

# BASI DI DATI I

- DML and Trigger



# DML & TRIGGER



# DML: DATA MANIPULATION LANGUAGE

- In SQL sono previsti tre comandi per modificare il database:
  - **INSERT**: popolamento tabelle
  - **DELETE**: rimozione di righe
  - **UPDATE**: cambiamento di valori nelle righe delle tabelle



# **DELETE**

- Il comando **DELETE** rimuove una o più tuple da una relazione.

**DELETE**

**FROM** <*NOME TABELLA*>  
**WHERE** <*CONDIZIONE*>;

- Cancella da <*NOME TABELLA*> tutte le righe che soddisfano <*CONDIZIONE*>
- Se la clausula **WHERE** è omessa allora è come se ci fosse la clausula

**WHERE TRUE**

e vengono cancellate tutte le righe della tabella

- **NB:** Il comando DELETE non rimuove la tabella
  - Il comando DROP <Nome Tabella> lo fa



# IL COMANDO DELETE - *ESEMPI*

```
DELETE FROM EMPLOYEE  
WHERE LNAME='Brown';
```

```
DELETE FROM EMPLOYEE  
WHERE DNO IN (SELECT DNUMBER  
                 FROM DEPARTMENT  
                 WHERE DNAME='Research');
```



# IL COMANDO INSERT

- Il comando **INSERT INTO** inserisce nuove righe in una relazione.

**INSERT INTO** <Nome Tabella> [ElencoAttributi]  
**VALUES** (*ListaValori*);

oppure

**INSERT INTO** <NomeTabella> [ElencoAttributi]  
**SELECT** *ElencoAttributi*  
**FROM** *Espressione*;



# IL COMANDO INSERT (2)

**INSERT INTO** <Nome Tabella> [ElencoAttributi]  
**VALUES** (*ListaValori*);

1. Se l'elenco degli attributi viene omesso, allora si intendono riempire i campi degli attributi in ordine di definizione della tabella
2. Se specifico l'elenco degli attributi, allora l'elenco deve includere tutti gli attributi TOTALI (NOT NULL). Decidiamo noi l'ordine.
3. Gli attributi PARZIALI non presenti, assumeranno il valore NULL o il valore di DEFAULT, se questo è specificato

---

**INSERT INTO** EMPLOYEE (FNAME, LNAME, SSN)  
**VALUES** ('Richard', 'Marini', '654765876');

---

**INSERT INTO** EMPLOYEE  
**VALUES** ('Richard', 'K', 'Marini', '654765876', '30-DEC-52', '98 Oak Forest, Katy, TX',  
'M', 37000, '987654321', 4);



# IL COMANDO INSERT - ESEMPIO

- Creare una tabella temporanea che ha nome, numero di impiegati e salari totali per ciascun dipartimento:

```
CREATE TABLE DEPTS_INFO ( DEPT_NAME VARCHAR(15),
                      NO_OF_EMPS INTEGER,
                      TOTAL_SAL INTEGER);
```

```
INSERT INTO DEPTS_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
SELECT DNAME, COUNT(*), SUM(SALARY)
FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER=DNO
GROUP BY DNAME;
```

- *Eventuali aggiornamenti successivi non influenzano la tabella originale. Per aggiornarla, è invece necessario definire una view.*



# IL COMANDO UPDATE

- Il comando **UPDATE** permette di modificare valori in una relazione:

**UPDATE** <NomeTabella> [alias]

**SET** <NomeAttributo1>= <Espressione> [, <NomeAttributoN>=<EspressioneN>]

[**WHERE** <condizione>]

1. Si lavora su NomeTabella
2. Si modificano le righe che soddisfano <condizione>
3. Per ogni riga si modificano gli attributi della clausola SET con il valore calcolato dall'espressione



# **IL COMANDO UPDATE - ESEMPIO**

**UPDATE PROJECT**

**SET PLOCATION='Bellaire', DNUM=5**

**WHERE PNUMBER=10;**

**UPDATE EMPLOYEE**

**SET SALARY=SALARY \* 1.1**

**WHERE DNO IN (SELECT DNUMBER**

**FROM DEPARTMENT**

**WHERE DNAME='Research');**



# ESEMPIO

*Magazzino (CodArticolo, Descrizione, Quantità)*

*Ordine (CodOrdine, Data, PIVA, DataInvio)*

*CompOrdine(CodOrdine\*, CodArticolo\*, Quantità, Prezzo)*

*Carrello(CodCarrello, Data, PIVA, Completo)*

*CompCarrello(CodCarrello\*, CodArticolo\*, Quantità, Prezzo)*

- Operazioni che devono essere fatte

Quando il carrello è completo

- a) Trasformo il carrello in ordine

1. Creare un nuovo ordine con la struttura del carrello
2. Rimuovere il carrello

- b) Aggiornare le scorte nel magazzino



# ESEMPIO

*Magazzino (CodArticolo, Descrizione, Quantità)*

*Ordine (CodOrdine, Data, PIVA, DataInvio)*

*CompOrdine(CodOrdine\*, CodArticolo\*, Quantità, Prezzo)*

*Carrello(CodCarrello, Data, PIVA, Completo)*

*CompCarrello(CodCarrello\*, CodArticolo\*, Quantità, Prezzo)*

- Operazioni che devono essere fatte

Quando il carrello è completo

- a) Trasformo il carrello in ordine

1. Creare un nuovo ordine con la struttura del carrello
2. Rimuovere il carrello

- b) Aggiornare le scorte nel magazzino

**Che operazioni di Manipolazione  
devo fare?**

**In che ordine devo fare le operazioni  
di manipolazione?**



# ESEMPIO (2)

Quando il carrello è completo

a) Trasformo il carrello in ordine

1. Creare un nuovo ordine con la struttura del carrello  
(INSERT in ORDINE e COMPOORDINE)
2. Rimuovere il carrello  
(DELETE in CARRELLO e COMPCARRELLO)

b) Aggiornare le scorte nel magazzino

(UPDATE su MAGAZZINO)



# ESEMPIO (3)

- Creare un nuovo ordine con la struttura del carrello (Lavoriamo con il carrello con il codice 'XYZ')

1) **INSERT INTO** Ordine (CodO, Data, PIVA, DataInvio)

```
(SELECT C.CodO, Data, PIVA, NULL  
FROM Carrello C  
WHERE C.CodO = 'XYZ')
```

Oppure

**INSERT INTO** Ordine (CodO, Data, PIVA)

```
(SELECT C.CodO, Data, PIVA  
FROM Carrello C  
WHERE C.CodO = 'XYZ')
```



# ESEMPIO (4)

- Creare un nuovo ordine con la struttura del carrello (Lavoriamo con il carrello con il codice 'XYZ')

2) **INSERT INTO** CompOrdine

```
(SELECT *  
FROM CompCarrello C  
WHERE C.CodC = 'XYZ')
```



# ESEMPIO (5)

- Creare un nuovo ordine con la struttura del carrello (Lavoriamo con il codice 'XYZ')

## 2) **DELETE**

**FROM** CompCarrello C  
**WHERE** C.CodC = 'XYZ'

## **DELETE**

**FROM** Carrello C  
**WHERE** C.CodC = 'XYZ'



# ESEMPIO (6)

- Aggiornare le scorte nel magazzino

- Per ogni articolo nell'ordine 'XYZ' devo rimuoverne la quantità venduta nel magazzino

- b) **UPDATE** Magazzino M

```
SET M.Quantità = M.Quantità - (SELECT H.Quantità
```

```
FROM CompOrdine H
```

```
WHERE H.CodO = 'XYZ' AND H.CodA = M.CodA)
```

```
WHERE M.CodA IN (SELECT C.CodA
```

```
FROM CompOrdine C
```

```
WHERE C.CodO = 'XYZ')
```



# TRANSACTION

- Sequenza di operazioni effettuate tra un

**BEGIN TRANSACTION**

***INSERT...***

***UPDATE...***

***DELETE...***

***procedure...***

**COMMIT**

**END TRANSACTION**

Parlammo già delle transaction...



# TRIGGER

- Un trigger è una di procedura/funzione di manutenzione della base di dati
- A differenza delle procedure standard, che sono chiamate in maniera esplicita, un trigger non è invocato.
- La loro attivazione è scatenata da una serie di eventi di diversa natura che avvengono nella base di dati
- Questi eventi scatenano una serie di operazioni che servono per mantenere la base di dati
  - Gli eventi che ci interessano, in particolare sono quelli che vanno ad alterare la base di dati
    - INSERT
    - UPDATE
    - DELETE



# CREAZIONE TRIGGER

- Quando voglio creare un trigger, chiaramente devo andare a definire i parametri del trigger:
  1. Quale evento scatena la reazione?
    - a) INSERT
    - b) DELETE
    - c) UPDATE
  2. Quando voglio scatenare la reazione? Prima o dopo l'evento?
    - a. Dopo: C'è stato, ad esempio, un UPDATE e voglio propagare l'evento
    - b. Prima: Ci sarà, ad esempio, una DELETE e voglio eseguire dei comandi prima della cancellazione
  3. Condizioni di Filtraggio che mi permettono di definire precisamente quale è l'evento che scatena la reazione
  4. Operazione globale/per ciascuna riga
  5. Reazione



# SIGNATURE TRIGGER

**CREATE TRIGGER** <nomeTrigger>

[ **AFTER/BEFORE/INSTEAD OF**] <operazione>

[**FOR EACH ROW**]

**WHERE** <condizione>

**BEGIN**

<CORPO DELLA PROCEDURA>

**END**

*INSERT ON <tabella>  
DELETE ON <tabella>  
UPDATE OF <ATTRIBUTO> ON <tabella>*

# TORNANDO ALL'ESEMPIO DI PRIMA

**CREATE TRIGGER** creaOrdine

**AFTER UPDATE ON** CARRELLO **OF** Completo

**FOR EACH ROW**

**WHEN OLD.Completo = 'Incompleto' AND NEW.Completo = 'Completo'**

**BEGIN**

**INSERT INTO** Ordine (CodO, Data, PIVA)

**(SELECT** C.CodO, Data, PIVA

**FROM** Carrello C

**WHERE** C.CodC = 'NEW.CodC');



```
INSERT INTO CompOrdine  
(SELECT *  
FROM CompCarrello C  
WHERE C.CodC = NEW.CodC);  
DELETE  
FROM CompCarrello C  
WHERE C.CodC = NEW.CodC;  
DELETE  
FROM Carrello C  
WHERE C.CodC = NEW.CodC;  
UPDATE Magazzino M  
SET M.Quantità = M.Quantità - (SELECT H.Quantità  
                 FROM CompOrdine H  
                 WHERE H.CodO = 'XYZ' AND H.CodA = M.CodA)  
WHERE M.CodA IN (SELECT C.CodA  
                 FROM CompOrdine C  
                 WHERE C.CodO = NEW.CodC);  
END;
```



# ESEMPIO DI APPLICAZIONE DI UN TRIGGER

Tabella (**ID**, Numero)

```
CREATE TRIGGER incremento  
AFTER UPDATE ON Tabella OF Numero  
FOR EACH ROW  
WHEN NEW.Numero > OLD.Numero  
BEGIN  
    UPDATE Tabella T  
    SET T.Numero = T.Numero+1  
    WHERE NEW.ID = T.ID  
END
```

```
UPDATE Tabella  
SET Numero = Numero+1  
WHERE Tabella.ID = '100'
```



# ESEMPIO

Tabella (**ID**, Numero)

```
CREATE TRIGGER incremento  
AFTER UPDATE ON Tabella OF Numero  
FOR EACH ROW  
WHEN NEW.Numero > OLD.Numero  
BEGIN  
    UPDATE Tabella T  
    SET T.Numero = T.Numero+1  
    WHERE NEW.ID = T.ID  
END
```

```
UPDATE Tabella  
SET Numero = Numero+1  
WHERE Tabella.ID = '100'
```

Quale sarà il valore di Numero,  
dopo l'esecuzione della UPDATE?



# ESEMPIO

Tabella (**ID**, Numero)

```
CREATE TRIGGER incremento  
AFTER UPDATE ON Tabella OF Numero  
FOR EACH ROW  
WHEN NEW.Numero > OLD.Numero  
BEGIN  
    UPDATE Tabella T  
    SET T.Numero = T.Numero+1  
    WHERE NEW.ID = T.ID  
END
```

```
UPDATE Tabella  
SET Numero = Numero+1  
WHERE Tabella.ID = '100'
```

Quale sarà il valore di Numero,  
dopo l'esecuzione della UPDATE?

In tal caso avremo un loop infinito.



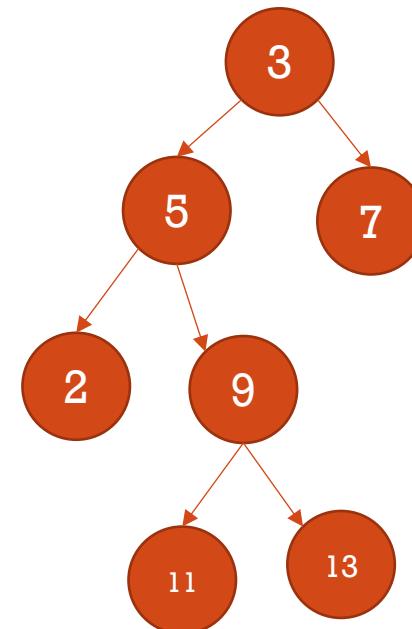
# ALTRÒ ESEMPIO

Tree (CodT, Radice)

Nodo (CodT, CodN, Etichetta)

Arco (CodT, NodoS, NodoD)

Quando incremento il valore di un nodo  
devo incrementare tutti i discendenti della  
stessa quantità.



Ipotizziamo inoltre che vogliamo propagare solo gli incrementi positivi.



**CREATE TRIGGER** propagazione

**AFTER UPDATE** Nodo **ON** Etichetta

**FOR EACH ROW**

**WHEN NEW.Etichetta – OLD.Etichetta > 0**

**BEGIN**

**UPDATE** Nodo N

**SET** N.Etichetta = N.Etichetta + (**NEW.Etichetta – OLD.Etichetta**)

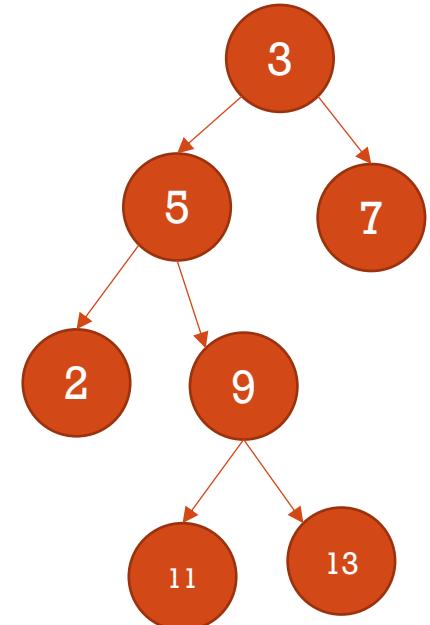
**WHERE** N.CodN **IN** (**SELECT** A.NodoS

**FROM** Arco A

**WHERE** A.CodT = **NEW.CodT**

**AND** A.NodoD = **NEW.CodN**)

**END**



**CREATE TRIGGER** propagazione

**AFTER UPDATE** Nodo **ON** Etichetta

**FOR EACH ROW**

**WHEN NEW.Etichetta – OLD.Etichetta > 0**

**BEGIN**

**UPDATE** Nodo N

**SET** N.Etichetta = N.Etichetta + (**NEW.Etichetta – OLD.Etichetta**)

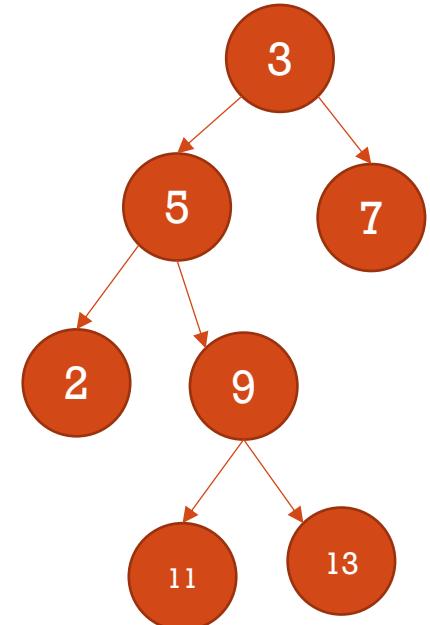
**WHERE** N.CodN **IN** (**SELECT** A.NodoS

**FROM** Arco A

**WHERE** A.CodT = **NEW.CodT**

**AND** A.NodoD = **NEW.CodN**)

**END**



Update sui figli  
di **NEW.CodN**



# BLOCCARE UN TRIGGER

1. Elimino il trigger

**DROP TRIGGER** <nometrigger>

2. Disabilo il trigger (un trigger si considera attivato alla sua azione)

**ALTER TRIGGER** <nometrigger> **DISABLE** (per disattivarlo)

**ALTER TRIGGER** <nometrigger> **ENABLE** (per riattivarlo)



# CLAUSOLA BEFORE

- La clausola BEFORE viene utilizzata quando faccio ad esempio degli inserimenti e voglio controllare i dati che vengono inseriti.
- ESEMPIO – *Chiavi sintetiche*

**CREATE SEQUENCE** *set\_id*

**START WITH** 1000

**INCREMENT BY** 1

**MAX VALUE** 10000



# CLAUSOLA BEFORE 2

Esempio (AutoIncID, Nome, Cognome)

```
INSERT INTO Esempio  
values(NULL, 'Ciro', 'Esposito')
```



# **CLAUSOLA BEFORE 3**

Esempio (AutoIncID, Nome, Cognome)

```
INSERT INTO Esempio  
values(NULL, 'Ciro', 'Esposito')
```

```
CREATE TRIGGER setKey  
BEFORE INSERT on Esempio  
FOR EACH ROW  
BEGIN  
    NEW.AutoIncID := set_id.NEXTVAL  
END
```



# CLAUSOLA BEFORE 3

Esempio (AutoIncID, Nome, Cognome)

```
INSERT INTO Esempio  
values(NULL, 'Ciro', 'Esposito')
```

```
CREATE TRIGGER setKey  
BEFORE INSERT on Esempio  
FOR EACH ROW  
BEGIN  
    NEW.AutoIncID := set_id.NEXTVAL  
END
```

```
CREATE TRIGGER setKey  
BEFORE INSERT on Esempio  
FOR EACH ROW  
BEGIN  
    NEW.AutoIncID := set_id.NEXTVAL;  
    NEW.Nome := UPPER(NEW.Nome);  
    NEW.Cognome := LOWER(NEW.Cognome);  
END
```



# CLAUSOLA INSTEAD OF

- La clausola **INSTEAD OF** non viene utilizzata sulle tabelle, ma è usata esclusivamente per operazioni di manipolazione sulle viste.
- Le Viste sono tabelle virtuali (non memorizzate)
  - Vanno bene per operazioni di lettura (**SELECT**)
  - Sono problematiche se devo fare variazioni di dati (**INSERT, UPDATE e DELETE**)



# ESEMPIO

- Studenti (Matricola, CF, Nome, Cognome, DataN, Residenza)

CF, Nome e Cognome sono attributi totali. DataN e Residenza sono parziali.

**CREATE VIEW** Studenti2 **AS**

**SELECT** Matricola, CF, Nome, Cognome

**FROM** Studenti

1- La vista contiene il campo chiave

2- Contiene gli attributi totali

3- Basata su una sola tabella

**DELETE FROM** Studenti2

**WHERE** Matricola = '340'

**INSERT INTO** Studenti2

**VALUES**('260', 'GLLCRR88M07H703Z', 'Ciro', 'Gallo', )

**UPDATE** Studenti2

**SET** Nome='Gianluca'

**WHERE** CF='GNCFRR91M09H333Q'



# ESEMPIO

- Studenti (Matricola, CF, Nome, Cognome, DataN, Residenza)

CF, Nome e Cognome sono attributi totali. DataN e Residenza sono parziali.

**CREATE VIEW** Studenti2 **AS**

**SELECT** Matricola, CF, Nome, Cognome  
**FROM** Studenti

- 1- La vista contiene il campo chiave
- 2- Contiene gli attributi totali
- 3- Basata su una sola tabella

**DELETE FROM** Studenti2

**WHERE** Matricola = '340'

**INSERT INTO** Studenti2

**VALUES**('260', 'GLLCRR88M07H703Z', 'Ciro', 'Gallo', )

**UPDATE** Studenti2

**SET** Nome='Gianluca'

**WHERE** CF='GNCFRR91M09H333Q'

Queste tre operazioni si ripercuotono  
sulla tabella originale



# ESEMPIO

Studente (Matricola, Nome, Cognome)  
Anagrafe (Matricola, CF, Residenza, DataN)

Relazione 1:1 tra Studente e Anagrafe

```
CREATE VIEW Info AS  
SELECT *  
FROM Studente INNER JOIN Anagrafe
```



```
CREATE TRIGGER Info_Stud  
INSTEAD OF INSERT ON Info OR DELETE ON Info OR UPDATE OF ATTRIBUTO ON...  
FOR EACH ROW  
BEGIN  
    IF INSERTING (TG_OP = INSERT)  
        INSERT INTO Studente (Matricola, Nome, Cognome)  
        VALUES(NEW.Matricola, NEW.Nome, NEW.Cognome)  
        INSERT INTO Anagrafe (Matricola, CF, Residenza, DataN)  
        VALUES(NEW.Matricola, NEW.CF, NEW.Residenza, NEW.DataN)  
    END IF  
    IF DELETING (TG_OP = 'DELETE')  
        DELETE FROM Anagrafe WHERE Matricola = OLD.Matricola  
        DELETE FROM Studente WHERE Matricola = OLD.Matricola  
    END IF  
    IF UPDATING (TG_OP = UPDATE)  
    ....  
    END IF  
END
```





# FINE

Per eventuali domande: (in ordine di preferenza personale)

- Ora.
- Chat di Teams
- Mail: [silvio.barra@unina.it](mailto:silvio.barra@unina.it)

