

SQL

```
SELECT s.Nome, e.Data
FROM   Studenti s, Esami e
WHERE  e.Materia = 'BD' AND e.Voto = 30 AND e.Matricola =
      s.Matricola
```

```
SELECT s.Nome As Nome, 2018 - s.AnnoNascita As Eta,
                                             0 As NumeroEsami
FROM   Studenti s
WHERE  NOT EXISTS (SELECT *
                  FROM   Esami e
                  WHERE  e.Matricola = s.Matricola )
```

SQL: Structured Query Language

- SQL è stato definito nel 1973 ed è oggi il linguaggio universale dei sistemi relazionali
- Standard: SQL-84, SQL-89, SQL-92 (o SQL2), SQL:1999 (o SQL3) (ANSI/ISO)
- SQL-92: entry, intermediate e full SQL.
- SQL:1999: a oggetti.
- SQL: DDL, DML, Query language.

SQL PER INTERROGARE: SELECT FROM WHERE

- SQL è un calcolo su multiinsiemi.
- Il comando base dell'SQL:
 - SELECT [DISTINCT] Attributo {, Attributo}
 - FROM Tabella [Ide] {, Tabella [Ide]}
 - [WHERE Condizione]
- Semantica: prodotto + restrizione + proiezione.
- Un attributo A di una tabella " $R \times$ " si denota come: A oppure $R.A$ oppure $x.A$

LA LISTA DEGLI ATTRIBUTI

- $\text{Attributi} ::= *$
| $\text{Expr} \text{ [[AS] Nuovonome] } \{, \text{Expr} \text{ [[AS] Nuovonome] } \}$
- $\text{Expr} ::= [\text{Ide.}] \text{Attributo} \mid \text{Const}$
| $(\text{Expr}) \mid [-] \text{Expr} [\text{Op Expr}]$
| $\text{COUNT}(*)$
| $\text{AggrFun} ([\text{DISTINCT}] [\text{Ide.}] \text{Attributo})$
- e AS x: dà un nome alla colonna di e
- $\text{AggrFun} ::= \text{SUM} \mid \text{COUNT} \mid \text{AVG} \mid \text{MAX} \mid \text{MIN}$
- AggrFun: o si usano tutte funzioni di aggregazione (e si ottiene un'unica riga) o non se ne usa nessuna.

LA LISTA DELLE TABELLE

- Le tabelle si possono combinare usando:
 - “,” (prodotto): FROM T1,T2
 - Giunzioni di vario genere:
 - Studenti s JOIN Esami e ON s.Matricola=e.Matricola
 - Studenti s JOIN Esami e USING Matricola
 - Studenti s NATURAL JOIN Esami e
 - Studenti s LEFT JOIN Esami e ON s.Matricola=e.Matricola
 - LEFT JOIN - USING
 - NATURAL LEFT JOIN
 - RIGHT JOIN
 - FULL JOIN

LA CONDIZIONE

- Combinazione booleana di predicati tra cui:
 - Expr Comp Expr
 - Expr Comp (Sottoselect che torna un valore)
 - [NOT] EXISTS (Sottoselect)
 - Expr Comp (ANY | ALL) (Sottoselect)
 - Expr [NOT] IN (Sottoselect) (oppure IN (v1,...,vn))
- Comp: <, =, >, <>, <=, >=

SINTASSI DELLA SELECT

- Sottoselect:

```
SELECT [DISTINCT] Attributi  
FROM Tabelle  
[WHERE Condizione]  
[GROUP BY A1,...,An [HAVING Condizione]]
```

- Select:

```
Sottoselect  
{ (UNION | INTERSECT | EXCEPT)  
  Sottoselect }  
[ ORDER BY Attributo [DESC] {, Attributo [DESC]} ]
```

ESEMPI: Proiezione

- Trovare il nome, la matricola e la provincia degli studenti:

```
SELECT Nome, Matricola, Provincia  
FROM    Studenti
```

Nome	Matricola	Provincia
Isaia	171523	PI
Rossi	167459	LU
Bianchi	179856	LI
Bonini	175649	PI

ESEMPI: Restrizione

- Trovare tutti i dati degli studenti di Pisa:

```
SELECT  *  
FROM    Studenti  
WHERE   Provincia = 'PI'
```

Nome	Matricola	Provincia	AnnoNascita
Isaia	171523	PI	1996
Bonini	175649	PI	1996

- Trovare la matricola, l'anno di nascita e il nome degli studenti di Pisa
(*Proiezione+Restrizione*):

```
SELECT  Nome, Matricola, AnnoNascita  
FROM    Studenti  
WHERE   Provincia = 'PI'
```

Nome	Matricola	AnnoNascita
Isaia	171523	1996
Bonini	175649	1996

ESEMPI: Prodotto e giunzione

- | | | |
|---|-------------------------|--|
| • Trovare tutte le possibili coppie
Studente-Esami: | SELECT
FROM | *
Studenti, Esami |
| • Trovare tutte le possibili coppie
Studente - Esame sostenuto dallo
studente: | SELECT
FROM
WHERE | *
Studenti s, Esami e
s.Matricola = e.Matricola |
| • Trovare il nome e la data degli
esami per gli studenti che hanno
superato l'esame di BD con 30: | SELECT
FROM
WHERE | Nome, Data
Studenti s, Esami e
e.Materia = 'BD' AND e.Voto = 30
AND e.Matricola = s.Matricola |

ESEMPI: ordinamenti e funzioni di aggregazione

- Studenti ordinati per Nome

```
SELECT *  
FROM   Studenti  
ORDER BY      Nome;
```

- Numero di elementi di Studenti

```
SELECT count(*)  
FROM   Studenti;
```

- Anno di nascita minimo, massimo e medio degli studenti:

```
SELECT min(AnnoNascita), max(AnnoNascita), avg(AnnoNascita)  
FROM   Studenti;
```

IL VALORE NULL

- Il valore di un campo di un'ennupla può mancare per varie ragioni; SQL fornisce il valore speciale NULL per tali situazioni.
- La presenza del NULL introduce dei problemi:
 - occorrono dei predicati per controllare se un valore è/non è NULL.
 - la condizione "reddito>8" è vera o falsa quando il reddito è uguale a NULL? Cosa succede degli operatori AND, OR e NOT?
 - Occorre una logica a 3 valori (vero, falso e unknown).
 - Va definita opportunamente la semantica dei costrutti. Ad es. il WHERE elimina le ennuple che non rendono vera la condizione.
 - Nuovi operatori sono utili (es. giunzioni esterne)

IL RAGGRUPPAMENTO

- Per ogni materia, trovare nome della materia e voto medio:
 - Per ogni materia:
 - Un attributo della materia
 - Una funzione aggregata sugli esami della materia
- Soluzione:

```
SELECT e.Materia, avg(e.Voto)
FROM Esami e
GROUP BY e.Materia
```

IL RAGGRUPPAMENTO

- Per ogni studente, nome e voto medio:

```
SELECT s.Nome, avg(e.Voto)
FROM Studenti s, Esami e
WHERE s.Matricola = e.Matricola
GROUP BY s.Matricola, ...
```

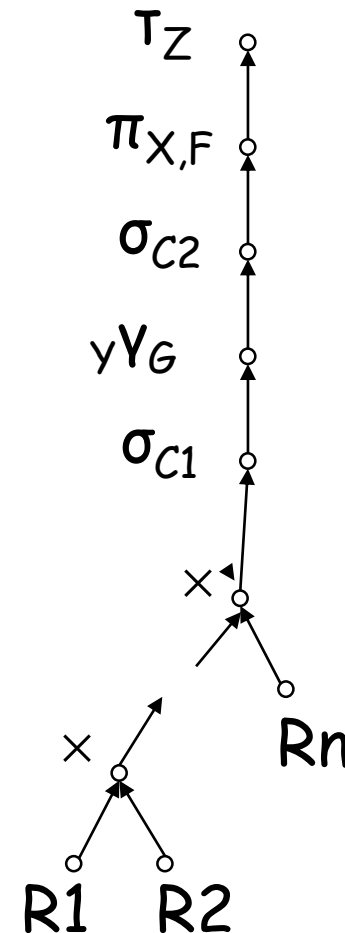
- È necessario scrivere:
 - `GROUP BY s.Matricola, s.Nome`
- Gli attributi espressi non aggregati nella select (`s.Nome`) devono essere inclusi tra quelli citati nella `GROUP BY` (`s.Matricola, s.Nome`)
- Gli attributi aggregati (`avg(e.Voto)`) vanno scelti tra quelli non raggruppati

IL RAGGRUPPAMENTO

- `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 `A1...An` (solo 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`

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)
- $X, Z \subseteq Y$, $F \subseteq G$, C2 nomina solo attributi in Y o espressioni in G



LA CLAUSOLA HAVING: IMPORTANTE

- Attenzione:
 - Se la `SELECT` contiene sia espressioni aggregate (`MIN`, `COUNT...`) che attributi non aggregati, allora DEVE essere presente la clausola `GROUP BY`
 - Le clausole `HAVING` e `SELECT` citano solo:
 - espressioni su attributi di raggruppamento;
 - funzioni di aggregazione applicate ad attributi non di raggruppamento.

ESECUZIONE DI GROUP BY

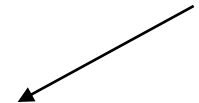
```
SELECT Matricola, count(*) AS NEsami, min(Voto), max(Voto), avg(Voto)
FROM Esami
GROUP BY Matricola
HAVING count(*) > 1;
```

Materia	Matricola	Voto	Docente
DA	1	20	10
LFC	2	30	20
MTI	1	30	30
LP	2	20	40



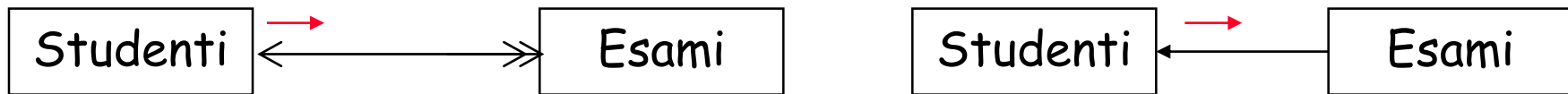
Materia	Matricola	Voto	Docente
DA	1	20	10
MTI	1	30	30
LFC	2	30	20
LP	2	20	40

Matricola	NEsami	min(Voto)	max(Voto)	Avg(Voto)
1	2	20	30	25
2	2	20	30	25



LA QUANTIFICAZIONE

- Tutte le interrogazioni su di una associazione multivalore vanno quantificate



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

LA QUANTIFICAZIONE

- Universale negata = esistenziale:
 - Non tutti i voti sono ≤ 24 = Almeno un voto > 24 (esistenziale)
- Esistenziale negata = universale:
 - Non esiste voto diverso da 30 = Tutti i voti sono uguali a 30 (universale)

LA QUANTIFICAZIONE ESISTENZIALE

- Gli studenti con almeno un voto sopra 27; servirebbe un quantificatore $\exists e \in \text{Esami-Di}(s): e.\text{Voto} > 27$ (stile OQL):

```
SELECT s.Nome
```

```
FROM Studenti s
```

```
WHERE EXIST Esami e WHERE e.Matricola = s.Matricola : e.Voto > 27
```

- Altra query esistenziale: gli studenti in cui non tutti gli esami hanno voto 30, ovvero: gli studenti in cui qualche esame ha voto diverso da 30:

```
SELECT s.Nome
```

```
FROM Studenti s
```

```
WHERE EXIST Esami e WHERE e.Matricola = s.Matricola : e.Voto <> 30
```

RICORDIAMO LA SINTASSI DEL WHERE

- Combinazione booleana di predicati tra cui:
 - Expr Comp Expr
 - Expr Comp (Sottoselect che torna un valore)
 - **[NOT] EXISTS (Sottoselect)**
- Inoltre:
 - Expr Comp (ANY | ALL) (Sottoselect)
 - Expr [NOT] IN (Sottoselect) (oppure IN (v1,...,vn))
- Comp: <, =, >, <>, <=, >=

LA QUANTIFICAZIONE ESISTENZIALE: EXISTS

- Gli studenti con almeno un voto sopra 27 stile OQL:

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

- In SQL diventa:

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

LA QUANTIFICAZIONE ESISTENZIALE: GIUNZIONE

- Gli studenti con almeno un voto sopra 27, tramite EXISTS:

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

- Stessa quantificazione esistenziale, tramite giunzione:

```
SELECT s.Nome  
FROM Studenti s, Esami e  
WHERE e.Matricola = s.Matricola AND e.Voto > 27
```

LA QUANTIFICAZIONE ESISTENZIALE: ANY

- ANY equivale ad EXISTS

- La solita query:

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

- Si può esprimere anche tramite ANY:

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

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

LA QUANTIFICAZIONE ESISTENZIALE: IN

- IN è solo un'abbreviazione di =ANY

- La solita query:

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

- Si può esprimere anche tramite IN:

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

RIASSUMENDO

- La quantificazione esistenziale si fa con:
 - Exists (il più espressivo)
 - Giunzione
 - =Any, >Any, <Any...
 - IN
- =Any, >Any, <Any, IN,... non aggiungono potere espressivo
- Il problema vero è: non confondere esistenziale con universale!

LA QUANTIFICAZIONE UNIVERSALE

- Gli studenti che hanno preso solo 30
- Errore comune (e grave):

```
SELECT s.Nome  
FROM Studenti s, Esami e  
WHERE e.Matricola = s.Matricola AND e.Voto = 30
```

- In stile OQL ($\forall e \in \text{Esami-Di}(s): e.Voto = 30$):

```
SELECT s.Nome  
FROM Studenti s  
WHERE FORALL Esami e WHERE e.Matricola = s.Matricola : e.Voto = 30
```

LA QUANTIFICAZIONE UNIVERSALE

- Gli studenti che hanno preso solo 30

```
SELECT s.Nome
```

```
FROM Studenti s
```

```
WHERE FORALL Esami e WHERE e.Matricola = s.Matricola : e.Voto = 30
```

- ????

```
SELECT s.Nome
```

```
FROM Studenti s
```

```
WHERE FORALL Esami e WHERE e.Voto = 30: e.Matricola = s.Matricola
```

LA QUANTIFICAZIONE UNIVERSALE

- Prima scrivete:

```
SELECT s.Nome FROM Studenti s
WHERE FORALL Esami e WHERE e.Matricola = s.Matricola : e.Voto = 30)
```

- Poi traducete $\forall e \in E. p$ in $\neg \exists e \in E. \neg p$
($\neg \exists e. e \in E \wedge \neg p = \forall e. \neg(e \in E \wedge \neg p) = \forall e. \neg e \in E \vee p = \forall e. (e \in E \Rightarrow p) = \forall e \in E. p$):

```
SELECT s.Nome FROM Studenti s
WHERE NOT EXIST Esami e WHERE e.Matricola = s.Matricola : e.Voto <>
30
```

- In SQL diventa:

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

LA QUANTIFICAZIONE UNIVERSALE CON ALL

- La query:

```
SELECT s.Nome FROM Studenti s
WHERE FORALL Esami e WHERE e.Matricola = s.Matricola : e.Voto =
    30)
```

- Diventa:

```
SELECT s.Nome FROM Studenti s
WHERE 30 =ALL (SELECT e.Voto FROM Esami e
                WHERE e.Matricola = s.Matricola )
```

LA QUANTIFICAZIONE UNIVERSALE CON ALL

- Sostituendo EXISTS con =ANY, la solita query (studenti con tutti 30):
SELECT s.Nome FROM Studenti s
WHERE **NOT EXISTS** (SELECT * FROM Esami e
 WHERE e.Matricola = s.Matricola
 AND e.Voto <> 30)
- Diventa:
SELECT s.Nome FROM Studenti s
WHERE **NOT**(s.Matricola =**ANY** (SELECT e.Matricola FROM Esami e
 WHERE e.Voto <> 30))
- Ovvero:
SELECT s.Nome FROM Studenti s
WHERE s.Matricola <>**ALL** (SELECT e.Matricola FROM Esami e
 WHERE e.Voto <> 30)
- Naturalmente, <>**ALL** è lo stesso di **NOT IN**...

LA QUANTIFICAZIONE UNIVERSALE E GLI INSIEMI VUOTI

- Trovare gli studenti che hanno preso solo trenta:

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

- Perché trovo anche Rossi? Cosa cambia se invece di **NOT EXISTS** uso **<>ALL** oppure **NOT IN**?

Nome	Matricola	Provincia	AnnoNascita
Bianco	1	PI	1996
Verdi	2	PI	1992
Rossi	3	PI	1992

Mater.	Matricola	Voto
RC	1	30
IS	2	30
RC	2	20

GLI INSIEMI VUOTI

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

```
SELECT s.Nome
FROM Studenti s
WHERE NOT EXISTS (SELECT *
                  FROM Esami e
                  WHERE e.Matricola = s.Matricola AND e.Voto < 27)
AND EXISTS (SELECT *
            FROM Esami e
            WHERE e.Matricola = s.Matricola)
```

- Oppure:

```
SELECT s.Nome
FROM Studenti s, Esami e
WHERE s.Matricola = e.Matricola
GROUP BY s.Matricola, s.Nome
HAVING Min(e.Voto) >= 27
```

OTTIMIZZARE IL NOT EXISTS

- Loop interno valutato una volta per ogni studente:

```
SELECT s.Nome  
FROM Studenti s  
WHERE NOT EXISTS (SELECT *  
                  FROM Esami e  
                  WHERE e.Matricola = s.Matricola AND e.Voto > 21)
```

- Loop interno valutato una sola volta (loop interno decorrelato):

```
SELECT s.Nome  
FROM Studenti s  
WHERE s.Matricola NOT IN (SELECT e.Matricola  
                          FROM Esami e  
                          WHERE e.Voto > 21)
```

NOT IN ED OUTER JOIN

- Loop interno valutato una sola volta (loop interno *decorrelato*):

```
SELECT s.Nome  
FROM Studenti s  
WHERE s.Matricola NOT IN (SELECT e.Matricola  
                           FROM Esami e  
                           WHERE e.Voto > 21)
```

- Outer join:

```
SELECT s.Nome  
FROM Studenti s LEFT JOIN (SELECT *  
                           FROM Esami e  
                           WHERE e.Voto > 21) USING Matricola  
WHERE e.Voto IS NULL
```

LEFT OUTER JOIN

R

A	B
1	a
2	b
3	c

S

A	C
1	x
3	y
5	z

SELECT *
FROM R
NATURAL JOIN
S;

A	B	C
1	a	x
3	c	y

Chiamato anche: natural inner join

R

A	B
1	a
2	b
3	c

S

A	C
1	x
3	y
5	z

SELECT *
FROM R
NATURAL LEFT JOIN
S;

A	B	C
1	a	x
2	b	
3	c	y

Chiamato anche : natural left outer join

OUTER JOIN: RIGHT, FULL

R

A	B
1	a
2	b
3	c

S

A	C
1	x
3	y
5	z

SELECT *
FROM R
NATURAL RIGHT JOIN
S;

A	B	C
1	a	x
3	c	y
5		z

Chiamato anche : natural right outer join

R

A	B
1	a
2	b
3	c

S

A	C
1	x
3	y
5	z

SELECT *
FROM R
NATURAL FULL JOIN
S;

A	B	C
1	a	x
2	b	
3	c	y
5		z

Chiamato anche : natural full outer join

SQL PER MODIFICARE I DATI

- INSERT INTO Tabella [(A1,...,An)]
(VALUES (V1,...,Vn) | AS Select)
- UPDATE Tabella
SET Attributo = Expr, ..., Attributo = Expr
WHERE Condizione
- DELETE FROM Tabella
WHERE Condizione