

Istruzioni DML di SQL

Pag. 99 par.4



Costrutto: INSERT INTO

(Inserimento di dati in tabelle)

■ Sintassi:

```
INSERT INTO <Nome_Tabella> [(<Attributo1>, <Attributo2>, ..., <AttributoN>)]  
VALUES (<Valore1>, <Valore2>, ... <ValoreN>);
```

- Se si omette l'elenco degli attributi allora l'ordine ed il numero dei valori inseriti deve corrispondere alla sequenza precisamente delle colonne della tabella.



Costrutto: INSERT INTO

(Inserimento di dati in tabelle)

■ Es:

- `INSERT INTO visita (Id_Visita, DataVisita, Peso, Altezza, PressMin, PressMax, Id_Paziente)
VALUES (NULL, '2005-02-12', 82, 1.72, 80, 160, 4);`
- `INSERT INTO visita (Id_Visita, DataVisita, Id_Paziente)
VALUES (NULL, '2005-02-12', 5);`
- `INSERT INTO visita
VALUES (NULL, '2005-02-13', 64, 1.58, 56, 115, 9);`



Costrutto: UPDATE

(Modifica dati di una singola tupla o più tuple)

■ Sintassi:

```
UPDATE <Nome_Tabella>  
SET   <Attributo1> = <Espressione1>,  
      <Attributo2> = <Espressione2>,  
      .....  
      <AttributoN> = <EspressioneN>  
[WHERE <Condizione>];
```

- La “condizione” determina eventualmente su quale tupla o insieme di tuple è necessario agire.



Costrutto: UPDATE

(Modifica dati di una singola tupla o più tuple)

■ Es:

- UPDATE visita
SET Peso= 69,
Altezza= 1.7,
PressMin= 48,
PressMax= 96,
Id_Paziente= 7
WHERE Id_Visita=10;

- UPDATE Alunno
SET Nome= "Cristian"
WHERE Matricola=50;

- UPDATE Dipendente
SET Stipendio = Stipendio + 100;



Costrutto: DELETE

(Per eliminare una o più tuple)

■ Sintassi:

```
DELETE FROM <Nome_Tabella>  
[WHERE <Condizione>];
```

- La “condizione” determina eventualmente su quale tupla o insieme di tuple è necessario agire.



Costrutto: DELETE

(Per eliminare una o più tuple)

■ Es:

- DELETE FROM alunno
WHERE nome = "Noemi";
- DELETE FROM alunno;
- DELETE FROM dipendente
WHERE StipendioLordo > 5000.00;
- DELETE FROM dipendente
WHERE Data <= "1990/12/31";


Costrutto: SELECT

(Per estrarre dei dati dal database → effettuare una query o interrogazione)

■ Sintassi:

```
SELECT [DISTINCT] <Attributo1>, <Attributo2>, ..., <AttributoN>  
FROM <Tabella1> [,<Tabella2>, ..., <TabellaN>]  
[WHERE <Condizione>];
```

- La query restituisce una tabella con le colonne relative agli attributi indicati (* indica tutte le colonne) della tabella specificata
(se sono presenti più di una tabella allora l'estrazione avverrà dalla tabella prodotto cartesiano di quelle indicate)
- La condizione ci determina quali righe visualizzare
- La clausola DISTINCT consente di visualizzare esclusivamente righe non duplicate

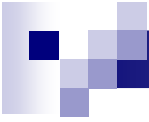


Costrutto: SELECT

(Per estrarre dei dati dal database → effettuare una query o interrogazione)

■ Es:

- `select * from Paziente;`
(vedi file 23-02-2009.txt riga 129)
- `select Nome, Cognome from Paziente;`
(vedi file 23-02-2009.txt riga 146)
- `select Nome, Cognome, CodASL from Paziente
where Provincia="MI";`
(vedi file 23-02-2009.txt riga 180)
- `select * from visita, paziente;`
(vedi file 12-03-2009.txt riga 42)
- `select Nome, Cognome, Peso
from visita, paziente
where Peso <= 80 And Peso >= 70;`
(vedi file 12-03-2009.txt riga 4)



Costrutto: SELECT


(Per estrarre dei dati dal database → effettuare una query o interrogazione)

■ Operazione di Selezione (o restrizione σ)

- L'operazione viene eseguita grazie al costrutto “select *...” con la clausola inserita nel “where”

■ Es:

- $\sigma_{\text{Nome} = \text{“Gabriele”}}$ (Alunni)
 - `Select * from Alunni
where Nome=“Gabriele”;`
- $\sigma_{\text{Matricola} \geq 60}$ (Alunni)
 - `Select * from Alunni
where Matricola>=60;`



Costrutto: SELECT

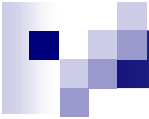
(Per estrarre dei dati dal database → effettuare una query o interrogazione)

■ Operazione di Proiezione (π)

- L'operazione viene eseguita grazie al costrutto “select ...” elencando gli attributi da estrapolare e senza la clausola “where”

■ Es:

- $\pi_{\text{matricola, nome}}$ (Alunni)
 - `Select matricola, nome from Alunni`



Costrutto: SELECT

(Per estrarre dei dati dal database → effettuare una query o interrogazione)


■ Operazione di Proiezione di Selezione

($\pi_{\langle \text{campi} \rangle} \sigma_{\langle \text{clausola} \rangle} (\langle \text{Tabella} \rangle)$)

- L'operazione viene eseguita grazie al costrutto “select” elencando gli attributi e specificando la clausola “where”

■ Es:

- $\pi_{\text{matricola, nome}} (\sigma_{\text{matricola} \leq 60} (\text{Alunni}))$
 - `Select matricola, nome from Alunni
where matricola < 60;`



Costrutto: SELECT

(Per estrarre dei dati dal database → effettuare una query o interrogazione)

- Operazione di creazione nuova tabella da dati estrapolati grazie al costrutto select

- create table prova

- select * from visita

- where Altezza < 1.70;

- (vedi file 23-02-2009.txt riga 436 e successive)



Operazione di join

■ Vi sono vari tipi di join noi ne analizzeremo solo tre tipi:

- **Join** → utilizzato per effettuare il prodotto cartesiano
 - Lo stesso risultato si può ottenere con la “,” fra le tabelle interessate
 - Oppure con “inner join senza la clausola “ON”

- **Inner Join** → utilizzato per effettuare il join semplice (con “ON” e clausola)
 - Lo stesso risultato si può ottenere con la “select *” e condizione opportuna “where”

- **Natural Join** → utilizzato per effettuare Equi-join
 - Lo stesso risultato si può ottenere con “select <Attribui>” e condizione opportuna “where”



Costrutto join

(Per effettuare il prodotto cartesiano)

■ Sintassi join:

```
SELECT *  
FROM <Tabella1> JOIN <Tabella2>;
```

■ Con l'utilizzo del costrutto "select *" e ","

```
SELECT *  
FROM <Tabella1> ,<Tabella2>;
```

■ Con l'utilizzo del costrutto "Inner Join"

```
SELECT *  
FROM <Tabella1> INNER JOIN <Tabella2>;
```



Costrutto join

(Per effettuare il prodotto cartesiano)

■ Es:

`select * from paziente join visita;`
(vedi file 12-03-2009.txt riga 325)

`select * from paziente, visita;`
(vedi file 12-03-2009.txt riga 462)

`select * from paziente inner join visita;`
(vedi file 12-03-2009.txt riga 599)



Costrutto Inner join

(Per effettuare il join semplice)

■ Sintassi Inner join:

```
SELECT *  
FROM <Tabella1> INNER JOIN <Tabella2> ON <Tabella1.Attributo> = <Tabella2.Attributo>;
```

■ Con l'utilizzo del costrutto “select *” e “where”

```
SELECT *  
FROM <Tabella1> , <Tabella2> where <Tabella1.Attributo> = <Tabella2.Attributo>;
```



Costrutto Inner join

(Per effettuare il join semplice)

■ Es:

- `select * from paziente inner join visita
ON paziente.id_paziente=visita.id_paziente;`
(vedi file 12-03-2009.txt riga 738)

- `select * from paziente, visita
where paziente.id_paziente=visita.id_paziente;`
(vedi file 12-03-2009.txt riga 759)



Costrutto Natural join

(Per effettuare Equi-join fra tabelle legate con chiave esterne)

■ Sintassi Natural join:

```
SELECT *  
FROM <Tabella1> NATURAL JOIN <Tabella2> ;
```

■ Con l'utilizzo del costrutto “select <Attributi>” e “where”

```
SELECT <Attributo1>, <Attributo2>,....<AttributoN>  
FROM <Tabella1> , <Tabella2> where <Tabella1.Attributo> = <Tabella2.Attributo>;
```



Costrutto Natural join

(Per effettuare Equi-join fra tabelle legate con chiave esterne)

■ Es:

- `select * from paziente natural join visita;`
(vedi file 12-03-2009.txt riga 779)



Costrutto: INTERSECT

(Per effettuare l'operazione di INTERSEZIONE fra relazioni compatibili)

■ Es:

- Consideriamo le seguenti due relazioni compatibili:
REGISTA (CodRegista, Cognome, Nome)
ATTORE (CodAttore, Cognome, Nome)

- Per ottenere una tabella che mi fornisca i registi che sono stati anche attori:
(Select Cognome, Nome From REGISTA)
INTERSECT
(Select Cognome, Nome From ATTORE);

Nota: Le parentesi sono obbligatorie



Costrutto: EXCEPT

(Per effettuare l'operazione di DIFFERENZA fra relazioni compatibili)

■ Es:

- Consideriamo le seguenti due relazioni compatibili:
REGISTA (CodRegista, Cognome, Nome)
ATTORE (CodAttore, Cognome, Nome)

- Per ottenere una tabella che mi fornisca i registi che NON sono MAI stati attori:

```
(Select Cognome, Nome From REGISTA)  
EXCEPT  
(Select Cognome, Nome From ATTORE);
```

Nota: Le parentesi sono obbligatorie



Costrutto: UNION

(Per effettuare l'operazione di UNIONE fra relazioni compatibili)

■ Es:

- Consideriamo le seguenti due relazioni compatibili:
REGISTA (CodRegista, Cognome, Nome)
ATTORE (CodAttore, Cognome, Nome)

- Per ottenere una tabella che mi fornisca TUTTI i registi e TUTTI gli attori:
`(Select Cognome, Nome From REGISTA)`
`UNION`
`(Select Cognome, Nome From ATTORE);`

Nota: Le parentesi sono obbligatorie



Funzioni di aggregazione

- In SQL vi sono alcune funzioni predefinite utili nelle circostanze in cui risulta necessario effettuare dei conteggi, somme, calcoli di medie o altro.
- **COUT** → Effettua il CONTEGGIO delle tuple con eventualmente una restrizione sulle tuple data da condizioni su una o più colonne
- **MIN** → Determina il valore MINIMO inserito all'interno di una determinata colonna con eventualmente una restrizione sulle tuple data da condizioni su una o più colonne
- **MAX** → Determina il valore MASSIMO inserito all'interno di una determinata colonna con eventualmente una restrizione sulle tuple data da condizioni su una o più colonne
- **SUM** → Determina il valore SOMMA dei dati di una determinata colonna con eventualmente una restrizione sulle tuple data da condizioni su una o più colonne
- **AVG** → Determina LA MEDIA dei dati all'interno di una determinata colonna con eventualmente una restrizione sulle tuple data da condizioni su una o più colonne



Funzioni di aggregazione

- **COUNT** → Effettua il CONTEGGIO delle tuple con eventualmente una restrizione sulle tuple data da condizioni su una o più colonne
- **Es:**
 - `select count(*) from paziente;`
Conteggio numero di tuple (vedi file 13-03-2009.txt riga 79)
 - `select count(*) from visita
where PressMin<80;`
Con condizione PressMin<80 (vedi file 13-03-2009.txt riga 96)
 - `select count(*) from visita
where PressMin<=60 And Peso<=80;`
Con condizione Composta (vedi file 13-03-2009.txt riga 114)
 - `select count(*) from paziente
where DataNascita >="1970-01-01";`
Con condizione su una data (vedi file 13-03-2009.txt riga 123)



Funzioni di aggregazione

- **MIN** → Determina il valore MINIMO inserito all'interno di una determinata colonna con eventualmente una restrizione sulle tuple data da condizioni su una o più colonne
- **Es:**
 - **select min(Peso) from visita;**
Individua il minimo peso rilevato nelle varie visite (vedi file 13-03-2009.txt riga 148)
 - **select min(Cognome) from paziente;**
Individua il minimo cognome (in ordine alfabetico) fra i vari pazienti (vedi file 13-03-2009.txt riga 156)
 - **select min(DataNascita) from paziente;**
Individua la minima DataNascita (relativo al più anziano) fra i vari pazienti (vedi file 13-03-2009.txt riga 172)



Funzioni di aggregazione

- **MAX** → Determina il valore MASSIMO inserito all'interno di una determinata colonna con eventualmente una restrizione sulle tuple data da condizioni su una o più colonne
- **Es:**
 - **select max(Peso) from visita;**
Individua il massimo peso rilevato nelle varie visite (vedi file 13-03-2009.txt riga 140)
 - **select max(Cognome) from paziente;**
Individua il massimo cognome (in ordine alfabetico) fra i vari pazienti (vedi file 13-03-2009.txt riga 164)
 - **select max(DataNascita) from paziente;**
Individua la massima DataNascita (relativo al più giovane) fra i vari pazienti (vedi file 13-03-2009.txt riga 180)



Funzioni di aggregazione

- **SUM** → Determina il valore SOMMA dei dati di una determinata colonna con eventualmente una restrizione sulle tuple data da condizioni su una o più colonne
- Es:
 - `select sum(Peso) from visita;`
Somma di tutti i pesi rilevati nelle varie visite
(vedi file 13-03-2009.txt riga 132)



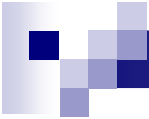
Funzioni di aggregazione

- **AVG** → Determina LA MEDIA dei dati all'interno di una determinata colonna con eventualmente una restrizione sulle tuple data da condizioni su una o più colonne
- **Es:**
 - `select avg(Peso) from visita;`
Determina la media fra i vari pesi rilevati nelle varie visite (vedi file 13-03-2009.txt riga 188)
 - `select avg(Peso) from visita
where Altezza < 1.70;`
Determina la media fra i vari pesi rilevati relativamente alle visite dove si è rilevato un'altezza < 1,70 (vedi file 13-03-2009.txt riga 197)



Clausola di raggruppamento

- **Le funzioni di aggregazione sono generalmente abbinate alla clausola di raggruppamento.**
Ci consente di raggruppare le tuple in funzione del valore che assumono in una determinata colonna.
Inoltre con la clausola “Having” riusciamo a filtrare solo alcuni gruppi e scartarne altri.
La clausola di raggruppamento viene inserita come ultima parte di un’eventuale interrogazione.
- **Sintassi:**
 - **GROUP BY <Attributo1>,....<AttributoN> [Having <Condizione gruppo>]**
- **Es: vedi testo a pag. 113**



Funzioni di Ordinamento

- In SQL vi sono alcune funzioni predefinite utili nelle circostanze in cui risulta necessario effettuare l'ordinamento dei dati in ordine crescente o decrescente rispetto ai valori inseriti in una o più colonne, con eventualmente una restrizione sulle tuple data da opportune condizioni su una o più colonne
- Si tratta di operare su opportune tabelle ottenute secondo le varie esigenze e come ultima parte della query inserire l'esigenza di ordinamento.
- L'ultima parte dovrà essere scritta secondo la seguente sintassi:

`ORDER BY <Attributo1>[ASC|DESC],....<AttributoN>[ASC|DESC]`

- ASC → Ordinamento Crescente
- DESC → Ordinamento Decrescente
- Qualora non si dovesse indicare `[ASC|DESC]` di default viene inteso ASC



Funzioni di Ordinamento

■ Es:

- `select * from paziente
order by cognome;`

Ordinamento delle tuple in base al cognome (Crescente A→Z)
(vedi file 13-03-2009.txt riga 206)

- `select * from paziente
order by cognome desc;`

Ordinamento delle tuple in base al cognome (Decrescente Z→A)
(vedi file 13-03-2009.txt riga 224)

- `select * from paziente
where Provincia = "MI"
order by cognome asc;`

Ordinamento delle tuple in base al cognome (Crescente A→Z)
dei pazienti che hanno come provincia di residenza "MI"
(vedi file 13-03-2009.txt riga 243)

Funzioni di Ordinamento

■ Es:

□ `select * from paziente
order by CodAsl desc, Cognome desc;`

Ordinamento delle tuple prima in base al CodAsl (Decrescente $Z \rightarrow A$) e fra quelli aventi lo stesso CodAsl, ordinati in base al cognome (Decrescente $Z \rightarrow A$) (vedi file 13-03-2009.txt riga 254)

□ `select * from paziente
order by CodAsl asc, Cognome asc;`

Ordinamento delle tuple prima in base al CodAsl (Crescente $A \rightarrow Z$) e fra quelli aventi lo stesso CodAsl, ordinati in base al cognome (Crescente $A \rightarrow Z$) (vedi file 13-03-2009.txt riga 272)

□ `select * from paziente
where CodAsl = "A040"
order by Cognome asc, Nome asc;`

Filtraggio delle sole tuple con CodAsl=A040 e queste proposte con Ordinamento delle tuple prima in base al Cognome (Crescente $A \rightarrow Z$) e fra quelli aventi lo stesso cognome, ordinati in base al Nome (Crescente $A \rightarrow Z$) (vedi file 13-03-2009.txt riga 292)



Interrogazioni e sottointerrogazioni annidate

- **Per rispondere a query complesse è possibile strutturare opportunamente più comandi SELECT. Ciò consente di costruire un'interrogazione al cui interno sono presenti altre interrogazioni, dette sottointerrogazioni o subquery.**
- La query secondaria o anche detta sottoquery può essere un solo valore (tabella scalare) o una sola riga (con una sola colonna) o una tabella con più righe e più colonne. Spesso si preferisce dare un nuovo nome alla tabella derivante dalla sottoquery utilizzando la clausole "AS"
- Es: vedi testo a pag. 114



Interrogazioni e sottointerrogazioni annidate

- **Per rispondere a query complesse è possibile strutturare opportunamente più comandi SELECT. Ciò consente di costruire un'interrogazione al cui interno sono presenti altre interrogazioni, dette sottointerrogazioni o subquery.**
- La query secondaria o anche detta sottoquery può essere un solo valore (tabella scalare) o una sola riga (con più colonne) o una tabella con più righe e più colonne. Spesso si preferisce dare un nuovo nome alla tabella derivante dalla sottoquery utilizzando la clausole "AS"
- Es: vedi testo a pag. 114



Conservazione dei risultati parziali

- **A volte è necessario conservare in una opportuna nuova tabella i risultati di una interrogazione per successive interrogazioni. Per far questo occorre far precedere il comando “Select” dal comando “Create Table” seguito dal nome della nuova tabella che si vuole creare.
La nuova tabella così creata entra a far parte del Database.**
- **Se possibile è sempre preferibile utilizzare il sistema dell’annidamento piuttosto che creare tabelle aggiuntive.**
- **Es: vedi testo a pag. 115**