# SQL:
# una visione panoramica

I lucidi presentati sono liberamente ispirati al contenuto del Capitolo 4 del libro
Atzeni, Ceri, Paraboschi, Torlone "Basi di dati" - McGraw-Hill, 1999

## SQL

- Structured Query Language
- è un linguaggio funzionalità di:
  - DDL (Data Definition Language, operante sui meta-dati)
  - DML (Data Manipulation Language, operante sui dati delle istanze del DB)
- ne esistono vari "dialetti"
- vediamo soltanto alcuni gli aspetti essenziali



## SQL: "storia"

- prima proposta: "SEQUEL" (1974);
- prime implementazioni in SQL/DS e Oracle (1981)
- dal 1983 ca. "standard di fatto"
- standard (1986, 1989, 1992, 1999)
  - Spesso recepito solo parzialmente



## Definizione dei dati

- Istruzione CREATE TABLE
  - definisce uno schema di relazione e ne crea un'istanza vuota
  - specifica attributi, domini e vincoli
- Es:

```
CREATE TABLE Impiegato(
        Matricola CHAR(6) PRIMARY KEY,
        Nome CHAR(20) NOT NULL,
        Cognome CHAR(20) NOT NULL,
        Dipart CHAR(15),
        Stipendio NUMERIC(9) DEFAULT 0,
        FOREIGN KEY(Dipart) REFERENCES
                Dipartimento(NomeDip),
        UNIQUE (Cognome,Nome)
)
```

## Domini elementari (predefiniti)

- Carattere: singoli caratteri o stringhe, anche di lunghezza variabile
- Bit: singoli booleani o stringhe
- Numerici, esatti e approssimati
- Data, ora, intervalli di tempo



## Domini definibili dall'utente

- Istruzione CREATE DOMAIN:
  - definisce un dominio (semplice), utilizzabile in definizioni di relazioni, anche con vincoli e valori di default
- Es:

```
CREATE DOMAIN Voto
        AS SMALLINT DEFAULT NULL
        CHECK ( value >=18 AND value <= 30 )
```



## Vincoli intra-relazionali

- NOT NULL
- UNIQUE definisce chiavi
- PRIMARY KEY: chiave primaria (una sola, implica NOT NULL)
- CHECK, vedremo più avanti



## UNIQUE e PRIMARY KEY

- due forme:
  - nella definzione di un attributo, se forma da solo la chiave
  - come elemento separato
- Es:

```
Matricola CHAR(6) PRIMARY KEY

Matricola CHAR(6),
…,
PRIMARY KEY (Matricola)
```

## Chiavi su più attributi

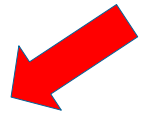
```
Nome            CHAR(20) NOT NULL,
Cognome         CHAR(20) NOT NULL,
UNIQUE (Cognome,Nome),

Nome      CHAR(20) NOT NULL UNIQUE,
Cognome CHAR(20) NOT NULL UNIQUE,
```

- Non è la stessa cosa!



## Vincoli inter-relazionali

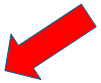
- CHECK, vedremo più avanti
- REFERENCES e FOREIGN KEY permettono di definire vincoli di *integrità referenziale*
- di nuovo: due sintassi
  - per singoli attributi
  - su più attributi
- E' possibile definire politiche di reazione alla violazione





## Vincoli inter-relazionali, esempio

```
CREATE TABLE Infrazioni(
   Codice CHAR(6) NOT NULL PRIMARY KEY,
   Data  DATE NOT NULL,
   Vigile        INTEGER NOT NULL
                 REFERENCES Vigili(Matricola),
   Provincia  CHAR(2),
   Numero     CHAR(6) ,
   FOREIGN KEY(Provincia, Numero)
                 REFERENCES Auto(Provincia, Numero)
 )
```





## Definizione dello schema DB: non solo SQL

- In molti sistemi (tra i quali MS Access) si utilizzano strumenti diversi dal codice SQL per definire lo schema della base di dati

# SQL, operazioni sui dati

- interrogazione:
  - SELECT
- modifica:
  - INSERT, DELETE, UPDATE



# Istruzione SELECT

SELECT <ListaAttributi>
FROM <ListaTabelle>
[ WHERE <Condizione> ]

- "target list"
- clausola FROM
- clausola WHERE



*Persone*

| nome | eta | reddito |
|---|---|---|
| Andrea | 27 | 21 |
| Aldo | 25 | 15 |
| Maria | 55 | 42 |
| Anna | 50 | 35 |
| Filippo | 26 | 30 |
| Luigi | 50 | 40 |
| Franco | 60 | 20 |
| Olga | 30 | 41 |
| Sergio | 85 | 35 |
| Luisa | 75 | 87 |

*Maternita*

| madre | figlio |
|---|---|
| Luisa | Maria |
| Luisa | Luigi |
| Anna | Olga |
| Anna | Filippo |
| Maria | Andrea |
| Maria | Aldo |

*Paternita*

| padre | figlio |
|---|---|
| Sergio | Franco |
| Luigi | Olga |
| Luigi | Filippo |
| Franco | Andrea |
| Franco | Aldo |



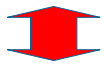
# Selezione

- Nome e reddito delle persone con meno di trenta anni

```
SELECT nome, reddito
FROM Persone
WHERE eta < 30
```

## SELECT, abbreviazioni

```
SELECT nome, reddito
FROM Persone
WHERE eta < 30
```



```
SELECT p.nome AS nome,
                    p.reddito AS reddito
FROM Persone p
WHERE p.eta < 30
```



## Selezione, tutti i campi

- Nome, età e reddito delle persone con meno di trenta anni

```
SELECT *
FROM Persone
WHERE eta < 30
```



## Espressioni nella target list

```
SELECT reddito/2 as redditoSemestrale
FROM Persone
WHERE nome = 'Luigi'
```



## Condizione complessa

```
SELECT *
FROM Persone
WHERE reddito > 25
    AND (eta < 30 or eta > 60)
```

# Condizione “LIKE”

- Le persone che hanno un nome che inizia per 'A' e ha una 'd' come terza lettera

```
SELECT *
FROM Persone
WHERE nome LIKE 'A_d%'
```

Stringa “modello”

- Caratteri “jolly” da usare nella stringa modello:
  - % (talvolta *) :qualsiasi sequenza di caratteri
  - _ (talvolta ?): qualsiasi carattere singolo



# Gestione dei valori nulli

- Gli impiegati la cui età è o potrebbe essere maggiore di 40

```
SELECT *
FROM Impiegati
WHERE eta > 40 OR eta IS NULL
```



# Ridenominazioni

- Possono essere necessarie ridenominazioni
  - nel “prodotto cartesiano” (FROM)
  - nella target list

```
SELECT X.A1 AS B1, ...
FROM R1 X, R2 Y, R1 Z
WHERE X.A2 = Y.A3 AND ...
```

Uso della stessa tabella con due diverse ridenominazioni



# Uso di “DISTINCT”

```
SELECT
      cognome, filiale
FROM Impiegati
```

| cognome | filiale |
|---|---|
| Neri | Napoli |
| Neri | Milano |
| Rossi | Roma |
| Rossi | Roma |

```
SELECT DISTINCT
      cognome, filiale
FROM Impiegati
```

| cognome | filiale |
|---|---|
| Neri | Napoli |
| Neri | Milano |
| Rossi | Roma |

# L'operazione di "join"

- I padri di persone che guadagnano più di venti milioni

```
SELECT DISTINCT padre
FROM Persone, Paternita
WHERE figlio = nome
      AND reddito > 20
```

Tabelle coinvolte

Condizione di join



# Join: passi logici

1. Eseguire il prodotto cartesiano tra le due tabelle
2. Nel risultato ottenuto, mantenere solo i record che soddisfano la condizione di Join (uguaglianza dei valori in campi corrispondenti di tabelle distinte)

- Es:
```
SELECT nome, regione
FROM T_A, T_B
WHERE T_A.citta = T_B.citta
```

T_A

| nome | citta |
|---|---|
| Franco | Napoli |
| Mara | Milano |

T_B

| citta | regione |
|---|---|
| Napoli | Campania |
| Roma | Lazio |

T_A × T_B

| nome | T_A.citta | T_B.citta | regione |
|---|---|---|---|
| Franco | Napoli | Napoli | Campania |
| Franco | Napoli | Roma | Lazio |
| Mara | Milano | Napoli | Campania |
| Mara | Milano | Roma | Lazio |

Result

| nome | regione |
|---|---|
| Franco | Campania |



# Join: altro esempio

- Le persone che guadagnano più dei rispettivi padri; mostrare nome, reddito e reddito del padre

```
SELECT f.nome, f.reddito, p.reddito
FROM Persone p, Paternita, Persone f
WHERE p.nome = padre AND
      figlio = f.nome AND
      f.reddito > p.reddito
```



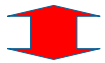
# SELECT con ridenominazione del risultato

```
SELECT figlio, f.reddito AS reddito,
       p.reddito AS redditoPadre

FROM Persone p, Paternita, Persone f

WHERE p.nome = padre AND figlio = f.nome
      AND f.reddito > p.reddito
```

# Join “esplicito”

- Nel DB con le tabelle Persona, Madre, Padre : mostrare padre e madre di ogni persona

```
SELECT paternita.figlio,padre, madre
FROM Maternita, Paternita
WHERE paternita.figlio = maternita.figlio
```



```
SELECT madre, paternita.figlio, padre
FROM maternita JOIN paternita ON
      paternita.figlio = maternita.figlio
```



# SELECT con join esplicito, sintassi

```
SELECT  …
FROM Tabella { … JOIN Tabella  ON CondDiJoin }, …
[ WHERE AltraCondizione ]
```



# Join “esplicito” : altro esempio

- Le persone che guadagnano più dei rispettivi padri; mostrare nome, reddito e reddito del padre

```
SELECT f.nome, f.reddito, p.reddito
FROM Persone p JOIN Paternita ON p.nome =
padre
     JOIN Persone f ON figlio = f.nome
WHERE f.reddito > p.reddito
```



# Join e valori NULL: "outer join"

- Padre e, se nota, madre di ogni persona

```
SELECT paternita.figlio, padre, madre
FROM paternita LEFT JOIN maternita
          on paternita.figlio = maternita.figlio
```

```
SELECT paternita.figlio, padre, madre
FROM paternita LEFT OUTER JOIN maternita
          on paternita.figlio = maternita.figlio
```

- OUTER è opzionale
- Il join visto in precedenza viene indicato anche come “inner join”

## Ordinamento del risultato

- Nome e reddito delle persone con meno di trenta anni in ordine alfabetico

```
SELECT nome, reddito
FROM Persone
WHERE eta < 30
ORDER BY nome
```



## Operatori aggregati

- Nelle espressioni della target list possiamo avere anche espressioni che calcolano valori a partire da insiemi di ennuple:
  - conteggio, minimo, massimo, media, totale
  - sintassi base (semplificata):

```
Funzione ( [ DISTINCT ] * )
Funzione ( [ DISTINCT ] Attributo )
```



## Operatori aggregati: COUNT

- Il numero di figli di Franco

```
SELECT COUNT(*) AS numFigliDiFranco
FROM Paternita
WHERE Padre = 'Franco'
```

- l'operatore aggregato (COUNT) viene applicato al risultato dell'interrogazione:

```
SELECT *
FROM Paternita
WHERE Padre = 'Franco'
```



## COUNT e valori nulli

```
SELECT COUNT(*) FROM Persone

SELECT COUNT(reddito) FROM Persone

SELECT COUNT(DISTINCT reddito) FROM Persone
```

Persone

| nome | eta | reddito |
|---|---|---|
| Andrea | 27 | 21 |
| Aldo | 25 | NULL |
| Maria | 55 | 21 |
| Anna | 50 | 35 |

# Altri operatori aggregati

- SUM, AVG, MAX, MIN
- Media dei redditi dei figli di Franco

```
SELECT AVG(reddito)
FROM Persone JOIN Paternita
     on nome=figlio
WHERE padre='Franco'
```



# Operatori aggregati e target list

- un'interrogazione scorretta:

```
SELECT nome, MAX(reddito)
FROM Persone
```

- di chi sarebbe il nome?
  La target list deve essere omogenea

```
SELECT MIN(eta), AVG(reddito)
FROM Persone
```



# Operatori aggregati e raggruppamenti

- Le funzioni possono essere applicate a partizioni delle relazioni
- Clausola GROUP BY:
  GROUP BY listaAttributi



# Operatori aggregati e raggruppamenti

- Il numero di figli di ciascun padre

```
SELECT padre, COUNT(*)  AS numFigli
FROM Paternita
GROUP BY padre
```

Paternita

| padre | figlio |
|---|---|
| Sergio | Franco |
| Luigi | Olga |
| Luigi | Filippo |
| Franco | Andrea |
| Franco | Aldo |

| padre | numFigli |
|---|---|
| Sergio | 1 |
| Luigi | 2 |
| Franco | 2 |

## Raggruppamenti e target list

scorretta

```
SELECT padre, AVG(f.reddito), p.reddito
FROM Persone f JOIN Paternita ON figlio = nome
        JOIN Persone p ON padre =p.nome
GROUP BY padre
```

corretta

```
SELECT padre, AVG(f.reddito), p.reddito
FROM Persone f JOIN Paternita ON figlio = nome
        JOIN Persone p ON padre =p.nome
GROUP BY padre, p.reddito
```



## Condizioni sui gruppi: HAVING

- I padri i cui figli sotto i 30 anni hanno un reddito medio maggiore di 20

```
SELECT padre, AVG(f.reddito)
FROM Persone f JOIN Paternita ON figlio = nome
WHERE eta < 30
GROUP BY padre
HAVING AVG(f.reddito) > 25
```



## Sintassi, riassumiamo

SelectSQL ::=

```
SELECT ListaAttributiOEspressioni
FROM ListaTabelle
[ WHERE CondizioniSemplici ]
[ GROUP BY ListaAttributiDiRaggruppamento ]
[ HAVING CondizioniAggregate ]
[ ORDER BY ListaAttributiDiOrdinamento ]
```



## Unione, intersezione e differenza

- La SELECT da sola non permette di fare unioni; serve un costrutto esplicito:

```
SELECT ...
UNION [ALL]
SELECT ...
```

- i duplicati vengono eliminati (a meno che si usi ALL); anche dalle proiezioni!

# Differenza

```
SELECT nome
FROM Impiegato
EXCEPT
SELECT cognome AS nome
FROM Impiegato
```

- vedremo che si può esprimere con select nidificate



# Intersezione

```
SELECT nome
   FROM Impiegato
   INTERSECT
   SELECT cognome AS nome
   FROM Impiegato
```

- equivale a

```
SELECT I.nome
FROM Impiegato I, Impiegato J
WHERE I.nome = J.cognome
```



# Interrogazioni nidificate

- le condizioni atomiche permettono anche
  - il confronto fra un attributo (o più, vedremo poi) e il risultato di una sottointerrogazione
  - quantificazioni esistenziali



- nome e reddito del padre di Mario

```
SELECT nome, reddito
FROM Persone, Paternita
WHERE nome = padre AND figlio = 'Franco'
```

```
SELECT nome, reddito
FROM Persone
WHERE nome =    (   SELECT padre
                    FROM Paternita
                    WHERE figlio = 'Franco')
```

## Interrogazioni nidificate, commenti

- La forma nidificata è “meno dichiarativa”, ma talvolta più leggibile (richiede meno variabili)
- La forma piana e quella nidificata possono essere combinate
- Le sottointerrogazioni non possono contenere operatori insiemistici (“l’unione si fa solo al livello esterno”); la limitazione non è significativa



- Nome e reddito dei padri di persone che guadagnano più di 20 milioni

```
SELECT DISTINCT P.nome, P.reddito
FROM Persone P, Paternita, Persone F
WHERE P.nome = padre and figlio = F.nome
      AND F.reddito > 20

SELECT nome, reddito
FROM Persone
WHERE nome IN (SEDLECT padre
               FROM Paternita
               WHERE figlio = ANY (SELECT nome
                                   FROM Persone
                                   WHERE reddito > 20))
```