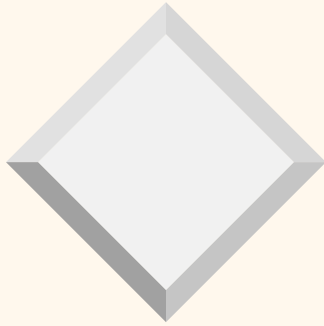


# *SQL: Definizione e Manipolazione di Relazioni*

## *Capitolo 2*



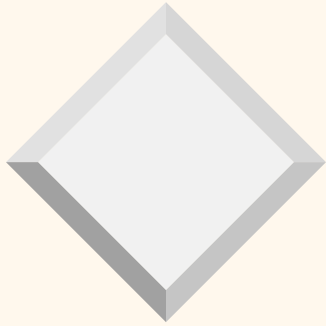
# *Sintassi della Creazione di uno Schema Relazionale*

```
CREATE TABLE <table> (  
<column_1> <data_type> [NOT NULL] [UNIQUE] [<column  
  constraint>], . . . , . . . ,  
<column_n> <data_type> [NOT NULL] [UNIQUE] [<column  
  constraint>],  
[<table constraint(s)>]  
) ;
```

```
[CONSTRAINT <name>] PRIMARY KEY | UNIQUE | NOT NULL
```

```
[CONSTRAINT <name>] [FOREIGN KEY (<column(s)>)] REFERENCES  
<table>[(<column(s)>)] [ON DELETE CASCADE]
```

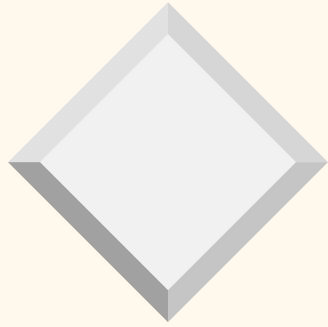
- ❖ La clausola `FOREIGN KEY` deve essere aggiunta alla clausola `REFERENCES` solo se la chiave esterna include più di una colonna. In questo caso il vincolo deve essere specificato come vincolo di tabella. Inoltre se viene riportato solo il nome della tabella che si sta referenziando allora la lista degli attributi che formano la chiave primaria di essa viene assunta come chiave esterna.



# *Creazione Schema Relazionale: vicoli su tabella*

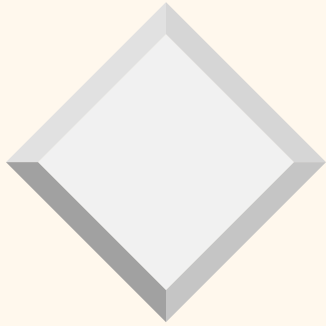
```
CREATE TABLE Studenti (sid CHAR(20),
                        nome CHAR(30),
                        login CHAR(20),
                        età INTEGER,
                        media REAL,
                        UNIQUE (nome,età),
                        CONSTRAINT ChiaveStudenti PRIMARY KEY (sid));
```

```
CREATE TABLE Iscrizioni (studid CHAR(20),
                          cid CHAR(20),
                          voto CHAR(10),
                          PRIMARY KEY (studid, cid),
                          FOREIGN KEY (studid) REFERENCES Studenti);
```



# *Creazione Schema Relazionale: vincoli su colonna*

```
CREATE TABLE EMP( empno NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,  
    . . . ,  
    deptno NUMBER(3) CONSTRAINT fk_deptno REFERENCES  
        DEPT(deptno) );
```

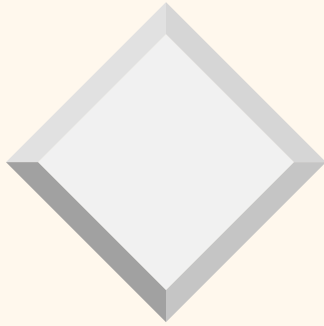


## *Altri esempi di Creazione di Schemi Relazionali*

```
CREATE TABLE Velisti (  vid INTEGER,  
                        vnome CHAR(20),  
                        esperienza INTEGER,  
                        età INTEGER,  
                        PRIMARY KEY (vid))
```

```
CREATE TABLE Barche (  bid INTEGER,  
                        bnome CHAR(30),  
                        colore CHAR(20),  
                        PRIMARY KEY (bid))
```

```
CREATE TABLE Prenotazioni (  vid INTEGER,  
                             bid INTEGER,  
                             giorno DATE,  
                             FOREIGN KEY (vid), REFERENCES Velisti,  
                             FOREIGN KEY (bid), REFERENCES Barche)
```



## *Sintassi per Eliminare/Alterare Schemi Relazionali*

```
ALTER TABLE <table>  
ADD(<column> <data type> [default <value>] [<column  
constraint>]);
```

```
ALTER TABLE <table> ADD (<table constraint>);
```

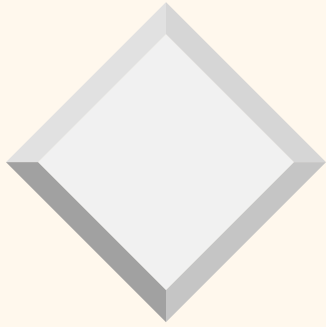
```
ALTER TABLE <table>  
MODIFY(<column> [<data type>] [DEFAULT <value>] [<column  
constraint>]);
```

```
DROP TABLE <table> [CASCADE CONSTRAINTS];
```

### ❖ Esempio:

```
ALTER TABLE Studenti  
ADD COLUMN nome-tutore CHAR(10);
```

```
DROP TABLE Studenti;
```

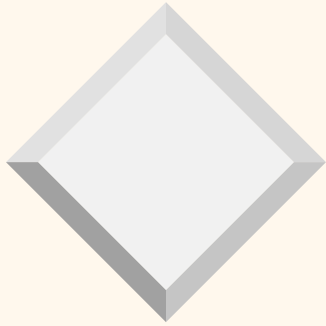


## *Sintassi di Inserzioni di tuple in uno Schema Relazionale*

```
INSERT INTO <table> [(<column i, . . . , column j>)]  
values (<value i, . . . , value j>);
```

```
INSERT INTO <table> [(<column i, . . . , column j>)]  
  <query>
```

- ❖ Per ogni colonna deve essere specificato un valore corrispondente del giusto tipo.
- ❖ Una inserzione non necessariamente deve seguire l'ordine degli attributi come specificato nella CREATE TABLE.
- ❖ Se una colonna viene omessa, allora per essa sarà utilizzato il valore NULL.
- ❖ Se non viene inserita nessuna lista di colonne, allora deve essere dato un valore per ogni colonna della relazione.



## *Esempio di Inserzioni*

### ❖ Esempio 1:

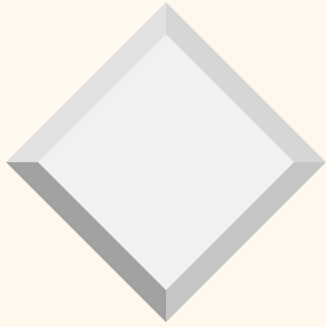
```
INSERT INTO Project(pno, pname, persons, budget, pstart)
VALUES(313, 'DBS', 4, 150000.42, '10-OCT-94');
```

Oppure

```
INSERT INTO Project
VALUES(313, 'DBS', 7411, null, 150000.42, '10-OCT-94', null);
```

### ❖ Esempio 2:

```
INSERT INTO Oldemp (eno, hdate)
SELECT empno, hiredate
FROM Emp
WHERE hiredate < '31-DEC-60';
```

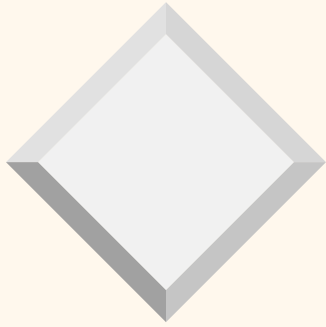


## *Altri Esempi di Inserzioni*

```
INSERT  
INTO Studenti (sid, nome, login, età, media)  
VALUES (53666, 'Jones', 'jones@cs', 18, 21)
```

```
INSERT  
INTO Studenti (sid, nome, login, età, media)  
VALUES (53688, 'Smith', 'smith@ee', 18, 24)
```

```
INSERT  
INTO Studenti (sid, nome, login, età, media)  
VALUES (53650, 'Smith', 'smith@smith', 19, 22)
```

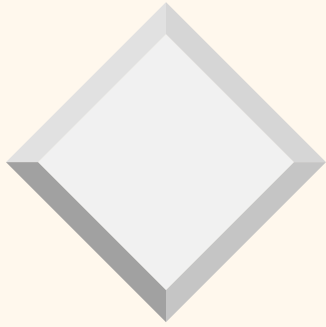


## *Sintassi per l'Aggiornamento di valori*

```
UPDATE <table> SET <column i> = <expression_i>, . . . ,  
    <column_j> = <expression_j>  
[WHERE <condition>];
```

```
UPDATE <table> SET(<column i, . . . , column j>) =  
    <query> [WHERE <condition>];
```

- ❖ Un'espressione può essere una costante, un'operazione aritmetica o un'operazione su stringhe, oppure una query SQL
- ❖ Ricordarsi che il nuovo valore che viene assegnato alla colonna <column\_i> deve appartenere al suo dominio
- ❖ Uno statement di UPDATE senza la clausola WHERE corrisponde a cambiare i valori delle colonne specificate per tutte le tuple della relazione



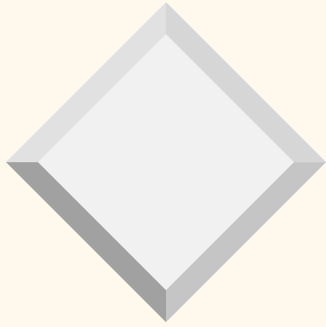
## *Esempio di Aggiornamento*

### ❖ Esempio 1:

```
UPDATE Emp SET  
job = 'MANAGER', deptno = 20, sal = sal + 1000  
WHERE ename = 'JONES';
```

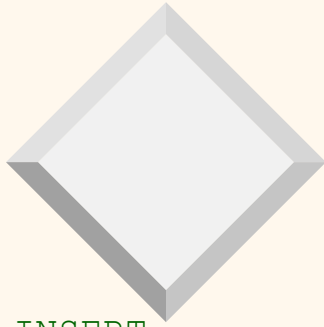
### ❖ Esempio 2:

```
UPDATE Emp SET  
sal = (SELECT min(sal) FROM Emp  
WHERE job = 'MANAGER')  
WHERE job = 'SALESMAN' and deptno = 20;
```



## *Istanze dopo le Inserzioni*

<b>sid</b>	<b>nome</b>	<b>login</b>	<b>età</b>	<b>media</b>
53666	Jones	jones@cs	18	21
53688	Smith	smith@ee	18	24
53650	Smith	smith@uk	19	22



## *Violazione dei VI di Chiave primaria*

```
INSERT  
INTO Studenti (sid, nome, login, età, media)  
VALUES (53688, 'Mike', 'mike@ee', 20, 22)
```

*Viola il vincolo di chiave primaria poiché c'è già una tupla con sid 53688*

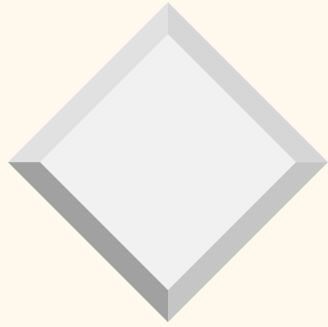
```
INSERT  
INTO Studenti (sid, nome, login, età, media)  
VALUES (null, 'Mike', 'mike@ee', 20, 22)
```

*Viola il vincolo per cui la chiave primaria non può contenere null*

```
UPDATE Studenti S  
SET S.sid = 53666  
WHERE S.sid = 53688
```

*Viola il vincolo di chiave primaria poiché c'è già una tupla con sid 53666*

*Esercizio: scrivere in SQL una inserzione che viola un vincolo di dominio per la relazione Studenti*

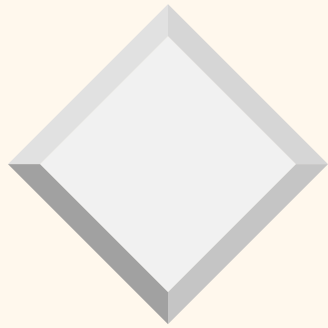


## *Violazione dei VI Referenziali*

```
INSERT  
INTO Iscrizioni (cid, voto, studid)  
VALUES ('Hindi101', 'B', 51111)
```

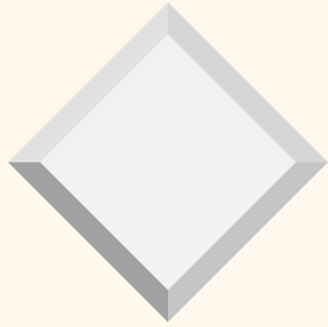
Inserimento illegale perché non ci sono tuple in *Studenti* con *sid* 51111

- ❖ Gli inserimenti di tuple in *Studenti* non violano l'integrità referenziale, e le cancellazioni di tuple in studenti **possono** causare violazioni
- ❖ Aggiornamenti su *Iscrizioni* e su *Studenti* che cambiano *studid* (o, rispettivamente, *sid*) **possono** potenzialmente violare l'integrità referenziale



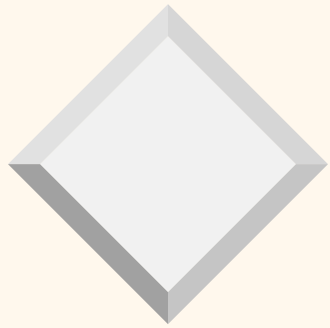
## *Violazione dei VI Referenziali*

- ❖ SQL fornisce diversi modi alternativi per gestire le violazioni di chiave esterna
- ❖ Domanda: *Cosa dovremmo fare se in "Iscrizioni" viene inserita una riga con un valore nella colonna "studid" che non appare in alcuna riga della tabella "Studenti"?* (in questo caso il comando INSERT viene ignorato)
- ❖ Inoltre: *Cosa dovremmo fare se una riga di "Studenti" viene cancellata?* Le opzioni sono:
  1. Cancellare tutte le righe di "Iscrizioni" che referenziano quella riga di "Studenti"
  2. Non permettere la cancellazione della riga "Studenti", se essa è referenziata da una riga di "Iscrizioni"
  3. Impostare la colonna *studid* al *sid* di qualche studente (esistente) di default, per ogni riga di "Iscrizioni" che referenzia la riga cancellata di "Studenti"
  4. Per ogni riga di "Iscrizioni" che la referenzia, impostare la colonna *studid* a *null*. Nell'esempio questa opzione è in conflitto col fatto che *studid* è parte della chiave primaria di *Iscrizioni* e quindi non può essere impostata a *null*.
- ❖ *Cosa dovremmo fare se il valore della chiave primaria di una riga di "Studenti" viene modificata?* Le opzioni sono come quelle precedenti.



## *Violazione dei VI Referenziali in SQL*

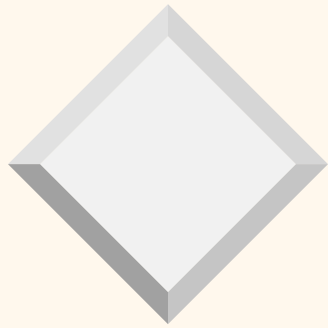
- ❖ SQL permette di scegliere una qualunque delle quattro opzioni per DELETE e UPDATE
- ❖ In SQL si può specificare che quando una riga di “Studenti” viene *cancellata* lo siano anche tutte le righe di “Iscrizione” che le referenziano,
- ❖ e che quando la colonna *sid* di una riga di “Studenti” viene modificata, questo aggiornamento viene rifiutato se una riga di “Iscrizioni” punta alla riga modificata di “Studenti”



## *Esempio*

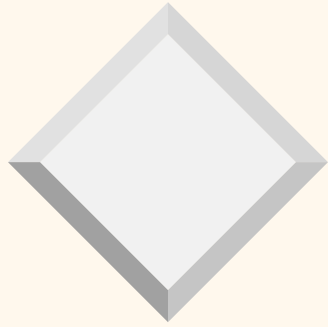
```
CREATE TABLE Iscrizioni (studid CHAR(20),
                          cid CHAR(20),
                          voto CHAR(10),
                          PRIMARY KEY (studid, cid),
                          FOREIGN KEY (studid) REFERENCES Studenti)
                          ON DELETE CASCADE
                          ON UPDATE NO ACTION
```

- ❖ Le opzioni vengono specificate come parte della dichiarazione della chiave esterna. L'operazione predefinita è `NO ACTION`, che significa che l'azione (`DELETE` o `UPDATE`) deve essere ignorata
- ❖ La parola `CASCADE` dice che se una riga di "Studenti" viene cancellata, in cascata tutte le righe di "Iscrizione" che la referenziano devono essere cancellata



## *Esercizio*

- ❖ Provate a sostituire la clausola `NO ACTION` con la clausola `CASCADE` anche per `UPDATE`. In questo caso, cosa succede se si modifica la colonna *sid* di una riga di “Studenti”?

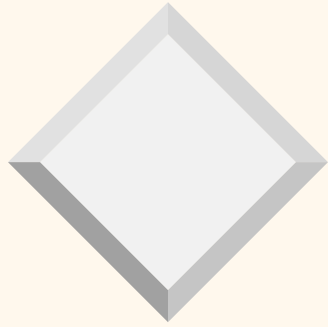


## *Gestione dei VI Referenziali in SQL*

- ❖ Se una riga di “Studenti” viene cancellata, è possibile spostare l’iscrizione ad uno studente di *default* usando ON DELETE SET DEFAULT. Lo studente di *default* è specificato nel campo *studid* di “Iscrizioni”

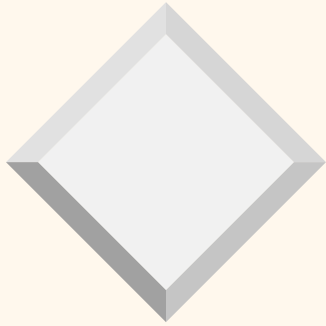
```
CREATE TABLE Iscrizioni (studid CHAR(20) DEFAULT '53666',  
                          cid CHAR(20),  
                          voto CHAR(10),  
                          PRIMARY KEY (studid, cid),  
                          FOREIGN KEY (studid) REFERENCES Studenti)  
                          ON DELETE CASCADE  
                          ON UPDATE NO ACTION
```

```
CREATE TABLE Iscrizioni (studid CHAR(20),  
                          cid CHAR(20),  
                          voto CHAR(10),  
                          PRIMARY KEY (studid, cid),  
                          FOREIGN KEY (studid) REFERENCES Studenti)  
                          ON DELETE SET NULL  
                          ON UPDATE NO ACTION
```



# *Transazioni e Vicoli in SQL*

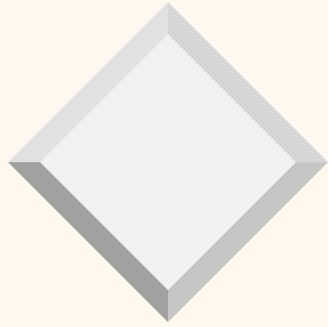
- ❖ Per default un vincolo viene verificato al termine di ogni istruzione SQL che potrebbe portare ad una violazione. Se la violazione si verifica allora il comando viene rifiutato. Questo approccio è troppo poco flessibile!!
- ❖ Esempio, consideriamo variante delle relazioni *Studenti* e *Corsi* (slide successiva): ogni studente deve partecipare ad un esame annuale, ed ogni corso deve avere un borsista che è un qualunque studente
- ❖ Quando viene inserita una tupla in *studenti* viene fatto un controllo per vedere se il corso annuale è nella relazione *Corsi*, e quando viene inserita in *Corsi* una tupla, viene fatto un controllo per vedere se il borsista è nella relazione *Studenti*. Come possiamo inserire il primo corso, o la prima tupla di *Studenti*?
- ❖ Soluzione: **differire** il controllo dei vincoli che normalmente sarebbe eseguito alla fine del comando `INSERT`



# *Esempio*

```
CREATE TABLE Studenti ( sid CHAR(20),  
                        nome CHAR(30),  
                        login CHAR(20),  
                        età INTEGER,  
                        annuale CHAR(10) NOT NULL,  
                        media REAL,  
                        PRIMARY KEY (sid),  
                        FOREIGN KEY (annuale) REFERENCES Corsi (cid))
```

```
CREATE TABLE Corsi ( cid CHAR(10),  
                     cnome CHAR(10),  
                     crediti INTEGER,  
                     borsista CHAR(20) NOT NULL,  
                     PRIMARY KEY (cid),  
                     FOREIGN KEY (borsista) REFERENCES Studenti (sid))
```

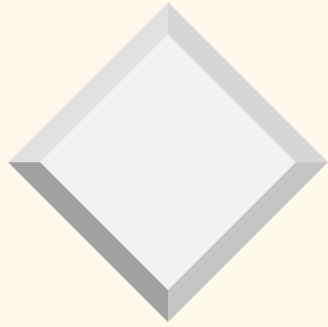


# *Transazioni e Vincoli in SQL*

- ❖ SQL permette di specificare che un vincolo è in modalità DEFERRED oppure IMMEDIATE

```
SET CONSTRAINT QualcheVincolo DEFERRED
```

- ❖ Un vincolo in modalità differita viene esaminato al termine della transazione.
- ❖ La transazione dell'esempio è la seguente:
  - \_ Dichiarazione dei vincoli di chiave esterna su *Corsi* e *Studenti* in modalità differita
  - \_ Inserimento di un corso con uno studente non esistente come borsista (stato BD inconsistente)
  - \_ Inserimento dello studente (ripristino consistenza)
  - \_ Termine transazione controllando che i vincoli siano soddisfatti



## *Sintassi per la Cancellazione di tuple*

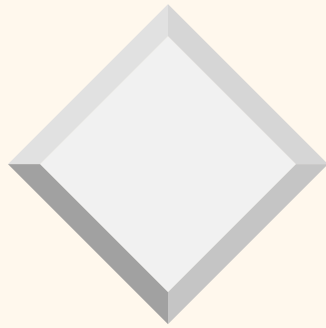
```
DELETE FROM <table> [WHERE <condition>];
```

- ❖ Se la clausola `WHERE` viene omessa, verranno cancellate dalla tabella tutte le tuple.
- ❖ Un comando alternativo per cancellare tutte le tuple è il seguente

```
TRUNCATE TABLE <table>;
```

- ❖ **Esempio:**

```
DELETE FROM Project WHERE pend < sysdate;  
TRUNCATE TABLE Project;
```



## *Istanze di esempio*

- ❖ Useremo nei nostri esempi queste istanze delle relazioni Velisti e Prenota
- ❖ Se la chiave per la relazione Prenota contenesse solo gli attributi *sid* e *bid*, in che modo la semantica sarebbe diversa?

*P1*

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

*V1*

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

*V2*

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0