

SQL

Lucia Ferrari

`lucia02.ferrari@edu.unife.it`

Introduzione

- SQL non rimuove di default i duplicati dalle operazioni di proiezione

Una query in SQL può contenere fino a 6 clausole di cui solo le prime due sono obbligatorie:

```
SELECT    <list – of – selection – expressions>  
FROM      <list – of – tables>  
WHERE     <condition>  
GROUP BY  <list – of – (grouping) – attributes>  
HAVING    <grouping – condition>  
ORDER BY  <list – of – (ordering) – attributes>
```

Il blocco fondamentale è creato dalle prime 3 clausole ed è definito 'mapping'

SELECT specifica gli attributi e/o le funzioni il cui valore deve essere estratto dalla valutazione della query. Utilizzando la keyword **DISTINCT** eliminiamo le tuple duplicate. Corrisponde all'operazione di proiezione.

FROM specifica da quali relazioni andiamo a prendere le informazioni per la query

WHERE specifica le condizioni, tramite un'espressione booleana per la selezione delle tuple da prendere dalle relazioni specificate nella clausola **FROM**. Corrisponde all'operazione di selezione.

Query di base

Sintassi del blocco fondamentale **SELECT-FROM-WHERE**:

```
SELECT attribute – expression [[AS] ALIAS]  
        { , attribute – expression [[AS] ALIAS]}  
FROM   table [[AS] ALIAS]  
        { , table [[AS] ALIAS]}  
WHERE  condition
```

Nel caso più semplice l'esecuzione del blocco è una proiezione sugli attributi specificati nella clausola **SELECT** del risultato della selezione delle tuple che rispettano la condizione della clausola **WHERE**, che si basa sul prodotto cartesiano delle relazioni specificate della clausola **FROM**.

Modello Relazionale che usiamo per le query

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

DEPT_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

WORKS_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

Figure 5.5
Schema diag
COMPANY r
database sch

Alias e asterisco(*)

Esempio 1 Trova il salario di tutti gli impiegati con cognome 'Thompson'

```
SELECT SALARY AS THOMPSON_SALARY  
FROM EMPLOYEE  
WHERE LNAME = 'Thompson '
```

Esempio 2. Recupera tutte le informazioni degli impiegati con cognome 'Thompson'

```
SELECT *  
FROM EMPLOYEE  
WHERE LNAME = 'Thompson '
```

Esempio 3. Trova il salario mensile di tutti gli impiegati con cognome 'Thompson' (Salary/12 esegue direttamente l'operazione sul salario)

```
SELECT SALARY/12 AS MONTHLY_SALARY  
FROM EMPLOYEE  
WHERE LNAME = 'Thompson'
```

Esempio 4. Per ogni impiegato recupera il suo codice SSN e il dipartimento in cui lavora

```
SELECT SSN, DNAME  
FROM EMPLOYEE, DEPARTMENT  
WHERE DEPT = DNUMBER
```

SELECT-FROM (Sub-)Block and DISTINCT

Esempio 5. recupera le informazioni del salario di ogni impiegato

```
SELECT SSN, SALARY  
FROM   EMPLOYEE
```

Esempio 6. Determina tutti i differenti salari esistenti

```
SELECT DISTINCT SALARY  
FROM   EMPLOYEE
```

Dot Notation e nomi di attributi ambigui

Esempio 7. Per ogni impiegato identifica nome e cognome e recupera il codice del progetto su cui lavora.

```
SELECT EMPLOYEE.SSN, EMPLOYEE.FNAME, EMPLOYEE.LNAME, W.PROJECT
FROM   EMPLOYEE, WORKS_ON AS W
WHERE  SSN = EMP
```

Esempio 8. Recupera le informazioni di nome e data di nascita di ogni dipendente supervisionato.

```
SELECT SSN, D.FNAME, D.BIRTH_DATE
FROM   EMPLOYEE, DEPENDENT D
WHERE  SSN = EMP
```

Per evitare ambiguità posso utilizzare la notazione puntata indicando
TABELLA.ATTRIBUTO

Esempio 9. Recupera nome e cognome di tutti i supervisori di impiegati che lavorano nel dipartimento 10

```
SELECT S.FNAME, S.LNAME  
FROM EMPLOYEE AS E S  
WHERE E.DEPT = 10 AND E.SUPERSSN = S.SSN
```

Esempio 10. Trova i dipartimenti che hanno il comune almeno una collocazione (location)

```
SELECT DISTINCT L1.DNUMBER, L2.DNUMBER  
FROM DEPT_LOCATIONS L1 L2  
WHERE L1.LOCATION = L2.LOCATION AND L1.DNUMBER < L2.DNUMBER
```

Usiamo $L1.DNUMBER < L2.DNUMBER$ per evitare di avere le stesse coppie ripetute due volte in ordine inverso NB: l'operazione `DISTINCT` lavora a livello di tuple (la considera per intero)

Espressioni booleane nella clausola WHERE

Esempio 11. Recupera nome e cognome di tutti gli impiegati uomini che guadagnano più di 4000 euro (assumendo non esistano omonimi)

```
SELECT FNAME, LNAME  
FROM   EMPLOYEE  
WHERE  SEX = 'M' AND SALARY > 40000
```

Esempio 12. Determina gli impiegati con cognome 'Thompson' e che lavorano per il dipartimento 2 o il dipartimento 3

```
SELECT SSN  
FROM   EMPLOYEE  
WHERE  LNAME = 'Thompson' AND (DEPT = 2 OR DEPT = 3)
```

NB: è importante inserire le parentesi per avere la giusta espressione

Operazioni insiemistiche

Le operazioni insiemistiche sono: **UNION**, **EXCEPT** (differenza insiemistiche) e **INTERSECT**. I duplicati ottenuti vengono rimossi a meno che non venga specificato il contrario (con la keyword **ALL**)

Esempio 13. Seleziona il nome di tutti gli impiegati e di tutti i dipendenti

```
SELECT FNAME  
FROM   EMPLOYEE  
UNION  
SELECT FNAME  
FROM   DEPENDENT
```


Se gli attributi hanno nomi differenti l'unione può comunque essere eseguita e il nome finale dell'attributo è quello del primo operatore

Esempio 14. Seleziona il nome e il cognome di tutti gli impiegati

```
SELECT FNAME  
FROM EMPLOYEE  
UNION  
SELECT LNAME  
FROM EMPLOYEE
```

Esempio 15. Seleziona il nome e il cognome di tutti gli impiegati del dipartimento 10 mantenendo i duplicati

```
SELECT FNAME  
FROM   EMPLOYEE  
WHERE  DEPT = 10  
  
UNION ALL  
SELECT LNAME  
FROM   EMPLOYEE  
WHERE  DEPT = 10
```

Esempio 16. Seleziona i cognomi che sono anche nomi (non necessariamente dello stesso impiegato)

```
SELECT LNAME
FROM   EMPLOYEE
INTERSECT
SELECT FNAME
FROM   EMPLOYEE
```

Esempio 17. Seleziona i nomi degli impiegati che non sono anche cognomi (di altri impiegati)

```
SELECT FNAME
FROM   EMPLOYEE
EXCEPT
SELECT LNAME
FROM   EMPLOYEE
```

Query innestate

Nella clausola **WHERE** SQL permette di confrontare, utilizzando sempre gli stessi operatori, un valore con il risultato dell'esecuzione di una query SQL (innestata).

Per confrontare questi valori si usano le keywords:

ANY: Le tuple soddisfano le condizioni di confronto se i valori dell'attributo della tupla in considerazione corrisponde con **ALMENO UNO** degli elementi forniti dalla valutazione della query innestata.

ALL: Le tuple soddisfano le condizioni di confronto se i valori dell'attributo della tupla in considerazione corrisponde con **TUTTI** gli elementi forniti dalla valutazione della query innestata.

NB: è richiesta la compatibilità tra gli attributi della query innestata e quelli con cui si confronta.

Esempio 18. Seleziona gli impiegati che appartengono ad un dipartimento con location = "Ferrara".

```
SELECT *  
FROM   EMPLOYEE  
WHERE  DEPT = ANY ( SELECT DNUMBER  
                     FROM   DEPT_LOCATIONS  
                     WHERE  DLOCATION = 'Ferrara' )
```

Esempio 19. Seleziona l'SSN degli impiegati il cui cognome è uguale ad alcuni impiegati che lavorano per il dipartimento 10.

```
SELECT SSN  
FROM   EMPLOYEE  
WHERE  LNAME = ANY ( SELECT LNAME  
                     FROM   EMPLOYEE  
                     WHERE  DEPT = 10 )
```

Esempio: ALL

Esempio 20. Seleziona tutti i dipartimenti che non hanno impiegati che guadagnano >80000 euro.

```
SELECT DNUMBER
FROM DEPARTMENT
WHERE DNUMBER <> ALL ( SELECT DEPT
                        FROM EMPLOYEE
                        WHERE SALARY > 80000 )
```

(<> significa diverso, in questo caso <> ALL indica 'diverso da tutti')

Esempio 21. Seleziona gli impiegati che ricevono il salario massimo

```
SELECT SSN
FROM EMPLOYEE
WHERE SALARY >= ALL ( SELECT SALARY
                      FROM EMPLOYEE )
```

Esempio 20': Seleziona tutti i dipartimenti che non hanno impiegati che guadagnano >80000 euro.

```
SELECT DNUMBER
FROM DEPARTMENT
EXCEPT
SELECT DEPT
FROM EMPLOYEE
WHERE SALARY > 80000
```

Esempio 21' Seleziona gli impiegati che ricevono il salario massimo

```
SELECT SSN
FROM EMPLOYEE
EXCEPT
SELECT E1.SSN
FROM EMPLOYEE AS E1 E2
WHERE E1.SALARY < E2.SALARY
```


Per verificare l'appartenenza (o non appartenenza) di un elemento/valore ad un insieme di elementi/valori ottenuti da una query innestata si può usare l'operatore **IN** (che è equivalente a = **ANY**) e **NOT IN** (equivalente a <> **ALL**).

Esempio 18'. Seleziona gli impiegati che appartengono ad un dipartimento con location = "Ferrara".

```
SELECT *  
FROM   EMPLOYEE  
WHERE  DEPT IN ( SELECT DNUMBER  
                  FROM   DEPT_LOCATIONS  
                  WHERE  DLOCATION = 'Ferrara' )
```

In tutte le query innestate che abbiamo visto finora, la query interna può essere eseguita una sola volta e prima di analizzare la query esterna. Il risultato della valutazione della query innestata non dipende da tuple specifiche della query esterna. In questo caso si parla di query innestata NON correlata con quella esterna.

A volte la query innestata fa riferimento alle tuple della query esterna: il risultato della valutazione della query interna quindi dipende dalle tuple della query esterna. Si parla di query innestata correlata con quella esterna.

Abbiamo quindi bisogno di un meccanismo di binding: gli attributi della query esterna devono essere utilizzabili dentro quella interna.

Regola di visibilità (scoping): gli attributi di una tabella sono utilizzabili solo nel contesto della query in cui la tabella è dichiarata o nella sua query innestata (a qualunque livello).

Utilizzo degli alias: possono esserci delle ambiguità quando gli attributi hanno lo stesso nome quindi bisogna usare gli alias, ovvero AS (regola di disambiguità).

Esempio violazione della regola della visibilità

Esempio 22. Seleziona gli impiegati che lavorano per il dipartimento di ricerca o per un dipartimento che ha una location in almeno una delle città in cui ci sono dipartimenti di ricerca.

```
SELECT SSN
FROM   EMPLOYEE
WHERE  DEPT IN ( SELECT DNUMBER
                  FROM   DEPARTMENT AS D1
                        DEPT_LOCATIONS AS L1
                  WHERE  D1.DNUMBER = L1.DNUMBER AND
                        D1.DNAME = 'Research' ) OR
        DEPT IN ( SELECT DNUMBER
                  FROM   DEPT_LOCATIONS AS L2
                  WHERE  L1.DLOCATION = L2.DLOCATION )
```

Questa soluzione è SBAGLIATA perchè la variabile L1 non è visibile nella seconda query innestata.

Esempio 23. Determina il nome e il cognome di tutti gli impiegati che hanno dipendenti con il loro stesso sesso e nome.

```
SELECT FNAME, LNAME
FROM   EMPLOYEE E
WHERE  SSN IN ( SELECT EMP
                 FROM   DEPENDENT
                 WHERE  E.FNAME = FNAME AND E.SEX = SEX )
```

Esempio 24. Seleziona tutti gli impiegati il cui salario è diverso da tutti i salari degli altri impiegati dello stesso dipartimento.

```
SELECT SSN
FROM   EMPLOYEE AS E
WHERE  SALARY NOT IN ( SELECT SALARY
                       FROM   EMPLOYEE
                       WHERE  SSN <> E.SSN AND DEPT = E.DEPT )
```

SQL introduce una funzione booleana, **EXISTS**, che permette di verificare se il risultato di una query innestata è vuoto o meno.

EXISTS: Il risultato di una relazione NON è vuoto

NOT EXISTS: Il risultato di una relazione è vuoto

Dato che non siamo interessati alle tuple di ritorno della query innestata ma solo dalla loro presenza o meno, la clausola **SELECT** della query innestata è generalmente nella forma: **SELECT ***.

Esempi: EXISTS e NOT EXISTS

Esempio 25. Seleziona tutti gli impiegati che non hanno dipendenti

```
SELECT SSN
FROM EMPLOYEE
WHERE NOT EXISTS ( SELECT *
                    FROM DEPENDENT
                    WHERE SSN = EMP )
```

Esempio 26. Restituisci il nome e il cognome di manager che hanno almeno un dipendente

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE EXISTS ( SELECT *
               FROM DEPARTMENT
               WHERE SSN = MANAGER ) AND
            EXISTS ( SELECT *
                    FROM DEPENDENT
                    WHERE SSN = EMP )
```

Operatore CONTAINS

SQL originale aveva l'implementazione dell'operatore **CONTAINS**, utilizzato per stabilire se un insieme ne contiene un altro.

Esempio 27. Trova gli impiegati che lavorano su tutti i progetti che sono controllati dal dipartimento 10.

```
SELECT SSN
FROM EMPLOYEE
WHERE ( SELECT PROJECT
        FROM WORKS_ON
        WHERE SSN = EMP )
CONTAINS
( SELECT PNUMBER
  FROM PROJECT
  WHERE DNUM = 10 )
```

L'operatore **CONTAINS** è stato però rimosso da SQL.

Come esprimere l'operatore CONTAINS

Esempio 27'. Trova gli impiegati che lavorano su tutti i progetti che sono controllati dal dipartimento 10.

```
SELECT SSN
FROM   EMPLOYEE
WHERE  NOT EXISTS (
        SELECT *
        FROM   PROJECT
        WHERE  DNUM = 10 AND
              NOT EXISTS ( SELECT *
                          FROM   WORKS_ON
                          WHERE  SSN = EMP AND
                                PNUMBER = PROJECT ) )
```

L'operatore booleano **UNIQUE** permette di verificare che il risultato di una query innestata non abbia duplicati.

Esempio 28. Trova gli impiegati che non hanno più di un dipendente

```
SELECT SSN
FROM EMPLOYEE
WHERE UNIQUE ( SELECT SEX
                FROM DEPENDENT
                WHERE SSN = EMP )
```

Operazioni di JOIN

SQL permette di specificare una tabella ottenuta come risultato di un'operazione di join tramite la clausola **FROM**, indicando la condizione di join nella clausola **WHERE** o direttamente nel **FROM**.

Tipologie di join:

- **INNER JOIN** (o semplicemente **JOIN**);
- **LEFT OUTER JOIN**, **RIGHT OUTER JOIN**, **FULL OUTER JOIN** (la keyword **OUTER** può essere omessa);
- **NATURAL JOIN**;
- **NATURAL LEFT/RIGHT/FULL OUTER JOIN**.

Esempio 29. Recupera il nome, cognome e indirizzo degli impiegati che lavorano per il dipartimento di ricerca

```
SELECT FNAME, LNAME, ADDRESS
FROM    ( EMPLOYEE JOIN DEPARTMENT ON DEPT = DNUMBER )
WHERE   DNAME = 'Research '
```

Esempio 30. Per ogni progetto realizzato a Bologna, restitisci il numero del progetto, il dipartimento che lo controlla e il cognome del manager di quel dipartimento.

```
SELECT PNUMBER, DNUMBER, LNAME
FROM    (( PROJECT JOIN DEPARTMENT ON DNUM = DNUMBER )
        JOIN EMPLOYEE ON MANAGER = SSN )
WHERE   PLOCATION = 'Bologna '
```

Esempio 29'. Recupera il nome, cognome e indirizzo degli impiegati che lavorano per il dipartimento di ricerca

```
SELECT FNAME, LNAME, ADDRESS  
FROM EMPLOYEE, DEPARTMENT  
WHERE DEPT = DNUMBER AND DNAME = 'Research'
```

Esempio 31. Restituire nome e cognome di ogni impiegato e del suo supervisore (se esiste).

```
SELECT E.FNAME AS FNAME_EMPLOYEE, E.LNAME AS LNAME_EMPLOYEE,  
       S.FNAME AS FNAME_SUPERVISOR, S.LNAME AS LNAME_SUPERVISOR  
FROM   ( EMPLOYEE AS E LEFT OUTER JOIN EMPLOYEE AS S  
         ON E.SUPERSSN = S.SSN )
```

Esempio 32. Restituisci tutti i dipartimenti e i progetti che controllano (se esistono)

```
SELECT PNUMBER, DNUMBER  
FROM   ( PROJECT RIGHT OUTER JOIN DEPARTMENT ON DNUM = DNUMBER )
```

Funzione aggregate

SQL ha delle funzioni built-in come: **COUNT**, **SUM**, **AVG**, **MIN** e **MAX** (altre funzioni per calcolare varianza e altro sono state aggiunte da SQL-99).

Esempio 33. Trovare il salario: totale, medio, massimo e minimo che l'azienda paga.

```
SELECT SUM(SALARY) , AVG(SALARY) , MAX(SALARY) , MIN(SALARY)  
FROM    EMPLOYEE
```

Le funzioni aggregate considerano tutti i valori che non sono NULL dell'attributo salario.

COUNT

L'operatore **COUNT** permette di contare il numero di righe di una tabella:

- utilizzando l'operatore * conta tutte le righe distinte
- utilizzando COUNT(attributo) conta anche le righe uguali
- utilizzando COUNT(DISTINCT attributo) esclude le righe uguali

I valori NULL non vengono considerati

Esempio 34. Determina il numero di impiegati del dipartimento 3

```
SELECT COUNT(*)  
FROM   EMPLOYEE  
WHERE  DEPT = 3
```

Esempio 35. Determina quanti sono i possibili salari distinti degli impiegati.

```
SELECT COUNT(DISTINCT SALARY)  
FROM   EMPLOYEE
```

Query inconsistenti (Mismatch)

Le funzioni aggregate non forniscono un meccanismo di selezione: sono applicate ad un insieme di valori e ne restituiscono uno solo.

Di conseguenza la clausola **SELECT** non può contenere uno o più attributi che generano un valore per ogni tupla e una o più funzioni aggregate, che generano un solo valore per l'intero insieme di tuple.

Il risultato non sarebbe corretto

Esempio di query inconsistente

```
SELECT FNAME, LNAME, MAX(SALARY)
FROM EMPLOYEE, DEPT_LOCATIONS
WHERE DEPT = DNUMBER AND DLOCATION = 'Ferrara'
```

Il comando **GROUP BY** ci permette di partizionare in gruppi le tuple di una relazione.

Esempio 36. Per ogni dipartimento, determina la somma dei salari dei suoi impiegati

```
SELECT  DEPT, SUM(SALARY)
FROM    EMPLOYEE
GROUP BY DEPT
```

Tramite la clausola **GROUP BY**, l'operatore **SELECT** può contenere SOLO: funzioni aggregate e un sottoinsieme degli attributi del **GROUP BY**.

La clausola **HAVING** è utilizzata in combinazione con la **GROUP BY** per introdurre delle condizioni sui gruppi.

Esempio 38. Seleziona tutti e soli i dipartimenti che spendono più di 1000000 euro in salari per i loro impiegati (e riporta il totale)

```
SELECT   DEPT, SUM(SALARY) AS TOTAL_AMOUNT_OF_SALARIES
FROM     EMPLOYEES
GROUP BY DEPT
HAVING    SUM(SALARY) > 1000000
```

Prima viene valutata la condizione del **WHERE** per selezionare un insieme di tuple, poi, dopo aver partizionato le tuple nei gruppi richiesti viene valutata la condizione del **HAVING** per eliminare alcune classi della partizione.

Esempio 39. Per ogni dipartimento che ha più di 5 impiegati, determina il numero di impiegati che guadagnano più di 6000 euro.

SOLUZIONE ERRATA

```
SELECT  DEPT, COUNT(*)  
FROM    EMPLOYEE  
WHERE   SALARY > 60000  
GROUP BY DEPT  
HAVING  COUNT(*) > 5
```

Il risultato di questa query offrirebbe i dipartimenti che hanno almeno 5 impiegati con salario > 60000

Esempio 39'. Per ogni dipartimento che ha più di 5 impiegati, determina il numero di impiegati che guadagnano più di 6000 euro.

SOLUZIONE CORRETTA

```
SELECT  DEPT, COUNT(*)
FROM    EMPLOYEE
WHERE   SALARY > 60000 AND
        DEPT IN ( SELECT  DEPT
                   FROM    EMPLOYEE
                   GROUP BY DEPT
                   HAVING  COUNT(*) > 5 )
GROUP BY DEPT
```

Esempio 40. Determinare il nome e il cognome degli impiegati che hanno più di 2 dipendenti (usando le funzioni aggregate).

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE ( SELECT COUNT(*)
        FROM DEPENDENT
        WHERE SSN = EMP ) >= 2
```


Varie

Esempio 41. Seleziona gli impiegati che dedicano lo stesso numero di ore in un progetto dell'impiegato 77.

```
SELECT DISTINCT EMP
FROM   WORKS_ON
WHERE  (PROJECT, WEEKLY_HOURS) IN ( SELECT PROJECT, WEEKLY_HOURS
                                   FROM   WORKS_ON
                                   WHERE  EMP = '77' )
```

Confronto una coppia di valori invece che un singolo

Esempio 42. Seleziona gli impiegati che lavorano nei progetti: 1,2 o 3

```
SELECT DISTINCT EMP
FROM   WORKS_ON
WHERE  PROJECT IN {1,2,3}
```

Operatore LIKE

L'operatore **LIKE** è utilizzato per fare pattern match sulle stringhe.

Esistono due caratteri utilizzati apposta per il LIKE:

1. %, rimpiazza un numero arbitrario (0 o più) di caratteri
2. _, rimpiazza un singolo carattere.

Esempio 43. Seleziona gli impiegati che vivono in una città che inizia con Bo (es: Bologna, Bolzano,...)

```
SELECT SSN
FROM EMPLOYEE
WHERE ADDRESS LIKE 'Bo%'
```

Esempio 44. Seleziona tutti gli impiegati che sono nati negli anni '60S(assumiamo che la data sia nel formato: YYYY-MM-DD).

```
SELECT SSN
FROM EMPLOYEE
WHERE DATE_OF_BIRTH LIKE '196 _ _ _ _'
```

ESCAPE e BETWEEN

Se `_` o `%` sono parte di una stringa da memorizzare nel database, ogni occorrenza deve essere preceduta da un escape character, specificato alla fine di una stringa utilizzando il keyword **ESCAPE**.

Esempio: la sequenza `'A!_B!%C'ESCAPE'!'` rappresenta la stringa `A_B%C` perchè il carattere di escaping è `!`.

Per effettuare confronti è utile l'operatore **BETWEEN**.

Esempio 45. Seleziona gli impiegati del dipartimento 3 che guadagnano un salario tra 30000 e 40000 (inclusi).

```
SELECT *  
FROM   EMPLOYEE  
WHERE  ( SALARY BETWEEN 30000 AND 40000 ) AND DEPT = 3
```

Il risultato di una query può essere ordinato, in modo crescente (**ASC** keyword) o decrescente (**DESC** keyword) sulla base di uno o più attributi tramite la clausola **ORDER BY**.

Esempio 46. Restituisci una lista degli impiegati insieme ai progetti su cui lavorano ordinati in modo decrescente sulla base del dipartimento, e per ogni dipartimento in ordine crescente di nome e cognome.

```
SELECT      DNAME, FNAME, LNAME, PNAME
FROM        EMPLOYEE, DEPARTMENT, PROJECT, WORKS_ON
WHERE       DEPT = DNUMBER AND SSN = EMP AND PROJECT = PNUMBER
ORDER BY    DNAME DESC, FNAME ASC, LNAME ASC
```

Di default **ORDER BY** utilizza l'ordine crescente quindi **ASC** può essere omesso

SQL permette di selezionare tuple che hanno valori NULL sull'attributo dato (oppure valore non NULL) tramite il predicato **IS NULL** e **IS NOT NULL**

Esempio 47. Seleziona tutti gli impiegati in cui salario è sconosciuto

```
SELECT *  
FROM   EMPLOYEE  
WHERE  SALARY IS NULL
```

SQL Data Definition

Elementi fondamentali

- Definizione di domini,
- Definizione di schema e delle sue tabelle,
- Definizione di vincoli,
- Definizione di valori di default,
- Vincoli interni alla relazione,
- Vincoli tra le relazioni,
- Modifiche ai valori delle tabelle,

Il dominio specifica qual è l'insieme di valori che gli attributi possono avere:

Esistono due tipi di domini:

- Domini elementari: forniti da SQL
- Domini definiti dall'utente: tramite comandi appositi.

I domini elementari principali riguardano:

- Caratteri
- Numeri esatti e approssimati
- Istanti e intervalli temporali

Dominio che consente di definire stringhe con un numero preciso oppure variabile di caratteri

Sintassi:

```
character [varying] [(length)]  
          [character set name - of - characters - family]
```

Examples:

- Stringa di esattamente 30 caratteri: `character(30)`, versione più usata: `char(30)`
- Stringa di al massimo 30 caratteri: `character varying(30)`, versione più usata: `varchar(15)`

Numeri esatti

Tipologie di numeri esatti: exact number, interi o decimali con parte decimale fissata

Sintassi possibili:

- `numeric [(precision [, scale])]`
- `decimal [(precision [, scale])]`
- `integer`
- `smallint`

`precision` indica il numero possibili di cifre mentre `scale` indica il numero di decimali (se `scale` non è specificato di default è zero)

Esempio: `numeric(6,4)` permette di rappresentare numeri tra -99.9999 e +99.9999.

`integer` e `smallint` si basano sulla rappresentazione binaria interna delle macchine

Numeri approssimati

Utilizzati per rappresentare valori reali

Sintassi possibili:

- `float` [(*precision*)] (*precision* indica il numero di cifre della mantissa, il numero di cifre dell'esponente dipende dall'implementazione)
- `real` (precisione fissata)
- `double precision` (precisione doppia rispetto a `real`)

In tutte le precedenti alternative ogni numero è descritto da una coppia mantissa e esponente. La mantissa è un numero frazionale e l'esponente un numero intero.

Esempi: $0.17E16$ rappresenta il numero $1.7 \cdot 10^{15}$, e $-0.4E-6$ rappresenta il numero $-4 \cdot 10^{-7}$.

Istanti temporali

Esistono 3 possibili sintassi

- **date**
- **time** [(*precision*)] [*with* – *the* – *time* – *zone*]
- **timestamp** [(*precision*)] [*with* – *the* – *time* – *zone*]

Il dominio **date** consente i campi: *anno*, *mese* e *giorno*.

Il dominio **time** consente i campi: *ora*, *minuti* e *secondi*.

Il dominio **timestamp** consente tutti i campi di **date** e **time**.

precision indica il numero di cifre decimali usate per rappresentare la frazione dei secondi

L'opzione *with* – *the* – *time* – *zone* consente la presenza di due ulteriori campi: *timezone_hour* e *timezone_minute*, che rappresentano la differenza tra il tempo locale e quello universale (Coordinated Universal Time o UTC). Esempio: 21 : 03 : 04 + 1 : 00.

Definizione dello schema

SQL definisce uno schema come una collezione di oggetti (es: domini, tabelle...)

Sintassi:

```
create schema [schema – name] [[ authorization ] schema – owner]
```

schema – owner è l'autore dello schema, se è ommesso si assume che sia quello che esegue il comando.

Se *schema – name* è ommesso si assume che il nome corrisponda al proprietario dello schema

Esempio: **CREATE SCHEMA** azienda **AUTHORIZATION** JohnDoe

Definizione delle tabelle dello schema

In SQL una tabella è data da un insieme ordinato di attributi, ognuno con i suoi domini e vincoli

Definition of a table:

```
CREATE TABLE DEPARTMENT (  
    DNUMBER      INTEGER PRIMARY KEY,  
    DNAME        VARCHAR(20) ,  
    MANAGER      CHAR(16) ,  
    START_DATE   DATE,  
    UNIQUE(DNAME) ,  
    FOREIGN KEY (MANAGER) REFERENCES EMPLOYEE(SSN)  
        ON DELETE SET NULL  
        ON UPDATE CASCADE )
```

Sintassi:

```
CREATE TABLE table_name (  
    attribute - name      domain [default - value] [constraints]  
    {, attribute - name domain [default - value] [constraints]}  
    other - constraints )
```

Inizialmente la tabella è vuota (non abbiamo nessuna riga).

Nella definizione di nuove tabelle i domini possono essere quelli standard oppure definiti dagli utenti:

Sintassi:

```
CREATE DOMAIN domain - name AS data - type [default - value] [constraint]
```

Esempio:

```
CREATE DOMAIN SSNUMBER AS CHAR(16)
```

default – *value* possono essere utilizzati nella definizione di una tabella o di un dominio e specificano il valore che un attributo deve avere nel caso in cui venga inserita una tupla senza un valore per quell'attributo. Se è ommesso il valore di default è **NULL**.

Sintassi:

```
DEFAULT ⟨generic – value | user | NULL⟩
```

Esempio:

```
DEPT SMALLINT DEFAULT 1
```

Esistono due tipologie di vincoli

- Vincoli interni alla relazione: si basano su una singola tabella e sono:
NOT NULL, UNIQUE, PRIMARY KEY.
- Vincoli esterni alla relazione si basano su più tabelle (sono i vincoli di integrità referenziale)..

NOT NULL: il valore **NULL** non è consentito come possibile valore dell'attributo.

Esempio:

```
LNAME VARCHAR(20) NOT NULL
```

UNIQUE: può essere applicato ad uno o più attributi della tabella quando è necessario che per le tuple ci siano valori tutti diversi per quell'attributo (un'eccezione è il valore NULL)

```
DNAME VARCHAR(20) UNIQUE
```

Vincoli interni alla relazione

Due diverse possibili sintassi per UNIQUE:

1)

```
FNAME VARCHAR(20) NOT NULL UNIQUE,  
LNAME VARCHAR(20) NOT NULL UNIQUE
```

2)

```
FNAME VARCHAR(20) NOT NULL,  
LNAME VARCHAR(20) NOT NULL,  
UNIQUE(FNAME,LNAME)
```

PRIMARY KEY: vincolo che può essere specificato una volta per ogni tabella

```
FNAME VARCHAR(20) ,  
LNAME VARCHAR(20) ,  
PRIMARY KEY(FNAME,LNAME)
```

FOREIGN KEY: rappresentano i vincoli di integrità referenziale.

Due possibili sintassi: 1)

```
DEPT SMALLINT REFERENCES DEPARTMENT(DNUMBER)
```

2)

```
MANAGER CHAR(16) ,  
FOREIGN KEY (MANAGER) REFERENCES EMPLOYEE(SSN)  
ON DELETE SET NULL  
ON UPDATE CASCADE
```

Il vincolo **FOREIGN KEY** crea un collegamento tra il valore dell'attributo nella tabella in cui è definito e il valore di un attributo nella stessa tabella o in una esterna.

L'attributo riferito deve avere almeno un vincolo **UNIQUE**. Generalmente ha vincolo **PRIMARY KEY**

Una variazione nella tabella interna (es: update di una tupla o inserimento) può causare una violazione del vincolo e quindi essere rifiutata.

Una variazione nella tabella esterna (es: update o eliminazione di una tupla) che causa una violazione del vincolo può essere gestita in più modi.

Possibili reazioni a operazioni di update o delete (sulla tabella riferita):

1. **CASCADE**: il nuovo valore è propagato anche nelle corrispondenti tuple delle tabelle interne. (in caso di delete anche le tuple delle tabelle interne sono eliminate)
2. **SET NULL**: il valore **NULL** è assegnato agli attributi della tab.interna
3. **SET DEFAULT**: il valore di default è assegnato agli attributi della tab. interna
4. **NO ACTION**: l'operazione di update/delete viene rifiutata.

Sintassi:

```
ON <DELETE | UPDATE>  
    <CASCADE | SET NULL | SET DEFAULT | NO ACTION>
```


Modifiche agli schema

Per modificare lo schema di un database si utilizzano i comandi: **ALTER** e **DROP**.

ALTER: consente la modifica dei DOMINI e delle TABELLE.

Sintassi modifica domini:

```
ALTER DOMAIN domain - name <SET DEFAULT default - value |  
DROP DEFAULT |  
ADD CONSTRAINT constraint - definition |  
DROP CONSTRAINT constraint - name>
```

Sintassi modifica tabelle:

```
ALTER TABLE table - name  
  <ALTER COLUMN attribute - name <SET DEFAULT | DROP DEFAULT> |  
  ADD CONSTRAINT constraint - definition |  
  DROP CONSTRAINT constraint - name |  
  ADD COLUMN attribute - definition |  
  DROP COLUMN attribute - name>
```

DROP: permette di rimuovere componenti da un database (schema, domini, tabelle)

Sintassi:

```
DROP <SCHEMA | DOMAIN | TABLE | VIEW | ASSERTION>  
      name – of – the – element [RESTRICT | CASCADE]
```

RESTRICT: il comando non è eseguito su componenti non vuote (default option)

CASCADE: tutte le componenti dell'elemento rimosso sono rimosse a loro volta.

Inserimento di tuple: **INSERT INTO**

Sintassi:

```
INSERT INTO table - name [(list - of - attributes)]  
    <VALUES (list - of - values) | SQL - query>
```

Agli attributi il cui valore non è specificato viene assegnato di default **NULL** value, se è presente un vincolo di **NOT NULL** non soddisfatto allora l'inserimento è rifiutato. Inoltre gli attributi devono essere inseriti nello stesso ordine con cui sono stati creati nella tabella.

Esempio. Inserimento di un singolo valore

```
INSERT INTO DEPARTMENT(DNUMBER, DNAME) VALUES (5 , ' Mathematics ')
```

Esempio. Inserimento di più valori

```
INSERT INTO EMP5 ( SELECT SSN, FNAME, LNAME, SALARY  
                  FROM    EMPLOYEE  
                  WHERE  DEPT = 5 )
```

Eliminazione di tuple: DELETE FROM

Sintassi:

```
DELETE FROM table – name [WHERE condition]
```

Il comando rimuove tutte e sole le tuple che soddisfano *condition*; se la clausola **WHERE** è ommessa allora vengono rimosse tutte le tuple.

Se c'è un vincolo di integrità referenziale con la politica **CASCADE** l'operazione di elimina è propagata anche a tutte le tuple della tabella con foreign key

DELETE FROM

Esempio elimina di più tuple

```
DELETE FROM DEPARTMENT WHERE DNUMBER NOT IN ( SELECT DEPT  
                                                FROM EMPLOYEE )
```

Esempio Eliminazione di una singola tupla

```
DELETE FROM DEPARTMENT WHERE DNAME = 'Research '
```

DELETE versus DROP:

```
DELETE FROM DEPARTMENT
```

```
DROP TABLE DEPARTMENT CASCADE
```

```
DROP TABLE DEPARTMENT RESTRICT
```

Aggiornamento di tuple: UPDATE

Sintassi:

```
UPDATE table – name  
    SET attribute =  $\langle \text{expression} \mid \text{SQL – query} \mid \text{NULL} \mid \text{DEFAULT} \rangle$   
        { , attribute =  $\langle \text{expression} \mid \text{SQL – query} \mid \text{NULL} \mid \text{DEFAULT} \rangle$  }  
[WHERE condition]
```

UPDATE consente di aggiornare uno o più attributi delle tuple di *table – name* che soddisfano la condizione del WHERE, se questo è ommesso allora l'aggiornamento è effettuato su tutte le tuple.

Il nuovo valore può essere:

- Il risultato di un'espressione sugli attributi delle tabelle
- Il risultato di una query SQL
- Il valore NULL
- Il valore di DEFAULT del dominio.

Esempio. Aggiornamento di una singola tupla

```
UPDATE EMPLOYEE SET SALARY = SALARY + 5000 WHERE SSN = 999888777
```

Esempio. Aggiornamento di un insieme di tuple

```
UPDATE EMPLOYEE SET SALARY = SALARY * 1.1 WHERE DEPT = 5
```

Esempio Aumento del salario degli impiegati che guadagnano meno di 25k del 15

```
UPDATE EMPLOYEE SET SALARY = SALARY * 1.15 WHERE SALARY <= 25000
```

```
UPDATE EMPLOYEE SET SALARY = SALARY * 1.1 WHERE SALARY > 25000
```

Consiglio pratico

Per inserire delle tuple con foreign key spesso può essere utile disabilitare temporaneamente il controllo sui vincoli delle chiavi:

```
SET FOREIGN_KEY_CHECKS=0;
```

Ricordatevi poi di riabilitarlo tramite:

```
SET FOREIGN_KEY_CHECKS=1;
```

Può anche essere utile disabilitare temporaneamente gli update safe(altrimenti siete sempre costretti ad indicare la chiave primaria per fare degli update)

```
SET SQL_SAFE_UPDATES = 0;
```

```
SET SQL_SAFE_UPDATES = 1;
```