

# Il linguaggio SQL: DML di base

Sistemi Informativi T

Versione elettronica: [04.2.SQL.DMLbase.pdf](#)

# Data Manipulation Language (DML)

- Le istruzioni del DML di SQL sono

<b>SELECT</b>	esegue interrogazioni ( <b>query</b> ) sul DB
<b>INSERT</b>	inserisce nuove tuple nel DB
<b>DELETE</b>	cancella tuple dal DB
<b>UPDATE</b>	modifica tuple del DB

**NB:** In ambito DBMS (e quindi nella manualistica) si parla spesso di:

- **TABLE**, intesa come sinonimo di relazione (ma c'è qualche differenza)
- **ROW**, sinonimo di tupla
- **COLUMN**, sinonimo di attributo

# DB di riferimento per gli esempi

## Università

### Studenti

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	13/04/1978	anna.rossi@yahoo.it
39654	Verdi	Marco	20/09/1979	mverdi@mv.com
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

### Corsi

CodCorso	Titolo	Docente	Anno
483	Analisi	Biondi	1
729	Analisi	Neri	1
913	Sistemi Informativi	Castani	2

### Esami

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
29323	913	26	NO
35467	913	30	NO

# L'istruzione SELECT

- È l'istruzione che permette di eseguire interrogazioni (*query*) sul DB
- La forma di base è:

```
SELECT    A1,A2,...,Am
FROM      R1,R2,...,Rn
WHERE     <condizione>
```

ovvero:

- clausola SELECT (cosa si vuole come risultato)
- clausola FROM (da dove si prende)
- clausola WHERE (che condizioni deve soddisfare)

## NB:

- Non è obbligatorio scrivere le clausole su righe separate
- In ogni istruzione SQL si possono inserire commenti,  
con -- se su singola riga  
oppure /\* in questo modo se si tratta  
di commenti su più righe \*/

# Estrarre tutti i dati

- La query:

```
SELECT *  
FROM Corsi  
-- la nostra prima query!
```

CodCorso	Titolo	Docente	Anno
483	Analisi	Biondi	1
729	Analisi	Neri	1
913	Sistemi Informativi	Castani	2

restituisce l'istanza della relazione Corsi (\* significa: tutti gli attributi)

- Con  

```
SELECT CodCorso, Titolo, Anno  
FROM Corsi
```

otteniamo informazioni solo per gli attributi specificati (**proiezione**)

CodCorso	Titolo	Anno
483	Analisi	1
729	Analisi	1
913	Sistemi Informativi	2

# Tabelle vs Relazioni: righe replicate

- Se tra le colonne su cui si proietta non compare nessuna chiave, può capitare che si generino delle righe duplicate
- Ad esempio:

```
SELECT Titolo  
FROM Corsi
```

Titolo
Analisi
Analisi
Sistemi Informativi

- La keyword **DISTINCT** elimina dal risultato le righe duplicate, che di default vengono mantenute:

```
SELECT DISTINCT Titolo  
FROM Corsi
```

Titolo
Analisi
Sistemi Informativi

# Espressioni nella clausola SELECT

- In output è anche possibile ottenere risultati di **espressioni** (numeriche, su stringhe, ecc.), che vengono valutate sulle tuple del risultato

```
SELECT  Matricola, Voto/3
FROM    Esami
```

Matricola	
29323	9
39654	10
29323	8
35467	10

- Si noti che:
  - la seconda colonna non ha un nome
  - il tipo dei valori non cambia (resta un intero)

# Casting

- Il secondo problema si risolve o eseguendo un **casting** esplicito di **Voto**:

```
SELECT Matricola, CAST(Voto AS Decimal(4,2))/3
FROM Esami
```

oppure forzando l'espressione a fornire un valore reale:

Matricola	
29323	9.33
39654	10.00
29323	8.66
35467	10.00

```
SELECT  Matricola, Voto/3.0
FROM    Esami
```

a cui volendo si può aggiungere un casting per limitare il numero di cifre decimali

Matricola	
29323	9.333333333333333333
39654	10.000000000000000000
29323	8.666666666666666666
35467	10.000000000000000000

# Ridenominare le colonne

- Per risolvere il primo problema si sfrutta la possibilità che SQL offre di **ridenominare le colonne**, caratteristica utile anche quando non si fa uso di espressioni al fine di migliorare la leggibilità del risultato:

```
SELECT Titolo AS NomeCorso, Docente AS Prof
FROM Corsi
```

in cui la keyword **AS** può anche essere omessa

NomeCorso	Prof
Analisi	Biondi
Analisi	Neri
Sistemi Informativi	Castani

- L'esempio visto si può quindi riscrivere come:

```
SELECT Matricola,
       CAST(Voto AS Decimal(4,2))/3 AS Decimi
FROM Esami
```

Matricola	Decimi
29323	9.33
39654	10.00
29323	8.66
35467	10.00

# Espressioni con stringhe

- Per lavorare con la stringhe è utile l'operatore CONCAT (esiste anche una funzione binaria analoga)

```
SELECT  Matricola,  
Nome CONCAT ' ' CONCAT Cognome  
        AS NomeCognome  
FROM  Studenti
```

Matricola	NomeCognome
29323	Giorgio Bianchi
35467	Anna Rossi
39654	Marco Verdi
42132	Lucia Neri

- E' possibile concatenare (senza bisogno di casting esplicito) numeri e date, e fare uso di costanti

```
SELECT  Matricola CONCAT ' ha preso '  
        CONCAT Voto CONCAT ' in '  
        CONCAT CodCorso  
FROM  Esami
```

29323 ha preso 28 in 483
39654 ha preso 30 in 729
29323 ha preso 26 in 913
35467 ha preso 30 in 913

# Filtrare il risultato: la clausola WHERE

- Per **selezionare** le sole tuple di interesse dobbiamo scrivere una **condizione**, che sia vera solo per tali tuple

```
SELECT Matricola, Voto, Lode
FROM   Esami
WHERE  CodCorso = 913
```

Matricola	Voto	Lode
29323	26	NO
35467	30	NO

- Si ottiene in questo modo:
  - La clausola FROM dice di prendere la tabella Esami
  - La clausola WHERE dice di prendere solo le tuple per cui CodCorso = 913
  - Infine, si estraggono i valori degli attributi nella SELECT list
- Equivale a  $\pi_{\text{Matricola,Voto,Lode}} (\sigma_{\text{CodCorso} = 913} (\text{Esami}))$

# Clausola WHERE = espressione logica

- La clausola WHERE consiste, nel caso generale, di una espressione logica (operatori AND, OR, NOT) di predicati
- Una tupla soddisfa la clausola WHERE se e solo se l'espressione risulta vera per tale tupla

```
SELECT *  
FROM   Esami  
WHERE  CodCorso = 913  
AND    Voto > 28
```

Matricola	CodCorso	Voto	Lode
35467	913	30	NO

# Alcuni utili operatori: LIKE

- L'operatore **LIKE**, mediante le “wildcard”
  - \_** (corrisponde a **un carattere arbitrario**) e
  - %** (corrisponde a **una stringa arbitraria**),permette di trovare stringhe che soddisfano un certo **“pattern”**

*Studenti la cui email finisce con '.it' e hanno una 'b' in seconda posizione*

```
SELECT *  
FROM Studenti  
WHERE Email LIKE '_b%.it'
```

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it

**gbianchi@alma.unibo.it**

# Alcuni utili operatori: BETWEEN

- L'operatore **BETWEEN** permette di esprimere condizioni di **appartenenza a un intervallo** (estremi inclusi)

*Esami con voto tra 26 e 29*

```
SELECT  *  
FROM    Esami  
WHERE   Voto BETWEEN 26 AND 29
```

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
29323	913	26	NO

- Lo stesso risultato si può ottenere scrivendo:

```
SELECT  *  
FROM    Esami  
WHERE   Voto >= 26 AND Voto <= 29
```

# Alcuni utili operatori: IN

- L'operatore **IN** permette di esprimere condizioni di **appartenenza a un insieme** di valori

*Esami dei corsi con codici 483 e 729*

```
SELECT  *  
FROM    Esami  
WHERE   CodCorso IN (483,729)
```

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	Sì

- Lo stesso risultato si può ottenere scrivendo:

```
SELECT  *  
FROM    Esami  
WHERE   CodCorso = 483  
OR      CodCorso = 729
```

# Valori nulli

- Vale quanto già visto in Algebra Relazionale, quindi data la relazione

## Studenti

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	NULL	anna.rossi@yahoo.it
39654	Verdi	Marco	20/09/1979	mverdi@mv.com
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

**SELECT**

**FROM**     **Studenti**

**WHERE**   **DataNascita > '31-12-1978'**

restituisce solo

Matricola	Cognome	Nome	DataNascita	Email
39654	Verdi	Marco	20/09/1979	mverdi@mv.com

# Verifica di valori nulli

- Né la condizione `DataNascita > '31-12-1978'` né il suo contrario (`DataNascita <= '31-12-1978'`) sono **vere** se il valore della data di nascita è NULL
- Per **verificare se un valore è NULL** si deve usare l'operatore **IS**

```
SELECT *  
FROM   Studenti  
WHERE  DataNascita IS NULL
```

Matricola	Cognome	Nome	DataNascita	Email
35467	Rossi	Anna	<b>NULL</b>	anna.rossi@yahoo.it

- **NOT (A IS NULL)**, che è vera se il valore dell'attributo A non è NULL, si scrive anche **A IS NOT NULL**

# Logica a 3 valori in SQL

- Nel caso di espressioni complesse, SQL ricorre alla **logica a 3 valori**: vero (V), falso (F) e “sconosciuto” (?)

Studenti	Matricola	Cognome	Nome	DataNascita	Email
	29323	Bianchi	Giorgio	21/06/1978	<b>NULL</b>
	35467	Rossi	Anna	<b>NULL</b>	anna.rossi@yahoo.it
	39654	Verdi	Marco	20/09/1979	mverdi@mv.com
	42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

```
SELECT Matricola
FROM Studenti
WHERE DataNascita <= '31-12-1978'
AND Email LIKE '%.edu'
```

Matricola
42132

```
WHERE DataNascita <= '31-12-1978'
OR Email LIKE '%.edu'
```

Matricola
29323
42132

# Ordinamento del risultato

- Per ordinare il risultato secondo i valori di una o più colonne si introduce la clausola **ORDER BY**, e per ogni colonna si specifica se l'ordinamento è per valori “ascendenti” (**ASC**, il default) o “discendenti” (**DESC**)
- Si ordina sulla prima colonna, a parità di valori di questa sulla seconda, e così via

```
SELECT      *  
FROM        Esami  
ORDER BY CodCorso, Voto DESC
```

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	Sì
35467	913	30	NO
29323	913	26	NO

```
SELECT      *  
FROM        Esami  
ORDER BY Voto DESC, CodCorso
```

Matricola	CodCorso	Voto	Lode
39654	729	30	Sì
35467	913	30	NO
29323	483	28	NO
29323	913	26	NO

# Interrogazioni su più tabelle

- A parte il caso del join naturale (peraltro non supportato da tutti i DBMS, incluso DB2), per combinare le tuple di due relazioni è necessario scrivere dei **predicati di join**
- Il problema è che se gli attributi su cui si fa il join hanno nomi uguali c'è ambiguità, ad es. che significa **Matricola = Matricola**?
- Il caso di join con condizione di join asimmetrica è illuminante:

Laureati

Matricola	VotoFinale
29323	89
39654	102

Concorsi

Codice	VotoFinale
XYZ	88
GHJ	99

- Data la richiesta *per ogni concorso trova i laureati il cui voto di laurea è maggiore o uguale al VotoFinale minimo richiesto per l'ammissione al concorso stesso*, scrivere **VotoFinale >= VotoFinale** non significa nulla!

# Pseudonimi per i nomi delle relazioni

- Per referenziare una colonna, in SQL si può anche fare uso della forma estesa **<nome tabella>.<nome colonna>**:

```
SELECT    Esami.CodCorso
FROM      Esami
WHERE     Esami.Matricola = '29323'
```

- Inoltre, per abbreviare la scrittura (ma non solo! vedi self-join) si può introdurre uno **pseudonimo** (*alias*) per la tabella

```
SELECT    E.CodCorso
FROM      Esami E                -- oppure Esami AS E
WHERE     Esami.Matricola = '29323'
```

# Interrogazioni con join

## ■ L'interrogazione

```
SELECT    S.Cognome , S.Nome , E.*  
FROM      Esami E,  Studenti S  
WHERE     E.Matricola = S.Matricola
```

si interpreta come segue:

- 1) FROM: Si genera il **prodotto Cartesiano** di Esami e Studenti
- 2) WHERE: Si applicano i predicati della clausola WHERE
- 3) SELECT: Si estraggono le colonne della SELECT list

Cognome	Nome	Matricola	CodCorso	Voto	Lode
Bianchi	Giorgio	29323	483	28	NO
Verdi	Marco	39654	729	30	Sì
Bianchi	Giorgio	29323	913	26	NO
Rossi	Anna	35467	913	30	NO

## ■ **E.Matricola = S.Matricola** è detto **predicato di join**

# Semantica di SQL ed esecuzione effettiva

- L'interpretazione (semantica) di una query SQL in termini di prodotto Cartesiano, applicazione dei predicati e quindi proiezione, **NON** va certo intesa come indicativa del modo in cui il risultato viene prodotto
- Ogni DBMS ha infatti le proprie strategie per determinare modalità efficienti di esecuzione, sfruttando regole di equivalenza che generalizzano quelle viste per l'Algebra Relazionale
  - Ad esempio il push-down delle selezioni
- Inoltre, a differenza dell'Algebra, in SQL è molto più laborioso ragionare su aspetti di equivalenza delle interrogazioni
- Questo verrà fatto solo per i casi più importanti, senza alcuna pretesa di completezza

# Altri esempi (1)

*i numeri di matricola degli studenti  
che hanno sostenuto l'esame di Analisi con il Prof. Biondi*

```
SELECT  E.Matricola
FROM    Corsi C, Esami E
WHERE   C.CodCorso = E.CodCorso
AND     C.Titolo = 'Analisi'
AND     C.Docente = 'Biondi'
```

CodCorso	Titolo	Docente	Anno
483	Analisi	Biondi	1
729	Analisi	Neri	1
913	Sistemi Informativi	Castani	2

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
29323	913	26	NO
35467	913	30	NO

Matricola	CodCorso	Voto	Lode	Titolo	Docente	Anno
29323	483	28	NO	Analisi	Biondi	1
39654	729	30	SÌ	Analisi	Neri	1
29323	913	26	NO	Sistemi Informativi	Castani	2
35467	913	30	NO	Sistemi Informativi	Castani	2

## Altri esempi (2)

*i docenti dei corsi di cui lo studente  
con matricola 29323 ha sostenuto l'esame*

```
SELECT  C.Docente
FROM    Corsi C, Esami E
WHERE   C.CodCorso = E.CodCorso
AND     E.Matricola = '29323'
```

CodCorso	Titolo	Docente	Anno
483	Analisi	Biondi	1
729	Analisi	Neri	1
913	Sistemi Informativi	Castani	2

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
29323	913	26	NO
35467	913	30	NO

Matricola	CodCorso	Voto	Lode	Titolo	Docente	Anno
29323	483	28	NO	Analisi	Biondi	1
<del>39654</del>	<del>729</del>	<del>30</del>	<del>SÌ</del>	<del>Analisi</del>	<del>Neri</del>	<del>1</del>
29323	913	26	NO	Sistemi Informativi	Castani	2
<del>35467</del>	<del>913</del>	<del>30</del>	<del>NO</del>	<del>Sistemi Informativi</del>	<del>Castani</del>	<del>2</del>

# Più di 1 tabella = 2,3,4,...

- Quanto fatto con 2 tabelle si generalizza al caso di 3 o più tabelle:

*i docenti dei corsi di cui lo studente*

*Giorgio Bianchi ha sostenuto l'esame*

```
SELECT C.Docente
FROM   Corsi C, Esami E, Studenti S
WHERE  C.CodCorso = E.CodCorso
AND    E.Matricola = S.Matricola
AND    S.Cognome = 'Bianchi'
AND    S.Nome = 'Giorgio'
```

# Self Join

- L'uso di alias è forzato quando si deve eseguire un self-join

*Chi sono i nonni di Anna?*

**Genitori G1**

Genitore	Figlio
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

**Genitori G2**

Genitore	Figlio
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

```
SELECT  G1.Genitore AS Nonno
FROM    Genitori G1, Genitori G2
WHERE   G1.Figlio = G2.Genitore
        AND      G2.Figlio = 'Anna'
```

Nonno
Giorgio
Silvia
Enzo

# Join espliciti

- Anziché scrivere i predicati di join nella clausola WHERE è possibile definire una( o più) *joined table* nella clausola FROM

```
SELECT S.*, E.CodCorso, E.Voto, E.Lode  
FROM  Studenti S JOIN Esami E ON (S.Matricola = E.Matricola)  
WHERE E.Voto > 26
```

in cui **JOIN** si può anche scrivere **INNER JOIN**

- Altri tipi di join espliciti sono:

**CROSS JOIN** (prodotto Cartesiano)

**LEFT [OUTER] JOIN**

**RIGHT [OUTER] JOIN**

**FULL [OUTER] JOIN**

**NATURAL JOIN** (non supportato da DB2)

# Outer join

- La semantica dell'outer join in SQL è:
  - PRIMA si applicano **TUTTI** i predicati del join
  - POI si completa il risultato con le tuple dangling
- Pertanto 

```
SELECT *  
FROM Studenti S LEFT JOIN Esami E ON  
  (S.Matricola = E.Matricola) AND (E.Anno = 2)  
WHERE E.CodCorso IS NULL
```

restituisce gli studenti senza esami nel secondo anno, mentre

```
SELECT *  
FROM Studenti S LEFT JOIN Esami E ON  
  (S.Matricola = E.Matricola)  
WHERE E.CodCorso IS NULL AND (E.Anno = 2)
```

ovviamente non restituisce nulla!

# Operatori insiemistici

- L'istruzione SELECT non permette di eseguire unione, intersezione e differenza di tabelle
- Ciò che si può fare è **combinare in modo opportuno i risultati di due istruzioni SELECT**, mediante gli operatori

**UNION, INTERSECT, EXCEPT**

- In tutti i casi gli elementi delle SELECT list devono avere tipi compatibili e gli stessi nomi se si vogliono colonne con un'intestazione definita
- **L'ordine degli elementi è importante (notazione posizionale)**
- Il risultato è in ogni caso privo di duplicati, per mantenerli occorre aggiungere l'opzione **ALL**:

**UNION ALL, INTERSECT ALL, EXCEPT ALL**

# Operatori insiemistici: esempi (1)

R

A	B
1	a
1	a
2	a
2	b
2	c
3	b

S

C	B
1	a
1	b
2	a
2	c
3	c
4	d

```
SELECT A
FROM R
UNION
SELECT C
FROM S
```

1
2
3
4

```
SELECT A
FROM R
UNION
SELECT C AS A
FROM S
```

A
1
2
3
4

```
SELECT A,B
FROM R
UNION
SELECT B,C AS A
FROM S
```

**Non corretta!**

```
SELECT B
FROM R
UNION ALL
SELECT B
FROM S
```

B
a
a
a
a
b
c
b
a
b
a
c
c
d

# Operatori insiemistici: esempi (2)

**R**

A	B
1	a
1	a
2	a
2	b
2	c
3	b

SELECT B FROM R  INTERSECT SELECT B FROM S	B a b c
---	------------------

SELECT B FROM S  EXCEPT SELECT B FROM R	B d
--	--------

**S**

C	B
1	a
1	b
2	a
2	c
3	c
4	d

SELECT B FROM R  INTERSECT ALL SELECT B FROM S	B a a b c
---	-----------------------

SELECT B FROM R  EXCEPT ALL SELECT B FROM S	B a b
--	-------------

# Istruzioni di aggiornamento dei dati

- Le istruzioni che permettono di aggiornare il DB sono:

**INSERT**      inserisce nuove tuple nel DB

**DELETE**      cancella tuple dal DB

**UPDATE**      modifica tuple del DB

- **INSERT** può usare il risultato di una query
- **DELETE** e **UPDATE** possono fare uso di condizioni per specificare le tuple da cancellare o modificare
- In ogni caso gli aggiornamenti riguardano una sola relazione

# Inserimento di tuple: valori espliciti

- Per inserire una o più tuple bisogna specificarne i valori, dicendo quale valore va assegnato a quale attributo

```
INSERT INTO Corsi(Titolo,CodCorso,Docente,Anno)
VALUES          ('StoriaAntica',456,'Grigi',3),
                ('StoriaModerna',457,'Gialli',2)
```

- Se la lista degli attributi viene omessa vale l'ordine con cui sono stati definiti

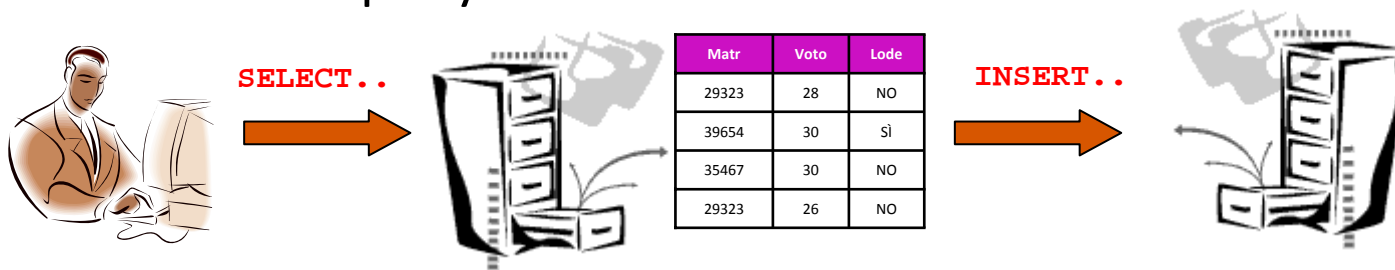
```
INSERT INTO Corsi
VALUES      (456,'StoriaAntica','Grigi',3)
```

- Se la lista non include tutti gli attributi, i restanti assumono valore NULL (se ammesso) o il valore di default (se specificato)

```
INSERT INTO Corsi(CodCorso,Titolo)
VALUES      (456,'StoriaAntica')
```

# Inserimento di tuple: risultato di query

- In alcuni casi si rende necessario inserire in una tabella le tuple che risultano da una query



- Si può fare direttamente! Ad esempio:

```
INSERT INTO StudentiSenzaEmail(Matr,Cog,Nom)
SELECT Matricola,Cognome,Nome
FROM Studenti
WHERE Email IS NULL
```

- Gli schemi del risultato e della tabella in cui si inseriscono le tuple possono essere diversi, l'importante è che i tipi delle colonne siano compatibili

# Cancellazione di tuple

- L'istruzione **DELETE** può fare uso di una **condizione** per specificare le tuple da cancellare:

```
DELETE FROM Corsi -- elimina i corsi di Biondi  
WHERE   Docente = 'Biondi'
```

- Per cancellare tutte le tuple (**attenzione!**):

```
DELETE FROM Corsi
```

- Che succede se la cancellazione porta a violare il vincolo di integrità referenziale? (ad es.: che accade agli esami dei corsi di Biondi?)

# Modifica di tuple

- Anche l'istruzione **UPDATE** può fare uso di una **condizione** per specificare le tuple da modificare e di espressioni per determinare i nuovi valori

```
UPDATE   Corsi
SET      Docente = 'Bianchi',
           Anno = 2
WHERE     Docente = 'Biondi'
```

```
UPDATE   Dipendenti
SET      Stipendio = 1.1*Stipendio -- aumento del 10%
WHERE     Ruolo = 'Programmatore'
```

- Anche l'UPDATE può portare a violare il vincolo di integrità referenziale

# Politiche di reazione (1)

- Anziché lasciare al programmatore il compito di garantire che a fronte di cancellazioni e modifiche i vincoli di integrità referenziale siano rispettati, si possono specificare opportune **politiche di reazione** in fase di definizione degli schemi
- **In assenza di specifiche opportune le violazioni non vengono ammesse.** Equivale a specificare esplicitamente:

```
CREATE TABLE Esami (  
    Matricola char(5) NOT NULL,  
    CodCorso  int     NOT NULL,  
    ...  
    FOREIGN KEY CodCorso REFERENCES Corsi  
        ON DELETE NO ACTION      -- cancellazioni non permesse  
        ON UPDATE NO ACTION      -- modifiche non permesse
```

# Politiche di reazione (2)

- Tuttavia sono possibili politiche alternative:

## DELETE:

### ON DELETE CASCADE

- vengono cancellate tutte le tuple che referenziano la tupla cancellata, e così via ricorsivamente ("in cascata")

### ON DELETE SET NULL

- se ammesso, la foreign key viene resa **NULL**

## UPDATE:

- lo standard prevede politiche analoghe a quelle del DELETE, ma non così DB2 che non ammette modifiche di chiavi referenziate

# Riassumiamo:

- L'istruzione **SELECT** consiste nella sua forma base di 3 parti: **SELECT**, **FROM** e **WHERE**
- A queste si aggiunge **ORDER BY**, per **ordinare** il risultato (e altre che vedremo)
- Per trattare i **valori nulli**, SQL ricorre a una **logica a 3 valori** (**vero**, **falso** e **sconosciuto**)