



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

# Il Linguaggio SQL DML/DQL

## Basi di Dati

*Corso di Laurea in Informatica per il Management*

*Alma Mater Studiorum - Università di Bologna*

---

**Prof. Marco Di Felice**

Dipartimento di Informatica – Scienza e Ingegneria

marco.difelice3@unibo.it

# Il Linguaggio SQL

---

- **LINGUAGGI** supportati dai **RDBMS**

## 3. SQL (*Structured Query Language*)

Diverse **versioni** del linguaggio:

- SQL-86 → Costrutti base
- SQL-89 → (**SQL1**) Integrità referenziale
- SQL-92 (**SQL2**) → SQL Interattivo, sistema tipi
- SQL:1999 (**SQL3**) → Modello ad oggetti
- SQL:2003 (**SQL3**) → Nuove parti: SQL/JRT, SQL/XML
- SQL:2006 (**SQL3**) → Estensione di SQL/XML
- SQL:2008 (**SQL3**) → Lievi aggiunte
- ....

<http://troels.arvin.dk/db/rdbms/>

# Il Linguaggio SQL

---

Due **componenti** principali:

- **DDL** (*Data Definition Language*)  
Contiene i costrutti necessari per la creazione e modifica dello **schema** della base di dati.



- **DML/DQL** (*Data Manipulation/Query Language*)  
Contiene i costrutti per le interrogazioni e di inserimento, eliminazione e modifica di **dati**.

# SQL: DML/DQL

---

**Es.** Estrarre il codice dello strutturato che riceve lo stipendio più alto.

## STRUTTURATI

<u>Codice</u>	Nome	Cognome	Tipo	Dipartimento	Stipendio
123	Marco	Marchi	Associato	Chimica	20000
124	Michele	Micheli	Associato	Fisica	20000
125	Lucia	Di Lucia	Ordinario	Fisica	30000
126	Dario	Rossi	Ordinario	Informatica	35000
127	Mario	Rossi	Ricercatore	Informatica	15000
129	Michele	Bianchi	Associato	Fisica	20000

# SQL: DML/DQL

---

**Es.** Estrarre il codice dello strutturato che riceve lo stipendio più alto.

```
SELECT CODICE, MAX(STIPENDIO)  
FROM STRUTTURATI
```

**ERRORE!**

- SELECT MAX(STIPENDIO) restituisce solo un valore!!
- SELECT CODICE restituisce più di un valore!!

# SQL: DML/DQL

---

Es. Estrarre il codice dello strutturato che riceve lo stipendio più alto.

```
SELECT CODICE  
FROM STRUTTURATI  
WHERE STIPENDIO= MAX(STIPENDIO)
```

ERRORE!

- L'operatore aggregato MAX si applica sulla SELECT e viene valutato dopo la WHERE ...

# SQL: DML/DQL

## QUERY ANNIDATE

Nella clausola where, oltre ad espressioni semplici, possono comparire espressioni complesse in cui il **valore di un attributo viene confrontato con il risultato di un'altra query**.

```
SELECT  
FROM  
WHERE (Attributo expr SELECT  
FROM  
WHERE)
```

**NOTA:** Si sta confrontando un singolo valore con il risultato di una query (quindi potenzialmente una tabella).

# SQL: DML/DQL

---

## QUERY ANNIDATE

E' possibile annidare due query SQL:

- ~~○ Nella clausola SELECT → codice poco leggibile~~
- ~~○ Nella clausola FROM → non presentato nelle slide~~
- Nella clausola WHERE → presentato a seguire
- ~~○ Nella clausola HAVING → non supportato da tutti i RDBMS~~



# SQL: DML/DQL

---

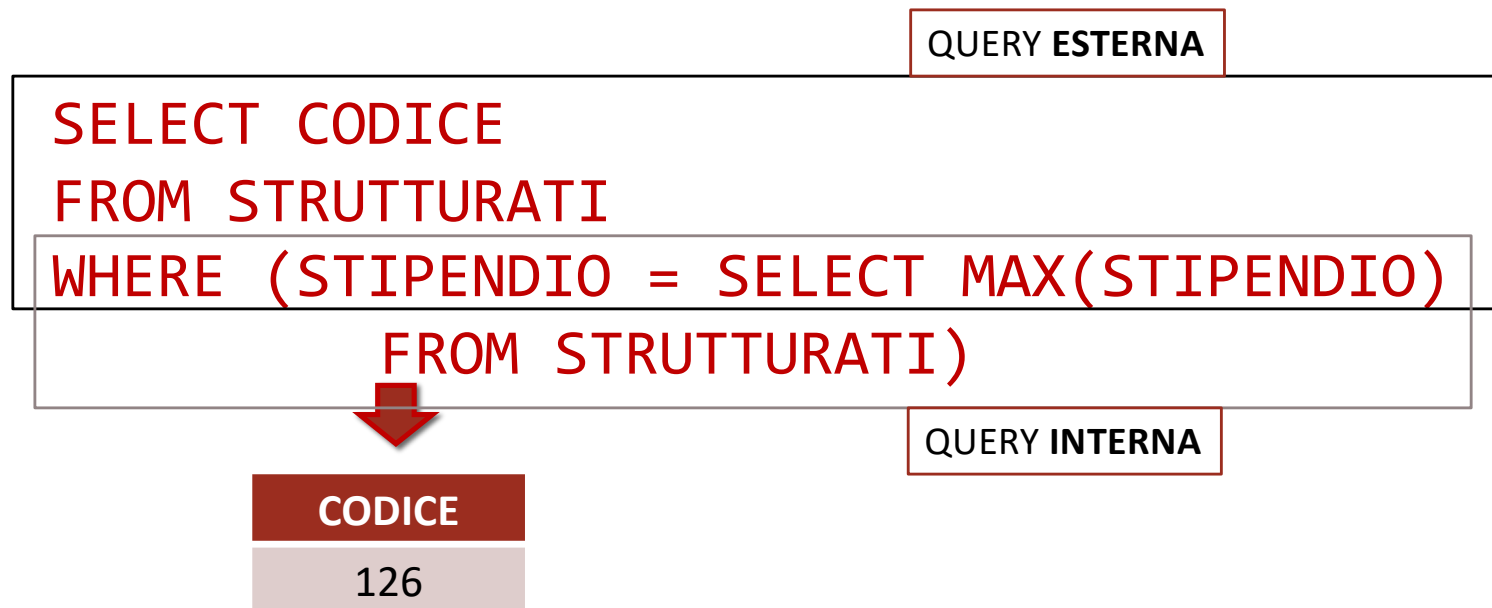
**Es.** Estrarre il codice dello strutturato che riceve lo stipendio più alto.

## STRUTTURATI

<u>Codice</u>	Nome	Cognome	Tipo	Dipartimento	Stipendio
123	Marco	Marchi	Associato	Chimica	20000
124	Michele	Micheli	Associato	Fisica	20000
125	Lucia	Di Lucia	Ordinario	Fisica	30000
126	Dario	Rossi	Ordinario	Informatica	35000
127	Mario	Rossi	Ricercatore	Informatica	15000
129	Michele	Bianchi	Associato	Fisica	20000

# SQL: DML/DQL

Es. Estrarre il codice dello strutturato che riceve lo stipendio più alto.



# SQL: DML/DQL

---

Nel caso precedente, la query interna restituisce solo un valore ...  
Cosa accade se la query interna **restituisce più di un valore?**

Gli operatori di confronto  $<,=,>$  non si possono utilizzare in questo caso !

**Es.** Estrarre nome e cognome degli strutturati del dipartimento di Informatica che guadagnano quanto un loro collega di Fisica.

# SQL: DML/DQL

---

Nel caso precedente, la query interna restituisce solo un valore ... Cosa accade se la query interna **restituisce più di un valore?**

```
SELECT NOME, COGNOME  
FROM STRUTTURATI  
WHERE (DIPARTIMENTO="INFORMATICA") AND  
      (STIPENDIO = (SELECT STIPENDIO  
                    FROM STRUTTURATI  
                    WHERE (DIPARTIMENTO="FISICA")))
```

**NON FUNZIONA!**

# SQL: DML/DQL

---

Esistono **operatori speciali di confronto** nel caso di interrogazioni annidate:

- **any** → la riga soddisfa la condizione se è vero il confronto tra il valore dell'attributo ed **ALMENO UNO** dei valori ritornati dalla query annidata.
- **all** → la riga soddisfa la condizione se è vero il confronto tra il valore dell'attributo e **TUTTI** i valori ritornati dalla query annidata.

# SQL: DML/DQL

---

Il costrutto **in** restituisce true se un certo valore è contenuto nel risultato di una interrogazione nidificata, **false** altrimenti.

```
SELECT ListaAttributi
FROM TabellaEsterna
WHERE Valore/i IN SELECT ListaAttributi2
                     FROM TabellaInterna
                     WHERE Condizione
```

Nel caso di più di 1 valore  
si utilizza il costruttore di  
tuple.

# SQL: DML/DQL

---

Il costrutto **exists** restituisce true se l'interrogazione nidificata restituisce un risultato non vuoto ( $\geq 1$  elemento trovato).

```
SELECT ListaAttributi
FROM TabellaEsterna
WHERE EXISTS SELECT ListaAttributi2
              FROM TabellaInterna
              WHERE Condizione
```

Controlla se il numero di  
righe della query  
interna  $> 0$

# SQL: DML/DQL

---

Es. Estrarre nome e cognome degli strutturati del dipartimento di Informatica che guadagnano **quanto un loro collega** di Fisica.

```
SELECT NOME, COGNOME  
FROM STRUTTURATI  
WHERE (DIPARTIMENTO="INFORMATICA") AND  
      (STIPENDIO = ANY (SELECT STIPENDIO  
                          FROM STRUTTURATI  
                          WHERE (DIPARTIMENTO="FISICA")))
```



# SQL: DML/DQL

---

Es. Estrarre nome e cognome degli strutturati del dipartimento di Informatica che guadagnano **più di tutti** i loro colleghi di Fisica.

```
SELECT NOME, COGNOME
FROM STRUTTURATI
WHERE (DIPARTIMENTO="INFORMATICA") AND
      (STIPENDIO > ALL (SELECT STIPENDIO
                          FROM STRUTTURATI
                          WHERE (DIPARTIMENTO="FISICA")))
```

# SQL: DML/DQL

---

Le interrogazioni **nidificate** possono essere:



- **Semplici** → non c'è **passaggio di binding** tra un contesto all'altro. Le interrogazioni vengono valutate dalla più interna alla più esterna.
- **Complesse** → c'è passaggio di binding attraverso **variabili condivise** tra le varie interrogazioni. In questo caso, le interrogazioni più interne vengono valutate su ogni tupla.

# SQL: DML/DQL

**STEP 1:** Viene valutata la query più interna...

```
SELECT NOME, COGNOME  
FROM STRUTTURATI  
WHERE (DIPARTIMENTO="INFORMATICA") AND  
      (STIPENDIO > ALL (SELECT STIPENDIO  
                        FROM STRUTTURATI  
                        WHERE (DIPARTIMENTO="FISICA"))))
```




Stipendio
20000
30000
20000

# SQL: DML/DQL

**STEP 2:** Viene confrontata ciascuna riga della tabella più esterna con il risultato della query interna ...

```
SELECT NOME, COGNOME  
FROM STRUTTURATI  
WHERE (DIPARTIMENTO="INFORMATICA") AND  
      (STIPENDIO > ALL (SELECT STIPENDIO
```

Codice	Nome	Cognome	Tipo	Dipartimento	Stipendio
123	Marco	Marchi	Associato	Chimica	20000
124	Michele	Micheli	Associato	Fisica	20000
125	Lucia	Di Lucia	Ordinario	Fisica	30000
<b>126</b>	<b>Dario</b>	<b>Rossi</b>	<b>Ordinario</b>	<b>Informatica</b>	<b>35000</b>
127	Mario	Rossi	Ricercatore	Informatica	15000
129	Michele	Bianchi	Associato	Fisica	20000



Stipendio
20000
30000
20000

**BASI DI DATI**

PROF. MARCO DI FELICE – CORSO DI LAUREA IN INFORMATICA PER IL MANAGEMENT

# SQL: DML/DQL

**STEP 2:** Viene confrontata ciascuna riga della tabella più esterna con il risultato della query interna ...

```
SELECT NOME, COGNOME  
FROM STRUTTURATI  
WHERE (DIPARTIMENTO="INFORMATICA") AND  
      (STIPENDIO > ALL (SELECT STIPENDIO  
                        FROM STRUTTURATI  
                        WHERE (DIPARTIMENTO="FISICA"))))
```

Nome	Cognome
Dario	Rossi

# SQL: DML/DQL

---

Le interrogazioni **nidificate** possono essere:

- **Semplici** → non c'è **passaggio di binding** tra un contesto all'altro. Le interrogazioni vengono valutate dalla più interna alla più esterna.



- **Complesse** → c'è passaggio di binding attraverso **variabili condivise** tra le varie interrogazioni. In questo caso, le interrogazioni più interne vengono valutate su ogni tupla.

# SQL: DML/DQL

---

**Es.** Estrarre nome/cognome degli impiegati che hanno omonimi (stesso nome/cognome di altri impiegati).

**IMPIEGATI**

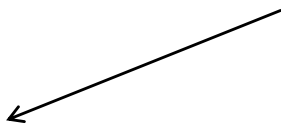
Codice	Nome	Cognome	Ufficio
1	Marco	Marchi	A
2	Dario	Rossi	B
3	Lucia	Di Lucia	C
4	Dario	Rossi	C
5	Mario	Rossi	A
6	Marco	Marchi	B

# SQL: DML/DQL

---

**Es.** Estrarre nome/cognome degli impiegati che hanno omonimi (stesso nome/cognome di altri impiegati).

```
A=IMPIEGATI
B=IMPIEGATI
for i=0 ... |A|
  for j=0 ... |B|
    if ((A[i].Nome==B[j].Nome)
        AND (A[i].Cognome== B[j].Cognome))
      Include IMPIEGATI[i] into the result
```



E' necessario valutare due volte la tabella IMPIEGATI!!



# SQL: DML/DQL

---

Es. Estrarre nome/cognome degli impiegati che hanno omonimi (stesso nome/cognome di altri impiegati).

```
SELECT NOME, COGNOME
FROM IMPIEGATI AS I
WHERE (I.NOME,I.COGNOME) = ANY
      (SELECT NOME, COGNOME
       FROM IMPIEGATI AS I2
       WHERE (I.NOME=I2.NOME)
            AND (I.COGNOME=I2.COGNOME)
            AND (I.CODICE <> I2.CODICE))
```

# SQL: DML/DQL

**Funzionamento:** La query più interna viene **valutata su ciascuna tupla** della query più esterna...

**I**

Codice	Nome	Cognome	Ufficio
1	Marco	Marchi	A
2	Dario	Rossi	B
3	Lucia	Di Lucia	C
4	Dario	Rossi	C
5	Mario	Rossi	A
6	Marco	Marchi	B

**I2**

Codice	Nome	Cognome	Ufficio
1	Marco	Marchi	A
2	Dario	Rossi	B
3	Lucia	Di Lucia	C
4	Dario	Rossi	C
5	Mario	Rossi	A
6	Marco	Marchi	B

# SQL: DML/DQL

**Funzionamento:** La query più interna viene **valutata su ciascuna tupla** della query più esterna...

**I**

Codice	Nome	Cognome	Ufficio
<b>1</b>	<b>Marco</b>	<b>Marchi</b>	A
2	Dario	Rossi	B
3	Lucia	Di Lucia	C
4	Dario	Rossi	C
5	Mario	Rossi	A
6	Marco	Marchi	B

**I2**

Codice	Nome	Cognome	Ufficio
1	Marco	Marchi	A
<b>2</b>	<b>Dario</b>	<b>Rossi</b>	B
3	Lucia	Di Lucia	C
4	Dario	Rossi	C
5	Mario	Rossi	A
6	Marco	Marchi	B

# SQL: DML/DQL

---

In alcuni casi, le **query annidate** possono essere riscritte usando costrutti di join tra tabelle o self-join (prodotto cartesiano + selezione).

```
SELECT NOME, COGNOME
FROM IMPIEGATI AS I, IMPIEGATI AS I2
WHERE (I.NOME=I2.NOME)
      AND (I.COGNOME=I2.COGNOME)
      AND (I.CODICE <> I2.CODICE))
```

# SQL: DML/DQL

---

In maniera equivalente, usando l'operatore `in` ed i **costruttori di tupla**:

```
SELECT CODICE
FROM IMPIEGATI AS I
WHERE (I.NOME,I.COGNOME) NOT IN
      (SELECT NOME, COGNOME
       FROM IMPIEGATI AS I2
       WHERE (I.NOME=I2.NOME)
            AND (I.COGNOME=I2.COGNOME)
            AND (I.CODICE <> I2.CODICE))
```

# SQL: DML/DQL

---

Es. Estrarre nome/cognome degli impiegati che **NON** hanno omonimi (stesso nome/cognome di altri impiegati).

IMPIEGATI

Codice	Nome	Cognome	Ufficio
1	Marco	Marchi	A
2	Dario	Rossi	B
3	Lucia	Di Lucia	C
4	Dario	Rossi	C
5	Mario	Rossi	A
6	Marco	Marchi	B

# SQL: DML/DQL

---

```
SELECT NOME, COGNOME
FROM IMPIEGATI AS I
WHERE NOT EXISTS (SELECT *
                  FROM IMPIEGATI AS I2
                  WHERE (I.NOME=I2.NOME) AND
                      (I.COGNOME=I2.COGNOME)
                  AND (I.CODICE <> I2.CODICE))
```

D. E' possibile scrivere una query equivalente *senza usare interrogazioni annidate?*

# SQL: Viste

---

Le **viste** rappresentano “tabelle virtuali” ottenute da dati contenute in altre tabelle del database. Ogni vista ha associato un **nome** ed una **lista di attributi**, e si ottiene dal **risultato di una select**.

```
create view NomeView [ListaAttributi]
as SELECT SQL
[with [local | cascade] check option]
```



# SQL: Viste

---

## PROPRIETA' delle VISTE

I **dati delle viste NON** sono fisicamente memorizzati a parte, in quanto dipendono da altre tabelle (ad eccezione delle **viste materializzate**).

- Le viste esistono **a livello di schema** ma **non hanno istanze proprie**.
- Le operazioni di **aggiornamento di viste** potrebbero non essere consentite in alcuni DBMS.

# SQL: Viste

---

## Q. A che serve definire una vista?



- Implementare meccanismi di **indipendenza** tra il livello logico ed il livello esterno.
- Scrivere **interrogazioni complesse**, semplificandone la sintassi.
- Garantire la **retro-compatibilità** con precedenti versioni dello schema del DB, in caso di ristrutturazione dello stesso.

# SQL: Viste

Data la tabella PROFESSORI, definire una **vista** “STUDENTI” in cui si mostrano solo le informazioni anagrafiche (nome, cognome, codice, data nascita) dei docenti.

PROFESSORI					
Codice	Nome	Cognome	Nascita	Livello	Stipendio
1	Marco	Marchi	10/04/1980	A1	20000
3	Michele	Micheli	12/05/1967	R	20000
5	Lucia	Di Lucia	12/05/1978	R2	30000
7	Dario	Rossi	24/01/1965	O2	32000

# SQL: Viste

---


Data la tabella PROFESSORI, definire una vista “STUDENTI” in cui si mostrano solo le informazioni anagrafiche (nome, cognome, codice, data nascita) dei docenti.

```
CREATE VIEW STUDENTI(CODICE,NOME,COGNOME,  
DATANASCITA) AS  
SELECT CODICE,NOME,COGNOME,NASCITA  
FROM PROFESSORI
```

# SQL: Viste

---

## Q. A che serve definire una vista?

- Implementare meccanismi di **indipendenza** tra il livello logico ed il livello esterno.
-  ○ Scrivere **interrogazioni complesse**, semplificandone la sintassi.
- Garantire la **retro-compatibilità** con precedenti versioni dello schema del DB, in caso di ristrutturazione dello stesso.

# SQL: Viste

---

**Es.** Estrarre il nome del dipartimento che ha la spesa più alta in stipendi.

## STRUTTURATI

Codice	Nome	Cognome	Tipo	Dipartimento	Stipendio
123	Marco	Marchi	Associato	Chimica	20000
124	Michele	Micheli	Associato	Fisica	20000
125	Lucia	Di Lucia	Ordinario	Fisica	30000
126	Dario	Rossi	Ordinario	Informatica	32000
127	Mario	Rossi	Ricercatore	Informatica	15000
129	Michele	Bianchi	Associato	Fisica	20000

# SQL: Viste

---

L'interrogazione seguente potrebbe **non essere consentita** su alcuni DBMS (annidamento nella clausola having) ...

```
SELECT DIPARTIMENTO
FROM STRUTTURATI
GROUP BY DIPARTIMENTO
HAVING SUM(STIPENDIO) >= ALL
      SELECT SUM(STIPENDIO)
      FROM STRUTTURATI
      GROUP BY DIPARTIMENTO
```

# SQL: Viste

---

**Soluzione.** Creare una vista che visualizzi la somma totale degli stipendi di ciascun dipartimento.

```
CREATE VIEW SPESEDIPARTIMENTI  
(NOMEDIP, SPESA) AS  
SELECT DIPARTIMENTO, SUM(STIPENDI)  
FROM STRUTTURATI  
GROUPBY DIPARTIMENTO
```



# SQL: Viste

---

**STEP 2.** Estrarre il nome del dipartimento che ha la spesa piu' alta in stipendi usando la vista SPESEDIPARTIMENTI.


```
SELECT NOMEDIP  
FROM SPESEDIPARTIMENTI  
WHERE SPESA=(SELECT MAX(STIPENDI)  
              FROM SPESEDIPARTIMENTI)
```

**D:** E' possibile scrivere *un'interrogazione equivalente senza usare una vista?*

# SQL: Viste

---

## Q. A che serve definire una vista?

- Implementare meccanismi di **indipendenza** tra il livello logico ed il livello esterno.
- Scrivere **interrogazioni complesse**, semplificandone la sintassi.
-  ○ Garantire la **retro-compatibilità** con precedenti versioni dello schema del DB, in caso di ristrutturazione dello stesso.

# SQL: Viste

---

Si supponga di avere un DB, in cui la tabella:

ESAMI(Matricola, Nome, Cognome, Data, Voto)

Viene **sostituita** con le tabelle:

STUDENTI(Matricola, Nome, Cognome)

PROVE(Matricola, Data, Voto)

Con le viste, è possibile mantenere anche la **struttura originaria**.

# SQL: Viste

---

Con le viste, è possibile mantenere anche la **struttura originaria** del database ...

```
CREATE VIEW ESAMI  
(MATRICOLA, NOME, COGNOME, DATA, VOTO)  
AS SELECT S.*, P.DATA, P.VOTO  
FROM STUDENTI AS S, PROVE AS P  
WHERE S.MATRICOLA=P.MATRICOLA
```


# SQL: Viste

---

In generale, l'**aggiornamento** di una vista è un' **operazione molto delicata**, ed è consentita solo in un sottoinsieme (limitato) di casi ...

In molti DBMS commerciali, non e' consentito l'aggiornamento di viste che sono ottenute da piu' di una tabella.

```
CREATE VIEW CAPI(NOME, TELEFONO) AS  
SELECT I.NOME,U.TEL  
FROM IMPIEGATI AS I,UFFICI AS U  
WHERE (I.NOME=U.NOME) AND (I.RUOLO="C")
```



Piu' di una tabella,  
vista non aggiornabile!!

# SQL: Viste

---

L'opzione `WITH CHECK OPTION` consente di definire **viste aggiornabili**, a condizione che le **tuple aggiornate continuino ad appartenere alla vista** (in pratica, la tupla aggiornata non deve violare la clausola `WHERE`).

```
CREATE VIEW  
PROFESSORIRICCHI(CODICE,NOME,COGNOME,STIPENDIO) AS  
SELECT CODICE,NOME,COGNOME,STIPENDIO  
FROM PROFESSORI  
WHERE (STIPENDIO>=30000)
```

# SQL: Viste

PROFESSORI

Codice	Nome	Cognome	Nascita	Livello	Stipendio
1	Marco	Marchi	10/04/1980	A1	20000
3	Michele	Micheli	12/05/1967	R	20000
5	Lucia	Di Lucia	12/05/1978	R2	30000
7	Dario	Rossi	24/01/1965	O2	32000

PROFESSORIRICCHI

Codice	Nome	Cognome	Stipendio
5	Lucia	Di Lucia	30000
7	Dario	Rossi	32000

UPDATE PROFESSORIRICCHI  
SET STIPENDIO=20000  
WHERE (CODICE=5)

Operazione NON consentita!!

# SQL: CTE

---

Le **Common Table Expression** (CTE) rappresentano **viste temporanee** che possono essere usate in una query come se fossero una vista a tutti gli effetti.

Differenza con la vista → una CTE non esiste a livello di schema del DB!

**WITH**

**NAME(Attributi) AS**

**SQL Query**



# SQL: CTE

---

**Es.** Estrarre il nome del dipartimento che ha la spesa più alta in stipendi.

```
WITH SPESEDIPARTIMENTI (NOMEDIP, SPESA) AS  
SELECT DIPARTIMENTO, SUM(STIPENDI)  
FROM STRUTTURATI  
GROUPBY DIPARTIMENTO
```

```
SELECT NOMEDIP  
FROM SPESEDIPARTIMENTI  
WHERE SPESA=(SELECT MAX(STIPENDI)  
              FROM SPESEDIPARTIMENTI)
```

La vista temporanea SPESEDIPARTIMENTI  
è valida solo nella query sottostante!

# SQL: Asserzioni

---

Le **asserzioni** (SQL2) sono un costrutto per definire **vincoli generici a livello di schema**.

`create assertion NomeAsserzione check Condizione`

- Consentono di definire **vincoli non altrimenti definibili con i costrutti visti fin qui**.
- Il vincolo può essere **immediato o differito** (ossia verificato al termine di una transazione).

# SQL: Asserzioni

---

Il voto deve essere compreso tra 18 e 30.

```
CREATE ASSERTION VotoValido CHECK (Voto IS  
NOT NULL AND (VOTO>=18) AND (Voto<=30))
```

La tabella STUDENTI non può essere vuota ...

```
CREATE ASSERTION TabellaValida CHECK  
(1>=SELECT COUNT(*) FROM STUDENTI)
```

# SQL: Asserzioni

---

```
CREATE SCHEMA IMP_SCHEMA;
```

```
CREATE TABLE IMPIEGATI (  
    NOME VARCHAR(20);  
    COGNOME VARCHAR(20);  
    SALARIO NUMERIC;  
    CODICE SMALLINT PRIMARY KEY;  
);
```

```
CREATE ASSERTION SALARIO_CONTROLLO  
    CHECK (NOT EXISTS (SELECT * FROM IMPIEGATI  
        WHERE (SALARIO > 35000)));
```

# SQL: Asserzioni

---

Elementi di uno schema SQL visti fin qui:

- **Tabelle**
- **Domini**
- **Viste**
- **Asserzioni**

**ALTRO?**



- Stored Procedures
- Trigger
- Regole d'accesso

# SQL: DML/DQL

---

Esistono altre **tre** varianti (poco usate) dell'operatore di JOIN

- **FULL join** → risultato dell'inner join + righe della tabella di sinistra/destra che non hanno un corrispettivo a destra/sinistra (completate con valori NULL)

```
SELECT ListaAttributi  
FROM Tabella FULL JOIN Tabella ON CondizioneJoin  
[WHERE Condizione]  
...
```