SQL

- SQL (Structured Query Language) è un linguaggio di interrogazione per basi di dati relazionali

- Contiene funzionalità anche di Data Definition Language (DDL) e di Data Manipulation Language (DML)
  - DDL: definizione di domini, tabelle, viste, indici, autorizzazioni, vincoli
  - DML: operazioni di modifica, aggiornamento, inserimento, cancellazione dei dati

Storia e standard

Storia

- Prima proposta: SEQUEL, IBM Research 1974
- Prime implementazioni in SQL/DS (IBM) e Oracle (1981)

Standardizzazione ISO e ANSI

- La standardizzazione è stata cruciale per il successo di SQL
- Prima versione 1986 (SQL-1), rivista nel 1989 (SQL-89)
- Seconda versione nel 1992 (SQL-2 o SQL- 92)
- Terza versione nel 1999 (SQL-3 o SQL- 99)

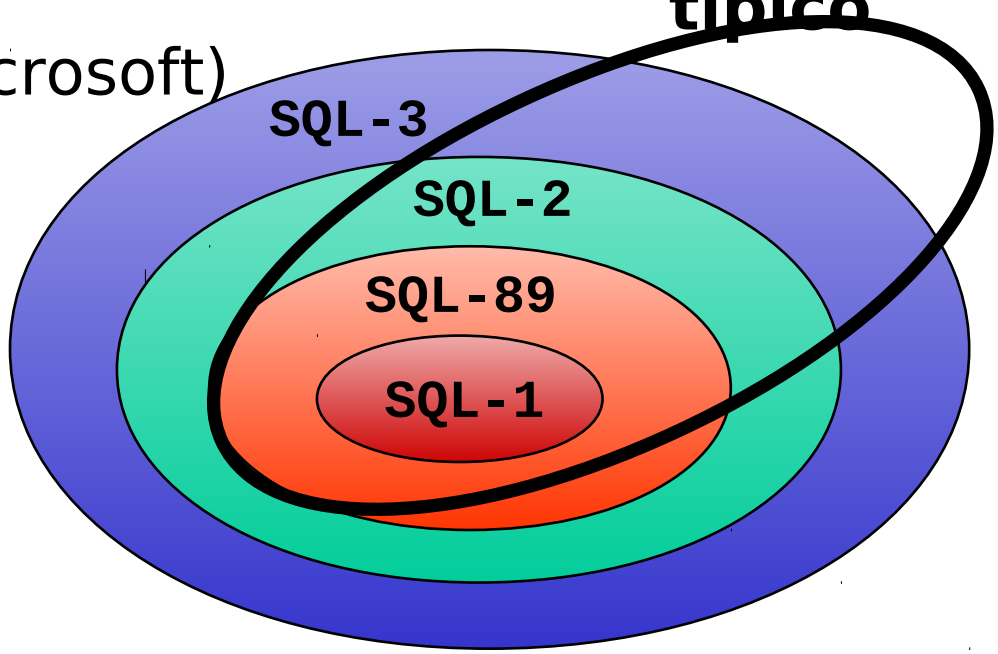
SQL-2

- È ricco e complesso e nessun sistema commerciale lo implementa in maniera completa
- Sono definiti 3 livelli di complessità che individuano dei sotto-standard che possono essere realizzati in un dato DBMS:
  - *Entry -* molto simile a SQL - 89,
  - *Intermediate*,
  - *Full*
- I sistemi commerciali spesso offrono funzionalità aggiuntive che non sono specificate nello standard e sono quindi dipendenti dal sistema usato

Implementazioni

Alcuni DBMS commerciali

- ORACLE
- DB2 (IBM)
- Access (Microsoft)
- MSSQL server (Microsoft)
- Informix
- Mysql
- **PostgreSQL**



Definizione di un database

- La definizione di un database avviene attraverso la definizione dello schema, ovvero dell'insieme di domini, tabelle, indici, viste, privilegi che lo costituiscono
- Un database contiene uno o più schema, che a loro volta contengono tabelle, domini, ecc.

Definizione di un database

- SQL mette a disposizione un apposito comando, CREATE SCHEMA, che consente di specificare quali sono i diritti degli utente sul DB e quali siano i suoi elementi componenti
- Alla creazione del database PostgreSQL automaticamente crea uno schema dal nome “public”

Creazione di un database

```
CREATE DATABASE name
    [ [ WITH ] [ OWNER [=] dbowner ]
            [ TEMPLATE [=] template ]
            [ ENCODING [=] encoding ]
            [ TABLESPACE [=] tablespace ]
            [ CONNECTION LIMIT [=] connlimit ] ]
```

```
CREATE DATABASE studente
CREATE DATABASE anagrafica WITH OWNER sarti
CREATE DATABASE biblioteca WITH OWNER universita
```

Diritti di accesso

- All’atto della creazione di un database l’utente creatore ne diventa il proprietario. Gli altri utenti possono leggere le informazioni ma non modificare il DB
- Per modificare i diritti di accesso è possibile utilizzare i comandi GRANT e REVOKE

GRANT e REVOKE

- Revoca dei diritti

  REVOKE [CREATE | CONNECT] ON DATABASE nomedatabase FROM [nome utente| public]

- Attribuzione diritti

  GRANT [CREATE | CONNECT] ON DATABASE nomedatabase TO [nome utente|public] <WITH GRANT OPTION>

Creazione di tabelle

<comando_di_creazione_tabella>::=
CREATE TABLE <nome_tabella>
(<definizione_elemento_tabella>,
…,<definizione_elemento_tabella>)

Dove:
<definizione_elemento_tabella>::=
<definizione_colonna>|<definizione_vincolo_di_tabella>

Quindi la creazione di una tabella avviene attraverso l'enumerazione delle colonne che la compongono.

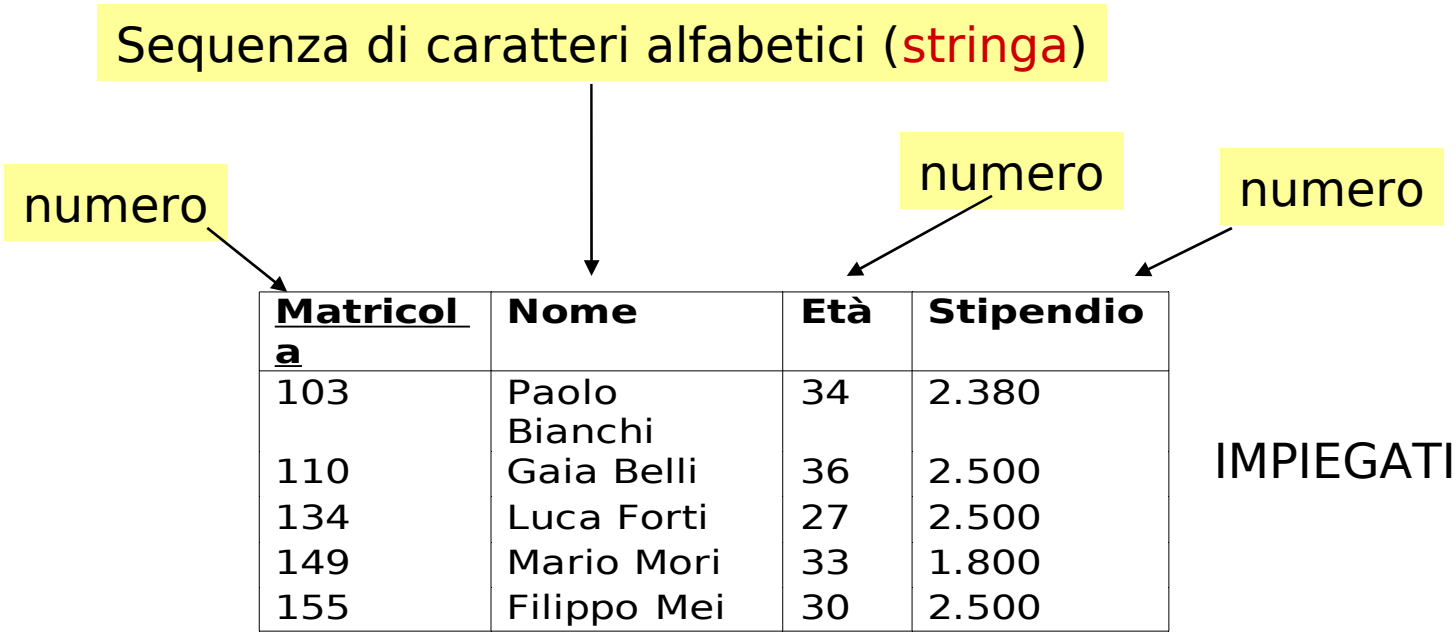
Definizione dei dati – 1/2

- Si possono usare 6 domini (tipi di dato) predefiniti:
  - Il tipo carattere
  - I tipi numerici esatti
  - I tipi numerici approssimati
  - Il tipo Date
  - Gli intervalli temporali

- I tipi di dato sono utilizzati per definire il tipo per gli attributi (colonne) delle relazioni, utilizzando la seguente sintassi:

```
<definizione_colonna> ::=
    <nome_colonna> <tipo_di_dati>
    [clausola_default]
    [definizione_vincolo_di_colonna]
```

Definizione dei dati – 2/2



Matricola	**Nome**	**Età**	**Stipendio**
103	Paolo Bianchi	34	2.380
110	Gaia Belli	36	2.500
134	Luca Forti	27	2.500
149	Mario Mori	33	1.800
155	Filippo Mei	30	2.500

IMPIEGATI

Il tipo carattere

- Rappresenta singoli caratteri alfanumerici oppure stringhe di lunghezza fissa o variabile

```
char
char(lunghezza)
varchar(lunghezza)
char varying (lunghezza)
```

Si definisce l'attributo `Nome` della relazione IMPIEGATI come sequenza dicaratteri di lunghezza massima 20

`Nome` **`char(20)`**

`Paolo_Bianchi_ _ _ _ _ _ _ _`

`Nome` **`varchar(20)`**

`Paolo_Bianchi`

Il tipo bit

- Corrisponde ad attributi che possono assumere solo due valori (0,1)
- Attributi di questo tipo (flag) indicano se l'oggetto rappresentato possiede o meno una certa proprietà

```
bit

bit(lunghezza)

bit varying (lunghezza)
```

Si definisce l'attributo `Lavoratore` nella relazione STUDENTI per indicare se lo studente è o meno lavoratore

```
Lavoratore bit
```

Tipi numerici esatti

- Rappresentano numeri interi o numeri decimali in virgola fissa (con un numero prefissato di decimali)

```
numeric    numeric(precisione) numeric(precisione,scala)

decimal    decimal(precisione) decimal(precisione,scala)
smallint

integer
```

Si definisce l'attributo `Eta` nella relazione IMPIEGATI

`Eta` **`decimal(2)`** Rappresenta tutti i numeri fra -99 e +99

Si definisce l'attributo `Cambio` nella relazione PAGAMENTO per indicare il valore del cambio del dollaro

`Cambio` **`numeric(5,4)`** Rappresenta tutti i numeri fra -9,9999 e +9,9999

Tipi numerici esatti e precisione

- La precisione dei tipi numerici esatti (il numero massimo e minimo rappresentabili) dipende dal DBMS
- **PostgreSQL**: smallint (signed,2byte), integer (signed,4byte), decimal e numeric consentono di scegliere la precisione desiderata.

- Il tipo dati DECIMAL consente, in generale, una precisione maggiore del tipo dati NUMERIC

- Il tipo dati INTEGER consente una precisione maggiore del tipo dati SMALLINT

Tipi numerici approssimati

- Sono utili per rappresentare ad esempio grandezze fisiche (rappresentazione in virgola mobile)
- Il parametro `precisione` definisce la lunghezza della mantissa

float	**float(***precisione***)**
real	
double precision	

Si definisce l'attributo `Massa` nella relazione ASTEROIDI

```
Massa real
```

0,17E16 = $1{,}7\ 10^{15}$

Precisione (double precision) > Precisione (real) > Precisione (float)

Date

- Permettono di rappresentare istanti di tempo

```
date
time        time(precisione)
timestamp   timestamp(precisione)
```

- Ciascuno di questi domini è decomponibile in un insieme di campi (anno, mese, giorno, ora, minuti, secondi)

```
DataDiNascita date
OraDiConsegna time
Arrivo timestamp
```

18/09/99

19.24.16

18/09/00 21.15.20

```
year(Arrivo) = 2000
```

```
minute(Arrivo) = 15
```

Intervalli temporali

- Permette di rappresentare intervalli di tempo come durate di eventi

```
interval PrimaUnitàDiTempo
interval PrimaUnitàDiTempo to UltimaUnitàDitempo
```

Esempi

Anzianità di servizio in anni e mesi

```
AnzianitaServizio interval year to month
```

Tempo di consegna in giorni ed ore

```
TempoConsegna interval day to hour
```

BLOB e CLOB

- Binary Large Object (BLOB) e Character Large Object (CLOB) permettono di includere direttamente nel database file molto grandi

- BLOB e CLOB sono tipi definiti solo in SQL-3, ma realizzati in diversi DBMS commerciali

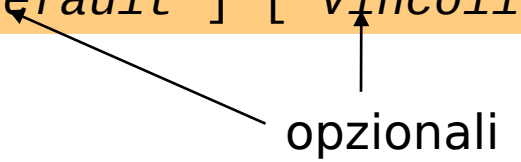
Una tabella con figure e documenti

```
CREATE TABLE quadri {
    nome VARCHAR(50),
    nomeAuthore VARCHAR(30),
    fotografia BLOB(10M),
    descrizione CLOB(100k)
}
```

Tipi definiti dall'utente

- Simile alla definizione di tipi nei linguaggi di programmazione ma non permette tipi strutturati
- Semplifica la scrittura del codice SQL e rende più semplice la modifica

```
CREATE DOMAIN NomeDominio AS DominioElementare
        [ Volore di default ] [ Vincoli ]
```



Esempi

Un prezzo in Euro

```
CREATE DOMAIN prezzo AS decimal(9,2) DEFAULT 0.00
```

Valori di default

- I valori di default specificano cosa deve essere assegnato all'attributo quando non si indica un valore esplicitamente

```
DEFAULT (valoreGenerico | user | null)
```

oppure

Esempio

```
costoColazione NUMERIC(5) DEFAULT 3000
```

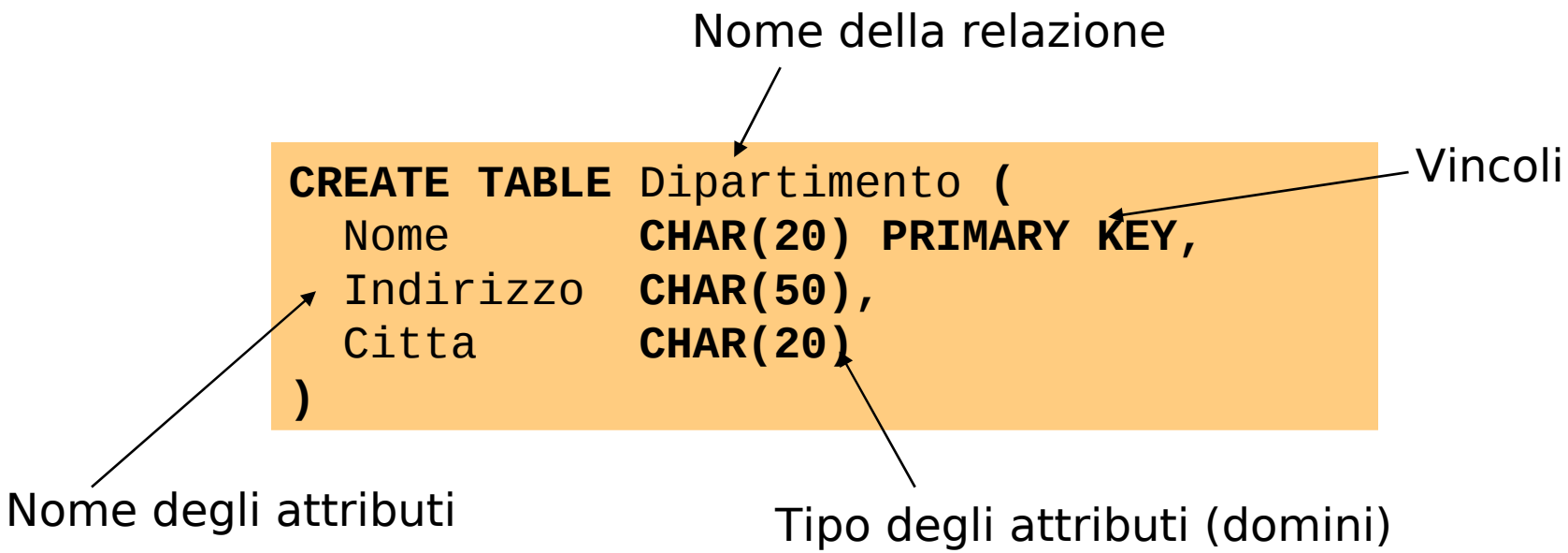
```
DatoInseritoDa  VARCHAR(8) DEFAULT null
```

Alcuni DBMS ammettono espressioni più complesse

```
lordo  NUMERIC(9) DEFAULT (netto+iva)
```

Definizione di tabella – Esempio 1

- Una tabella è costituita da un insieme ordinato di attributi e di vincoli



Definizione di Tabella – Esempio 2

```
CREATE TABLE Studenti (
  Matricola      CHAR(9) PRIMARY KEY,
  Cognome        VARCHAR(50),
  Nome           VARCHAR(50),
  DataDiNascita  DATE,
  Lavoratore     BIT DEFAULT NULL
)
```

Se non si specifica un valore si inserisce per default il valore NULL

Studenti

Matricola	Cognome	Nome	Data di nascita	Lavoratore

Definizione di Tabella – Esempio 3

```
CREATE TABLE Veicoli
    (Targa CHAR(10),
    Cod_Modello CHAR(3),
    Cod_Categoria CHAR(2),
    Cilindrata NUMERIC(4),
    Cod_Combustibile CHAR(2),
    Cavalli_Fiscali NUMERIC(3),
    Velocita NUMERIC(3),
    Posti NUMERIC(2) DEFAULT 5,
    Immatricolazione DATE)
```

Inserimento – Esempio 1

```
CREATE TABLE Studenti (
  Matricola      CHAR(9) PRIMARY KEY,
  Cognome        VARCHAR(50),
  Nome           VARCHAR(50),
  DataDiNascita  DATE,
  Lavoratore     BIT DEFAULT NULL
)
```

```
INSERT INTO Studenti VALUES (
    '680100465', 'Castelli', 'Ilaria', '1983-06-07', '0'
)
```

```
INSERT INTO Studenti VALUES (
    '680100467', 'Rossi', 'Antonio', '1981-12-07'
)
```

Inserimento – Esempio 1

	matricola [PK] character(9)	cognome character var	nome character var	datadinascita date	lavoratore bit(1)
1	680100465	Castelli	Ilaria	1983-06-07	0
2	680100467	Rossi	Antonio	1981-12-07	
*					

Inserimento – Esempio 1

- Una sintassi alternativa consente di elencare i nomi dei campi esplicitamente...

```
INSERT INTO Studenti
    (Matricola, Cognome, Nome, DataDiNascita, Lavoratore)
VALUES ('680100470', 'Verdi', 'Marco', '1980-11-04', '1' )
```

...anche in ordine diverso o con omissioni

```
INSERT INTO Studenti
    (Matricola, Nome, Cognome, Lavoratore)
VALUES ('680100473', 'Lorenzo', 'Neri', '0' )
```

Inserimento – Esempio 1

	matricola [PK] character(9)	cognome character var	nome character var	datadinascita date	lavoratore bit(1)
1	680100465	Castelli	Ilaria	1983-06-07	0
2	680100466	Rossi	Antonio	1981-12-07	
3	680100470	Verdi	Marco	1980-11-04	1
4	680100473	Neri	Lorenzo		0
*					

Vincoli intrarelazionali semplici

- Operano su un solo attributo della relazione

NOT NULL L'attributo non può assumere il valore NULL

UNIQUE Non possono esistere due righe che hanno gli stessi valori per l'attributo o insieme di attributi specificati

PRIMARY KEY Identifica la chiave primaria. Può essere specificato per una sola colonna oppure come vincolo di tabella

Esempi di vincoli - 1

Column constraint

```
CREATE TABLE Impiegato (
  Cognome        VARCHAR(50) NOT NULL UNIQUE,
  Nome           VARCHAR(50) NOT NULL UNIQUE,
  Dipartimento   INTEGER,
  Stipendio      INTEGER DEFAULT 0
)
```

Non sono la stessa cosa!

Table constraint

```
create table Impiegato (
  Cognome        VARCHAR(50) NOT NULL,
  Nome           varchar(50) NOT NULL,
  Dipartimento   INTEGER,
  Stipendio      INTEGER DEFAULT 0,
  UNIQUE(Cognome,Nome)
)
```

Esempi di vincoli - 2

```
CREATE TABLE Veicoli (
Targa  CHAR(10) PRIMARY KEY,

       ...

)
```

```
CREATE TABLE Veicoli (
Targa  CHAR(10) PRIMARY KEY,

Cod_Proprietario CHAR(5) PRIMARY KEY,

       ...

)
```

```
CREATE TABLE Veicoli (
Targa  CHAR(10),

Cod_Proprietario CHAR(5),

       ...

PRIMARY KEY (Targa,Cod_Proprietario)

)
```

Esempi di vincoli - 3

- Tecnicamente, il vincolo PRIMARY KEY è una combinazione dei vincoli UNIQUE e NOT NULL

```
CREATE TABLE prodotti (
  id INTEGER UNIQUE NOT NULL,
  nome VARCHAR(80),
  prezzo NUMERIC(5)
)
```

```
CREATE TABLE prodotti (
  id INTEGER PRIMARY KEY,
  nome VARCHAR(80),
  prezzo NUMERIC(5)
)
```

PostgreSQL - Serial

- Non è propriamente un tipo di dato
- È comodo per la creazione di attributi identificativi (unici)

```
CREATE TABLE prodotti (
  id SERIAL PRIMARY KEY,
  nome VARCHAR(80),
  prezzo NUMERIC(5)
)
```

Il valore di default dell'attributo è assegnato da un generatore di sequenza

```
INSERT INTO prodotti (id, nome, prezzo)
    VALUES (DEFAULT, 'penna', '2.00')
```

```
INSERT INTO prodotti (nome, prezzo)
    VALUES ('matita', '1.00')
```

Vincoli di integrità referenziale

- Creano un legame fra i valori dell'attributo della tabella corrente (tabella interna) e i valori di un attributo di un'altra tabella (tabella esterna)
- È richiesto che il valore dell'attributo, se non è nullo, sia presente tra i valori dell'attributo di riferimento della tabella esterna
- L'attributo della tabella esterna a cui si fa riferimento deve essere dichiarato unique o essere chiave
- Si possono utilizzare i costrutti:
  - references()
  - foreign key () references()

references

```
CREATE TABLE Impiegato (
  Matricola     CHAR(6) PRIMARY KEY,
  Cognome       VARCHAR(50) NOT NULL,
  Nome          VARCHAR(50) NOT NULL,
  Diparti       CHAR(15)
                REFERENCES Dipartimento(NomeDip),
  Stipendio     INTEGER DEFAULT 0,
  UNIQUE(Cognome,Nome)
)
```

Il campo `Dipart` può assumere solo i valori che compaiono nel campo `NomeDip` della tabella `Dipartimento`.
References permette di specificare vincoli di colonna

foreign key () references()

```
CREATE TABLE Impiegato (
  Matricola     CHAR(6) PRIMARY KEY,
  Cognome       VARCHAR(50) NOT NULL,
  Nome          VARCHAR(50) NOT NULL,
  Diparti       CHAR(15)
                REFERENCES Dipartimento(NomeDip),
  Stipendio     INTEGER DEFAULT 0,
  UNIQUE(Cognome,Nome),
  FOREIGN KEY (Nome,Cognome)
            REFERENCES Anagrafica(Nome,Cognome)
)
```

La coppia di campi (`Nome,Cognome`) può assumere solo le coppie di valori che compaiono nei campi (`Nome,Cognome`) della tabella `Anagrafica`. **Foreign key() references() permette di specificare un vincolo di tabella**

Cosa accade quando si viola un vincolo

- Se la violazione è causata da un accesso alla tabella **interna** il comando è **sempre rifiutato**
- Se la violazione è causata da un accesso alla tabella **esterna** si **può specificare** come e se la modifica si propaga
  - **cascade:** si ripete l'operazione della tabella esterna
  - **set null:** l'attributo viene impostato a **null**
  - **set default:** l'attributo viene impostato al valore di default
  - **no action:** nessuna azione
- **Se non si specifica niente** il comando è rifiutato

Come si specificano le operazioni da effettuare dopo la violazione di unvincolo?

- Le azioni che possono causare una violazione sono:
  - Update
  - Delete

```
CREATE TABLE Impiegato (
  Matricola     CHAR(6) PRIMARY KEY,
  Cognome       VARCHAR(50) NOT NULL,
  Nome          VARCHAR(50) NOT NULL,
  Diparti       CHAR(15)
                REFERENCES Dipartimento(NomeDip),
  Stipendio     INTEGER DEFAULT 0,
  UNIQUE(Cognome,Nome),
  FOREIGN KEY (Nome,Cognome)
  REFERENCES Anagrafica(Nome,Cognome)

  ON DELETE CASCADE, ON UPDATE NO ACTION
)
```

Vincoli di controllo

- I vincoli di controllo sono utilizzati per verificare generiche condizioni sui valori di una colonna
- La parola chiave che permette di specificare un vincolo di controllo è CHECK

```
CREATE TABLE Veicoli (
    Targa CHAR(10) PRIMARY KEY,
    Cilindrata NUMERIC(4) CHECK(Cilindrata < 4000),
    ...)
```

Vincoli di controllo

- La definizione del vincolo è effettuata dopo il tipo di dato (come i valori di DEFAULT)
- Valori di DEFAULT e vincoli CHECK possono essere scambiati di ordine

```
Cilindrata NUMERIC(4) DEFAULT 1000 CHECK(Cilindrata < 4000)
```

```
Cilindrata NUMERIC(4) CHECK(Cilindrata < 4000) DEFAULT 1000
```

Vincoli di controllo

- Possono riferirsi a colonne diverse della stessa tabella

```
CREATE TABLE prodotti (
    id integer PRIMARY KEY,
    nome VARCHAR(80),
    prezzo NUMERIC(5) CHECK (prezzo > 0),
    prezzo_scont NUMERIC(5) CHECK (prezzo_scont > 0),
    CHECK (prezzo > prezzo_scont)
)
```

Vincoli – visti da PostgreSQL

- pgAdmin3 consente di visualizzare il codice SQL che ha generato l'oggetto correntemente selezionato

```
CREATE TABLE veicoli
(
  targa character(7) NOT NULL,
  cilindrata numeric(4,0) DEFAULT 1000,
  CONSTRAINT veicoli_pkey PRIMARY KEY (targa),
  CONSTRAINT veicoli_cilindrata_check CHECK (cilindrata < 4000::numeric)
)
```

- Ad ogni vincolo viene attribuito automaticamente un nome, visualizzabile dopo la parola chiave CONSTRAINT

Vincoli – visti da PostgreSQL

- È possibile esplicitare il nome da dare ai vincoli

```
CREATE TABLE veicoli1 (
targa CHAR(7)
  CONSTRAINT my_pk_constraint PRIMARY KEY,
cilindrata NUMERIC(4) DEFAULT 1000
  CONSTRAINT my_cilindrata_constraint CHECK (cilindrata < 4000)
)
```

```
CREATE TABLE veicoli1
(
  targa character(7) NOT NULL,
  cilindrata numeric(4,0) DEFAULT 1000,
  CONSTRAINT my_pk_constraint PRIMARY KEY (targa),
  CONSTRAINT my_cilindrata_constraint CHECK (cilindrata < 4000::numeric)
)
```

Indici

- Servono per velocizzare le interrogazioni a spese di un incremento della complessità degli aggiornamenti
- Si introduce un ordinamento sugli attributi specificati
- Non è una caratteristica standard di SQL

Crea un indice per effettuare ricerche più veloci utilizzando gli attributi `Nome` e `Cognome` della tabella `Studenti`

```
create index NomeCognome on Studenti(Cognome,Nome)
```

Crea un indice secondo un ordine discendente

```
create index NomeCognome on Studenti(Cognome DESC,Nome DESC)
```

Modifica dello schema

- Esistono due comandi per modificare i componenti dello schema: **drop** e **alter**
- Il comando drop permette di rimuovere le componenti eventualmente eliminando anche le componenti dipendenti

Esempi

Elimina la tabella dipendenti, il suo contenuto e le viste connesse

```
DROP TABLE dipendenti CASCADE
```

Elimina la tabella dipendenti solo se è vuota e non ci sono oggetti connessi

```
DROP TABLE dipendenti RESTRICT
```

Modifica dello schema

- Il comando ALTER permette di modificare le componenti: inserimento/ rimozione colonne delle tabelle, inserimento/rimozione vincoli ....
- Le modifiche possono generare errori

Esempi

Elimina/inserisci colonne

```
ALTER TABLE dipendenti ADD COLUMN stipendio NUMERIC(9)
ALTER TABLE dipendenti DROP COLUMN stipendio
```

I vincoli di tabella legati alla colonna vengono a loro volta eliminati

Se altre tabelle hanno come FOREIGN KEY la colonna?

```
ALTER TABLE dipendenti DROP COLUMN stipendio CASCADE
```

Modifica dello schema

Alcuni database permettono modifiche agli attributi

```
ALTER TABLE dipendenti ALTER COLUMN stipendio
   NUMERIC(10) NOT NULL
```

Aggiunta di vincoli

```
ALTER TABLE dipendenti CHECK (stipendio > 0)
```

```
ALTER TABLE dipendenti

   ADD FOREIGN KEY (Nome, Cognome)

   REFERENCES anagrafica (Nome, Cognome)
```

Interrogazioni semplici

- Le interrogazioni esprimono **cosa** si cerca e **non come** si cerca
- Il come è prodotto dall'interprete SQL del DBMS che traduce l'interrogazione SQL nelle operazioni interne sui dati che producono la risposta
- L'utente non deve quindi preoccuparsi di come si trova il risultato
- Le operazioni di interrogazione vengono specificate con l'istruzione **`select`**

select

Matricola	Cognome	Nome	Data di nascita
A80198760	Bianchi	Anna	22/03/1967
A80293450	Rossi	Andrea	13/04/1968
A80198330	Neri	Luca	04/08/1970
A80295640	Rossi	Lorenzo	25/02/1969
A80197456	Melli	Mara	17/10/1966

Studenti

Campi considerati

```
SELECT Matricola AS NumeroMatricola
FROM Studenti
WHERE Cognome = 'Rossi'
```

Tabelle da cui si selezionano

Condizioni

Nuovo nome per l'attributo

NumeroMatricola
A80293450
A80295640

Risultato: una relazione!

select *

Matricola	Cognome	Nome	Data di nascita
A80198760	Bianchi	Anna	22/03/1967
A80293450	Rossi	Andrea	13/04/1968
A80198330	Neri	Luca	04/08/1970
A80295640	Rossi	Lorenzo	25/02/1969
A80197456	Melli	Mara	17/10/1966

Studenti

Tutti i campi sono considerati

```
SELECT *
FROM Studenti
WHERE Cognome = 'Rossi'
```

Matricola	Cognome	Nome	Data di nascita
A80293450	Rossi	Andrea	13/04/1968
A80295640	Rossi	Lorenzo	25/02/1969

select (espressioni)

Prodotto	Cliente	Quantita	PrezzoUnitario
Chiodo 35mm	Bianchi	100	15
Vite 45mm	Rossi	50	20
Dado 5mm	Neri	200	8
Vite 8mm	Rossi	20	25
Bulloni 5cm	Melli	40	50

```
SELECT Prodotto, Quantita*PrezzoUnitario AS PrezzoTotale
FROM Ordini
WHERE Cliente = ‘Rossi’
```

Prodotto	PrezzoTotale
Vite 45mm	1000
Vite 8mm	500

Espressione che combina più attributi

Funzioni

SQL mette a disposizione **funzioni predefinite**

- stringhe (SUBSTR, TRIM, LENGTH, | ,....)
- date, intervalli (+, -, DATE, DAYOFWEEK,...)
- matematiche (* , + , - , / , TAN, SQRT, SIN,...)
- informazioni di sistema (USER, CURRENT_DATE,...)

Esempi

Nome e cognome degli studenti

```
SELECT  cognome | ‘ ‘ | nome AS nomeCompleto
FROM studenti
```

Matricola	Cognome	Nome
A80198760	Bianchi	Anna
A80293450	Rossi	Andrea
A80198330	Neri	Luca
A80295640	Rossi	Lorenzo
A80197456	Melli	Mara

NomeCompleto
Bianchi Anna
Rossi Andrea
Neri Luca
Rossi Lorenzo
Melli Mara

Esempi

Mese di nascita degli studenti

```
SELECT  monthname(“data di nascita”) AS meseDiNascita
FROM studenti
```

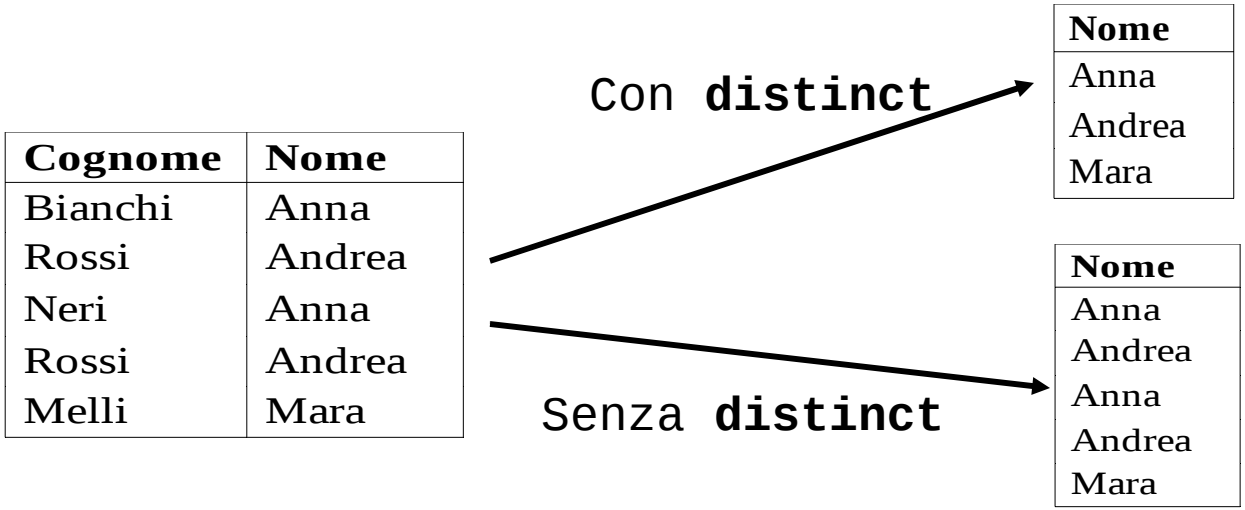
Cognome	Nome	Data di nascita
Bianchi	Anna	22/03/1967
Rossi	Andrea	13/04/1968
Neri	Luca	04/08/1970
Rossi	Lorenzo	25/02/1969
Melli	Mara	17/10/1966

meseDiNascita
marzo
aprile
agosto
febbraio
ottobre

Valori distinti

- A differenza del calcolo o dell'algebra relazionale, i risultati delle interrogazioni SQL contengono duplicati
- I duplicati possono essere rimossi usando **distinct**

```
SELECT DISTINCT nome FROM studenti
```



Ordinamento

- La clausola **order by** definisce l'ordine con cui i record devono essere restituiti
- Senza la clausola, l'ordinamento dipende dal sistema, e potrebbe anche cambiare ad ogni interrogazione

Studenti ordinati per cognome (ordine inverso) e nome

```
SELECT cognome, nome FROM studenti
ORDER BY cognome DESC, nome ASC
```

Cognome	Nome
Rossi	Andrea
Rossi	Lorenzo
Neri	Luca
Melli	Mara
Bianchi	Anna

Select ... where

- La clausola **where** ammette come argomento una condizione costruita da
  - condizioni semplici (ad esempio confronti con =, <> ,<,>,<=,>= fra costanti o valori degli attributi)
  - espressioni ottenute usando gli operatori **and**, **or** e **not**
- Si possono usare le funzioni predefinite
- Vengono selezionate solo le righe per cui la condizione è vera

Esempi

Ricerca degli impiegati con reddito maggiore di 100 milioni e età non compresa fra 40 e 50 anni

```
SELECT *
FROM impiegati
WHERE reddito >= 100 AND NOT (eta <=50 AND eta>=40)
```

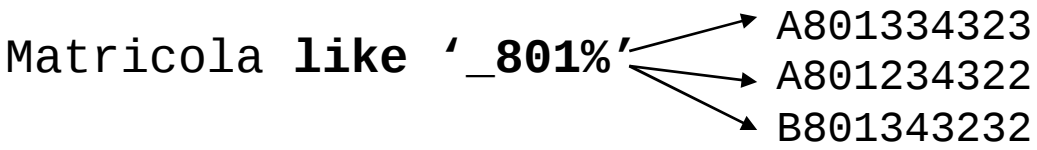
La precedenza fra gli operatori **and** e **or** non è definita dallo standard

Ricerca degli studenti nati di domenica

```
SELECT *
FROM studenti
WHERE dayofweek(“data di nascita”) = 1
```

Ricerca di stringhe

- Per il confronto di stringhe si può usare l'operatore **like** che supporta i caratteri speciali _ (qualsiasi carattere) e **%** (qualsiasi sequenza di caratteri)



Ricerca degli studenti il cui indirizzo inizia per "via"

```
SELECT *
FROM studenti
WHERE indirizzo LIKE 'via%'
```

Valori nulli

I valori nulli possono significare

- valore non conosciuto
- valore non applicabile
- non si sà se il valore è sconosciuto o non applicabile

- SQL-89 usa una logica a due valori
  - un confronto con **null** produce **FALSE**
- SQL-2 usa una logica a tre livelli
  - un paragone con **null** produce **UNKNOWN**

select ..... from

- Per accedere a righe appartenenti a più di una tabella, si usa come argomento della clausola **from** la lista della tabelle a cui si vuole accedere
- Sul prodotto cartesiano delle tabelle elencate vengono applicate le condizioni della clausola **where**
- Se si vuole effettuare un **join** occorre scrivere in modo esplicito le condizioni che esprimono il legame tra le diverse tabelle

Select e prodotto cartesiano

```
SELECT  * FROM tab1, tab2 WHERE attr1='A'
```

tab1

Attr1	Attr2
A	B
B	A

tab2

Attr3	Attr4
C	B
D	B

Prodotto cartesiano

Attr1	Attr2	Attr3	Attr4
A	B	C	B
B	A	C	B
A	B	D	B
B	A	D	B

Selezione

Attr1	Attr2	Attr3	Attr4
A	B	C	B
A	B	D	B

Select e prodotto cartesiano

```
SELECT  * FROM tab1, tab2 WHERE attr1='A'
```

è equivalente a

```
SELECT  * FROM tab1 CROSS JOIN tab2 WHERE attr1='A'
```

- Scrivendo **FROM tab1, tab2** la prima operazione che viene eseguita è il prodotto cartesiano tra le tabelle.
- Il prodotto cartesiano viene poi "filtrato" in base alle clausole **WHERE, GROUP BY, HAVING**

Il join con select

Matricola	Cognome	Nome	Data di nascita
A80198760	Bianchi	Anna	22/03/1967
A80293450	Rossi	Andrea	13/04/1968
A80295640	Felici	Lorenzo	25/02/1969

Studenti

Studente	Voto	Corso
A80198760	28	Analisi I
A80293450	30	Basi di Dati
A80295640	27	Analisi I
A80198760	30L	Fisica I
A80293450	21	Chimica

Esami

Attributi di tabelle

```
SELECT Studenti.Cognome,Studenti.Nome,Esami.Voto
FROM Studenti,Esami
WHERE (Studenti.Matricola = Esami.Studente) AND
      (Esami.Corso = 'Analisi I')
```

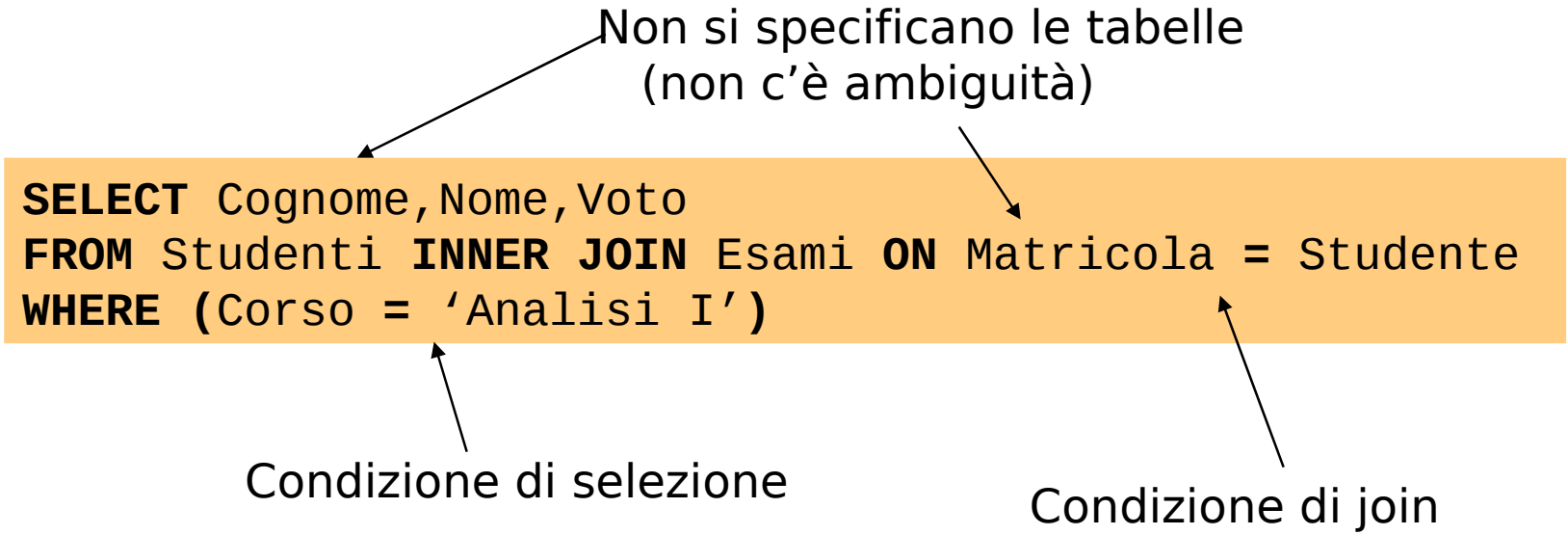
prodotto cartesiano

Condizione di join

Cognome	Nome	Voto
Bianchi	Anna	28
Felici	Lorenzo	27

join interno

- Modo alternativo per indicare il join di due tabelle che distingue le condizioni di join da quelle di selezione delle righe



- Per ogni riga R1 di T1, la tabella risultato ha una riga per ogni riga di T2 che soddisfa la condizione di join con R1

join esterno

- Quando si fa il join può capitare che alcune delle righe di una tabella non vengano selezionate perché non c'è nessuna riga dell'altra tabella che soddisfa la condizione di join
- Il **join esterno** esegue il join interno mantenendo però tutte le righe che fanno parte di una o di entrambe le tabelle coinvolte come segue:
  - **OUTER LEFT** sono aggiunte le righe della relazione T1 (a sinistra del join) che non hanno righe corrispondenti nella tabella di destra T2
  - **OUTER RIGHT** sono aggiunte le righe della relazione T2 (a destra del join) che non hanno righe corrispondenti nella tabella di sinistra T1
  - **OUTER FULL** compaiono tutte le righe delle due tabelle

outer left join

Guidatore

Cognome	Nome	NroPatente
Bianchi	Anna	VR 2030020Y
Rossi	Andrea	PZ 1012436B
Felici	Lorenzo	AP 4544442R

Automobile

Targa	Marca	Modello	NroPatente
AB574WW	Fiat	Punto	VR2030020Y
AA652FF	Renault	Clio	VR2030020Y
BJ747XX	Ford	Focus	PZ1012436B
BB421JJ	Renault	Megane	MI2020030U

Associa un altro nome alla tabella (alias)

```
SELECT *
FROM Guidatore G LEFT JOIN Automobile A
        ON (G.NroPatente = A.NroPatente)
```

ci sarebbe stata ambiguità

Cognome	Nome	NroPatente	Targa	Marca	Modello
Bianchi	Anna	VR 2030020Y	AB574WW	Fiat	Punto
Bianchi	Anna	VR 2030020Y	AA652FF	Renault	Clio
Rossi	Andrea	PZ 1012436B	BJ747XX	Ford	Focus
Felici	Lorenzo	AP 4544442R	NULL	NULL	NULL

inner/outer

Le parole chiave INNER e OUTER sono opzionali:

- INNER è il default.

T1 JOIN T2 ON attributo1 = attributo2
implicitamente significa inner join.

- LEFT, RIGHT, FULL, implicano outer join.

Join - riassunto

T1

num	nome
1	a
2	b
3	c

T2

num	valore
1	xxx
3	yyy
5	zzz

```
SELECT  * FROM T1 INNER JOIN T2 ON T1.num = T2.num
```

```
SELECT  * FROM T1, T2 WHERE T1.num = T2.num
```

num	nome	num	valore
1	a	1	xxx
3	c	3	yyy

Join - riassunto

```
SELECT  * FROM T1 CROSS JOIN T2
```

```
SELECT  * FROM T1, T2
```

num	nome	num	valore
1	a	1	xxx
1	a	3	yyy
1	a	5	zzz
2	b	1	xxx
2	b	3	yyy
2	b	5	zzz
3	c	1	xxx
3	c	3	yyy
3	c	5	zzz

Join - riassunto

```
SELECT  * FROM T1 LEFT JOIN T2 ON T1.num = T2.num
```

num	nome	num	valore
1	a	1	xxx
2	b		
3	c	3	yyy

```
SELECT  * FROM T1 RIGHT JOIN T2 ON T1.num = T2.num
```

num	nome	num	valore
1	a	1	xxx
3	c	3	yyy
		5	zzz

Join - riassunto

```
SELECT  * FROM T1 FULL JOIN T2 ON T1.num = T2.num
```

num	nome	num	valore
1	a	1	xxx
2	b		
3	c	3	yyy
		5	zzz

Uso di variabili

- Permette di far riferimento a più esemplari della stessa tabella

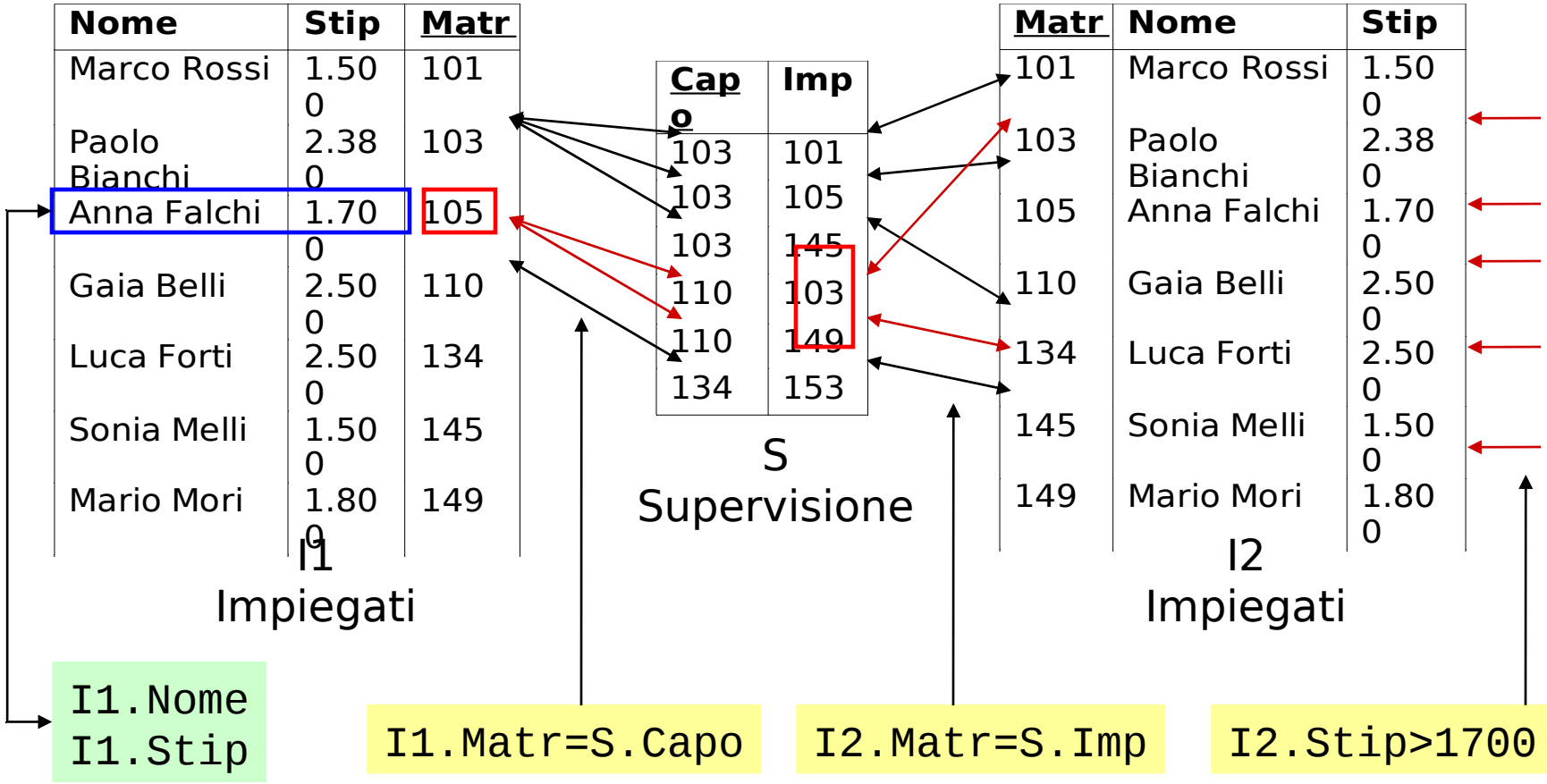
Trovare nome e stipendio dei capi degli impiegati che guadagnano più di 1.700 Euro

```
SELECT I1.Nome,I1.Stip
FROM Impiegati I1, Supervisione S, Impiegati I2
WHERE I1.Matr = S.Capo AND
      I2.Matr = S.Imp AND
      I2.Stip > 1700
```

Copia I1: usata per ricavare le informazioni sul capo

Copia I2: usata per riferirsi agli impiegati

Esempio di uso di variabili



Operatori aggregati

- Permettono di costruire interrogazioni che coinvolgono più tuple contemporaneamente
- Ad esempio: contare il numero di studenti che hanno superato un esame, trovare il massimo voto preso da uno studente ad un esame, calcolare la media di uno studente
- Si definiscono gli operatori aggregati che combinano fra loro più tuple

count	conteggio righe
sum	somma
max	massimo
min	minimo
avg	media

Conteggio, massimo, media...

Studente	Voto	Corso
A80198760	28	01
A80293450	30	04
A80198760	27	03
A80293450	25	03
A80293450	21	05

```
SELECT count(*) FROM esami
WHERE Studente = 'A80293450'
```

```
SELECT max(Voto) FROM esami
WHERE Studente = 'A80293450'
```

```
SELECT avg(Voto) FROM esami
WHERE Studente = 'A80293450'
```

Informazioni sui voti dello studente A80293450

```
SELECT count(distinct Studente)
FROM esami
```

Conteggio di quanti studenti hanno fatto almeno un esame

Interrogazioni con raggruppamento

- Il raggruppamento permette di applicare gli operatori aggregati distintamente a sottoinsiemi di righe
  - Prima si esegue la query senza **group by** e operatori aggregati
  - Poi si esegue il raggruppamento in accordo agli attributi in **group by**
  - Infine si calcolano gli operatori aggregati

Calcolare la media di ogni studente

```
SELECT Studente,avg(Voto) AS Media
FROM Esami
GROUP BY Studente
```

Interrogazioni con raggruppamento II

Calcolare la media di ogni studente

Studente	Voto	Corso
A80198760	28	01
A80293450	30	04
A80198760	27	03
A80293450	25	03
A80293450	21	05

```
SELECT Studente,avg(Voto) AS Media
FROM Esami
GROUP BY Studente
```

Studente	Voto
A80198760	28
A80198760	27
A80293450	30
A80293450	25
A80293450	21

Studente	Media
A80198760	27.500
A80293450	25.333

Restrizioni della sintassi

- Nelle interrogazioni con raggruppamento, nella clausola **select** possono comparire solo gli attributi che compaiono anche nella clausola **group by**, oppure operatori aggregati

Una interrogazione scorretta

```
SELECT cognome, nome, “data di nascita”
FROM studenti LEFT JOIN Esami ON Matricola = Studente
GROUP BY cognome, nome
```

Una interrogazione corretta

```
SELECT cognome, nome, “data di nascita”
FROM studenti LEFT JOIN Esami ON Matricola = Studente
GROUP BY cognome, nome,”data di nascita”
```

Restrizioni della sintassi

```
=> SELECT * FROM test1;
 x | y
---+---
 a | 3
 c | 2
 b | 5
 a | 1
```

```
=> SELECT x FROM test1 GROUP BY x;
 x
---
 a
 b
 c
```

- Non si poteva fare “SELECT * FROM test1 GROUP BY x” perché non esiste nessun valore nella colonna y che potrebbe essere associato ad ogni gruppo.

- È invece possibile usare operatori aggregati

```
=> SELECT x, sum(y) FROM test1 GROUP BY x;
 x | sum
---+-----
 a |   4
 b |   5
 c |   2
```

GROUP BY e DISTINCT

- Se si usa GROUP BY senza nessun operatore aggregato, ciò che viene restituito è l'insieme dei valori distinti presenti nella colonna.
- Lo stesso risultato può essere ottenuto con DISTINCT

```
SELECT x FROM test1 GROUP BY x
```

```
SELECT DISTINCT x FROM test1
```

Interrogazioni con raggruppamento e selezione

- Si possono selezionare solo alcuni raggruppamenti in base a condizioni di tipo aggregato

Trovare la media degli studenti che hanno sostenuto almeno 3 esami

Studente	Voto	Corso
A80198760	28	01
A80293450	30	04
A80198760	27	03
A80293450	25	03
A80293450	21	05

```
SELECT Studente,avg(Voto) AS Media
FROM Esami
GROUP BY Studente
HAVING count(*)>=3
```

Studente	Voto
A80198760	28
A80198760	27
A80293450	30
A80293450	25
A80293450	21

Studente	Media
A80293450	25.333

Having vs where

- La clausola **having**
  - è applicata dopo il raggruppamento
  - si applica solo ad attributi presenti nel **group by** e ad operatori aggregati
- La clausola **where** si applica prima del raggruppamento

```
SELECT Studente,avg(Voto)
FROM Esami GROUP BY Studente
HAVING avg(Voto)>25
```

Gli studenti con media maggiore di 25

Fare la media usando solo i voti maggiori di 25

```
SELECT Studente,avg(Voto)
FROM Esami GROUP BY Studente
WHERE voto>25
```

Interrogazioni nidificate

- Nella clausola **where** si possono utilizzare valori prodotti da altre istruzioni **select** utilizzando **any** (qualsiasi) o **all** (tutti) insieme agli operatori di confronto

Trovare nome, cognome e matricola degli studenti che non hanno fatto esami

```
SELECT Matricola,Nome,Cognome FROM studenti
WHERE matricola <> ALL (SELECT studente
                        FROM esami
                        GROUP BY studente)
```

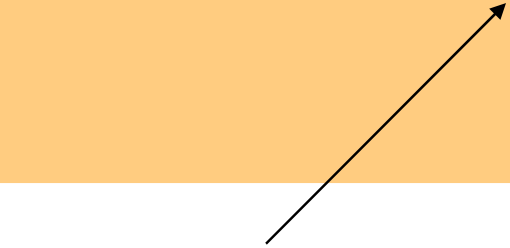
"diversa da tutte"
(si può usare anche **not in**)

Matricole degli studenti che hanno fatto almeno un esame

Un' altra interrogazione

Trovare il nome degli studenti che hanno fatto almeno 3 esami

```
SELECT Nome FROM studenti
WHERE matricola = ANY (SELECT studente
                       FROM esami
                       GROUP BY studente
                       HAVING count(*)>=3)
```



"uguale ad almeno una"
(si può usare anche **in**)

Matricole degli studenti che hanno fatto almeno 3 esami

Interrogazioni nidificate semplici e complesse - 1

Studenti

Matricola	Cognome	Nome	Data di nascita
A80198760	Bianchi	Anna	22/03/1967
A80293450	Rossi	Andrea	13/04/1968
A80295640	Felici	Lorenzo	25/02/1969

Esami

Studente	Voto	Corso
A80198760	28	Analisi I
A80293450	30	Basi di Dati
A80295640	27	Analisi I
A80198760	30L	Fisica I
A80293450	21	Chimica

Interrogazioni nidificate semplici e complesse - 2

- Nei casi più complessi la sottointerrogazione deve essere eseguita una volta per ogni riga dell'interrogazione principale

Interrogazione semplice: cercare il cognome degli studenti con almeno un 30

```
SELECT Cognome FROM studenti
WHERE matricola =
          ANY (SELECT studente FROM esami WHERE voto=30)
```

Interrogazione complessa: cercare gli studenti con un voto uguale al giorno di nascita (c'è infatti da valutare la funzione "day" sulla data di nascita)

```
SELECT Matricola FROM studenti
WHERE matricola =
      ANY (SELECT studente FROM esami WHERE
            voto=day("data di nascita"))
```

Interrogazioni equivalenti

- Alcune interrogazioni possono essere espresse in diverse forme
- L'utente sceglie la forma più leggibile, il DBMS tenta di eseguire la forma più efficiente

Due interrogazioni equivalenti

```
SELECT Matricola FROM studenti
WHERE matricola = ANY (SELECT studente FROM esami WHERE voto=30)
```

```
SELECT Matricola FROM studenti, esami
WHERE matricola = studente AND voto=30
```

Operatori su insiemi

- SQL mette a disposizione gli operatori **union** (unione), **except** (differenza), **intersect** (intersezione)

Cognomi e nomi di studenti che hanno sostenuto almeno un esame

```
SELECT cognomi AS nomeOppureCognome FROM studenti
WHERE matricola = ANY (SELECT studente FROM esami)

UNION

SELECT nome AS nomeOppureCognome FROM studenti
WHERE matricola = ANY (SELECT studente FROM esami)
```

Differenza

Cognomi che non sono anche nomi di studenti che hanno Sostenuto almeno un esame

```
SELECT cognomi AS nomeOppureCognome FROM studenti
WHERE matricola = ANY (SELECT studente FROM esami)

EXCEPT

SELECT nome AS nomeOppureCognome  FROM studenti
WHERE matricola = ANY (SELECT studente FROM esami)
```

- UNION, EXCEPT, INTERSECT tra più query si possono usare solo se esse restituiscono lo stesso numero di colonne e le colonne corrispondenti hanno tipi di dati compatibili

Cancellazione e modifica di tuple

Cancellazione

Cancella lo studente con matricola A80198760

```
DELETE FROM studenti
WHERE Matricola = 'A80198760'
```

Cancella gli studenti che non hanno esami

```
DELETE FROM studenti
WHERE Matricola NOT IN (SELECT Matricola
                        FROM esami
                        GROUP BY Matricola)
```

Modifica

```
UPDATE esami SET Voto = 30
WHERE Matricola = 'A80198760'
      AND corso = 'Analisi I'
```

Cancellazione: osservazioni

- Si eliminano tutti record indicati dalla clausola **where**
- Se la clausola where non è presente, si eliminano tutti i record della tabella

Rimuovere tutti gli studenti

```
DELETE FROM studenti
```

- Si può causare la cancellazione di record in altre tabelle (se i vincoli di integrità referenziale sono definiti con politiche di reazione **cascade**)

Modifica: osservazioni

- L'ordine dei comandi di aggiornamento è importante

Aumentare di 1 il voto degli studenti con voto maggiore di 25 e di 2 il voto degli altri studenti

```
UPDATE esami SET voto=voto+2 WHERE voto<=25

UPDATE esami SET voto=voto+1 WHERE voto>25
```

Chi aveva 25 dopo ha 28

```
UPDATE esami SET voto=voto+1 WHERE voto>25

UPDATE esami SET voto=voto+2 WHERE voto<=25
```

Chi aveva 25 dopo ha 27

Vincoli di integrità generici: check

- E' possibile specificare vincoli di integrità generici con la clausola **check**(*condizione*)

Sostituisce references

Vincoli sul valore del voto

```
CREATE TABLE esami (
  studente char(9) not null
        check(studente = any(select matricola from studenti))
  voto     integer not null check(voto<=30 and voto>=0),
  lode     char(1) check(lode=' ' or lode='L'),
  corso    char(20) not null,
  UNIQUE(studente,corso),
  CHECK(not((voto<>30) and (lode='L')))
)
```

Vincolo su lode solo con 30

Vincoli sul valore della lode

Viste – 1/2

- Corrispondono a tabelle virtuali ovvero che non esistono fisicamente ma il cui contenuto è ottenuto da altre tabelle
- In SQL si ottengono assegnando una lista di attributi e un nome ad una interrogazione con **`select`**

```
CREATE VIEW StudentiMeritevoli
(Matricola,Nome,Cognome)
AS SELECT Matricola,Nome,Cognome
   FROM Studenti
   WHRE Matricola IN (SELECT studente
                      FROM esami
                      GROUP BY studente
                      HAVING avg(Voto)>=28)
```

Viste – 2/2

- E’ possibile fare delle modifiche attraverso la vista, purché ogni riga della vista corrisponda ad una sola riga della tabella

```
CREATE VIEW StudentiEsami
(Nome,Cognome,Corso,Voto,Matricola)
AS SELECT Nome,Cognome,Corso,Voto,Studente
   FROM Studenti, Esami
   WHERE Matricola=Studente
```

Comando permesso

```
UPDATE StudentiEsami SET voto=30
WHERE cognome=‘rossi’ AND corso=‘analisi’
```