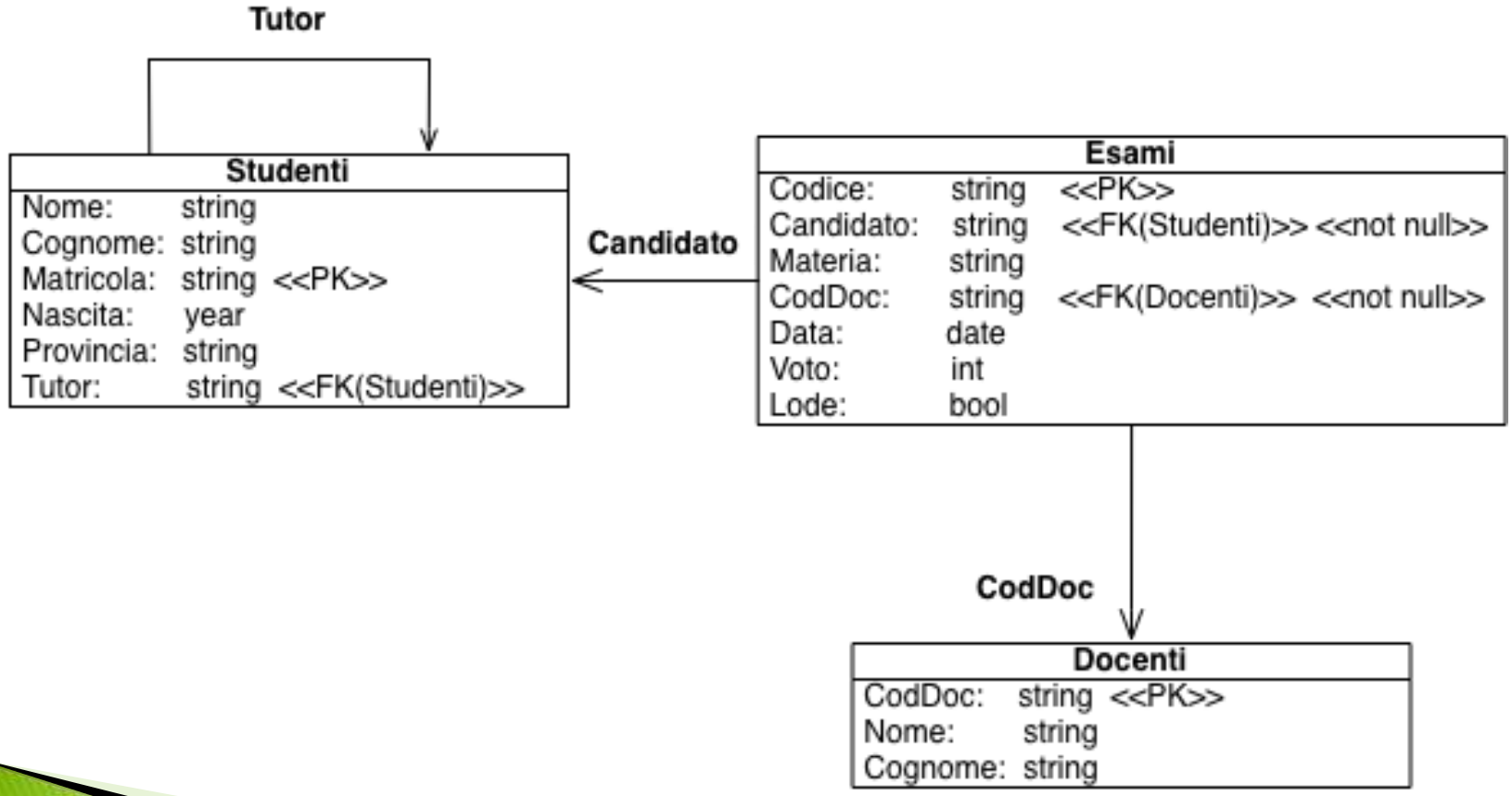
Tutorato di Base di Dati
Lezione 5

Andrea Gasparetto

Schema di riferimento

Consideriamo lo schema relazionale



Pattern Matching

- Sulle stringhe
  - **WHERE** *Expr* **LIKE** *pattern*
- Il pattern può contenere caratteri e i simboli speciali
  - `%` sequenza di 0 o più caratteri qualsiasi
  - `_` un carattere qualsiasi
- Studenti con il nome di almeno due caratteri che inizia per A

```
SELECT *
   FROM   Studenti
   WHERE  Nome LIKE 'A_%'
```

Pattern Matching

- Studenti con il nome che inizia per ‘A’ e termina per ‘a’ oppure ‘i’

```
SELECT *
FROM   Studenti
WHERE  Nome LIKE 'A%a' OR Nome LIKE ‘A%i’
```

- stessa query usando le espressioni regolari

```
SELECT *
FROM   Studenti
WHERE  Nome REGEXP '^A.*(a|i)$’
```

Clausola WHERE

- La clausola WHERE è piu complicata di come l'abbiamo vista finora.
- Combinazione booleana (`AND, OR, NOT`) di predicati tra cui:
  - *Expr Comp Expr*
  - *Expr Comp* (*Sottoselect che torna esattamente un valore*)
  - *Expr* [**NOT**] **IN** (*Sottoselect*) (oppure **IN** (v1,..,vn))
  - [**NOT**] **EXISTS** (*Sottoselect*)
  - *Expr Comp* (**ANY** | **ALL**) (*Sottoselect*)
- Comp: `<, =, >, <>, <=, >=` (e altri)

Select annidate

- Alcune interrogazioni richiedono di estrarre dati dalla BD e usarli in operazioni di confronto
- E' possibile specificare select annidate, inserendo nel campo **`WHERE`** una condizione che usa una select (che a sua volta può contenere sottoselect ...)
- Si può
  - eseguire confronti con l'insieme di valori ritornati dalla sottoselect (sia quando questo è un singoletto, sia quando contiene più elementi)
  - verificare la presenza/assenza di valori dati nell'insieme ritornato dalla sottoselect
  - verificare se l'insieme di valori ritornato dalla sottoselect è o meno vuoto

Sottoselect con un solo valore

- Nel campo WHERE

  Expr Comp (***Sottoselect che torna esattamente un valore***)

- Studenti che vivono nella stessa provincia dello studente con matricola 71346, escluso lo studente stesso

```
SELECT   *
FROM     Studenti
WHERE    (Matricola <> '71346') AND
         Provincia = (SELECT Provincia
                      FROM   Studenti
                      WHERE  Matricola='71346')
```

Sottoselect con un solo valore

- E’ indispensabile la sottoselect?

```
SELECT  altri.*
FROM    Studenti altri, Studenti s
WHERE   altri.Matricola <> '71346' AND
        s.Matricola = '71346' AND altri.Provincia = s.Provincia
```

Sottoselect con un solo valore

- … è un join

```
SELECT   altri.*
  FROM    Studenti altri JOIN Studenti
  s USING (Provincia)
  WHERE   altri.Matricola <> '71346'
          AND s.Matricola = '71346';
```

Quantificazione

- Le interrogazioni su di una associazione multi-valore vanno quantificate

- Non: gli studenti che hanno preso 30 (ambiguo!) ma:
  - Gli studenti che hanno preso sempre (solo, tutti) 30: universale
  - Gli studenti che hanno preso qualche (almeno un) 30: esistenziale
  - Gli studenti che non hanno preso mai 30 (senza alcun 30): universale
  - Gli studenti che non hanno preso sempre 30: esistenziale

Quantificazione

- Universale negata = esistenziale:
  - Non tutti i voti sono = 30 (universale) = esiste un voto ≠30 (esistenziale)
  - Più formalmente

$$\neg\forall x.P(x) \equiv \exists x.\neg P(x)$$

- Esistenziale negata = universale:
  - Non esiste un voto diverso da 30 (esistenziale) = Tutti i voti sono uguali a 30 (universale)
  - Più formalmente

$$\neg\exists x.P(x) \equiv \forall x.\neg P(x)$$

Quantificazione Esistenziale

- Gli studenti con almeno un voto > 27; servirebbe un quantificatore

```
EXISTS e IN s.HaSostenuto: e.Voto > 27
```

(stile **OQL**)

- Sarebbe bello poterlo tradurre come ...

```
SELECT   *
FROM     Studenti s
WHERE    EXISTS e IN Esami WHERE
e.Candidato = s.Matricola:e.Voto > 27
```

Quantificazione Esistenziale

- Altra query esistenziale
  - gli studenti che non hanno preso 30 in tutti gli esami, ovvero
  - gli studenti per i quali qualche esame ha voto diverso da 30:
- diverrebbe:

```
SELECT *
FROM   Studenti s
WHERE  EXIST e IN Esami WHERE
e.Candidato = s.Matricola: e.Voto <> 30
```

- Purtroppo un costrutto così non c'è in SQL ma ...

EXISTS

- Come condizione nel **WHERE** possiamo usare

```
SELECT ...
FROM   ...
WHERE [NOT] EXISTS (Sottoselect)
```

- Per ogni tupla (o combinazione di tuple) `t` della select esterna
  - calcola la sottoselect
  - verifica se ritorna una tabella [non] vuota e in questo caso seleziona `t`

Quantificazione esistenziale: EXISTS

- La query studenti con almeno un voto > 27

```
SELECT   *
FROM     Studenti s
WHERE    EXIST e IN Esami WHERE e.Candidato
= s.Matricola:e.Voto > 27
```

- in SQL diventa:

```
SELECT *
FROM   Studenti s
WHERE  EXISTS (SELECT *
               FROM  Esami e
               WHERE e.Candidato = s.Matricola
                     AND e.Voto > 27)
```

Quantificazione esistenziale: Giunzione+Proiezione

- Query con EXISTS:

```
SELECT  *
  FROM  Studenti s
  WHERE EXISTS SELECT *
               FROM Esami e
               WHERE e.Candidato = s.Matricola
                     AND e.Voto > 27)
```

- La stessa query, ovvero gli studenti con almeno un voto > 27, tramite giunzione:

```
SELECT s.*
FROM   Studenti s JOIN Esami e
       ON (e.Candidato = s.Matricola)
WHERE  e.Voto > 27
```

Quantificazione esistenziale: ANY

- Un altro costrutto che permette una quantificazione esistenziale
  - **SELECT** ...
  - **FROM** ...
  - **WHERE** *Expr Comp* **ANY** (*Sottoselect*)

- Per ogni tupla (o combinazione di tuple) `t` della select esterna
  - calcola la sottoselect
  - verifica se `Expr` è in relazione `Comp` con almeno uno degli elementi ritornati dalla select

Quantificazione Esistenziale: ANY

- La solita query
  - "Studenti che hanno preso almeno un voto > 27" si può esprimere anche tramite ANY ...

```
SELECT *
FROM   Studenti s
WHERE  s.Matricola = ANY(SELECT e.Candidato
                         FROM Esami e
                         WHERE e.Voto >27)
```

```
SELECT *
FROM   Studenti s
WHERE  27 < ANY(SELECT e.Voto
                FROM   Esami e
                WHERE  e.Candidato =s.Matricola)
```

Quantificazione Esistenziale: ANY

- ANY non fa nulla in più di EXISTS

```
SELECT *
FROM   Tab1
WHERE  attr1 op ANY (SELECT attr2
                     FROM   Tab2
                     WHERE  C);
```

- diventa

```
SELECT *
FROM   Tab1
WHERE  EXISTS (SELECT *
               FROM   Tab2
               WHERE  C AND
                      attr1 op attr2);
```

Quantificazione Esistenziale: IN

- Forma ancora più blanda di quantificazione esistenziale:

```
SELECT ...
FROM   ...
WHERE  Expr IN (sottoselect)
```

- Nota: abbreviazione di =ANY
- La solita query si può esprimere anche tramite IN:

```
SELECT *
FROM   Studenti s
WHERE  s.Matricola IN (SELECT e.Candidato
                       FROM   Esami e
                       WHERE  e.Voto >27)
```

Ancora su IN

- Può essere utilizzato con ennuple di valori

  Expr **IN** (val_1, val_2, ..., val_n)

- Gli studenti di Padova, Venezia e Belluno

```
SELECT   *
FROM     Studenti
WHERE    Provincia IN
('PD','VE','BL');
```

Riassumendo ...

- La quantificazione esistenziale si fa con:
  - `EXISTS` (il più "espressivo")
  - Giunzione
  - `=ANY, >ANY, <ANY, ...`
  - `IN`
- Però: `=ANY, >ANY, <ANY, IN,...` non aggiungono nulla, `EXISTS` basta e avanza
- Il problema vero è: non confondere esistenziale con universale!

Quantificazione Universale

- Gli studenti che hanno preso solo 30
- Errore comune (e grave):
  - ```
    SELECT s.*
    FROM   Studenti s, Esami e
    WHERE  e.Candidato = s.Matricola AND e.Voto = 30
    ```
- In stile OQL: `FORALL e IN s.HaSostenuto: e.Voto=30`
  - ```
    SELECT *
    FROM   Studenti s
    WHERE  FORALL e IN Esami WHERE e.Candidato =
    s.Matricola:e.Voto = 30
    ```

Quantificazione Universale

- In SQL non c'è un operatore generale esplicito **FOR ALL**. Si usa l'equivalenza logica

$$\forall e \in E.\ P \equiv \neg(\exists e \in E.\ \neg P)$$

- Quindi da:

```
SELECT *
FROM   Studenti s
WHERE  FORALL e IN Esami WHERE e.Candidato
= s.Matricola:e.Voto = 30
```

- ... si può passare a ...

Quantificazione Universale

- ... si può passare a

  - ```
    SELECT *
    FROM   Studenti
    WHERE NOT EXIST e IN Esami WHERE
    e.Candidato = s.Matricola:
                                    e.Voto <> 30
    ```

    dove `NOT(e.Voto = 30)` è diventato `e.Voto <> 30`

- In SQL diventa:

```
SELECT *  FROM Studenti s
WHERE NOT EXISTS (SELECT *
                  FROM   Esami e
                  WHERE  e.Candidato = s.Matricola
                         AND e.Voto <> 30)
```
```

Quantificazione universale: ALL

- E` disponibile un operatore duale rispetto a ANY, che è ALL:

```
WHERE Expr Comp ALL (Sottoselect)
```

- Sostituendo EXISTS con =ANY, la solita query (studenti con tutti 30):
  - ```
    SELECT * FROM Studenti s
    WHERE NOT EXISTS (SELECT *
                      FROM Esami e
                      WHERE e.Candidato = s.Matricola
                            AND e.Voto <> 30)
    ```
- Diventa:
  - ```
    SELECT *
    FROM Studenti s
    WHERE NOT(s.Matricola =ANY (SELECT e.Candidato
                                FROM Esami e
                                WHERE e.Voto <> 30))
    ```

Quantificazione universale: ALL

- …
  - ```
    SELECT *
    FROM Studenti s
    WHERE NOT(s.Matricola =ANY (SELECT e.Candidato
                                FROM   Esami e
                                WHERE  e.Voto <> 30))
    ```
- Ovvero:
  - ```
    SELECT * FROM Studenti s
    WHERE s.Matricola <>ALL (SELECT e.Candidato
                             FROM Esami e
                             WHERE e.Voto <> 30)
    ```
- E naturalmente, **<>ALL** è lo stesso di **NOT IN** …

```
SELECT * FROM Studenti s
WHERE s.Matricola NOT IN (SELECT e.Candidato
                          FROM Esami e
                          WHERE e.Voto <> 30)
```

Quantificazione Universale e insiemi vuoti

- Supponiamo che la BD sia tale che la query
- **SELECT** s.Nome, s.Cognome, e.Materia, e.Voto
  FROM Studenti s **LEFT JOIN** Esami e
  ON
  s.Matricola=e.Candidato;
- ritorni

Nome	Cognome	Materia	Voto
Chiara	Scuri	NULL	NULL
Giorgio	Zeri	NULL	NULL
Paolo	Verdi	BD	27
Paolo	Verdi	ALG	30
Paolo	Poli	ALG	22
Paolo	Poli	FIS	22
Anna	Rossi	BD	30
Anna	Rossi	FIS	30

Quantificazione Universale e insiemi vuoti

- Qual è l'ouput della query 'studenti che hanno preso solo trenta'?
  - ```
    SELECT s.Cognome
    FROM   Studenti s
    WHERE NOT EXISTS (SELECT *
                      FROM  Esami e
                      WHERE e.Candidato =
    s.Matricola AND e.Voto <> 30)
    ```

```
+---------+
| Cognome |
+---------+
| Scuri   |
| Zeri    |
| Rossi   |
+---------+
```
```

Quantificazione Universale e insiemi vuoti

- Qual è l'ouput della query 'studenti che hanno preso solo trenta'?
  - ```
    SELECT s.Cognome
    FROM   Studenti s
    WHERE NOT EXISTS (SELECT *
                      FROM  Esami e
                      WHERE e.Candidato =
                            s.Matricola
                        AND e.Voto <> 30)
    ```

- Cosa cambia se invece di **NOT EXISTS** uso **<>ALL** oppure **NOT IN** ?
    ```

Quantificazione Universale e insiemi vuoti

- Se voglio gli studenti che hanno preso solo trenta, e hanno superato qualche esame:

```
SELECT *
FROM Studenti s
WHERE NOT EXISTS (SELECT *
                  FROM  Esami e
                  WHERE e.Candidato = s.Matricola
                        AND e.Voto <> 30)
      AND EXISTS (SELECT *
                  FROM  Esami e
                  WHERE e.Candidato = s.Matricola)
```

Quantificazione Universale e insiemi vuoti

- Oppure:
  - ```
    SELECT s.Matricola, s.Cognome
    FROM   Studenti s JOIN Esami e ON
           (s.Matricola = e.Candidato)
    GROUP BY s.Matricola, s.Cognome
    HAVING Min(e.Voto) = 30;
    ```
    ```

Raggruppamento

- Per ogni materia, trovare nome della materia e voto medio degli esami in quella materia [selezionando solo le materie per le quali sono stati sostenuti più di tre esami]:
  - Per ogni materia vogliamo
    - Il nome, che è un attributo di Esami
    - Una funzione aggregata sugli esami della materia
- Soluzione:

```
SELECT    e.Materia, avg(e.Voto)
FROM      Esami e
GROUP BY e.Materia
[HAVING    count(*)>3]
```

Raggruppamento

- Costrutto:

```
SELECT … FROM … WHERE …
GROUP BY A1,..,An
[HAVING condizione]
```

- Semantica:
  - Esegue le clausole `FROM` – `WHERE`
  - Partiziona la tabella risultante rispetto all'uguaglianza su tutti i campi A_1, …, A_n (in questo caso, si assume `NULL = NULL`)
  - Elimina i gruppi che non rispettano la clausola `HAVING`
  - Da ogni gruppo estrae una riga usando la clausola `SELECT`

Esecuzione di GROUP BY

```
SELECT Candidato, count(*) AS NEsami,
       min(Voto), max(Voto), avg(Voto)
FROM   Esami
GROUP BY Candidato
HAVING avg(Voto) > 23;
```

Codice	Materia	Candidato	Data	Voto	Lode	CodDoc
B112	BD	71523	2006-07-08	27	N	AM1
B247	ALG	71523	2006-12-28	30	S	NG2
B248	BD	76366	2007-07-18	29	N	AM1
B249	ALG	71576	2007-07-19	22	N	NG2
F313	FIS	76366	2007-07-08	26	N	GL1
F314	FIS	71576	2007-07-29	22	N	GL1

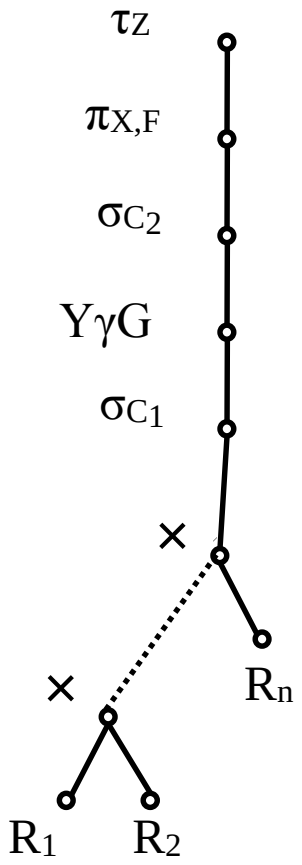
Esecuzione di GROUP BY

Codice	Materia	Candidato	Data	Voto	Lode	CodDoc
B112	**BD**	**71523**	**2006-07-08**	**27**	**N**	**AM1**
B247	**ALG**	**71523**	**2006-12-28**	**30**	**S**	**NG2**
B249	ALG	71576	2007-07-19	22	N	NG2
F314	FIS	71576	2007-07-29	22	N	GL1
B248	**BD**	**76366**	**2007-07-18**	**29**	**N**	**AM1**
F313	**FIS**	**76366**	**2007-07-08**	**26**	**N**	**GL1**

Candidato	NEsami	min(Voto)	max(Voto)	avg(Voto)
71523	2	27	30	28.5000
76366	2	26	29	27.5000

SQL → ALGEBRA

```
SELECT DISTINCT X, F
FROM  R1,…,Rn
WHERE C1
GROUP BY Y
HAVING C2
ORDER BY Z
```



- X, Y, Z sono insiemi di attributi
- F, G sono insiemi di espressioni aggregate, tipo count(*) o sum(A)
- $X,Z \subseteq Y$, $F \subseteq G$, C_2 nomina solo attributi
  in Y o espressioni in G

Raggruppamento

- Per ogni studente, cognome e voto medio:
  - `SELECT  s.Cognome, avg(e.Voto)`
  - `FROM  Studenti s, Esami e`
  - `WHERE  s.Matricola = e.Candidato`
  - `GROUP BY s.Matricola`
- È necessario scrivere:
  - `GROUP BY s.Matricola, s.Cognome`
- Gli attributi espressi non aggregati nella select (`s.Cognome`) e in **`HAVING`** se presenti (`s.Matricola`) devono essere inclusi tra quelli citati nella **`GROUP BY`**
- Gli attributi aggregati (`avg(e.Voto)`) vanno scelti tra quelli non raggruppati

Clausola HAVING: importante

- Anche la clausola `HAVING` cita solo:
  - espressioni su attributi di raggruppamento;
  - funzioni di aggregazione applicate ad attributi non di raggruppamento.
- Non va ...

```
SELECT   s.Cognome, avg(e.Voto)
FROM     Studenti s JOIN Esami e
         ON (s.Matricola = e.Candidato)
GROUP BY s.Matricola, s.Cognome
HAVING   YEAR(Data) > 2006;
```

Raggruppamento e NULL

- Nel raggruppamento si assume (è uno dei pochi casi) **`NULL = NULL`**
- Es: Matricole dei tutor e relativo numero di studenti di cui sono tutor

```
SELECT Tutor, COUNT(*) AS NStud
FROM   Studenti
GROUP BY Tutor;
```

```
+-------+-------+
| Tutor | NStud |
+-------+-------+
| NULL  |     2 |
| 71347 |     2 |
| 71523 |     1 |
+-------+-------+
```

Sintassi della select completa

- Sottoselect:
  - **SELECT** [**DISTINCT**] *Attributi*
    FROM *Tabelle*
    [**WHERE** *Condizione*]
    [**GROUP** BY A_1,..,A_n [**HAVING** *Condizione*]]
- Select:
  - Sottoselect
    { (**UNION** [**ALL**] | **INTERSECT** [**ALL**] | **EXCEPT** [**ALL**])
    Sottoselect }
    [**ORDER BY** *Attributo* [**DESC**] {, *Attributo* [**DESC**]}]