**, da una relazione.*

$$R=\sigma_p(A)$$

- **GENERA UNA RELAZIONE R**
  Avente gli stessi **attributi di A** e un sottoinsieme delle **tuple di A** che verificano il **predicato P.**

- **IL PREDICATO P**
  E' un'espressione booleana di confronto tra attributi e costanti.
  AND OR NOT

PROIEZIONE

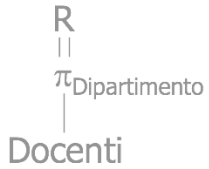
*L'operatore di selezione estrae un sottoinsieme di colonne, o **verticale**, da una relazione. **Elimina le tuple duplicati,** per mantenere la relazione.*

$$R=\pi_L(A)$$

- **GENERA UNA RELAZIONE R**
  Avente gli **attributi di L** e tutte le **tuple di A**.

- **IL PREDICATO L**
  E' un'espressione booleana di confronto tra attributi e costanti.
  AND OR NOT

- **ELIMINA LE TUPLE FINALI DUPLICATE**
  Per la definizione di operatore relazionale stesso. Se è eseguita su una **chiave candidata** l'operazione di rimozione è **superflua**.



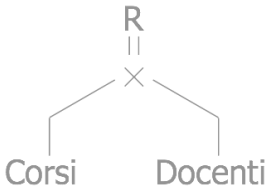
PRODOTTO CARTESIANO

L'operatore di prodotto cartesiano genera delle coppie formate dal tuple di due relazioni A,B,

$$R=A\times B$$

- **GENERA UNA RELAZIONE R**
  Avente come schema **l'unione degli schemi** delle relazioni **A,B**.
  Le tuple sono tutte le combinazioni che posso generare tra le tuple di A e B

- **ASSOCIATIVA E COMMUTATIVA**



JOIN

*L'operatore JOIN è un **prodotto derivato**, espresso mediante operatori elementari.*

Il prodotto cartesiano è inutile se non vi è **legame sematico tra gli attributi**.

- **NATURAL JOIN**
- **THETA-JOIN**
- **SEMI-JOIN**

NATURAL JOIN

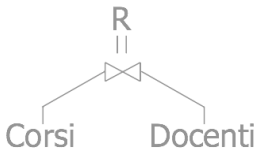
*Il natural join di due relazioni A,B si basa su **specifiche relazioni di legame**, generando un risultato R tale che. Richiede che la **condizione di legame sia implicita.***

$$R=A<>B$$

- **GENERA UNA RELAZIONE R**
  -Gli attributi presenti nello **schema di A** ma non di B.
  -Gli attributi presenti nello **schema di B** ma non di A.
  -Una **sola coppia degli attributi comuni**

- **CONTIENTE TUTTE LE TUPLE**
  Aventi i valori degli **attributi comuni uguali.**

- **COMMUTATIVO E ASSOCIATIVO**



R

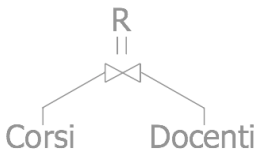
Corsi. Codice	Corsi. NomeCorso	Corsi. Semestre	MatrDocente	Docenti. NomeDoc	Docenti. Diparimento
M2170	Informatica 1	1	D102	Verdi	Informatica
M4880	Sistemi digitali	2	D104	Bianchi	Elettronica
F1401	Elettronica	1	D104	Bianchi	Elettronica
F0410	Basi di dati	2	D102	Verdi	Informatica

THETA-JOIN JOIN

*Il tetha join specifica una **generica condizione di legame** (non solo uguaglianza) con attributi con nomi diversi.*

$$R = A \bowtie_p B$$

- **CONDIZIONE DI LEGAME**
  Rispetto al naturale posso specificare l'**operatore che lega gli attributi e i nomi di questi.**
  -Unione degli schemi di **A e B**
  -Contiene tutte le coppie contenute nelle tuple di A,B per cui è **vero il predicato p.**
- **IL PREDICATO E' UN OPERATORE LOGICO**
- **POSSO EFFETTUARE CONTI**
- **COMMUTATIVO ED ASSOCIATIVO**



EX

CONTARE CON IL JOIN

Selezionare tutti i docenti che hanno almeno 2 corsi.
Effettuo un **theta-join**, creando una **tupla con i duplicati** degli attributi.

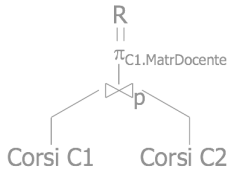
C1

Codice	NomeCorso	Semestre	MatrDocente
M2170	Informatica 1	1	D102
M4880	Sistemi digitali	2	D104
F1401	Elettronica	1	D104
F0410	Basi di dati	2	D102

C2

Codice	NomeCorso	Semestre	MatrDocente
M2170	Informatica 1	1	D102
M4880	Sistemi digitali	2	D104
F1401	Elettronica	1	D104
F0410	Basi di dati	2	D102

Trovare la matricola dei docenti che sono titolari di almeno due corsi



p: C1.MatrDocente=C2.MatrDocente $\wedge$ C1.Codice<>C2.Codice

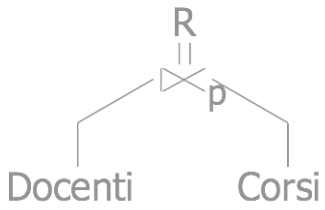
$R = \pi_{C1.MatrDocente}((Corsi\ C1) \bowtie_p (Corsi\ C2))$

SEMI-JOIN

*Il semi-join di A,B seleziona tutte le tuple di A **semanticamente legate** ad una di B.*

$$R = A \ltimes_p B$$

- **RISULTATO**
  Il risultato ha unicamente lo **schema di A**
  -Contiene tutte tuple di A per cui è vero il **predicato che lo lega con le tuple di B.**
- **IL PREDICATO E' UN OPERATORE LOGICO**
- **NON E' COMMUTATIVO**



OUTER JOIN

L'outer join permette di conservare le informazioni relative alle tuple ***non legate senticamente al predicato*** *del JOIN.*

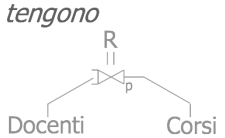
$$R = A \bowtie_p B$$

- **COMPLETA LE TUPLE**
  Con valori null, se prive di legame semantico
- **TIPI DI OUTER JOIN**
  Left: Tuple del primo operando
  Right: Tuple del secondo operando
  Full: entrambe le tuple

LEFT OUTER JOIN

- **UNA TUPLA A,B**
  Con i campi semanticamente legati. Per quelli non legati **vedo NULL**
- **SELEZIONO UNA TUPLA DI A**
  E completo con le **tuple di B se è valido il predicato**. altrimenti **null.**

Trovare le informazioni sui docenti e sui corsi che tengono



$R = Docenti \bowtie_p Corsi$

R p: Docenti.MatrDocente=Corsi.MatrDocente

Docenti. MatrDocente	Docenti. NomeDoc	Docenti. Dipartimento	Corsi. Codice	Corsi. NomeCorso	Corsi. Semestre	Corsi. MatrDocente
D102	Verdi	Informatica	M2170	Informatica 1	1	D102
D102	Verdi	Informatica	F0410	Basi di dati	2	D102
D104	Bianchi	Elettronica	M4880	Sistemi digitali	2	D104
D104	Bianchi	Elettronica	F1401	Elettronica	1	D104
D105	Neri	Informatica	null	null	null	null

RIGHT OUTER JOIN

Opposto del left Outer.

- **UNA TUPLA A,B**
  Con i campi semanticamente legati. Per quelli non legati **vedo NULL**
- **SELEZIONO UNA TUPLA DI b**
  E completo con le **tuple di A se è valido il predicato**. altrimenti null.

$$R = A \bowtie_p B$$

FULL OUTER JOIN

Unione degli attributi di A,B, completati col null. E' ***commutativo***

$$R = A \bowtie_p B$$

UNIONE

L'operatore genera una relazione finale avente lo stesso schema di A,B, contenente tutte le tuple appartenenti ad A e tutte quelle di B.

$$R = A \cup B$$

- **GENERA UNA RELAZIONE R**
  Avente l'unione delle tuple, **eliminando i duplicati.**

- **CONDIZIONE DI COMPATIBILITA'**
  Le relazioni A,B devono avere lo **stesso schema**, **numero e nome di attributi.**
- **COMMUTATIVA ED ASSOCIATIVA**

INTERSEZIONE

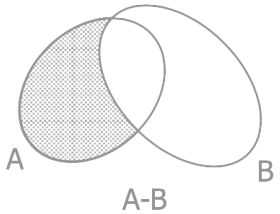
*L'operatore genera una relazione finale avente lo stesso schema di A,B, contenente tutte le tuple appartenenti **sia ad A che a B.***

- **GENERA UNA RELAZIONE R**
  Avente l'intersezione delle tuple, **eliminando i duplicati.**

- **CONDIZIONE DI COMPATIBILITA'**
  Le relazioni A,B devono avere lo **stesso schema**, **numero e nome di attributi.**
- **COMMUTATIVA ED ASSOCIATIVA**

- **LEGAME COL JOIN**
  Se ho un pericato di uguaglianza su **tutti gli attributi** ottengo un JOIN.

DIFFERENZA

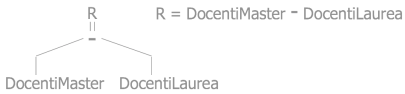


*Seleziona le tuple **presenti nel primo** operando e **assenti nel secondo.***
*E' un operatore fondamentale che **non è definibile tramite altri***

$$R = A - B$$

- **GENERA UNA RELAZIONE R**
  Avente lo **stesso schema di A,B (compatibilità)** contenente le tuple **appartenenti ad A e non a B.**

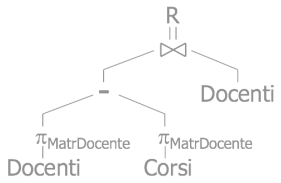
- **CONDIZIONE DI COMPATIBILITA'**
  Le relazioni A,B devono avere lo **stesso schema**
- **NON E' COMMUTATIVA**



EX

DIFFERENZA

Trovare Matricola, Nome e Dipartimento dei docenti che non tengono corsi



$$R = Docenti \bowtie ((\pi_{MatrDocente} Docenti) - (\pi_{MatrDocente} Corsi))$$

ANTI-JOIN

E' una differenza che ***elimina la compatibilità di schema.***
L'antijoin tra A,B, seleziona tutte le tuple ***semanticamente non legate*** *alle tuple di B.*

$$R = A \overline{\bowtie}_p B$$

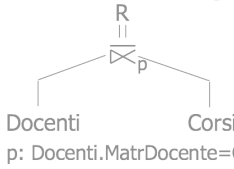
- **GENERA UNA RELAZIONE R**
  Avente lo **stesso schema di A**
  Contenente tutte le tuple di A per le quali non esiste **nessuna tupla di B per il quale il predicato è verificato.**
- **NON RICHIEDE COMPATIBILITA' DI SCHEMA!**

- **NON E' COMMUTATIVO O ASSOCIATIVO**

EX

ANTIJOIN

Trovare Matricola, Nome e Dipartimento dei docenti che non tengono corsi



$$R = Docenti \overline{\bowtie}_p Corsi$$

R

MatrDocente	NomeDoc	Dipartimento
D105	Neri	Informatica

DIVISIONE

Operazione elementare ***non definibile tramite altre.***

$$R = A / B$$

- **GENERA UNA RELAZIONE R**
  Avente come schema **Schema(A) - Schema(B)**
  Contenente tutte le tuple di A tali che per ogni **tupla (Y:y) di B esiste una tupla (X:x),(Y,y) di A.**

- **NON E' COMMUTATIVA O ASSOCIATIVA**

EsamiSuperati

MatrStudente	CodCorso
S1	C1
S1	C2
S1	C3
S1	C4
S1	C5
S1	C6
S2	C1
S2	C2
S3	C2
S4	C2
S4	C4
S4	C5

CorsiPrimoAnno

CodCorso
C1
C2
C3
C4
C5
C6

R

MatrStudente
S1

164

SQL

IL LINGUAGGIO

è un linguaggio standardizzato per database basati sul modello relazionale, in grado di estrapolare dati e definire la base dati.
DML + DDL

- **LINGUAGGIO DI SET**
  Gli operatori operano e restituiscono unicamente **relazioni** al massimo **dengeneri.**

- **LINGUAGGIO DICHIARATIVO**
  Non si deve prestare attenzione al come l'operazione viene eseguita.

- **DML**
  SELECT, UPDATE, REMOVE, INSERT
- **DDL**
  CREATE, ALTER, DROP **TABLE or VIEW or INDEX**
  GRANT, REVOKE
  COMMIT, ROLLBACK

GRAMMATICA

- **PARENTESI < >**
  Isolano un termine della sintassi
- **PARENTESI [ ]**
  Il termine è opzionale
- **PARENTESI { }**
  Il termine può non comparire o essere ripetuto
- **PARENTESI |**
  Indica l'OR

DB EXEMP

P

CodP	NomeP	Colore	Taglia	Magazzino
P1	Maglia	Rosso	40	Torino
P2	Jeans	Verde	48	Milano
P3	Camicia	Blu	48	Roma
P4	Camicia	Blu	44	Torino
P5	Gonna	Blu	40	Milano
P6	Bermuda	Rosso	42	Torino

F

CodF	NomeF	NSoci	Sede
F1	Andrea	2	Torino
F2	Luca	1	Milano
F3	Antonio	3	Milano
F4	Gabriele	2	Torino
F5	Matteo	3	Venezia

FP

CodF	CodP	Qta
F1	P1	300
F1	P2	200
F1	P3	400
F1	P4	200
F1	P5	100
F1	P6	100
F2	P1	300
F2	P2	400
F3	P2	200
F4	P3	200
F4	P4	300
F4	P5	400

SELECT

Istruzione di selezione dati. (include selezione e proiezione)

SELECT [**ALL** | **DISTINCT** | TOP] lista_elementi_selezione [**AS** rename]
FROM lista_riferimenti_tabella
[**WHERE** espressione_condizionale]
[**GROUP BY** lista_colonne [**HAVING** Condizione]]
[**ORDER BY** lista_colonne]
[**LIMIT** numeroris];

- **CONGRUENZA ALGEBRA**
  Non coincide sempre con gli operatori di algebra relazionale

- **DISTINCT**
  Applica la rimozione dei duplicati sul risultato della selezione
  EX: *SELECT **DISTINCT** CodP FROM FP;*

- **AS**
  Rinomina temporaneamente (in ram) un attributo in **rename**
  EX: *SELECT DISTINCT CodP as **Codicione** FROM FP;*

- **HAVING**
  Condizione di selezione **sui gruppi, associata alla group by definita sui valori aggregati.**
  EX: *SELECT CodP, **SUM(Qta)** FROM FP **GROUP BY** CodP **HAVING SUM(Qta)>=600;***

- **WHERE**
  Permette di definire il **predicato di selezione**.
  Puo' essere costituito da: **espressioni algebriche, booleane, LIKE** (_ e %)**.**
  Può essere multiplo tramite la concatenazione con **OR, AND.**
  Sono ammessi **IS, NOT, LIKE**
  EX: *SELECT DISTINCT CodP FROM FP WHERE **Sede='Milano' OR Sede='Torino' AND NSoci>2;***

- **GROUP BY**
  Crea delle partizioni per l'attributo. Le **funzioni aggregate** sono calcolate **sulle partizioni.**
  Se nel **group by ho la chiave primaria della tabella**, posso aggiungere altri attributi della tabella senza cambiare il risultato.
  Nella clausola select **possono apparire solo attributi di Group by o funzioni aggregate.**
  EX: *SELECT CodP, **SUM(Qta)** FROM FP **GROUP BY CodP;***

- **ORDER BY**
  Permette di ordinare i risultati in maniera **ASC** o **DESC** in base alle **colonne di order.** Le colonne di order **devono comparire nella lista_elementi_selezione.**
  EX: *SELECT DISTINCT CodP FROM FP **ORDER by CodP DESC;***

- ***lista_elementi_selezione***
  Campi, separati da virgola, selezionati. Puo' essere una **wildcard** *
- ***lista_riferimenti_tabella***
  l'elenco delle **tabelle** da cui estrarre i dati
- ***numeroris***

JOIN

Istruzione di selezione dati da più tabelle

- **CONGRUENZA ALGEBRA**
  Non coincide sempre con gli operatori di algebra relazionale

- **WHERE**
  Devono esserci **almeno (N-1)** clausole con **N tabelle di JOIN**
  EX: *SELECT FX.CodF, FY.CodF FROM F AS FX, F AS FY WHERE FX.Sede=FY.Sede;*

- ***lista_riferimenti_tabella***
  l'elenco delle **tabelle** deve essere **maggiore di 2 per essere un JOIN**

JOIN 2

Istruzione di selezione dati. (include selezione e proiezione)

SELECT [DISTINCT] Attributi
  FROM Tabella
  < INNER | [FULL | LEFT | RIGHT] OUTER > JOIN Tabella
  ON CondizioneDiJoin
  [WHERE CondizioniDiTupla**];**

- **INNER JOIN**
  Una inner join **crea una nuova tabella combinando i valori delle due tabelle** di partenza (A and B) basandosi su una **certa regola di confronto**. La query compara ogni riga della tabella A con ciascuna riga della tabella B cercando di soddisfare la regola di confronto definita.
- **EQUI JOIN**
  E' unicamente ammesso l'**operatore =**
- **NATURAL JOIN**
  I campi delle tabelle devono avere **nome uguale.**
- **OUTER JOIN**
  Una outer join **non richiede che ci sia corrispondenza esatta tra le righe** di due tabelle. La tabella risultante da una outer join trattiene tutti quei record che **non hanno alcuna corrispondenza tra le tabelle.**
  LEFT Trattiene le righe della tabella di SX
  RIGHT Trattiene le righe della tabella di DC
  FULL Trattiene le righe di entrambe le tabelle

FUNZIONI AGGREGATE

*Gli operatori aggregati si caratterizzano per restituire un valore in corrispondenza di **un gruppo di valori o dei valori** che formano una colonna di una tabella contenuta in un database*

COUNT SUM AVG MIN MAX

- ***lista_elementi_selezione***
  Nella query select può contenere le funzioni aggregate.
- **Valutate dopo la clausola where**

———**COUNT**
COUNT (<| [DISTINCT | ALL] ListaAttributi >)*
*EX: SELECT **COUNT(DISTINCT CodF)** FROM FP;*

INTERROGAZIONI NIDIFICATE

Un'interrogazione nidificata è un'istruzione SELECT ***contenuta all'interno di un'altra interrogazione.*** *La nidificazione di interrogazioni permette di* ***suddividere un problema complesso*** *in sottoproblemi più semplici.*

Posso nidificare la selezione nelle clausole di:

- **WHERE**
- **HAVING**
- **FROM**

- **UTILIZZO UGUALE**
  EX: SELECT CodF FROM F WHERE Sede = (SELECT Sede FROM F WHERE CodF='F1');
  È possibile **utilizzare '=' esclusivamente se è noto a priori che il risultato della SELECT nidificata è sempre unico

- **EQUIVALENZA JOIN**
  EX: SELECT FY.CodF FROM F AS FX, F AS FY WHERE FX.Sede=FY.Sede ANDFX.CodF='F1';
  Molto spesso le selezioni nidificate equivalgono a delle query di JOIN.
  In ogni caso questo **non è sempre possibile.**

Oltre all'uguale ci sono **altri operatori** per la nidificazione

- **IN**
  Operatore di appartenenza all'insieme
  NomeAttributo IN (InterrogazioneNidificata)
  EX: SELECT NomeF FROM F WHERE CodF **IN** (SELECT CodF FROM FP WHERE CodP='P2');

```
SELECT CodF
FROM FP
WHERE CodP IN
        (SELECT CodP
         FROM FP
         WHERE CodF IN
                (SELECT CodF
                 FROM FP
                 WHERE CodP IN
                        (SELECT CodP
                         FROM P
                         WHERE Colore='Rosso')))
```

Codici dei fornitori di prodotti forniti da fornitori di prodotti rossi

```
SELECT ...
FROM F, FP AS FPA, FP AS FPB, FP AS FPC, P
WHERE F.CodF=FPA.CodF AND
      FPA.CodP=FPB.CodP AND
      FPB.CodF=FPC.CodF AND
      FPC.CodP=P.CodP AND
      Colore='Rosso'
```

- **NOT IN**
  Concetto di esclusione. Da ricordare che **diverso e negativo** sono diversi!
  NomeAttributo NOT IN (InterrogazioneNidificata)
  EX: SELECT NomeF FROM F WHERE CodF **NOT IN** (SELECT CodF FROM FP WHERE CodP='P2');

  Trovare il nome dei fornitori che **forniscono solo il prodotto P2** =Trovare il nome dei fornitori di P2 che **non hanno mai fornito prodotti diversi da P2**
  Insieme escluso: **fornitori di prodotti diversi da P2**

```
SELECT NomeF
FROM F
WHERE F.CodF NOT IN (SELECT CodF
                     FROM FP
                     WHERE CodP<>'P2')
      AND F.CodF IN (SELECT CodF
                     FROM FP);
```

COSTRUTTORE DI TUPLA

Permette di definire la struttura temporanea di una tupla tramite ***l'elencazione degli attributi necessari.***

- **ESTENSIONE FUNZIONALITA' IN-NOTIN**

```
SELECT LuogoPartenza, LuogoArrivo
FROM VIAGGIO
WHERE (LuogoPartenza, LuogoArrivo) NOT IN
 DBG
(SELECT LuogoPartenza, LuogoArrivo FROM VIAGGIO
WHERE OraArrivo-OraPartenza>2);
```

EXIST

Permette di controllare ***l'esistenza di un risultato di una subquery.***

```
SELECT [column_name... | expression1 ]
FROM [table_name]
WHERE [NOT] EXISTS (subquery)
```

- **CLAUSOLA SELECT INTERNA**
  I parametri selezionati sono irrilevanti, poichè **exist verifica unicamente l'esistenza dei risultati.**
  Deve essere presente una clausola di correlazione

CORRELAZIONE TRA INTERROGAZIONI

Permette di correlare degli attributi a ***livelli diversi di nidificazione.***

```
SELECT CodP, CodF FROM FP AS FPX
WHERE Qta = (SELECT MAX(Qta)
             FROM FP AS FPY WHERE FPY. CodP=FPX.CodP);
```

- **INDICATA NEL WHERE INTERNO**
- **LEGA FROM DIVERSI**
  Dell'interrogazione interna ed esterna

DIVISIONE

Permette di implementare l'operatore ***divisione*** *nel linguaggio SQL*

```
SELECT CodF FROM FP GROUP BY CodF
HAVING COUNT(*)=(SELECT COUNT(*) FROM P);
```

Estrae i fornitori che forniscono tutti i prodotti.

- **IMPLEMENTATA STEP BY STEP**

AGGREGATO A 2 LIVELLI

Permette di implementare funzioni aggregate a doppio livello

- **IMPLEMENTATA STEP BY STEP**

```
SELECT MAX(MediaStudenti)
FROM (SELECT Matricola, AVG(Voto) AS MediaStudenti
      FROM ESAME-SUPERATO GROUP BY Matricola)
     AS MEDIE;
```

STUDENTE (Matricola, AnnoIscrizione)
ESAME-SUPERATO (Matricola, CodC, Data, Voto)

- Trovare la media massima (conseguita da uno studente)
- Risoluzione in 2 passi
  - trovare la media per ogni studente
  - trovare il valore massimo della media

TABLE FUNCTION

```
(SELECT Matricola, AVG(Voto) AS MediaStudenti
      FROM ESAME-SUPERATO GROUP BY Matricola)
```

Definisce una tabella temporanea utilizzata nelle operazioni di calcolo.

Gode delle seguenti proprietà:
- **INIZIA CON UNA SELECT**
- **E' DEFINITA ALL'INTERNO DI UNA CLAUSOLA FROM**

Permette di:
- **CALCOLARE AGGREGATI A DUE LIVELLI**
- **IMPLEMENTARE LA CORRELAZIONE**

```
SELECT AnnoIscrizione, MAX(MediaStudente)
FROM STUDENTE,
      (SELECT Matricola, AVG(Voto) AS MediaStudente
       FROM  ESAME-SUPERATO
       GROUP BY Matricola) AS MEDIE
WHERE STUDENTE.Matricola=MEDIE.Matricola
GROUP BY AnnoIscrizione;
```

OPERATORI INSIEMISTICI

Implementazioni di operatori insiemistici nel linguaggio SQL

UNION INTERSECT EXCEPT

UNION

Definisce un risultato UNIONE delle istruzioni A,B.

Gode delle seguenti proprietà:
- **UNION** con rimozione duplicati
- **UNION ALL** senza rimozione duplicati
- **RICHIEDE COMPATIBILITA' DI SCHEMA**

```
SELECT CodP FROM P
   WHERE Colore='Rosso'
UNION
SELECT CodP FROM FP
  WHERE CodF='F2'
```

Fornisce tutti i codice prodotto di prodotti rossi o forniti da F2.

INTERSECT

Definisce un risultato INTERSEZIONI delle istruzioni A,B.

Gode delle seguenti proprietà:
- **RICHIEDE COMPATIBILITA' DI SCHEMA**
- **PUO' ESSERE REALIZZATO CON JOIN E IN**

```
SELECT Sede FROM F
INTERSECT
SELECT Magazzino FROM P;
```

```
SELECT SedeFROM F, P
WHERE F.Sede=P.Magazzino;
```

Fornisce le sedi che sono anche magazzini, con equivalenza con una query di JOIN.

EXCEPT *Definisce un risultato sottraendo B al risultato di A.*

Gode delle seguenti proprietà:

- **RICHIEDE COMPATIBILITA' DI SCHEMA**
- **PUO' ESSERE REALIZZATO CON NOT-IN**

```
SELECT Sede FROM F
EXCEPT
SELECT Magazzino FROM P;
```

Fornisce le sedi che sono sedi di fornitori, ma non magazzino prodotti.
Equivalenza con il NOT IN di seguito

```
SELECT SedeFROM F
WHERE Sede NOT IN (SELECT Magazzino FROM P);
```

INSERT *Istruzione di selezione dati.*

INSERT INTO NomeTabella
[(ElencoColonne)]
VALUES (ElencoCostanti);

- **VINCOLI INTEGRITA' REFERENZIALE**
  Devono essere rispettati per la **coerenza della base dati.**

- **PUO' CONTENERE SUBQUERY**
  Inserisco valori risultanti da altre query.

```
INSERT INTO FORNITURE-TOTALI
  (CodP, TotQta)
  (SELECT CodP, SUM(Qta) FROM FP GROUP BY CodP);
```

- ***elenco_colonne***
  E' opzionale. se non messo, l'istruzione rischia di non funzionare a seguito di un **cambio di schema.**

CREATE

Istruzione di creazione della tabella

CREATE TABLE NomeTabella
(NomeAttributo Dominio [ValoreDiDefault] [Vincoli]
{ , NomeAttributo Dominio [ValoreDiDefault] [Vincoli]} AltriVincoli);

Istruzione di creazione della tabella

```
CREATE TABLE example(
 column1 INTEGER,
 column2 VARCHAR(50),
 column3 DATE NOT NULL,
 PRIMARY KEY (column1, column2)
);
```

CREATE TABLE NomeTabella
(NomeAttributo Dominio [ValoreDiDefault] [Vincoli]
{ , NomeAttributo Dominio [ValoreDiDefault] [Vincoli]} AltriVincoli);

- **VINCOLI INTEGRITA' REFERENZIALE**
  Devono essere rispettati per la **coerenza della base dati.**

- **DOMINIO**

  [CHARACTER SET NomeFamigliaCaratteri]
  BIT [VARYING] [(Lunghezza)]
  NUMERIC [(Precisione, Scala)]
  DECIMAL [(Precisione, Scala)]
  INTEGER
  SMALLINT
  (Precisione=digits - Scala=numero dopo la virgola)
  FLOAT [(n)]
  REAL
  DOUBLE PRECISION
  INTERVAL PrimaUnitàDiTempo [TO UltimaUnitàDiTempo]
  TIMESTAMP
  DATETIME

- **VALORI DI DEFAULT**
  < GenericoValore | USER | CURRENT_USER |SESSION_USER | SYSTEM_USER | NULL>

- **CREAZIONE DI DOMINIO**
  EX: CREATE DOMAIN Voto AS SMALLINT DEFAULT NULLCHECK (Voto >= 18 and Voto <=30)

-

- **VINCOLI INTEGRITA' REFERENZIALE**
  Devono essere rispettati per la **coerenza della base dati.**

- **DOMINIO**

  [CHARACTER SET NomeFamigliaCaratteri]
  BIT [VARYING] [(Lunghezza)]
  NUMERIC [(Precisione, Scala)]
  DECIMAL [(Precisione, Scala)]
  INTEGER

ALTER

Istruzione di modifica della tabella

ALTER TABLE NomeTabella
<
ADD COLUMN <Definizione-Attributo> |
ALTER COLUMN NomeAttributo
< SET <Definizione-Valore-Default> | DROP DEFAULT>|
DROP COLUMN NomeAttributo < CASCADE | RESTRICT > |
ADD CONSTRAINT[NomeVincolo] < definizione-vincolo-unique > |
< definizione-vincolo-integrità-referenziale > | < definizione-vincolo-check > |
DROP CONSTRAINT [NomeVincolo] < CASCADE | RESTRICT >

- **RESTRICT**
  Non rimuove l'elemento se è utilizzato in altre definizioni
- **CASCADE**
  Tutti gli elementi dipendenti dall'elemento in eliminazione vengono rimossi.

DROP

Istruzione di eliminazione della tabella

DROP TABLE NomeTabella [< CASCADE | RESTRICT >]

- **RESTRICT**
  Non rimuove l'elemento se è utilizzato in altre definizioni
- **CASCADE**
  Tutti gli elementi dipendenti dall'elemento in eliminazione vengono rimossi.

METADATI

*Informazioni sui dati, memorizzati all'interno del **dizionario dei dati.***

- **CONTIENE INFO SU TUTTE LE STRUTTURE**
  Elenco delle tabelle, colonne, indici e viste, **stored procedures, utenti e privilegi.**
  Può essere consultato con **istruzioni SQL.**

- **TABELLA**
  Nome e struttura fisica **della tabella**
  Nome e struttura degli **attributi**
  Nome degli **indici**
  Vincoli di **integrità**

```
SELECT Column_Name, Num_Distinct, Num_Nulls
FROM USER_TAB_COL_STATISTICS
WHERE Table_Name = 'FP'
ORDER BY Column_Name;
```

```
SELECT Table_Name, Num_Rows
FROM USER_TABLES;
```

Integrità dei dati

REGOLE DI INTEGRITA'

*I dati all'interno di una base di dati sono corretti se soddisfano un insieme di **regole di correttezza**.*
L'alterazione della base dati non sempre rispetta questi vincoli.

- **3 TIPI DI REGOLE**
  Procedure **applicative, vincoli di integrità, trigger.**

PROCEDURE APPLICATIVE

*Il controllo sui dati è all' **application level***

Gode delle seguenti proprietà:
- **APPROCCIO** molto efficiente
- **AGGIRARE** le verifiche agendo sul DBMS
- **CONOSCENZA** della base dati è insita nell'applicazione **embedded**

VINCOLI DI INTEGRITA'

Il controllo sui dati è a livello del DBMS

Gode delle seguenti proprietà:
- **DEFINIZIONE** tramite gli ALTER o CREATE
- **DICHIARATIVI**
- **CENTRALIZZAZIONE VERIFICA**
- **RALLENTANO** l'esecuzione dell'applicazione
- **NON POSSO DEFINIRE** tutti i tipi di dati (**EX: dati aggregati**)

Nello standard SQL-92 esistono:
VINCOLI DI TABELLA
VINCOLI DI INTEGRITA' REFERENZIALE

TRIGGER

*Il trigger, nelle basi di dati, è una **procedura** che viene eseguita in maniera automatica in coincidenza di un **determinato evento***

Gode delle seguenti proprietà:
- **DEFINIZIONE** tramite CREATE TRIGGER e memorizzati nel sistema.
- **VINCOLI COMPLESSI**
- **CENTRALIZZAZIONE VERIFICA**
- **COMPLESSI** a livello applicativo
- **RALLENTANO** l'esecuzione dell'applicazione

RIPARAZIONE DELLE VIOLAZIONI

Se un'applicazione tenta di eseguire un'operazione che violerebbe un vincolo, il sistema può:

- **IMPEDIRE L'OPERAZIONE**
- **GENERARE UN'AZIONE COMPENSATIVA**
  EX: quando si cancella un fornitore, cancellare anche tutte le sue forniture

SQL92 VINCOLI I TABELLA

Vincoli definiti su una o più colonne della tabella.

Tipologie di vincolo:
- **CHIAVE PRIMARIA**
- **VALORE NULLO**
- **UNICITA'**
- **GENERALI DI TUPLA**

A livello di sistema, si comportano in questo modo
- **VERIFICA**
  Dopo ogni istruzione di inserimento o modifica dati.Se il vincolo è violato viene generata **un'eccezione.**

CHIAVE PRIMARIA
Posso specificare una solo chiave primaria per tabella.
PRIMARY KEY (ElencoAttributi)

EX: CREATE TABLE FP (CodF CHAR(5), CodP CHAR(6), Qta INTEGER **PRIMARY KEY (CodF, CodP)**);

VALORE NULLO
Indica l'ammissibilità o meno del valore nullo (**default: ammesso)**
NomeAttributo Dominio NOT NULL

EX: CREATE TABLE F (CodF CHAR(5),NomeF CHAR(20) **NOT NULL**,NSoci SMALLINT, Sede CHAR(15));

UNICITA'
Un attributo o un insieme di attributo deve essere nullo
UNIQUE (ElencoAttributi)

EX: CREATE TABLE F (CodF CHAR(5),**UNIQUE** NomeF CHAR(20),NSoci SMALLINT, Sede CHAR(15));

CHIAVE CANDIDATA
Alcuni attributi possono essere chiavi candidate, **uniche e non nulle**
NomeAttributo Dominio UNIQUE NOT NULL

EX: CREATE TABLE F (CodF CHAR(5),**UNIQUE NOT NULL** NomeF CHAR(20),NSoci SMALLINT, Sede CHAR(15));

VINCOLO CHECK

Permettono di esprimere condizioni di tipo generale su ogni tupla

NomeAttributo Dominio CHECK (Condizione)

```
CREATE TABLE F (CodF    CHAR(5) PRIMARY KEY,
                NomeF   CHAR(20) NOT NULL,
                NSoci   SMALLINT
                        CHECK (NSoci>0),
                Sede    CHAR(15));
```

SQL92 VINCOLI REFERENZIALI

Vincoli definiti con legami tra tabelle della base dati

Tipologie di vincolo:

- **FOREIGN KEY**

FOREIGN KEY

Permette di relazionare le **chiavi di tabelle distinte.**

FOREIGN KEY (ElencoAttributiReferenzianti)
    REFERENCES NomeTabella [(ElencoAttributiReferenziati)]

Se gli attributi referenziati hanno **lo stesso nome di quelli referenzianti**, **non è obbligatorio specificarli**

EX: CREATE TABLE FP (CodF CHAR(5), CodP CHAR(6),
Qta INTEGER,
PRIMARY KEY (CodF, CodP), FOREIGN KEY (CodF)
REFERENCES F(CodF), FOREIGN KEY (CodP)
REFERENCES P(CodP));

Gestione delle transazioni

TRANSAZIONE

*Sequenza di operazioni che rappresenta un lavoro elementare, che si conclude con un **successo o insuccesso.***

Tipologia di ritorno.

- **SUCCESSO**
  Consolido l'aggiornamento
  COMMIT[WORK]

- **INSUCCESSO**
  Rollbackdella base dati.

START TRANSACTION
[...]
COMMIT [WORK] OR ROLLBACK [WORK]

```
START TRANSACTION; UPDATE Conto-Corrente
SET Saldo = Saldo + 100
WHERE IBAN='IT92X0108201004300000322229'; UPDATE Conto-Corrente
SET Saldo = Saldo - 100
WHERE IBAN= 'IT32L0201601002410000278976'; DBG COMMIT;
```

Proprietà delle transazioni **ACID**

A. **ATOMICITA'**
   La transazione è indivisibile: **devono essere eseguite tutte le sottoperazioni.** NO STATO INTERMEDIO

B. **CONSISTENZA**
   Lo stato iniziale e finale devono essere **consistenti.**
   VINCOLI DI INTEGRITA'

C. **ISOLAZIONE**
   Gli stati **intermedi non sono visibili e accessibili.**

D. **DURABILITA'**
   Lo stato finale **persistente.**

Modello ER

FASI DELLA PROGETTAZIONE

La progettazione di una base di dati è una delle attività del processo di sviluppo di un sistema informativo

- **STUDIO DI FATTIBILITA'**

- **RACCOLTA E ANALISI REQUISITI**
  Raccolta informale di informazioni utili a definire **funzionalità e proprietà** dell'applicazione

- **PROGETTAZIONE (CONCETTUALE E LOGICOFISICA)**
  Nella fase **concettuale** si rappresenta il **contenuto informativo della base dati,** indipendentemente dagli aspetti implementativi.
  Nella fase **logica** si si rappresenta l'organizzazione logica delle informazioni, la cui qualità si verifica con tecniche formali **normalizzazione**
  Nella fase **fisica** si produce un modello fisico che **dipende dal DBMS.**

- **PROTOTIPAZIONE**

- **IMPLEMENTAZIONE**

- **VALIDAZIONE E TESTING**

- **FUNZIONAMENTO**

Ogni fase si può suddividere nei seguenti passaggi:
- **DECOMPOSIZIONE PROBLEMA**
- **STRATEGIE PER AFFRONTARE IL PROBLEMA**
- **CREAZIONE DEL MODELLO**

Nelle basi dati, la metodologia **più coerente di progettazione,** consiste nel suddividere il problema in:
- **COSA RAPPRESENTARE - CONCETTUALE**
- **COME RAPPRESENTARE - LOGICO/FISICO**

MODELLO ER

*Il modello ER è un **modello per la rappresentazione concettuale** dei dati ad un alto livello di astrazione, formalizzato dal prof. Peter Chen nel 1976*

- **FORMALE**
- **INDIPENDETE DAL DBMS**
- **MODELLO GRAFICO**

Principali costrutti del modello ER:
- **ENTITA'**
- **RELAZIONI**
- **ATTRIBUTI**
- **IDENTIFICATORI**
- **GENERALIZZAZIONI E SOTTOINSIEMI**

ENTITA'

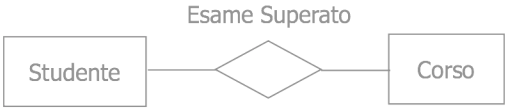
*Rappresentano **classi di oggetti** (fatti, cose, persone, ...) che hanno **proprietà comuni** ed **esistenza autonoma** ai fini dell'applicazione di interesse.*

- **PROPRIETA' COMUNI**
- **ESISTENZA AUTONOMA**
  Un'interessante conseguenza di questo fatto è che un'occorrenza di entità ha un'**esistenza indipendente dalle proprietà ad essa associate**.
- **PUO' ESSERE GENERALIZZATA**
  Persona - DONNA e UOMO (SituazioneMilitare)

RELAZIONE

*Le relazioni (dette anche associazioni) rappresentano un **legame tra due o più entità**.*

Esempio di relazione tra entità:



OCCORRENZA

Un'occorrenza di una relazione **è una n-upla** (coppia nel caso di relazione binaria) costituita da **occorrenze di entità**, una per ciascuna delle entità coinvolte

CARDINALITA'

Vengono specificate per ciascuna entità che partecipa a una relazione e **dicono quante volte**, in una relazione tra entità, **un'occorrenza di una di queste entità può essere legata ad occorrenze delle altre** entità coinvolte nella relazione

(MIN,MAX)

MIN: **0 - 1** (obbligatorio o meno)
MAX: **1 - N** (univoca o 1:N)

ATTENZIONE:

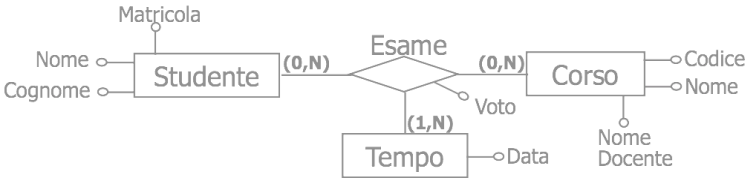
Le cardinalità **minime raramente sono 1 per tutte le entità** coinvolte in una relazione
Le **cardinalità massime di una relazione n-aria sono (praticamente) sempre N**

ATRRIBUTI

*Le entità e le relazioni possono essere descritte **usando una serie di attributi.** Tutti gli oggetti della stessa classe entità (associazione) hanno gli stessi attributi*

Nome attributo

- **ENTITA' UGUALI HANNO ATTRIBUTI UGUALI**
- **DOMINIO**



DOMINIO

Insieme di **valori ammissibili** per l'attributo.

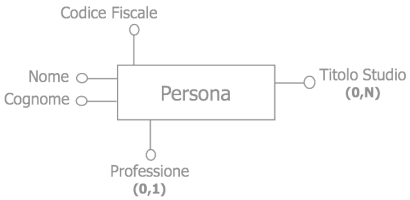
ATTRIBUTO COMPOSTO

Attributo composto da **sotto-attributi**

CARDINALITA'

Vincoli di cardinalità per il singolo attributo:

- **INDICARE OPZIONALITA'** **(0,1)**
- **INDICARE ATTRIBUTI MULTIVALORE** **(0,N)**



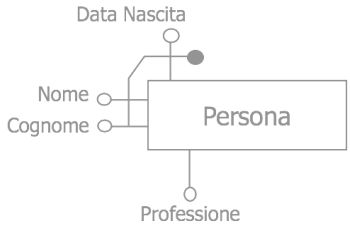
IDENTIFICATORI

*Costituiscono un sottoinsieme degli attributi di un'entità che identificano in maniera univoca **ogni occorrenza della stessa entità**.*

- **OGNI ENTITA' DEVE AVERE UN IDENTIFICATORE**
  Se non dispone di una chiave, si dice **entità debole.**
  L'entità debole deve partecipare con **cardinalità 1-1 in ognuna delle relazioni che forniscono parte dell'identificatore**
- **SONO AMMESSI PIU' IDENTIFICATORI**
- **LE RELAZIONI NON HANNO IDENTIFICATORI**

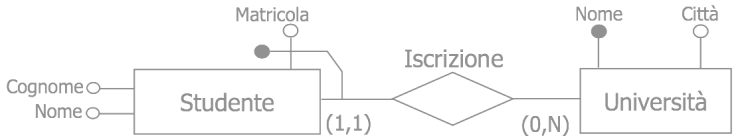
SEMPLICE O COMPOSTO

Può essere costituito da **uno o più attributi**



ENTITA' DEBOLE

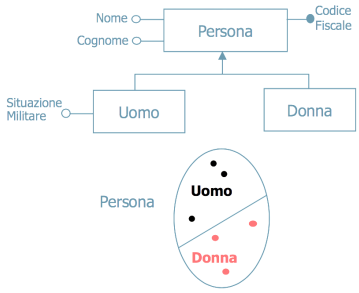
Entità che non possiede **internamente attributi identificatori sufficienti.**
L'entità debole deve **partecipare con cardinalità (1,1)** in ognuna delle **relazioni che forniscono parte dell'identificatore**



GENERALIZZAZIONI

Rappresentano dei legami logici esistenti tra due o più entità. Tra le entità coinvolte si distinguono:

- **UNICA ENTITA' PADRE**
- **UNA O PIU' ENTITA FIGLIE**



Proprietà della generalizzazione:

- **OCCORRENZA FIGLIA**
  E' anche occorrenza dell'**entità padre**
- **PROPRIETA' ENTITA' PADRE**
  E' anche proprietà dell'**entità figlia**
- **UN ENTITA' PUO' AVERE PIU' GENERALIZZAZIONI**

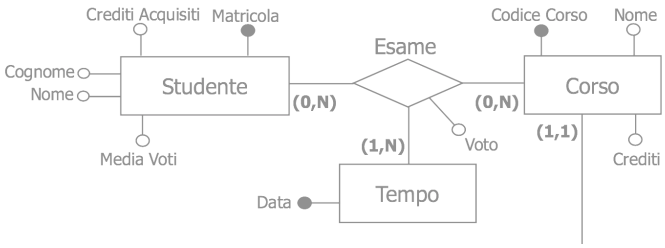
Proprietà ORTOGONALI della generalizzazione:

- **TOTALE O PARZIALE**
  Una generalizzazione è **totale** quando **padre =unione figli**
- **ESCLUSIVA O SOVRAPPOSTA**
  si dice esclusiva quando l'intersezione dei sottoinsiemi dei figli è vuota

DOCS

Il modello ER viene arricchito con ***dizionario e schema dei vincoli di integrità.***

GESTIONE DEL TEMPO



Gestione delle viste

DEFINIZIONE

Una vista è rappresentata da una query (SELECT), il cui risultato può essere utilizzato come se fosse una tabella.

```
(CREATE | ALTER | DROP) VIEW NomeVista
    [(ElencoAttributi) ] AS InterrogazioneSQL
    [WITH [LOCAL|CASCADED] CHECK OPTION];
```

- **RISULTATO RICALCOLATO**
  Ad ogni apertura della vista
- **OGGETTO INTERROGABILE**
  Come se **fosse una tabella**
- **DECOMPOSIZIONE PROBLEMA**
  Problemi complessi vengono scomposti tramite l'utilizzo delle viste.

- **DEFINIZIONE ATTRIBUTI**

```
CREATE VIEW NUMFORNITORI_PER_PRODOTTO
   (CodP, NumFornitori) AS
   SELECT CodP, COUNT(*) FROM FP GROUP BY CodP;
```

  E' necessario definirli se **sono risultato di espressioni, funzioni, costanti o due colonne hanno lo stesso nome.**

- **AGGIORNABILITA' VISTE**
  sono aggiornabili le viste in cui una sola riga di ciascuna tabella di base corrisponde a una sola riga della vista **CORRISPONDENZA UNIVOCA**

- **CHECK OPTION**
  Vieta operazioni di inserimento e aggiornamento per le quali **una tupla possa diventare non visibile.**

- **CASCADED | LOCAL**
  Se la vista è definita tramite altre viste, il controllo check-option può venire effettuato sulla vista più esterna **LOCAL** o su tutte con **CASCADED**

NON SONO AGGIORNABILI:

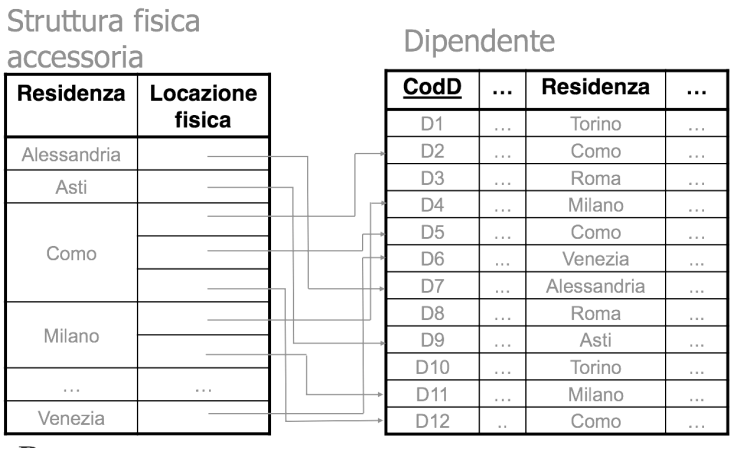
- **NO CHIAVE PRIMARIA**
  Nella vista
- **JOIN MULTIPLI**
  Corrispondenze 1:n o n:n. Risolvo realizzando il JOIN con **IN**
- **FUNZIONI AGGREGATE**
- **DISTINCT**

Gestione degli indici

DEFINIZIONE

"We make datas faster" ®

- **HASH TABLE | TREE**
  Strutture di organizzazione degli indici.
- **ORGANIZZAZIONE FISICA**
  l'organizzazione fisica dei dati all'interno di un file influenza il tempo di accesso alle informazioni
- **CREAZIONI INDICI**
  Per evitare grande carico dovuto alle letture sequenziali di tutta la tabella interrogata.



- **LOCAZIONE FISICA**
  Permette di **accedere ad un dato direttamente**, conoscendo il suo indirizzo!
- **STRUTTURE FISICHE**
  Le strutture fisiche di accesso descrivono il **modo in cui i dati sono organizzati** in **memoria secondaria** per garantire operazioni di ricerca e modifica dei dati efficienti

NON SONO AGGIORNABILI:
- **OCCUPAZIONE MAGGIORE SPAZIO**
- **POCHI INDICI**

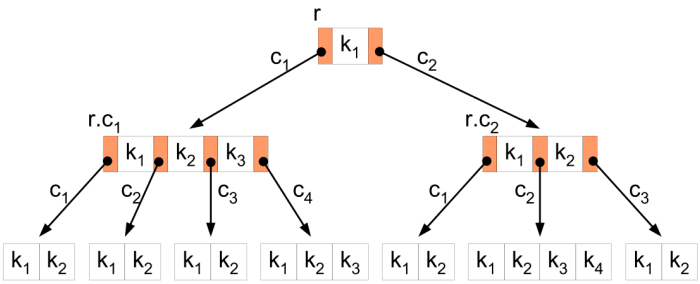
STRUTTURA SEQUENZIALE

Disposizione sequenziale dei record all'interno del file

- **BLOCCHI DI MEMORIA CONSECUTIVA**
- **CHIAVE DI ORDINAMENTO**
  Composta da uno o più attributi

STRUTTURA AD ALBERO

Disposizione TREE o BTREE+

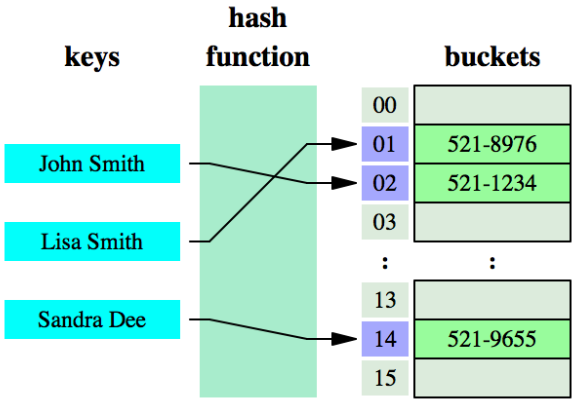


- **BLOCCHI DI MEMORIA AD INDIRIZZO**
- **CHIAVE DI ORDINAMENTO**
  Composta da uno o più attributi

STRUTTURA AD ACCESSO CALCOLATO

I record sono inseriti nel file nell'ordine determinato applicando una funzione di hash ai valori di un campo, detto campo di hash

HASH TABLE - HASH MAP



- **HASH FUNCTION**
  Che riceve come parametro la **chiave di ordinamento**
- **CHIAVE DI ORDINAMENTO**
  Composta da uno o più attributi

DEFINIZIONE SQL

- **STRUTTURA FISICA**
  Ordinata o meno
- **ATTRIBUTI INDICIZZATI**
- **DEFINIZIONE TIPO INDICE**
- **VARIAZIONI DELLO SCHEMA**

Normalizzazione

DEFINIZIONE

*" La normalizzazione è un procedimento volto **all'eliminazione della ridondanza informativa** e del **rischio di incoerenza** dal database."*

ESEMPIO

MatrStudente	Residenza	CodCorso	NomeCorso	Voto
s94539	Milano	04FLYCY	Calcolatori elettronici	30
s94540	Torino	01FLTCY	Basi di dati	26
s94540	Torino	01KPNCY	Reti di calcolatori	28
s94541	Pescara	01KPNCY	Reti di calcolatori	29
s94542	Lecce	04FLYCY	Calcolatori elettronici	25

Nella tabella, NomeCorso è **ridondante**, così come la residenza dello studente. Inoltre, se la residenza studente cambiasse, vi sarebbe **rischio di incoerenza.**

ESISTONO VARI TIPI DI NORMALIZZAZIONE

- **PRIMO-SECONDO-TERZO TIPO**
- **BOYCE-CODD**

CAUSA DELLE ANOMALIE:

- **CONCETTI INDPENDENTI**
  Tra di loro all'interno della stessa relazione
- **DIPENDENZE FUNZIONALI**
  X->Y che permettono la presenza di **più tuple con lo stesso valore di X**

DIPENDENZA FUNZIONALE

*" La dipendenza funzionale è un **vincolo d'integrità** definito **sugli attributi tra le tuple**"*

Una **relazione r** soddisfa la **dipendenza funzionale X ->Y** se, per ogni coppia **t1, t2 di tuple** di r, aventi **gli stessi valori per gli attributi in X**, t1 e t2 hanno **gli stessi valori anche per gli attributi in Y**

Esempi

MatrStudente → Residenza

MatrStudente CodCorso → NomeCorso

FORMA BOYCE-CODD

*"Una **relazione r** è in BCNF se, per ogni dipendenza funzionale (non banale) X -> Y definita su di essa, **X contiene una chiave di r** (X è superchiave di r) "*

NORMALIZZAZIONE

*"Processo di **sostituzione di una relazione non normalizzata** con due o più relazioni in BCNF"*

CRITERIO

una relazione che rappresenta **più concetti indipendenti** è **decomposta in relazioni più piccole**, una per ogni concetto, per mezzo delle dipendenze funzionali

- Da

  R (MatrStudente, Residenza, CodCorso, NomeCorso, Voto)

- Le relazioni in BCNF sono

  R_1 (MatrStudente, Residenza) = $\pi_{\text{MatrStudente, Residenza}}$ R

  R_2 (CodCorso, NomeCorso) = $\pi_{\text{CodCorso, NomeCorso}}$ R

  R_3 (MatrStudente, CodCorso, Voto) = $\pi_{\text{MatrStudente, CodCorso, Voto}}$ R

R_1

MatrStudente	Residenza
s94539	Milano
s94540	Torino
s94541	Pescara
s94542	Lecce

R_2

CodCorso	NomeCorso
04FLYCY	Calcolatori elettronici
01FLTCY	Basi di dati
01KPNCY	Reti di calcolatori

R_3

MatrStudente	CodCorso	Voto
s94539	04FLYCY	30
s94540	01FLTCY	26
s94540	01KPNCY	28
s94541	01KPNCY	29
s94542	04FLYCY	25

DECOMPOSIZIZ. SENZA PERDITA

DECOMPOSIZIONE SENZA PERDITA

La decomposizione di una **relazione r** su due insiemi di attributi **X1 e X2** è senza perdita di informazione se **il join delle proiezioni di r su X1 e X2 è uguale a r stessa**

TUTTE LE DECOMPOSIZIONI DI NORMALIZZAZIONE DEVONO:

- ESSERE SENZA PERDITA
- PRESERVARE LE DIPENDENZE FUNZIONALI

Note finali

Alcuni dei contenuti presenti nelle seguenti dispense sono stati liberamente tratti dai materiali didattici disponibili al Politecnico di Torino.
Le dispense sono state elaborate dal sottoscritto come complemento allo studio e non intendono in alcun modo sostituire la completezza dei libri di testo e delle lezioni dalle quali sono state liberamente tratte.
Le dispense sono state scritte per l'esame di Basi di Dati dell'A.A. 2016-2017, docente Elena Baralis, corso di laurea in Ingegneria Gestionale L8.
E' doveroso quindi citare alcuni delle fonti da cui sono stati liberamente tratti alcune parti di esercizi e/o metodologie di soluzione:

- Elena Maria Baralis, Luca Cagliero, Whiteboard e appunti del corso di Basi di Dati, A.A. 2016-2017.
- wikipedia.org