Università degli Studi di Verona

Dipartimento di Informatica
A.A. 2024 - 2025

Tecnologie per le basi di dati

Corso tenuto da
S. MIGLIORINI

Appunti a cura di
NICK

Made with LaTeX

Indice

1 PostgreSQL **2**
1.1 Introduzione . . . 2
1.1.1 Server . . . 2
1.1.2 Client(s) . . . 2
1.1.3 Esempio di connessione al server tramite psql . . . 2
1.2 Esempio applicativo psql . . . 3

2 Structured Query Language - SQL **4**
2.1 Notazione . . . 4
2.2 Creazione di una tabella . . . 4
2.3 Domini elementari . . . 4
2.3.1 Caratteri . . . 4
2.3.2 Boolean . . . 4
2.3.3 Tipi numerici esatti . . . 5
2.3.4 Tipi numerici approssimati . . . 5
2.3.5 Istanti temporali . . . 5
2.3.6 Intervalli temporali . . . 6
2.4 Domini definiti dall'utente . . . 6
2.5 Vincoli intrarelazionali . . . 6
2.5.1 Vincolo `NOT NULL` e `DEFAULT` . . . 7
2.5.2 Vincolo `UNIQUE` . . . 7
2.5.3 Vincolo `PRIMARY KEY` . . . 7
2.5.4 Vincolo `CHECK` . . . 8
2.6 Vincoli interrelazionali . . . 8
2.6.1 Sintassi per singoli attributi . . . 8
2.6.2 Sintassi per più attributi . . . 8
2.7 Esempio di creazione di una tabella . . . 9
2.8 Modifiche degli schemi . . . 9
2.8.1 Comando `ALTER TABLE` . . . 10

1 PostgreSQL

1.1 Introduzione

PostgreSQL è un RDBMS con alcune funzionalità orientate agli oggetti. E' multipiattaforma, open-source e gestito da un gruppo di volontari. La documentazione si può trovare sul sito ufficiale di `http://www.postgresql.org`. PostgreSQL implementa gran parte dello standard SQL: la versione 16 implementa quasi tutte le funzioni SQL 2023, per approfondire si rimanda alla documentazione.

L'interazione tra un utente e le basi di dati gestite da PostgreSQL avviene secondo il modello *client-server*.

1.1.1 Server

Il daemon postmaster supervisiona tutti i processi specifici di PostgreSQL.

1.1.2 Client(s)

Qualsiasi programma in grado di gestire l'interazione con postmaster (input di comandi, invio al postmaster, visualizzazione esito comando, ecc). Il client standard è **psql**, un'applicazione a linea di comando. Esiste un'alternativa interessante: **pgAdmin IV**, un client grafico più intuitivo.

1.1.3 Esempio di connessione al server tramite psql

```
psql -h db-srv.di.univr.it -U <userid> <nomeid>
Inserisci la password per l'utente <userid>
psql(16.0)
connessione SSL (protocollo: TLSv1.2, cifrario ECDHE-RSA-AES256-GCM-
    SHA384,
bit: 256, compressione: disattivato)

Digita 'help' per avere un aiuto
<usrid> =>
```

psql è un interprete iterativo da terminale. Accetta due tipi di comando: comando di tipo SQL (Structured Query Language) o comandi interni.

- Comandi SQL: permettono di creare/gestire e interrogare le basi di dati.
- Comandi interni (riconoscibili perché iniziano con '
  '): permettono di configurare il client o avere aiuto per scrivere comandi SQL.

1.2 Esempio applicativo psql

Esempio – comando interno \l: elenca tutti le basi di dati

```
id051563 => \l
                 List of databases
Name      | Owner     | Encoding | Collate       | Ctype
----------+-----------+----------+---------------+--------------
template0 | postgres  | UTF8     | it_IT .UTF -8 | it_IT .UTF -8
Template1 | postgres  | UTF8     | it_IT .UTF -8 | it_IT .UTF -8
id051563  | id051563  | UTF8     | it_IT .UTF -8 | it_IT .UTF -8
...
id051563 =>
```

Esempio – comando SQL CREATE

```
id051563 => CREATE TEMP TABLE inutile (id INTEGER , inutile VARCHAR );
CREATE TABLE
id051563 =>
```

2 Structured Query Language - SQL

2.1 Notazione

- **[** e **]** indicano parti opzionali, il termine può non comparire o comparire una sola volta;
- **{** e **}** indicano che il termine può non comparire o comparire un numero arbitrario di volte;
- L'istruzione `CREATE TABLE` definisce uno schema di relazione e ne crea un'istanza vuota: attributi, domini e vincoli.

2.2 Creazione di una tabella

Esempio di sintassi per la creazione di una tabella:

```
CREATE TABLE tabella (
  attributo dominio [ valoreDefault ] { vincolo }
  {, attributo dominio [valoreDefault } { vincolo } }
  {, vincoloTabella }
]);
```

2.3 Domini elementari

2.3.1 Caratteri

Permette di rappresentare singoli caratteri oppure stringhe. Un carattere o una stringa di caratteri sono rappresentati tra apici singoli, ad esempio `'a'` oppure `'questa è una stringa'`.

La lunghezza delle stringhe può essere fissa o variabile.

```
CHARACTER [VARYING] [(lunghezza)]

CHARACTER          -- carattere singolo
CHARACTER(20)      -- stringa di lunghezza fissa, se si inseriscono meno
   di 20 caratteri viene riempita di spazi.
CHARACTER VARYING(20)   -- stringa di lunghezza variabile, max 20
   caratteri.
TEXT               -- stringa di lunghezza variabile senza limite fissato.
   Estensione PostgreSQL
```

2.3.2 Boolean

Permette di rappresentare i valori booleani `TRUE/FALSE`, rappresentati anche come `t/f, 1/0, yes/NO`, più lo stato unkonown (valore nullo), rappresentato come `NULL`.

```
BOOLEAN      -- valore booleano singolo
```

2.3.3 Tipi numerici esatti

Permettono di rappresentare valori esatti interi o con parte decimale di lunghezza prefissata.

Valori interi

```
SMALLINT    -- valori interi (2 bytes da -32768 a +32767)
INTEGER     -- valori interi (4 bytes da - 2147483648 a +2147483647)
```

Valori decimali

```
NUMERIC [ (precisione [, scala]) ]
```

Dove **precisione** è il numero totale di cifre significative, cioè sia a sinistra che a destra della virgola; mentre **scala** è il numero di cifre dopo la virgola, se vuoto = 0. Equivalente a `NUMERIC` esiste anche il tipo `DECIMAL`:

```
DECIMAL [ (precisione [, scala]) ]
```

Esempio utilizzo di `NUMERIC`:

```
NUMERIC(5,2) -- permette di rappresentare valori come 100.01 o 100.99 (
    arrotondato a 101.00)
-- ma non rappresenta valori come 1000.01
```

2.3.4 Tipi numerici approssimati

Permettono di rappresentare valori in virgola mobile.

```
REAL -- precisione di 6 cifre decimali.

DOUBLE PRECISION -- precisione di 15 cifre decimali.
```

Ciascun numero è rappresentato da una coppia di valori: la mantissa e l'esponente. La mantissa è un numero frazionario, l'esponente è un numero intero. Il valore si ottiene moltiplicando la mantissa per la potenza di 10 con grado pari all'esponente, ad esempio: $0.176E16 = 1.7 \times 10^{15}$ oppure $-0,4E-6 = -4 \times 10^{-7}$

Nota: `REAL` e `DOUBLE PRECISION` sono comunque valori approssimati. Se si devono rappresentare importi di denaro che contengono anche decimali, **mai usare questi tipi, ma usare** `NUMERIC(precisione)`.

2.3.5 Istanti temporali

Permettono di rappresentare gli istanti di tempo

DATE

Un valore `DATE` si rappresenta tra apici (') e lo standard ISO prescrive il formato YYYY-MM-DD.

```
DATE                -- anno, mese, giorno
```

TIME

Istanti del tipo hour, minute, second. Precisione è il numero di cifre decimali per rappresentare frazioni del secondo. `WITH TIME ZONE` permette di specificare anche $\pm$hour, minute, che rappresentano la differenza con l'ora di Greenwich, o nomeFusoOrario.

```
TIME [ (precisione)] [WITH TIME ZONE] -- hour, minute, second
```

TIMESTAMP

```
TIMESTAMP [ (precisione)] [WITH TIME ZONE]  -- date + time
```

2.3.6 Intervalli temporali

Permettono di rappresentare intervalli di tempo, come la durata di un evento. La sintassi è

```
INTERVAL primaUnitaDiTempo [TO ultimaUnitaDiTempo]
```

`primaUnitaDiTempo` e `ultimaUnitaDiTempo` definiscono le unità di misura che deveono essere utilizzate, dalla piú precisa alla meno precisa. Esempio:

```
INTERVAL year TO month -- indica un intervallo di tempo misurato in
    numero di anni e mesi
```

L'insieme delle unità di misura è diviso in due insiemi distinti: da *year* a *month* e da *day* a *second*. Poiché non è possibile paragonare esattamente giorni e mesi (un mese può avere da 28 a 31 giorni).

Si può specificare una precisazione per la `primaUnitaDiTempo`: numero di cifre in base 10 per la **rappresentazione**. Se l'`ultimaUnitaDiTempo` è **second**, si può specificare una precisione che rappresenta il numero di cifre decimali dopo la virgola. Ad esempio:

```
INTERVAL year(5) TO month -- intervalli della durata fino a 99 999 anni e
    11 mesi

INTERVAL day(4) TO second(2) -- intervalli della durata fino a 9999
    giorni, 23 ore, 59 minuti e 59.99 secondi
```

2.4 Domini definiti dall'utente

L'istruzione `CREATE DOMAIN` definisce un dominio, utilizzabile in definizioni di relazioni, anche con vincoli e valori di default. Il dominio può essere definito a partire da un dominio elementare oppure da un altro dominio definito dall'utente.

Sintassi:

```
CREATE DOMAIN nome AS tipoBase [valoreDefault] [vincolo]
```

Per esempio creiamo il dominio `giorniSettimana` che indica i giorni della settimana con 3 caratteri:

```
CREATE DOMAIN giorniSettimana AS CHARACTER(3)
  CHECK (VALUE IN ('LUN', 'MAR', 'MER', 'GIO', 'VEN', 'SAB', 'DOM'));
```

Un altro esempio potrebbe essere un dominio per rappresentare un voto:

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

2.5 Vincoli intrarelazionali

Abbiamo quattro vincoli intrarelazionali:

- `NOT NULL`: il valore nullo non è ammesso per l'attributo, quindi il valore deve sempre essere specificato, o si assegna un valore di default.

- `UNIQUE`: definisce (super)chiavi.
- `PRIMARY KEY`: chiave primaria (una sola, implica `NOT NULL`).
- `CHECK`

2.5.1 Vincolo `NOT NULL` e `DEFAULT`

Il vincolo `NOT NULL` determina che il valore nullo non è ammesso come valore dell'attributo. Nel caso di vincoli `NOT NULL` può essere utile specificare un valore di default per l'attributo.

L'istruzione `DEFAULT valore` specifica un valore di default per un attributo quando un comando di inserimento dati non specifica nessun valore per l'attributo.

Esempio:

```
37 nome VARCHAR(20) NOT NULL
38 cognome VARCHAR(20) NOT NULL DEFAULT 'VUOTO'
```

2.5.2 Vincolo `UNIQUE`

Il vincolo `UNIQUE` impone che righe differenti della tabella non possono avere gli stessi valori: concetto di superchiave il quale, però, richiede anche il vincolo di `NOT NULL`.

Si può definire su un solo attributo o su un insieme di attributi.

UNIQUE su una coppia ≠ UNIQUE sui due attributi	

`nome VARCHAR(20),`
`cognome VARCHAR(20),`
`UNIQUE(nome, cognome)`
	`('n','c'), ('n', NULL), ('c', NULL),`
	`(NULL, NULL), (NULL, NULL).`

`nome VARCHAR(20) UNIQUE,` `cognome VARCHAR(20) UNIQUE`	`('n','c'), ('n', NULL), (NULL, 'c'),` `(NULL, NULL), (NULL, NULL).`

2.5.3 Vincolo `PRIMARY KEY`

Il vincolo `PRIMARY KEY` identifica l'attributo che rappresenta la chiave primaria della relazione. Si usa una volta per tabella ed implica il vincolo `NOT NULL`.

Esistono due forme per la specifica:

- nella definizione dell'attributo, se è l'unico componente della chiave primaria;

```
39 matricola CHAR(6) PRIMARY KEY;
```

- come definizione separata a livello di tabella (vincolo di tabella), se invece la *chiave primaria* è composta da più attributi.

```
40 nome VARCHAR(20),
41 cognome VARCHAR(20),
42 PRIMARY KEY(nome, cognome);
```

2.5.4 Vincolo `CHECK`

Il vincolo `CHECK` specifica un vincolo generico che devono soddisfare le tuple della tabella. Un vincolo `CHECK` è soddisfatto se la sua espressione è vera o nulla.

```
43 CREATE TABLE Impiegato (
44 matricola CHAR(6) PRIMARY KEY,
45 nome VARCHAR(20) NOT NULL,
46 cognome VARCHAR(20) NOT NULL,
47 qualifica VARCHAR(20),
48 stipendio NUMERIC(8,2) DEFAULT 500.00 NOT NULL
49 CHECK( stipendio >= 0.0 ), --check di attributo
50 UNIQUE( cognome, nome ),
51 CHECK( nome <> cognome ) --check di tabella
52 );
```

2.6 Vincoli interrelazionali

Un vincolo di **integrità referenziale** (o interrelazionale) crea un legame tra i valori di un attributo, o di un insieme di essi, A della tabella corrente (detta interna/slave) e i valori di un attributo, o di un insieme di essi, B di un'altra tabella (detta esterna/master).

Il vincolo interrelazionale impone che, in ogni tupla della tabella interna, il valore di A, **se diverso dal valore nullo**, sia presente tra i valori di B nella tabella esterna. L'attributo B della tabella esterna deve essere soggetto a un vincolo `UNIQUE` o `PRIMARY KEY`.

Nota: l'attributo (o insieme di attributi) B può anche non essere la chiave primaria, ma deve essere identificante per le tuple della tabella esterna.

Le keyword `REFERENCES` e `FOREIGN KEY` permettono di definire vincoli di integrità referenziale.

2.6.1 Sintassi per singoli attributi

Se è coinvolto un singolo attributo, basta usare il costrutto sintattico `REFERENCES` nella dichiarazione dell'attributo, specificando la tabella esterna e il relativo attributo coinvolto.

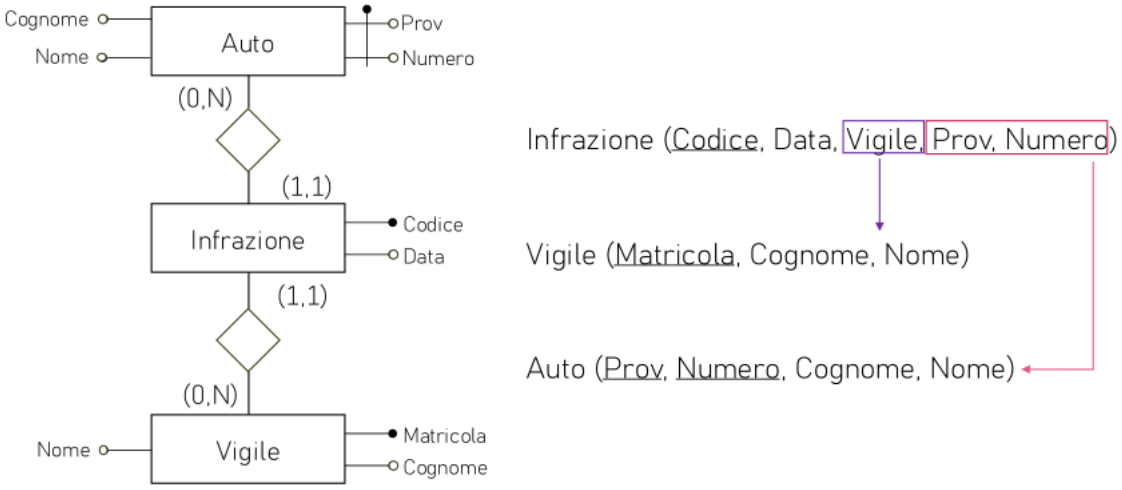
```
53 CREATE TABLE Impiegato (
54   Matricola = CHARACTER(6) PRIMARY KEY,
55   Nome = CHARACTER VARYING(20) NOT NULL,
56   Cognome = CHARACTER VARYING(20) NOT NULL,
57   Dipart = CHARACTER VARYING(15)
58         REFERENCES Dipartimento(NomeDip), -- singolo attributo
59   Ufficio = NUMERIC(3),
60   Stipendio = NUMERIC(9) DEFAULT 0,
61   UNIQUE (Cognome, Nome)
62 );
```

2.6.2 Sintassi per più attributi

Quando il legame coinvolge più attributi si usa il costrutto `FOREIGN KEY` alla fine della definizione degli attributi, elencando gli attributi coinvolti e seguito dalla definizioni dei corrispondenti attributi della tabella esterna, sempre grazie a `REFERENCES`.

```sql
CREATE TABLE Impiegato (
   Matricola = CHARACTER(6) PRIMARY KEY,
   Nome = CHARACTER VARYING(20) NOT NULL,
   Cognome = CHARACTER VARYING(20) NOT NULL,
   Dipart = CHARACTER VARYING(15)
         REFERENCES Dipartimento(NomeDip),
   Ufficio = NUMERIC(3),
   Stipendio = NUMERIC(9) DEFAULT 0,
   FOREIGN KEY(Nome, Cognome) REFERENCES Anagrafica (Nome, Cognome) -- due
      attributi
);
```

2.7 Esempio di creazione di una tabella




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

2.8 Modifiche degli schemi

L'SQL offre delle primitive per modificare schemi già introdotti. Il comando `ALTER` effettua modifiche a domini e tabelle. Ad esempio, è possibile aggiungere una colonna a una relazione. Il comando `DROP (DOMAIN/TABLE)` permette di rimuovere i vari componenti di domini e tabelle. Ad esempio è possibile cancellare una tabella.

2.8.1 Comando `ALTER TABLE`

La struttura di una tabella si può modificare dopo la sua creazione con questo comando. Le modifiche più comuni sono:

- Aggiunta di un nuovo attributo `ADD COLUMN`

```
-- Sintassi
ALTER TABLE tabella ADD COLUMN attributo tipo;
-- Esempio
ALTER TABLE impiegato ADD COLUMN stipendio NUMERIC(8,2);
```

- Rimozione di un attributo `DROP COLUMN`

```
-- Sintassi
ALTER TABLE tabella DROP COLUMN attributo;
```

- Modifica valore di default di un attributo `ALTER COLUMN`

```
-- Sintassi
ALTER TABLE tabella ALTER COLUMN attributo { SET DEFAULT valore |
   DROP DEFAULT};
-- Esempio
ALTER TABLE impiegato ALTER COLUMN stipendio SET DEFAULT 1000.00;
```

2.9 Cancellazione dati in una tabella

Le tuple di una tabella vengono cancellate con il comando `DELETE`:

```
DELETE FROM tabella [WHERE condizione ];
```

La **condizione** è un'espressione booleana che seleziona quali tuple cancellare. Se `WHERE` non è presente, tutte le tuple saranno cancellate, infatti una tabella viene cancellata con il comando `DROP TABLE`:

```
DROP TABLE tabella;
```

Esempi

```
-- Cancellazione di una tupla
DELETE FROM impiegato   WHERE matricola = 'A001';

-- Cancellazione di una tabella
DROP TABLE impiegato;
```