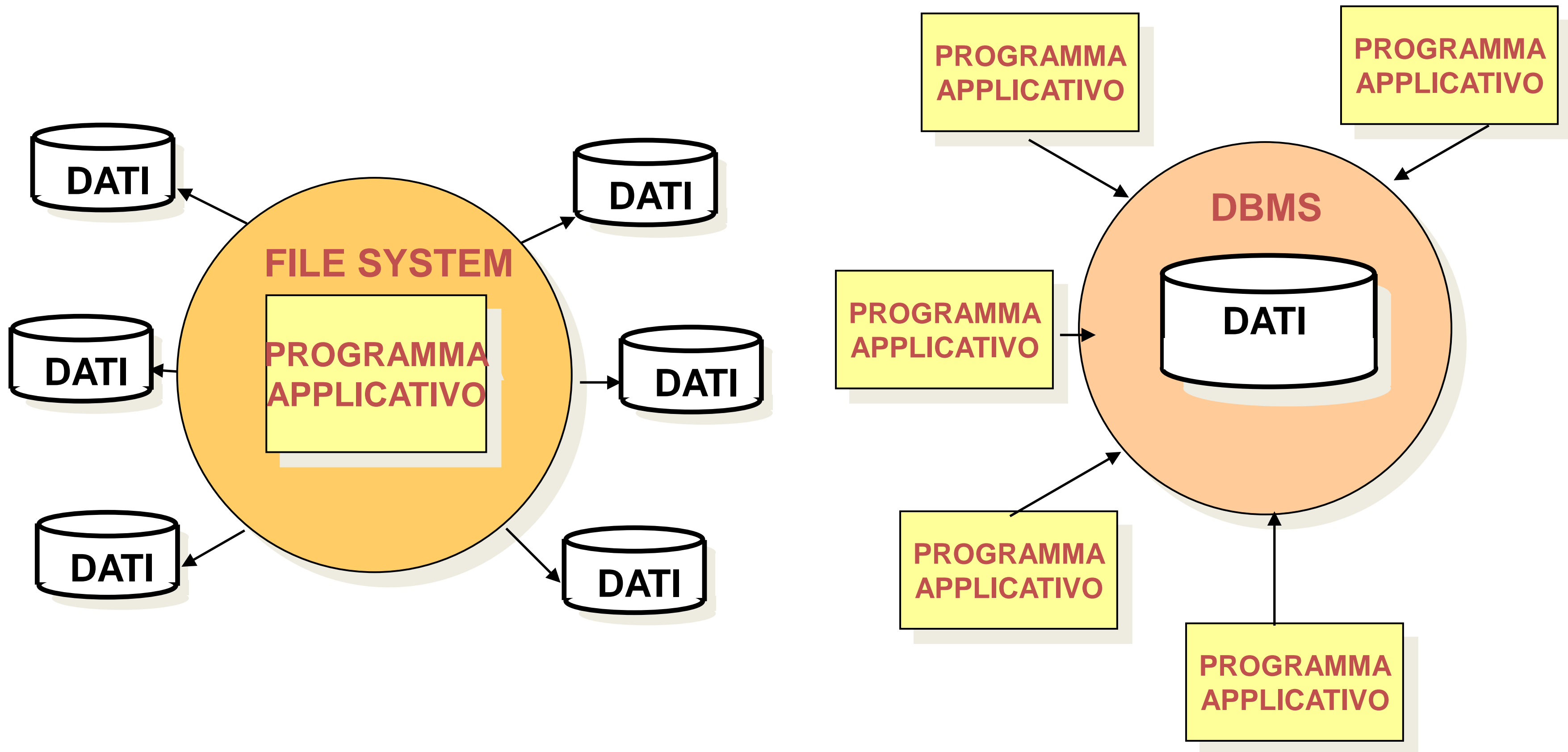
Introduzione
alle Basi di Dati

Sides Prof. Alessandro Campi

Materiale tratto da…

- Teoria: bastano le slide
  - Materiale costruito partendo da:
    - Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Riccardo Torlone
      Basi di dati: Modelli e linguaggi di interrogazione
- Esercizi: bastano le slide
  - Riusati identici in:
    - Danile Braga, Marco Brambilla, Alessandro Campi
      Eserciziario basi di dati
      Editrice Esculapio

BASE DI DATI E FILE SYSTEM A CONFRONTO



Principali caratteristiche dei DBMS

- **condivisione dei dati**
  - assenza di replicazione nei file
  - concorrenza
- **qualità dei dati**
  - vincoli di integrità
- **efficienza**
  - caricamento, query, sort
- **controllo dell'accesso**
  - privatezza
- **robustezza**

L'elemento base: la tabella

studente

MATR	NOME	CITTA'	CCS
123	Carlo	Bologna	Inf
415	Paola	Torino	Inf
702	Antonio	Roma	Ges

Il modello relazionale

Cronologia del modello relazionale

- **Inventato da T. Codd, 1970**
  (IBM Research di Santa Teresa, Cal)
- **Primi progetti:**
  SYSTEM R (IBM), Ingres (Berkeley Un.)
- **Principali scoperte tecnologiche: 1978-1980**
- **Primi sistemi commerciali:**
  inizio anni ‘80 (Oracle, IBM-SQL DS e DB2, Ingres, Informix, Sybase)
- **Successo commerciale: dal 1985.**

Definizione informale

colonna

schema

studente

MATR	NOME	CITTA'	INDS
123	Carlo	Bologna	Inf
107	Giovanni	Milano	Ges
415	Paola	Torino	Inf
702	Antonio	Roma	Ges

istanza

riga

Definizione formale

- **Dominio D: qualunque insieme di valori**

  D1, D2, …, Dn (n insiemi anche non distinti)
- Il prodotto cartesiano D1×D2×…×Dn, è l'insieme di tutte le n-uple ordinate <d1, d2, …, dn> tali che

  $d1 \in D1, d2 \in D2, \ldots, dn \in Dn$.
- Una relazione matematica su D1, D2, …, Dn è un sottoinsieme del prodotto cartesiano D1×D2×…×Dn.
- D1, D2, …, Dn sono i domini della relazione. Una relazione su n domini ha grado n .
- Il numero di n-uple è la cardinalità della relazione. Nelle applicazioni reali, la cardinalità è sempre finita.

Esempio

- D_1 = (a,b)
- D_2 = (1,2,3)

- D_1 x D_2 = (<a,1>, <b,1>, <a,2>, <b,2>,
  <a,3>, <b,3>)

- R1 = (<a,1>, <b,3>)
- R2 = (<c,1>, <b,3>, <a,2>)
- R3 = ( )
- R4 = (<a,1>, <b,1>, <a,2>, <b,2>,
  <a,3>, <b,3>)

Proprietà

- **Grado della relazione:**

  numero di domini (n)

- **Cardinalità della relazione:**

  numero di n-uple (o tuple)

- **Attributo:**

  nome dato al dominio in una relazione

  I nomi di attributo in una relazione devono essere tutti distinti fra loro

Proprietà

In base alle definizioni, una relazione matematica è un **insieme** di n-uple **ordinate**:

<d1, d2, …, dn> tali che d1 $\in$ D1, d2 $\in$ D2, …, dn $\in$ Dn

Una relazione è un **insieme**; quindi:

- non è definito alcun ordinamento fra le n-uple
- le n-uple di una relazione sono distinte l'una dall'altra
- le n-uple sono internamente **ordinate**: l'i-esimo valore di ciascuna proviene dall' i-esimo dominio (è cioè definito un ordinamento fra i domini)
- i riferimenti fra dati in relazioni diverse sono rappresentati per mezzo di valori dei domini che compaiono nelle ennuple.

studenti

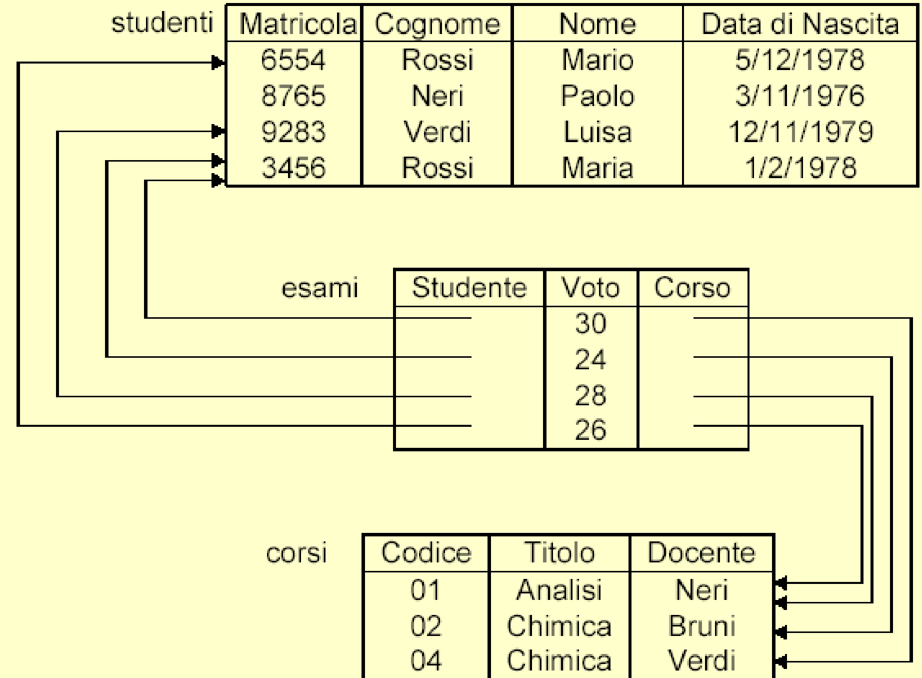
Matricola	Cognome	Nome	Data di Nascita
6554	Rossi	Mario	5/12/1978
8765	Neri	Paolo	3/11/1976
9283	Verdi	Luisa	12/11/1979
3456	Rossi	Maria	1/2/1978

esami

Studente	Voto	Corso
3456	30	04
3456	24	02
9283	28	01
6554	26	01

corsi

Codice	Titolo	Docente
01	Analisi	Neri
02	Chimica	Bruni
04	Chimica	Verdi



studenti

Matricola	Cognome	Nome	Data di Nascita
6554	Rossi	Mario	5/12/1978
8765	Neri	Paolo	3/11/1976
9283	Verdi	Luisa	12/11/1979
3456	Rossi	Maria	1/2/1978

esami

Studente	Voto	Corso
	30	
	24	
	28	
	26	

corsi

Codice	Titolo	Docente
01	Analisi	Neri
02	Chimica	Bruni
04	Chimica	Verdi

Perché sui valori?

- Indipendenza dalle strutture fisiche (si potrebbe avere anche con puntatori di alto livello) che possono cambiare anche dinamicamente
- Si rappresenta solo ciò che è rilevante dal punto di vista dell'applicazione (dell'utente); i puntatori sono meno comprensibili per l'utente finale (senza, l'utente finale vede gli stessi dati dei programmatori)
- I dati sono portabili più facilmente da un sistema ad un altro
- I puntatori sono direzionali

Informazione incompleta

- Il modello relazionale impone ai dati una struttura rigida:
  - le informazioni sono rappresentate per mezzo di ennuple
  - solo alcuni formati di ennuple sono ammessi: quelli che corrispondono agli schemi di relazione
- I dati disponibili possono non corrispondere esattamente al formato previsto, per varie ragioni.

Informazione incompleta

Città	Prefettura
Roma	Via IV novembre
Firenze	
Tivoli	
Prato	

- Firenze è provincia, ma non conosciamo l'indirizzo della prefettura
- Tivoli non è provincia: non ha prefettura
- Prato è “nuova” provincia: ha la prefettura?

NULL

- Tre casi differenti
  - **valore sconosciuto:** esiste un valore del dominio,ma non è noto (Firenze)
  - **valore inesistente:** non esiste un valore del dominio (Tivoli)
  - **valore senza informazione:** non è noto se esista o meno un valore del dominio (Prato)
- I DBMS non distinguono i tipi di valore nullo (e quindi implicitamente adottano il valore **senza informazione**)

Informazione incompleta

Studenti

Matricola	Cognome	Nome	Nascita
276545	Rossi	Maria	NULL
NULL	Neri	Anna	23/04/1972
NULL	Verdi	Fabio	12/02/1972

Esami

Studente	Voto	Corso
276545	28	01
NULL	27	NULL
200768	24	NULL

Corsi

Codice	Titolo	Docente
01	Analisi	Giani
03	NULL	NULL
NULL	Chimica	Belli

Vincoli di integrità

- Esistono istanze di basi di dati che, pur sintatticamente corrette, non rappresentano informazioni possibili per l'applicazione di interesse.

Esami

Studente	Voto	Lode	Corso
276545	28	e lode	01
276545	32		02
788854	23		03
200768	30	e lode	03

Vincoli di integrità

- Definizione
  - proprietà che deve essere soddisfatta dalle istanze che rappresentano informazioni corrette per l'applicazione ogni vincolo può essere visto come una funzione booleana (o un predicato) che associa ad ogni istanza il valore **vero** o **falso**.
- Tipi di vincoli:
  - vincoli intrarelazionali; casi particolari:
    - vincoli su valori (o di dominio)
    - vincoli di ennupla
  - vincoli interrelazionali

Vincoli di ennupla

- Esprimono condizioni sui valori di ciascuna ennupla, indipendentemente dalle altre ennuple.
- Una possibile sintassi: espressione booleana (con AND, OR e NOT) di atomi che confrontano valori di attributo o espressioni aritmetiche su di essi.
- Un vincolo di ennupla è un **vincolo di dominio** se coinvolge un solo attributo
- Esempi:
  - (Voto >= 18) AND (Voto <= 30)
  - (Voto =30) OR NOT (Lode = “e lode”)
  - Lordo = (Ritenute + Netto)

Nozione di chiave

Sottoinsieme degli attributi dello schema che ha la proprieta' di unicita' e minimalita'

unicità: non esistono due tuple con chiave uguale
minimalità: sottraendo un qualunque attributo alla chiave si perde la proprieta' di unicità

Se il sottoinsieme non è minimale si parla di SUPERCHIAVE

Chiavi nell'esempio : gestione degli esami universitari

studente

<u>MATR</u>	NOME	CITTA’	CCS

esame

<u>*MATR*</u>	<u>*COD-CORSO*</u>	DATA	VOTO

corso

<u>COD-CORSO</u>	TITOLO	DOCENTE

Esiste sempre una chiave?

- Poiché le relazioni sono insiemi, ogni relazione non può contenere ennuple distinte ma uguali fra loro:
  - ogni relazione ha come superchiave l’insieme degli attributi su cui è definita;
- Poiché l’insieme di tutti gli attributi è una superchiave per ogni relazione, ogni schema di relazione ha tale insieme come superchiave;
- Poiché l’insieme di attributi è finito, ogni schema di relazione ha (almeno) una chiave

Chiavi e valori nulli

- In presenza di valori nulli, i valori degli attributi che formano la chiave
  - non permettono di identificare le ennuple come desiderato
  - né permettono di realizzare facilmente i riferimenti da altre relazioni
- La presenza di valori nulli nelle chiavi deve essere limitata
- Soluzione pratica: per ogni relazione scegliamo una chiave (la chiave primaria) su cui non ammettiamo valori nulli

Foreign Key

- Informazioni in relazioni diverse sono correlate attraverso valori comuni
- In particolare, valori delle chiavi (primarie, di solito)
- Un **vincolo di integrità referenziale** fra un insieme di attributi X di una relazione R1 e un'altra relazione R2 impone ai valori su X di ciascuna ennupla dell'istanza di R1 di comparire come valori della chiave (primaria) dell'istanza di R2

Chiavi esterne: esempio

studente

MATR	NOME	CITTA’	CCS

esame

MATR	*COD-CORSO*	DATA	VOTO

corso

COD-CORSO	TITOLO	DOCENTE

Integrità referenziale

- Esprime un legame gerarchico (padre-figlio) fra tabelle
- Alcuni attributi della tabella figlio sono definiti FOREIGN KEY
- I valori contenuti nella FOREIGN KEY devono essere sempre presenti nella tabella padre

SQL

SQL

- Il nome sta per *Structured Query Language*
- Le interrogazioni SQL sono dichiarative
  - l’utente specifica quale informazione è di suo interesse, ma non come estrarla dai dati
- Le interrogazioni vengono tradotte dall’ottimizzatore (query optimizer) nel linguaggio procedurale interno al DBMS
- Il programmatore si focalizza sulla leggibilità, non sull’efficienza
- È l'aspetto più qualificante delle basi di dati relazionali

Interrogazioni SQL

- Le interrogazioni SQL hanno una struttura `select-from-where`
- Sintassi:

  `select` *AttrEspr* {, *AttrEspr*}
  `from` *Tabella* {, *Tabella*}
  [`where` *Condizione*]

- Le tre parti della query sono chiamate:
  - clausola `select` / target list
  - clausola `from`
  - clausola `where`
- La query effettua il prodotto cartesiano delle tabelle nella clausola `from`, considera solo le righe che soddisfano la condizione nella clausola `where` e per ogni riga valuta le espressioni nella `select`
- Sintassi completa:

  `select` *AttrEspr* [[`as`] *Alias*] {, *AttrEspr* [[`as`] *Alias*] }
  `from` *Tabella* [[`as`] *Alias*] {, *Tabella* [[`as`] *Alias*] }
  [`where` *Condizione*]

Esempio:
gestione degli esami universitari

Studente

MATR	NOME	CITTA’	CCS
123	Carlo	Bologna	Inf
415	Alex	Torino	Inf
702	Antonio	Roma	Ges

Esame

MATR	COD-CORSO	DATA	VOTO
123	1	7-9-23	30
123	2	8-1-23	28
702	2	7-9-23	20

Corso

COD-CORSO	TITOLO	DOCENTE
1	matematica	Barozzi
2	informatica	Meo

Proiezione

SELECT Nome, CCS

FROM STUDENTE

è una tabella con

- schema :
  gli attributi Nome e CCS (grado <=)
- istanza :
  la restrizione delle tuple sugli attributi
  Nome e CCS (cardinalità <=)

Nome	CCS
Carlo	Inf
Alex	Inf
Antonio	Ges

Proiezione

SELECT *

FROM STUDENTE

Prende tutte le colonne della tabella STUDENTE

Duplicati

- In SQL, le tabelle prodotte dalle interrogazioni possono contenere più righe identiche tra loro
- I duplicati possono essere rimossi usando la parola chiave `distinct`

```
Select distinct CCS
from Studente
```

CCS
Inf
Ges

```
select CCS
from Studente
```

CCS
Inf
Inf
Ges

Selezione

```
SELECT *
FROM STUDENTE
WHERE Nome='Alex'
```

È una tabella con

- schema: lo stesso schema di STUDENTE (grado =)
- istanza: le tuple di STUDENTE che soddisfano il predicato di selezione (cardinalità <=)

Matr	Nome	Città	CCS
415	Alex	Torino	Inf

Sintassi del predicato di selezione

espressione booleana di predicati semplici

operazioni booleane :

- AND (P1 AND P2)
- OR (P1 OR P2)
- NOT (P1)

predicati semplici :

- TRUE, FALSE
- termine
  comparatore
  termine

comparatore :

- =, <>, <, <=, >, >=

termine :

- costante, attributo
- espressione
  aritmetica di
  costanti e attributi

Sintassi della clausola `where`

- Espressione booleana di predicati semplici
- Estrarre gli studenti di informatica originari di Bologna:

```
select *
from Studente
where CCS = 'Inf' and Città = 'Bologna'
```

- Estrarre gli studenti originari di Bologna o di Torino:

```
select *
from Studente
where Città = 'Bologna' or Città = 'Torino'
```

  – Attenzione: estrarre gli studenti originari di Bologna e originari di Torino

Espressioni booleane

- Estrarre gli studenti originari di Roma che frequentano il corso in Informatica o in Gestionale:

```
select *
from Studente
where Città = 'Roma' and
      (CCS = 'Inf' or
       CCS = 'Ges')
```

- Risultato:

Matr	Nome	Città	CCS
702	Antonio	Roma	Ges

Gestione dei valori nulli

- Per fare una verifica sui valori nulli:

  Attributo **is** [**not**] **null**

```
select *
from Studente
where CCS = 'Inf' or CCS <> 'Inf'
```

è equivalente a:

```
select *
from Studente
where CCS is not null
```

Esempio di selezione

SELECT *
FROM STUDENTE
WHERE (Città='Torino') OR ((Città='Roma') AND NOT (CCS='Ges'))

MATR	NOME	CITTA'	CCS
~~123~~	~~Carlo~~	~~Bologna~~	~~Inf~~
415	Alex	Torino	Inf
~~702~~	~~Antonio~~	~~Roma~~	~~Ges~~

Selezione e proiezione

Matr	Nome	Città	CCS
123	Carlo	Bologna	Inf
415	Alex	Torino	Inf
702	Antonio	Roma	Ges

- Estrarre il nome degli studenti iscritti al corso in informatica?

```
SELECT Nome
FROM STUDENTE
WHERE CCS='Inf'
```

NOME
Carlo
Alex

Selezione e proiezione

Matr	Nome	Città	CCS
123	Carlo	Bologna	Inf
415	Alex	Torino	Inf
702	Antonio	Roma	Ges

- Nome degli studenti di Gestionale non di Milano

SELECT NOME

FROM STUDENTE

WHERE CCS='Ges' AND Città<>'Milano'

NOME
Antonio

https://sqlzoo.net/wiki/SELECT_basics
https://sqlzoo.net/wiki/SELECT_from_WORLD_Tutorial
https://sqlzoo.net/wiki/SELECT_from_Nobel_Tutorial

Prodotto cartesiano

R , S

è una tabella (priva di nome) con

- schema :

  gli attributi di R e S
  (grado(RxS)= grado(R)+grado(S))

- istanza :

  tutte le possibili coppie di tuple di R e S
  (card(RxS)=card(R)*card(S))

Esempio

R(A,B)

A	B
a	1
b	3

S(C,D)

C	D
c	1
d	5
a	2

R,S (A,B,C,D)

A	B	C	D
a	1	c	1
a	1	d	5
a	1	a	2
b	3	c	1
b	3	d	5
b	3	a	2

```
Select *
FROM R,S
```

Prodotto cartesiano con condizione →Join

```
SELECT *
FROM STUDENTE , ESAME
WHERE STUDENTE.Matr=ESAME.Matr
```

Join

FROM STUDENTE JOIN ESAME

ON STUDENTE.Matr=ESAME.Matr

è equivalente alla seguente espressione (operatore derivato):

FROM STUDENTE , ESAME

WHERE STUDENTE.Matr=ESAME.Matr

attributi omonimi sono resi non ambigui usando la notazione “puntata”:

ESAME.Matr

STUDENTE.Matr

Join

FROM STUDENTE JOIN ESAME

ON STUDENTE.Matr=ESAME.Matr

produce una tabella con

- **schema:** la concatenazione degli schemi di STUDENTE e ESAME
- **istanza:** le tuple ottenute concatenando quelle tuple di STUDENTE e di ESAME che soddisfano il predicato

STUDENTE. Matr	Nome	Città	CCS	ESAME. Matr	Cod-Corso	Data	Voto
123	Carlo	Bologna	Inf	123	1	7-9-23	30
123	Carlo	Bologna	Inf	123	2	8-1-23	28
702	Antonio	Roma	Ges	702	2	7-9-23	20

Sintassi del predicato di join

espressione congiuntiva di predicati semplici:

ATTR1 comp ATTR2

ove ATTR1 appartiene a TAB1

ATTR2 appartiene a TAB2

comp: =, <>, <, <=, >, >=

Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di “Gestionale” che hanno preso almeno un 30

```
select Matr, Nome
from Studente, Esame
where Studente.Matr = Esame.Matr
  and CCS = ’Ges’ and Voto = 30
```

MATR	NOME
123	Carlo

Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di “Gestionale” che hanno preso almeno un 30

```
select distinct Matr, Nome
from Studente, Esame
where Studente.Matr = Esame.Matr
   and CCS = 'Ges' and Voto = 30
```

MATR	NOME
123	Carlo

Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di “Gestionale” che hanno preso sempre 30

```
select distinct Matr, Nome
from Studente, Esame
where Studente.Matr = Esame.Matr
   and CCS = 'Ges' and Voto = 30
```

Interrogazione semplice con tre tabelle

- Estrarre il nome degli studenti che hanno preso 30 in "Matematica"

```
select Matr, Nome
from Studente, Esame, Corso
where  Studente.Matr = Esame.Matr
  and Corso.CodCorso = Esame.CodCorso
  and Titolo = 'Matematica' and Voto = 30
```

Join in SQL

- SQL ha una sintassi per i join, li rappresenta esplicitamente nella clausola `from`:

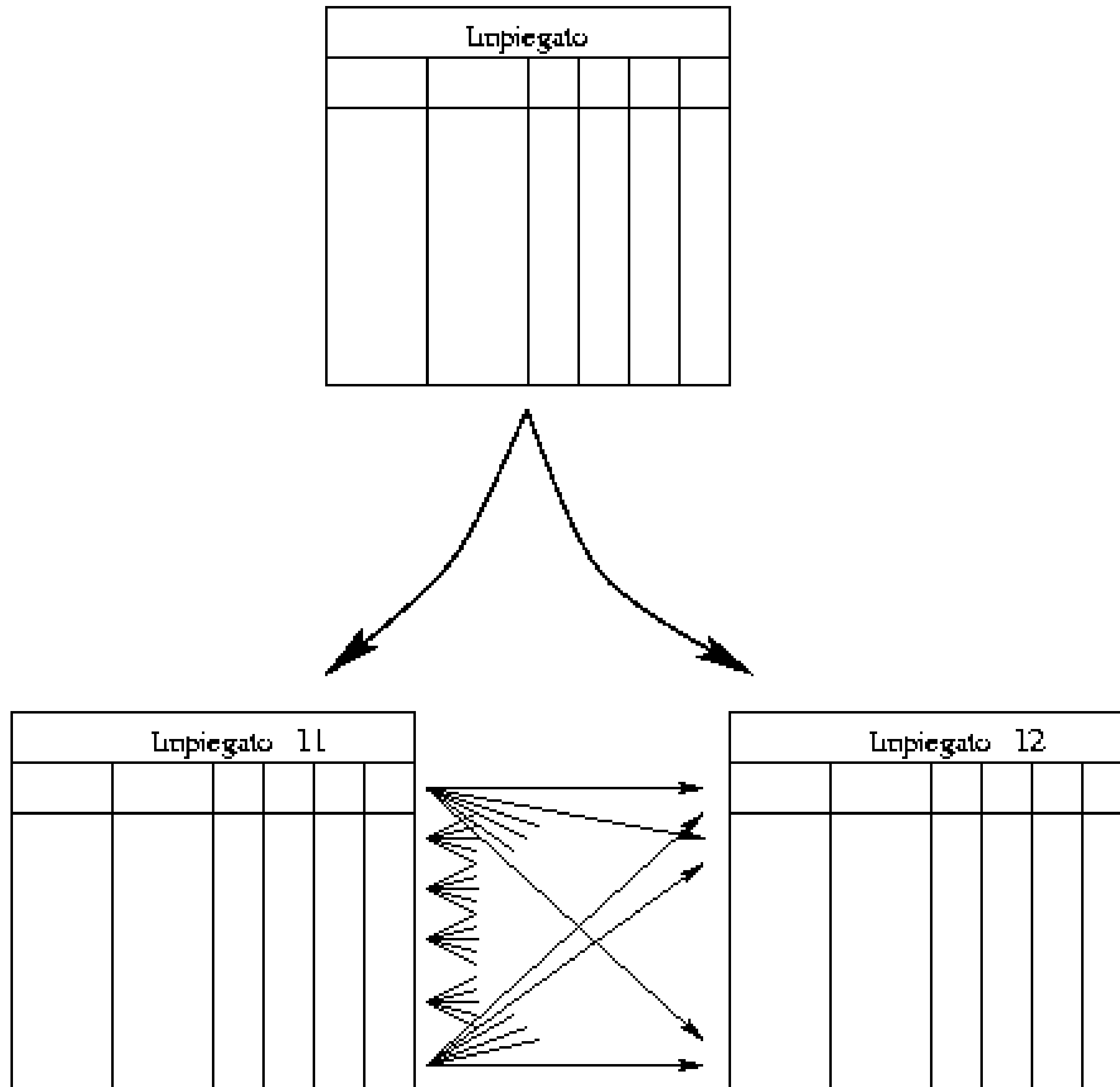
`select` *AttrEspr* {, *AttrEspr*}
`from` *Tabella* { [*TipoJoin*] **`join`** *Tabella* **`on`** *Condizioni* }
[**`where`** *AltreCondizioni*]

```
SELECT S1.Matr, S1.Nome
FROM Studente S1, Studente S2
WHERE S1.Nome = S2.Nome AND S1.Matr<>S2.Matr
```

```
SELECT distinct S1.Nome
FROM Studente S1, Studente S2
WHERE S1.Nome= S2.Nome AND S1.Matr<>S2.Matr
```

```
SELECT S1.Nome
FROM Studente S1, Studente S2
WHERE S1.Nome= S2.Nome AND S1.Matr > S2.Matr
```

Variabili in SQL



Interrogazione semplice con variabili relazionali

Chi sono i dipendenti di Giorgio?

Impiegato

Matr	Nome	DataAss	Salario	MatrMgr
1	Piero	1-1-95	3 M	2
2	Giorgio	1-1-97	2,5 M	null
3	Giovanni	1-7-96	2 M	2

Chi sono i dipendenti di Giorgio?

```
select I1.Nome,I1.MatrMgr,I2.Matr,I2.Nome
from Impiegato I1, Impiegato I2
where I1.MatrMgr = I2.Matr
   and I2.Nome = 'Giorgio'
```

X.Nome	X.MatrMgr	Y.Matr	Y.Nome
Piero	2	2	Giorgio
Giovanni	2	2	Giorgio

https://sqlzoo.net/wiki/The_JOIN_operation
https://sqlzoo.net/wiki/Old_JOIN_Tutorial
https://sqlzoo.net/wiki/More_JOIN_operations
https://sqlzoo.net/wiki/Self_join

Ordinamento

- La clausola `order by`, che compare in coda all'interrogazione, ordina le righe del risultato
- Sintassi:

  `order by` *AttributoOrdinamento* [`asc` | `desc`]
  {, *AttributoOrdinamento* [`asc` | `desc`] }

- Le condizioni di ordinamento vengono valutate in ordine
  - a pari valore del primo attributo, si considera l'ordinamento sul secondo, e così via

Query con ordinamento

```
select  *
from Studente
order by Matricola

select  *
from Studente
order by Matricola ASC

select  *
from Studente
order by Matricola DESC

select  *
from Studente
order by Cognome, Nome
```

Esempio: gestione ordini

Cliente

<u>CODCLI</u>	INDIRIZZO	P-IVA

Ordine

<u>CODORD</u>	CODCLI	DATA	IMPORTO

Dettaglio

<u>CODORD</u>	<u>CODPROD</u>	QTA

Prodotto

<u>CODPROD</u>	NOME	PREZZO

Istanza di ordine

Ordine

CODORD	CODCLI	DATA	IMPORTO
1	3	1-6-23	50.000
2	4	3-8-23	8.000
3	3	1-9-23	5.500
4	1	1-7-23	12.000
5	1	1-8-23	1.500
6	3	3-9-23	27.000

Funzioni aggregate

- Il risultato di una query con funzioni aggregate dipende dalla valutazione del contenuto di un insieme di righe
- Cinque operatori aggregati:
  - `count` cardinalità
  - `sum` sommatoria
  - `max` massimo
  - `min` minimo
  - `avg` media

Operatore `count`

- `count` restituisce il numero di righe o valori distinti; sintassi:

  `count(`< * | [`distinct` | `all`] *ListaAttributi* >)

- Estrarre il numero di ordini:

```
select count(*)
from Ordine
where codCliente=1
```

- Estrarre il numero di valori distinti dell'attributo CodCli per tutte le righe di Ordine:

```
select count(distinct CodCli)
from Ordine
where importo>10000
```

- Estrarre il numero di righe di Ordine che posseggono un valore non nullo per l'attributo CodCli:

```
select count(all CodCli)
from Ordine
```

`sum`, `max`, `min`, `avg`

- Sintassi:
  < `sum` | `max` | `min` | `avg` > ([`distinct` | `all`] *AttrEspr*)

- L'opzione `distinct` considera una sola volta ciascun valore
  - utile solo per le funzioni `sum` e `avg`

- L'opzione `all` considera tutti i valori diversi da *null*

Query con massimo

- **Estrarre l’importo massimo degli ordini**

```
select max(Importo) as MaxImp
from Ordine
```

MaxImp
50.000

Query con sommatoria

- **Estrarre la somma degli importi degli ordini relativi al cliente numero 1**

```
select sum(Importo) as SommaImp
from Ordine
where CodCliente = 1
```

SommaImp
13.500

Funzioni aggregate con join

- Estrarre l'ordine massimo tra quelli contenenti il prodotto con codice 'ABC' :

```
select max(Importo) as MaxImportoABC
from Ordine, Dettaglio
where Ordine.CodOrd = Dettaglio.CodOrd and
      CodProd = 'ABC'
```

Funzioni aggregate e target list

- Query scorretta:

```
select Data, max(Importo)
from Ordine, Dettaglio
where Ordine.CodOrd = Dettaglio.CodOrd and
      CodProd = 'ABC'
```

- La data di quale ordine? La target list deve essere omogenea
- Estrarre il massimo e il minimo importo degli ordini:

```
select max(Importo) as MaxImp,
       min(Importo) as MinImp
from Ordine
```

Funzioni aggregate e target list

- Estrarre il massimo e il minimo importo degli ordini:

```
select max(Importo) as MaxImp,
       min(Importo) as MinImp
from Ordine
```

MaxImp	MinImp
50.000	1.500

https://sqlzoo.net/wiki/SUM_and_COUNT

Solo da 1 a 5

Query con raggruppamento

- Nelle interrogazioni si possono applicare gli operatori aggregati a sottoinsiemi di righe
- Si aggiungono le clausole
  - **`group by`** (raggruppamento)
  - **`having`** (selezione dei gruppi)

```
select …
from …
where …
group by …
having …
```

Query con raggruppamento

- Estrarre la somma degli importi degli ordini successivi al 10-6-23 per quei clienti che hanno emesso almeno 2 ordini

```
select CodCli, sum(Importo)
from Ordine
where Data > 10-6-23
group by CodCli
having count(*) >= 2
```

Passo 1: Valutazione `where`

CodOrd	CodCli	Data	Importo
2	4	3-8-23	8.000
3	3	1-9-23	5.500
4	1	1-7-23	12.000
5	1	1-8-23	1.500
6	3	3-9-23	27.000

Passo 2 : Raggruppamento

- si valuta la clausola **`group by`**

CodOrd	CodCli	Data	Importo
4	1	1-7-23	12.000
5	1	1-8-23	1.500
3	3	1-9-23	1.500
6	3	3-9-23	5.500
2	4	3-8-23	8.000

Passo 3 : Calcolo degli aggregati

- si calcolano **`sum(Importo)`** e **`count(*)`** per ciascun gruppo

CodCli	sum(Importo)	count(*)
1	13.500	2
3	32.500	2
4	5.000	1

Passo 4 : Estrazione dei gruppi

- si valuta il predicato `count(*) >= 2`

CodCli	sum (Importo)	count(*)
1	13.500	2
3	32.500	2
~~4~~	~~5.000~~	~~1~~

Passo 5 : Produzione del risultato (esecuzione della clausola Select)

CodCli	sum(Importo)
1	13.500
3	32.500

Query con group by e target list

- **Query scorretta:**

```
select Importo
from Ordine
group by CodCli
```

- **Query scorretta:**

```
select O.CodCli, count(*), C.Città
from Ordine O join Cliente C
     on (O.CodCli = C.CodCli)
group by O.CodCli
```

- **Query corretta:**

```
select O.CodCli, count(*), C.Città
from Ordine O join Cliente C
     on (O.CodCli = C.CodCli)
group by O.CodCli, C.Città
```

`where` o `having`?

- Soltanto i predicati che richiedono la valutazione di funzioni aggregate dovrebbero comparire nell'argomento della clausola `having`

Raggruppamento e ordinamento

Estrarre la somma degli importi degli ordini successivi al 10-6-23 per quei clienti che hanno emesso almeno 2 ordini, in ordine decrescente di codice cliente

```
select CodCli, sum(Importo)
from Ordine
where Data > 10-6-23
group by CodCli
having count(*) >= 2
order by CodCli desc
```

Risultato dopo la clausola di ordinamento

CodCli	sum(Importo)
3	32.500
1	13.500

Doppio raggruppamento

- Estrarre la somma delle quantità dei dettagli degli ordini emessi da ciascun cliente per ciascun prodotto, purché la somma superi 50

```
select CodCli, CodProd, sum(Qta)
from Ordine as O, Dettaglio as D
Where O.CodOrd = D.CodOrd
group by CodCli, CodProd
having sum(Qta) > 50
```

Situazione dopo il join e il raggruppamento

Ordine		Dettaglio			
CodCli	Ordine.CodOrd	Dettaglio.CodOrd	CodProd	Qta	
1	3	3	1	30	gruppo 1,1
1	4	4	1	20	
1	3	3	2	30	gruppo 1,2
1	5	5	2	10	
2	3	3	1	60	gruppo 2,1
3	1	1	1	40	gruppo 3,1
3	2	2	1	30	
3	6	6	1	25	

Estrazione del risultato

- si valuta la funzione aggregata **`sum(Qta)`** e il predicato **`having`**

CodCli	CodProd	sum(Qta)
~~1~~	~~1~~	~~50~~
~~1~~	~~2~~	~~40~~
2	1	60
3	1	95

https://sqlzoo.net/wiki/SUM_and_COUNT

Solo da 6 a 8

Query nidificate

- Nella clausola **`where`** e nella clausola **`having`** possono comparire predicati che:
  - confrontano un attributo (o un’espressione sugli attributi) con il risultato di una query SQL; sintassi:

    AttrExpr Operator < **`any`** | **`all`** > *SelectSQL*

    - **`any`**: il predicato è vero se almeno una riga restituita dalla query *SelectSQL* soddisfa il confronto
    - **`all`**: il predicato è vero se tutte le righe restituite dalla query *SelectSQL* soddisfano il confronto
    - *Operator*: uno qualsiasi tra =, <>, <, <=, >, >=
- La query che appare nella clausola **`where`** e nella clausola **`having`** è chiamata query nidificata
- Nelle query nidificate posso usare variabili definite esternamente

Uso di **`any`** e **`all`**

```
select CodOrd
from Ordine
where Importo > any
         select Importo
         from Ordine
```

```
select CodOrd
from Ordine
where Importo >= all
         select Importo
         from Ordine
```

COD-ORD	IMPORTO	ANY	ALL
1	50	F	F
2	300	V	V
3	90	V	F

Query nidificate con any

- Estrarre gli ordini di prodotti con un prezzo superiore a 100

```
select distinct CodOrd
from Dettaglio
where CodProd = any(select CodProd
                    from Prodotto
                    where Prezzo > 100)
```

- Equivalente a (senza query nidificata)

```
select distinct CodOrd
from Dettaglio D, Prodotto P
where D.CodProd = P.CodProd
   and Prezzo > 100
```

Query nidificate con all

- Estrarre gli ordini di prodotti con NESSUN prezzo superiore a 100

```
select distinct CodOrd
from Dettaglio
where CodProd <> all(select CodProd
                     from Prodotto
                     where Prezzo > 100)
```

```
select distinct CodOrd
from Dettaglio D, Prodotto P
where D.CodProd = P.CodProd
    and Prezzo <= 100
```

Query nidificate con all

- Estrarre gli ordini di prodotti con NESSUN prezzo superiore a 100

```
select distinct CodOrd
from Dettaglio
where CodOrd <>
     all( select CodOrd
          from Dettaglio D, Prodotto P
          where D.CodProd = P.CodProd
             and Prezzo > 100 )
```

Negazione con query nidificate

- Estrarre gli ordini che non contengono il prodotto ‘ABC’:

```
select CodOrd
from Ordine
where CodOrd <> all (select CodOrd
                     from Dettaglio
                     where CodProd = 'ABC')
```

Query nidificate

- *AttrExpr Operator* < **`in`** | **`not in`** > *SelectSQL*
  - **`in`**: il predicato è vero se almeno una riga restituita dalla query *SelectSQL* e' presente nell'espressione
  - **`not in`**: il predicato è vero se nessuna riga restituita query e' presente nell'espressione

Operatori **in** e **not in**

- L'operatore `in` è equivalente a `= any`

```
select CodProd
from Dettaglio
where CodOrd in
        (select CodOrd
         from Dettaglio
         where CodProd = 'ABC')
```

- L'operatore `not in` è equivalente a `<> all`

```
select distinct CodOrd
from Ordine
where CodOrd not in (select CodOrd
                     from Dettaglio
                     where CodProd = 'ABC')
```

max con query nidificata

- Gli operatori aggregati `max` (e `min`) possono essere espressi tramite query nidificate
- Estrarre l'ordine con il massimo importo
  - Con una query nidificata, usando `max`:

```
select CodOrd
from Ordine
where Importo in (select max(Importo)
                          from Ordine)
```

  - con una query nidificata, usando all:

```
select CodOrd
from Ordine
where Importo >= all (select Importo
                              from Ordine)
```

Costruttore di tupla

- Il confronto con la query nidificata può coinvolgere più di un attributo
- Gli attributi devono essere racchiusi da un paio di parentesi tonde (costruttore di tupla)
- Esempio: estrarre gli omonimi

```
select *
from Persona P
where (Nome,Cognome) in
         (select Nome, Cognome
          from Persona P1)
```

Costruttore di tupla

- Il confronto con la query nidificata può coinvolgere più di un attributo
- Gli attributi devono essere racchiusi da un paio di parentesi tonde (costruttore di tupla)
- Esempio: estrarre gli omonimi

```
select *
from Persona P
where (Nome,Cognome) in
        (select Nome, Cognome
         from Persona P1
         where P1.CodFisc <> P.CodFisc)
```

Costruttore di tupla

```
select *
from Persona P, Persona P1
where P.Nome=P1.Nome and
                 P.Cognome=P1.Cognome
  and P1.CodFisc <> P.CodFisc
```

Costruttore di tupla

- Esempio: estrarre le persone che non hanno omonimi

```
select *
from Persona P
where (Nome,Cognome) not in
        (select Nome, Cognome
         from Persona P1
         where P1.CodFisc <> P.CodFisc)
```

https://sqlzoo.net/wiki/SELECT_within_SELECT_Tutorial

Viste

- Offrono la "visione" di tabelle virtuali (schemi esterni)
- Le viste possono essere usate per formulare query complesse
  - Le viste decompongono il problema e producono una soluzione più leggibile
- Le viste sono talvolta necessarie per esprimere alcune query:
  - query che combinano e nidificano diversi operatori aggregati
  - query che fanno un uso sofisticato dell'operatore di unione
- Sintassi:

`create view` *NomeVista* [(*ListaAttributi*)] **`as`** *SelectSQL*

Composizione delle viste con le query

- Vista:

```
create view OrdiniPrincipali as
    select *
    from Ordine
    where Importo > 10000
```

- Query:

```
select CodCli
from OrdiniPrincipali
```

Viste e query

- Estrarre il cliente che ha generato il massimo fatturato (senza usare le viste):

```
select CodCli
from Ordine
group by CodCli
having sum(Importo) >= all
            (select sum(Importo)
             from Ordine
             group by CodCli)
```

Viste e query

- Estrarre il cliente che ha generato il massimo fatturato (usando le viste):

```
create view CliFatt(CodCli,FattTotale) as
select CodCli, sum(Importo)
from Ordine
group by CodCli

select CodCli
from CliFatt
where FattTotale = (select max(FattTotale)
                    from CliFatt)
```

Viste e query

- Estrarre il numero medio di ordini per cliente:
  - Soluzione scorretta (SQL non permette di applicare gli operatori aggregati in cascata):

```
select avg(count(*))
from Ordine
group by CodCli
```

  - Soluzione corretta (usando una vista):

```
create view CliOrd(CodCli,NumOrdini) as
select CodCli, count(*)
from Ordine
group by CodCli

select avg(NumOrdini)
from CliOrd
```

https://sqlzoo.net/wiki/SELECT_.._SELECT

C'è solo un esempio già fatto

Trovate esercizi (anche su cose che non abbiamo fatto) qui:

https://www.w3schools.com/sql/exercise.asp