



UNIVERSITÀ DEGLI STUDI DI SALERNO

Atzeni, Ceri, Fraternali, Paraboschi, Torlone
Basi di dati *Quinta edizione*
McGraw-Hill Education, 2018

1

BASI DI DATI

Concetti Base SQL-DDL – quarta parte



Matteo Gaeta

Full Professor – Senior Member IEEE

BASI DI DATI

Concetti base – SQL

COMANDI FONDAMENTALI

TABLE	↔	Tabella	↔	Relazione
ROW	↔	Riga	↔	Tupla
COLUMN	↔	Colonna	↔	Attributo

➤ **SQL come DDL ha tre comandi fondamentali:**

Create

Database, Schema, Table, Domain, Constraint,

Alter

Database, Schema, Table, Domain, Constraint,

Drop

Database, Schema, Table, Domain, Constraint,

A pensare male degli altri si fa peccato, ma spesso si indovina.
(Giulio Andreotti)

BASI DI DATI

Concetti base – SQL

NOT-NULL CONSTRAINTS

- Un vincolo **NOT NULL** specifica semplicemente che una colonna non deve assumere il valore nullo. Un esempio di sintassi:
***CREATE TABLE products (
product_no integer NOT NULL,
name text NOT NULL,
price numeric
);***
- Un vincolo NOT NULL viene sempre scritto come vincolo di colonna.
- Un vincolo **NOT NULL** (nome_colonna NOT NULL), **è funzionalmente equivalente alla creazione di un vincolo ad hoc di controllo CHECK** ma in PostgreSQL è più efficiente creare un vincolo esplicito NOT NULL .
- Lo svantaggio dichiarandolo (come vincolo di colonna) è che non è possibile assegnare nomi espliciti a vincoli NOT NULL creati in questo modo.

Il potere logora chi non ce l'ha
(Giulio Andreotti)

BASI DI DATI

Concetti base – SQL

NOT-NULL CONSTRAINTS

- Una colonna può avere più di un vincolo. Basta scrivere i vincoli uno dopo l'altro e l'ordine non ha importanza:
***CREATE TABLE products (
product_no integer NOT NULL,
name text NOT NULL,
price numeric NOT NULL CHECK (price > 0)
);***
- Il vincolo NOT NULL ha un «inverso»: il vincolo NULL. Ovvero esplicita il comportamento predefinito del valore nella colonna che potrebbe essere NULL.
- Il vincolo NULL (presente in PostgreSQL, non è presente nello standard SQL e non deve essere utilizzato nelle applicazioni portatili. È stato aggiunto a PostgreSQL solo per essere compatibile con altri sistemi di database.
- Nella maggior parte dei progetti di database, la maggior parte delle colonne dovrebbe essere contrassegnata come NOT NULL.

La prudenza è una grande virtù, ma l'incertezza è di enorme danno, sia nella vita pubblica che in quella privata. (Giulio Andreotti)

BASI DI DATI

Concetti base – SQL

UNIQUE CONSTRAINTS

- I vincoli **UNIQUE** assicurano che i dati contenuti in una colonna o in un gruppo di colonne siano univoci tra tutte le righe della tabella.

La sintassi come vincolo di colonna è:

```
CREATE TABLE products (  
  product_no integer UNIQUE,  
  name text,  
  price numeric  
);
```

La sintassi come vincolo di Tabella è:

```
CREATE TABLE products (  
  product_no integer,  
  name text,  
  price numeric,  
  UNIQUE (product_no)  
);
```

- Per definire il vincolo **UNIQUE** per un gruppo di colonne dobbiamo scriverlo come vincolo di Tabella con i nomi delle colonne separati da virgole:

```
CREATE TABLE example (  
  a integer,  
  b integer,  
  c integer,  
  UNIQUE (a, c)  
);
```

- Questa modalità deve essere usata allorquando desideriamo specificare che la combinazioni di valori nelle colonne di interesse è univoca in tutta la tabella, anche se una qualsiasi colonna può assumere un valore non univoco nella tabella
- *Due valori NULL non vengono mai considerati uguali in questo confronto.*

- Anche per un vincolo di tipo UNIQUE è possibile assegnare un nome come avviene per altri vincoli:
***CREATE TABLE prodotti (
product_no integer CONSTRAINT must_be_different UNIQUE,
testo del nome,
prezzo numerico
);***
- L'aggiunta di un vincolo UNIQUE creerà automaticamente un indice B-tree univoco sulla colonna o sul gruppo di colonne elencate nel vincolo.
- Una restrizione di unicità che copre solo alcune righe non può essere scritta come vincolo univoco, ma è possibile applicare tale restrizione creando un indice parziale univoco.

BASI DI DATI

8

Concetti base – SQL

PRIMARY KEYS

- Un vincolo di chiave primaria indica che una colonna, o un gruppo di colonne, può essere utilizzato come **identificatore univoco** per le righe nella tabella. Ciò richiede che i valori in gioco siano **sia UNIQUE che NOT NULL**. Può esserci un numero qualsiasi di vincoli UNIQUE e NOT NULL, che sono funzionalmente quasi la stessa cosa di una definizione PRIMARY KEY, ma la Primary Key è una sola.

- **Quindi, le seguenti due definizioni di tabella accettano gli stessi dati:**

```
CREATE TABLE products (  
product_no integer UNIQUE NOT NULL,  
name text,  
price numeric );
```

```
CREATE TABLE products (  
product_no integer PRIMARY KEY,  
name text,  
price numeric );
```


BASI DI DATI

Concetti base – SQL

PRIMARY KEYS

- Le chiavi primarie possono occupare più di una colonna e quindi viene scritto come vincolo di tabella; la sintassi è simile ai vincoli UNIQUE:

```
CREATE TABLE example (  
  a integer,  
  b integer,  
  c integer,  
  PRIMARY KEY (a, c)  
);
```

- L'aggiunta di una chiave primaria creerà automaticamente un indice B-tree univoco sulla colonna o sul gruppo di colonne elencate nella chiave primaria e costringerà la colonna o le colonne a essere contrassegnate come NOT NULL.
- **Una tabella può avere al massimo una chiave primaria.**
- La teoria dei DB relazionali impone che ogni tabella debba avere una Primary Key.
- Questa regola non è applicata da PostgreSQL, ma di solito è bene seguirla.

BASI DI DATI

10

Concetti base – SQL

FOREIGN KEYS

- Un vincolo di chiave esterna specifica che i valori in una colonna (o un gruppo di colonne) devono corrispondere ai valori che appaiono in una riga di un'altra tabella. Tale Vincolo mantiene **l'integrità referenziale** tra due tabelle correlate.
- Supponiamo di avere la tabella dei prodotti che abbiamo già utilizzato più volte:
CREATE TABLE products (←TABELLA REFERENZIATA, MASTER, ESTERNA, PRINCIPALE
product_no integer PRIMARY KEY,
name text,
price numeric);
- Supponiamo di avere una tabella che memorizza gli ordini di quei prodotti e che vogliamo assicurarci che essa contenga solo ordini di prodotti in tabella dei prodotti. **definiamo un vincolo di chiave esterna nella tabella degli ordini che fa riferimento alla tabella dei prodotti:**
CREATE TABLE orders (← TABELLA REFERENZIANTE, SLAVE, INTERNA, SECONDARIA
order_id integer PRIMARY KEY,
product_no integer REFERENCES products(product_no),
quantity integer);

BASI DI DATI

11

Concetti base – SQL

FOREIGN KEYS

- Ora è **impossibile** creare ordini con voci product_no diverse da NULL che non compaiono nella tabella dei prodotti.
- E' possibile abbreviare il comando (in colonna) perché in assenza di un elenco di colonne, la Primary Key della tabella di riferimento viene usata come colonna di riferimento.

```
CREATE TABLE orders (  
  order_id integer PRIMARY KEY,  
  product_no integer REFERENCES products,  
  quantity integer);
```

- Una chiave esterna può anche vincolare e far riferimento a un gruppo di colonne, usando la forma dei vincoli di tabella ed il numero e il tipo delle colonne vincolate devono corrispondere al numero e al tipo delle colonne di riferimento.

```
CREATE TABLE t1 (  
  a integer PRIMARY KEY,  
  b integer,  
  c integer,  
  FOREIGN KEY (b, c) REFERENCES other_table (c1, c2) );
```

BASI DI DATI

12

Concetti base – SQL

FOREIGN KEYS

- E' possibile assegnare un nome per un vincolo di chiave esterna, nel solito modo. Una tabella può avere più di un vincolo di chiave esterna. **Viene utilizzato per implementare relazioni multi-a-molti tra le tabelle.**

```
CREATE TABLE products (  
product_no integer PRIMARY KEY,  
name text,  
price numeric );  
CREATE TABLE orders (  
order_id integer PRIMARY KEY,  
shipping_address text,  
... );  
CREATE TABLE order_items (  
product_no integer REFERENCES products,  
order_id integer REFERENCES orders,  
quantity integer,  
PRIMARY KEY (product_no, order_id) );
```

- Cosa succede se un prodotto viene rimosso dopo la creazione di un ordine che lo tratta? Intuitivamente, abbiamo alcune opzioni: 1) Non consentire l'eliminazione di un prodotto di riferimento; 2) Elimina anche gli ordini 3) Qualcos'altro?
- Realizziamo la seguente Policy:
 - ❖ quando vogliamo rimuovere un prodotto che è referenziato da un ordine (tramite order_items), non lo consentiamo.
 - ❖ Se rimuoviamo un ordine, vengono rimossi anche gli articoli dell'ordine:

```
CREATE TABLE order_items (  
product_no integer REFERENCES products ON DELETE RESTRICT,  
order_id integer REFERENCES orders ON DELETE CASCADE,  
quantity integer,  
PRIMARY KEY (product_no, order_id) );
```

- Con riferimento alle possibili opzioni **ON DELETE** e **ON UPDATE in REFERENCES**, la limitazione **NO ACTION (RESTRICT)** e le eliminazioni a cascata **CASCADE** sono le due opzioni più comuni:
 - ❖ **NO ACTION** significa che se esistono ancora righe di riferimento quando il vincolo è selezionato, viene generato un errore; questo è il comportamento predefinito se non si specifica nulla. Le regole di vincolo NO ACTION vengono verificate dopo il completamento dell'istruzione e di tutte le altre operazioni (come i trigger).
 - ❖ **RESTRICT** impedisce la cancellazione di una riga referenziata; la modifica del valore non è affatto consentita, nessuna modifica o eliminazione è consentita se sono presenti record in tabelle correlate. Le regole di vincolo RESTRICT vengono verificate prima di qualsiasi altra operazione
- ❖ Nella maggior parte dei casi, non c'è differenza tra le due opzioni.
- ❖ **CASCADE** specifica che quando una riga di riferimento viene eliminata, le righe che fanno riferimento ad essa devono essere automaticamente anche cancellate.

- Sono disponibili altre due opzioni: **SET NULL** e **SET DEFAULT**.
- Le opzioni SET NULL oppure SET DEFAULT fanno sì che le colonne di riferimento nelle righe di riferimento vengano impostate rispettivamente su NULL o sui loro valori predefiniti di default, quando una riga di riferimento viene eliminata **ON DELETE**.
- **Ovviamente questi valori devono essere compatibili con gli altri vincoli presenti.**
 - ✓ Ad esempio, se un'azione specifica SET DEFAULT ma il valore predefinito non soddisfa il vincolo di chiave esterna, l'operazione avrà esito negativo.
- Analogamente a ON DELETE, esiste **ON UPDATE** che viene richiamato quando una colonna di riferimento deve essere aggiornata.

- In definitiva le azioni possibili di ON UPDATE sono le stesse di ON DELETE.
- In questo caso, CASCADE significa che i valori aggiornati delle colonne referenziate devono essere copiati nelle righe referenziate.
- Normalmente, una riga di riferimento non deve soddisfare il vincolo di chiave esterna se una delle sue colonne di riferimento è nulla.
- Se MATCH FULL è aggiunto alla dichiarazione di chiave esterna, una riga di riferimento soddisfa il vincolo solo se **tutte** le sue colonne di riferimento sono nulle (combinazioni di valori nulli e non nulli fanno fallire un vincolo MATCH FULL).
- Se non si desidera che le righe di riferimento siano in grado di evitare di soddisfare il vincolo di chiave esterna, dichiarare la/e colonna/e di riferimento NOT NULL.

- **Una chiave esterna deve fare riferimento a colonne che sono una chiave primaria o formano un vincolo univoco.** Ciò significa che le colonne referenziate hanno sempre un indice (quello sottostante la chiave primaria o vincolo univoco);
- Poiché un DELETE di una riga dalla tabella di riferimento o un UPDATE di una colonna di riferimento richiederà una scansione della tabella di riferimento per le righe che corrispondono al vecchio valore, è spesso una buona idea indicizzare anche le colonne di riferimento.
- Poiché questo non è sempre necessario e sono disponibili molte scelte su come indicizzare, la dichiarazione di un vincolo di chiave esterna non crea automaticamente un indice sulle colonne di riferimento.

BASI DI DATI

18

Concetti base – SQL

CREATE TABLE: UN ESEMPIO

➤ Synopsis

```
CREATE TABLE Nome Tabella (  
    NomeAttributo1 tipoDati(Dominio) [valore di default] [vincoli],  
    { NomeAttributo1 tipoDati(Dominio) [valore di default] [vincoli] }  
    AltriVincoli (es. di Tabella) );
```

➤ Esempio

```
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) );
```

Warning: DB2 vuole NOT NULL per la chiave primaria

➤ Le operazioni di Modifying Table variano la struttura della tabella (non i dati), esse sono:

- Add columns
- Remove columns
- Add constraints
- Remove constraints
- Change default values
- Change column data types
- Rename columns
- Rename tables

➤ Tutte queste azioni vengono eseguite utilizzando il comando

❖ **ALTER TABLE**

➤ *Adding a Column*

- Per aggiungere una colonna, il comando è:
 - ❖ **ALTER TABLE products ADD COLUMN description text;**
- La nuova colonna viene inizialmente riempita con il valore predefinito fornito (NULL se non si specifica una clausola DEFAULT).
- E' possibile definire vincoli sulla colonna allo stesso tempo, usando la solita sintassi con il comando CHECK:
 - ❖ **ALTER TABLE products ADD COLUMN descr text CHECK (descr <> "");**
- Tutte le opzioni che possono essere applicate alla descrizione di una colonna in CREATE TABLE possono essere utilizzate anche in questo caso.
- il valore predefinito deve soddisfare i vincoli dati, altrimenti ADD fallirà. E' possibile aggiungere vincoli dopo aver definito correttamente la nuova colonna.

➤ *Removing a Column*

- Per rimuovere una colonna, usa un comando:
 - ❖ **ALTER TABLE products DROP COLUMN description;**
- I dati nella colonna vengono eliminati in una alla volta.
- I vincoli di tabella che coinvolgono la colonna vengono eliminati.
- Tuttavia, se la colonna è referenziata da un vincolo di chiave esterna di un'altra tabella, PostgreSQL non eliminerà «silenziosamente» quel vincolo. Puoi autorizzare l'eliminazione di tutto ciò che dipende dalla colonna aggiungendo CASCADE:
 - ❖ **ALTER TABLE products DROP COLUMN description CASCADE;**

➤ *Adding a Constraint*

- Per aggiungere un vincolo, viene utilizzata la sintassi del vincolo di tabella, es:
 - ❖ **ALTER TABLE products ADD CHECK (name <> '');**
 - ❖ **ALTER TABLE products ADD CONSTRAINT some_name UNIQUE (product_no);**
 - ❖ **ALTER TABLE products ADD FOREIGN KEY (product_group_id) REFERENCES product_groups;**
- Per aggiungere un vincolo non nullo, che non può essere scritto come vincolo di tabella, utilizzare questa sintassi:
 - ❖ **ALTER TABLE products ALTER COLUMN product_no SET NOT NULL;**

Il vincolo verrà verificato immediatamente, quindi i dati della tabella devono soddisfare il vincolo prima di essere aggiunti.

➤ *Removing a Constraint*

- Per rimuovere un vincolo è necessario conoscerne il nome. Se noto è facile, altrimenti il sistema ha assegnato un nome <generato>, che è necessario conoscere. In tal caso in PostgreSQL il comando `psql \d tablename` può essere utile; altri RDBMS potrebbero anche fornire un modo per ispezionare i dettagli della tabella.
- il comando è:
 - ❖ **ALTER TABLE products DROP CONSTRAINT some_name;**
- Come per l'eliminazione di una colonna, è necessario aggiungere CASCADE se si desidera eliminare un vincolo che implementa dipendenze, come ad un vincolo di chiave esterna
- Funziona allo stesso modo per tutti i tipi di vincoli tranne i vincoli NOT NULL.
- Per eliminare un vincolo non nullo è possibile usare:
 - ❖ **ALTER TABLE products ALTER COLUMN product_no DROP NOT NULL;**

➤ Changing a Column's Default Value

- Per impostare un nuovo valore predefinito per una colonna, utilizza un comando:
❖ **ALTER TABLE products ALTER COLUMN price SET DEFAULT 7.77;**
- Nota che questo non influisce sui dati di nessuna riga esistente nella tabella, cambia solo l'impostazione predefinita per il futuro INSERT comandi.
- Per rimuovere qualsiasi valore predefinito, usare:
❖ **ALTER TABLE products ALTER COLUMN price DROP DEFAULT;**
- Questo comando equivale ed è effettivamente lo stesso dell'impostazione del valore predefinito su NULL. Infatti, basta ricordare che il valore predefinito di base è implicitamente il valore nullo.

➤ *Changing a Column's Data Type*

- Per convertire una colonna in un tipo di dati diverso, il comando è:
❖ **ALTER TABLE products ALTER COLUMN price TYPE numeric(10,2);**
- Ciò avrà successo solo se ogni voce esistente nella colonna può essere convertita nel nuovo tipo (CAST implicito).
- Se è necessaria una conversione più complessa, è possibile aggiungere una clausola **USING** che specifica come per calcolare i nuovi valori dal vecchio.
- PostgreSQL tenterà di convertire il valore predefinito della colonna (se presente) nel nuovo tipo, così come eventuali vincoli che coinvolgono la colonna. Ma queste conversioni potrebbero fallire o produrre risultati sorprendenti
- Spesso è meglio eliminare qualsiasi vincolo sulla colonna prima di modificarne il tipo, quindi poi aggiungerli nuovamente opportunamente modificati

➤ *Renaming a Column*

➤ Il comando è:

❖ ***ALTER TABLE products RENAME COLUMN product_no TO product_number;***

➤ *Renaming a Column*

Il comando è:

❖ ***ALTER TABLE products RENAME TO items;***

BASI DI DATI

27

Concetti base – SQL

SCRIPT

Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres@Postgi

Query Editor Query History Sc

```
1  /* Esempio Creazione di una tabella DIPARTIMENTO */
2  DROP TABLE IF EXISTS DIPARTIMENTO cascade;
3  CREATE TABLE DIPARTIMENTO (
4  Codice          CHAR(4) not null,
5  Nome            VARCHAR(20) not null unique,
6  constraint PK_DIPARTIMENTO primary key (CODICE)
7  );
8
9
10
11
12 |
13 /* Esempio Inserire 3 colonne nello schema della tabella DIPARTIMENTO */
```

Data Output Explain Messages Notifications

CREATE TABLE

Query returned successfully in 59 msec.

BASI DI DATI

28

Concetti base – SQL

SCRIPT

Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres@Post

Query Editor Query History

```
12
13 /* Esempio Inserire 3 colonne nello schema della tabella DIPARTIMENTO */
14 ALTER TABLE DIPARTIMENTO
15 add column DIP_IND varchar (50);
16
17 ALTER TABLE DIPARTIMENTO
18 add DIP_CITTA varchar(20),
19 add PROVA varchar(2);
20
21 /* Esempio eliminare colonne dallo schema della tabella DIPARTIMENTO */
22 ALTER TABLE DIPARTIMENTO
23 drop PROVA;
24
```

Data Output Explain Messages Notifications

ALTER TABLE

Query returned successfully in 45 msec.

BASI DI DATI

29

Concetti base – SQL

SCRIPT

Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres

Query Editor Query History

```
24
25
26 /* Creare una tabella DIPARTIMENTO */
27 DROP TABLE IF EXISTS DIPARTIMENTO;
28 CREATE TABLE DIPARTIMENTO (
29 CODICE      CHAR(4) not null,
30 NOME        VARCHAR(20) not null unique,
31 Dip_ind     varchar (50),
32 Dip_Citta   varchar(20),
33 constraint PK_DIPARTIMENTO primary key (CODICE)
34 );
35 |
36 /* Esempio Inserire occorrenze in una tabella DIPARTIMENTO */
```

Data Output Explain Messages Notifications

CREATE TABLE

Query returned successfully in 58 msec.

BASI DI DATI

Concetti base – SQL

SCRIPT

Query Editor Query History

```

36  /* Esempio Inserire occorrenze in una tabella DIPARTIMENTO */
37  INSERT INTO  DIPARTIMENTO (CODICE, NOME, DIP_IND, DIP_CITTA)
38  VALUES ('AMMZ', 'Amministrazione', 'Via Tito Livio, 27', 'Milano' );
39  INSERT INTO  DIPARTIMENTO (CODICE, NOME, DIP_IND, DIP_CITTA)
40  VALUES ('PROD', 'Produzione', 'P.le Lavater, 3', 'Torino' );
41  INSERT INTO  DIPARTIMENTO (CODICE, NOME, DIP_IND, DIP_CITTA)
42  VALUES ('DIST', 'Distribuzione', 'Via Segre, 9', 'Roma' );
43  INSERT INTO  DIPARTIMENTO (CODICE, NOME, DIP_IND, DIP_CITTA)
44  VALUES ('DIRE', 'Direzione', 'Via Tito Livio, 27', 'Milano' );
45  INSERT INTO  DIPARTIMENTO (CODICE, NOME, DIP_IND, DIP_CITTA)
46  VALUES ('RICE', 'Ricerca', 'Via Venosa, 6', 'Milano' );
47
48  Select *
49  From Dipartimento;
50

```

Data Output Explain Messages Notifications

	codice [PK] character (4)	nome character varying (20)	dip_ind character varying (50)	dip_citta character varying (20)
1	AMMZ	Amministrazione	Via Tito Livio, 27	Milano
2	PROD	Produzione	P.le Lavater, 3	Torino
3	DIST	Distribuzione	Via Segre, 9	Roma
4	DIRE	Direzione	Via Tito Livio, 27	Milano
5	RICE	Ricerca	Via Venosa, 6	Milano

BASI DI DATI

31

Concetti base – SQL

SCRIPT

Query Editor Query History

```
51  /* Esempio Creare una tabella IMPIEGATO */
52  drop table IF EXISTS impiegato cascade;
53  create table impiegato (
54      matricola char(6) primary key,
55      nome varchar(20),
56      cognome varchar (20),
57      dipart varchar (20) references DIPARTIMENTO(nome),
58      ufficio numeric (3),
59      stipendio numeric (9) default 0,
60      citta character varying(50),
61      unique (cognome, nome)
62  );
63
64
65
```

Data Output Explain Messages Notifications

CREATE TABLE

Query returned successfully in 57 msec.

BASI DI DATI

32

Query Editor Query History

Concetti base – SQL

SCRIPT

```
64 /* Esempio Inserire occorrenze in una tabella IMPIEGATO */
65 INSERT INTO impiegato (matricola, nome, cognome, dipart, ufficio, stipendio, citta)
66 VALUES ('000001', 'Mario', 'Rossi', 'Amministrazione', 10, 45, 'Milano');
67 INSERT INTO impiegato (matricola, nome, cognome, dipart, ufficio, stipendio, citta)
68 VALUES ('000002', 'Carlo', 'Bianchi', 'Produzione', 20, 36, 'Torino');
69 INSERT INTO impiegato (matricola, nome, cognome, dipart, ufficio, stipendio, citta)
70 VALUES ('000003', 'Giovanni', 'Verdi', 'Amministrazione', 20, 40, 'Roma');
71 INSERT INTO impiegato (matricola, nome, cognome, dipart, ufficio, stipendio, citta)
72 VALUES ('000004', 'Franco', 'Neri', 'Distribuzione', 16, 45, 'Napoli');
73 INSERT INTO impiegato (matricola, nome, cognome, dipart, ufficio, stipendio, citta)
74 VALUES ('000005', 'Carlo', 'Rossi', 'Direzione', 14, 80, 'Milano');
75
76 select *
77 from impiegato;
```

Data Output Explain Messages Notifications

	matricola [PK] character (6)	nome character varying (20)	cognome character varying (20)	dipart character varying (20)	ufficio numeric (3)	stipendio numeric (9)	citta character varying (50)	
1	000001	Mario	Rossi	Amministrazione	10	45	Milano	
2	000002	Carlo	Bianchi	Produzione	20	36	Torino	
3	000003	Giovanni	Verdi	Amministrazione	20	40	Roma	
4	000004	Franco	Neri	Distribuzione	16	45	Napoli	
5	000005	Carlo	Rossi	Direzione	14	80	Milano	

BASI DI DATI

Materiale utilizzato e bibliografia

- **Le slide utilizzate dai docenti per le attività frontali sono in gran parte riconducibili e riprese dalle slide originali (con alcuni spunti parziali ripresi dai libri indicati) realizzate da:**
 - ✓ autori del libro *Basi di Dati* (Atzeni e altri) testo di riferimento del corso *Basi di Dati* e sono reperibili su internet su molteplici link oltre che laddove indicato dagli stessi autori del libro;
 - ✓ Prof.ssa Tiziana Catarci e dal dott. Ing. Francesco Leotta – corso di *Basi di Dati* dell'Università degli Studi La Sapienza di Roma al seguente link ed altri: <http://www.dis.uniroma1.it/~catarci/basidatGEST.html> (molto Interessanti anche le lezioni su YouTube).
 - ✓ Proff. Luca Allulli e Umberto Nanni, *Libro Fondamenti di basi di dati*, editore HOEPLI (testo di facile lettura ed efficace).
- **Diverse slide su specifici argomenti utilizzate dai docenti per le attività frontali sono anche in parte riconducibili e riprese dalle slide originali – facilmente reperibili e accessibili su internet - realizzate da:**

Prof.ssa Roberta Aiello – corso *Basi di Dati* dell'Università di Salerno

Prof. Dario Maio - corso *Basi di Dati* dell'Università di Bologna al seguente link ed altri: <http://bias.csr.unibo.it/maio>

Prof. Marco Di Felice - corso *Basi di Dati* dell'Università di Bologna al seguente link ed altri: <http://www.cs.unibo.it/difelice/dbsi/>

Prof. Marco Maggini e prof. Franco Scarselli - corso *Basi di Dati* dell'Università di Siena ai seguenti link ed altri: [http://staff.icar.cnr.it/pontieri/didattica/LabSI/lezioni/_preliminari-DB1%20\(Maggini\).pdf](http://staff.icar.cnr.it/pontieri/didattica/LabSI/lezioni/_preliminari-DB1%20(Maggini).pdf)

Prof.ssa Raffaella Gentilini - corso *Basi di Dati* dell'Università di Perugia al seguente link ed altri: <http://www.dmi.unipg.it/raffaella.gentilini/BD.htm>

Prof. Enrico Giunchiglia - corso *Basi di Dati* dell'Università di Genova al seguente link ed altri: <http://www.star.dist.unige.it/~enrico/BasiDiDati/>

Prof. Maurizio Lenzerini - corso *Basi di Dati* dell'Università degli Studi La Sapienza di Roma al seguente link ed altri: <http://didatticainfo.altervista.org/Quinta/Database2.pdf>

➤ The PostgreSQL Global Development Group - PostgreSQL nn.xx Documentation

➤ PostgreSQL (appendice - scaricabile dal sito del libro (area studenti) e www.postgresql.org)