

- 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 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 x" si denota come: A oppure $R.A$ oppure $x.A$

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

- **Le tabelle si possono combinare usando:**
 - **"," (prodotto): FROM T1,T2**
 - **Giunzioni di vario genere.**

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

- 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]} ]
```

- 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

- Trovare tutti i dati degli studenti di Pisa:

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

Nome	Matricola	Provincia	AnnoNascita
Isaia	171523	PI	1985
Bonini	175649	PI	1985

- 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	1985
Bonini	175649	1985

- Trovare tutte le possibili coppie (Studente, Esame):

```
SELECT *  
FROM Studenti, Esami
```

- Trovare tutte le possibili coppie (Studente, Esame sostenuto dallo studente):

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

- 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 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)

- 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
```

- 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:

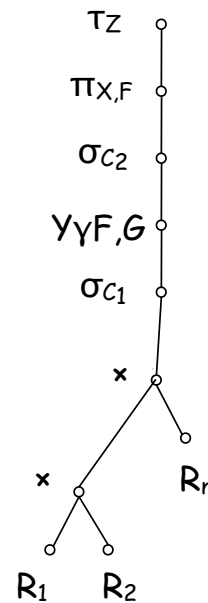
```
GROUP BY s.Matricola, s.Nome
```

- 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

- `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...A_n (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

SELECT DISTINCT X, F
 FROM R_1, \dots, R_n
 WHERE C_1
 GROUP BY Y
 HAVING C_2
 ORDER BY Z

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



LA CLAUSOLA HAVING: IMPORTANTE

- **Attenzione:**
 - Le clausole **HAVING** e **SELECT** citano solo:
 - espressioni su attributi di raggruppamento;
 - funzioni di aggregazione applicate ad attributi non di raggruppamento.


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

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



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

Candidato	Count(*)	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

- 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)

- 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  $\neq$  30
```

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

- 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)
```

- 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
```

- ANY non fa nulla in più di 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)
```

- 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)
```

- 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!

- 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
```

- 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 <>
```

- 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...

- 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        | 1970        |
| Verdi  | 2         | PI        | 1980        |
| Rossi  | 3         | PI        | 1980        |

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

- 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 Max(e.Voto) >= 27
```

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