Insegnamento di Laboratorio di Basi di Dati

Lezione 1: Introduzione a PostgreSQL e a SQL

Roberto Posenato

ver. 2.4, 08/01/2018

Sommario

Modulo di “Laboratorio di Basi di Dati”

Obiettivo

- Introdurre all’uso di un Relational Database Management System (RDBMS) e allo sviluppo di applicazioni web basate su basi di dati con un approccio applicativo.

Modulo di “Laboratorio di Basi di Dati”

Prerequisiti

- Saper programmare in un linguaggio orientato agli oggetti (Java ad esempio).
- Conoscere i concetti spiegati nel corso di teoria di basi di dati. In questo corso si assume che lo studente conosca il linguaggio SQL.

Modulo di “Laboratorio di Basi di Dati”

Programma

1. Introduzione al *relational data base management system (RDBMS)* PostgreSQL.
2. Introduzione all’uso di SQL-2 in PostgreSQL (con estensioni).
3. Ottimizzazione di query.
4. Introduzione alle transazioni.
5. Introduzione al linguaggio Python.
6. Accesso alla basi dati da programmi Java o Python.
7. Introduzione al microframework Flask (Python) e allo sviluppo di semplici applicazioni web basate su basi di dati.

Organizzazione del modulo (attività formativa)

Il modulo è di **3** crediti.

In aula

36 ore di lezioni frontali suddivise in 18 lezioni in **9** settimane. Ciascuna settimana prevede:

- 1 lezione di 2 ore in aula.
- 1 lezione di 2 ore in laboratorio.

Nel mese di gennaio si svolgono 8 lezioni. Le rimanenti 10 saranno svolte a partire da Marzo.

Lavoro individuale

40 ore circa di studio personale in cui si richiede di studiare i concetti visti in aula, svolgere gli esercizi e approfondire eventuali temi proposti dal docente.

Organizzazione del modulo (attività formativa)

Modalità d'esame

- Quest'anno c'è la possibilità di svolgere prove parziali a Febbraio e a Giugno. Chi supera entrambe le prove con 18, ha come voto di laboratorio la media aritmetica dei voti delle prove parziali.
- Chi non supera o non fa le prove parziali, ha 3 appelli: Luglio, Settembre e Febbraio.
- L'esame di un appello consiste in una *prova scritta* da tenersi in un giorno diverso dalla prova scritta del modulo di teoria del corso.

Prova scritta: Risoluzione di esercizi in ordine crescente di difficoltà.

Organizzazione del modulo (attività formativa)

Materiale didattico

Non ci sono testi da acquistare ma manuali e/o documenti disponibili in rete presso i siti:

- PostgreSQL (http://www.postgresql.org).
- PostgreSQL Tutorial (http://www.tutorialspoint.com/postgresql/)
- Python (https://www.python.org/).
- Flask (http://flask.pocoo.org/).

Organizzazione del modulo (attività formativa)

Aggiornamenti

Il programma ufficiale, i riferimenti bibliografici, gli avvisi e il link al sito di eLearning relativi all'insegnamento sono pubblicati alla pagina:

- http://www.di.univr.it/?ent=oi&codiceCs=S24&codins=12700&cs=420&aa=2017%2F2018 per il corso di laurea in Informatica.
- http://www.di.univr.it/?ent=oi&codiceCs=S23&codins=10099&cs=419&aa=2017%2F2018 per il corso di laurea in Bioinformatica.

Si comincia. . .

PostgreSQL

- Si pronuncia post-gress-chiu-el.
- È un *relational data base management system (RDBMS o DBMS)* con alcune funzionalità orientate agli oggetti.
- È multipiattaforma (Linux,Mac OS X, Windows, ecc.), di pubblico dominio, gestita da un gruppo di volontari.
- Il sito ufficiale è `http://www.postgresql.org`
- Rappresenta un'alternativa *open source* ai DBMS di Oracle, Microsoft, etc.
- Implementa gran parte dello standard SQL:
  - la versione 9.6 implementa quasi tutte le funzioni di SQL:2011.
  - `https://www.postgresql.org/docs/9.6/static/unsupported-features-sql-standard.html` indica le funzioni NON implementate di SQL:2011.
  - In sintesi: SQL-92 (SQL2) è completamente implementato. SQL:1999 (SQL3) è implementato nelle principali funzionalità.
- In PostgreSQL la integrità dei dati e l'affidabilità sono la priorità.

PostgreSQL
versus Oracle

- Un confronto tra PostgreSQL e Oracle è disponibile:
  - http://db-engines.com/en/system/Oracle%3BPostgreSQL
  - http://www.sql-workbench.net/dbms_comparison.html
- In sintesi, le linee-guida per scegliere tra i due possono essere:

PostgreSQL:

- Il budget è limitato/piccolo.
- Si prevede di usare procedure complesse personalizzate.
- La base di dati sarà grande e complessa, con alto livello di concorrenza e tipi diversi di query.
- Si prevedono molte operazioni di scrittura e la velocità delle query di lettura non è molto importante.
- Si sta sviluppando un prodotto per sviluppatori.

Oracle:

- L'assistenza software da parte di una società è necessaria.
- Si richiede una flessibilità in termini di controllo transazioni.
- Si prevede di installare base di dati enormi.
- Si richiede un alto livello di scalabilità.
- Si vuole avere una base di dati indipendente dal sistema operativo.

PostgreSQL

- L'interazione tra un utente (programmatore/utente finale) e le basi di dati gestite da PostgreSQL avviene secondo il modello client-server.

Server

Il *daemon* `postmaster` supervisiona tutti i processi specifici di PostgreSQL.

Client(s)

Qualsiasi programma in grado di gestire l'iterazione con `postmaster`: input comandi, invio al `postmaster`, visualizzazione esito comando, ecc.

- Il client standard è `psql`, un'applicazione a linea di comando.
- Un'alternativa interessante è *pgAdmin IV*, `http://www.pgadmin.org/`, client grafico più intuitivo.

- Per le esercitazioni in laboratorio si usa il server PostgreSQL `dbserver.scienze.univr.it`.
- Il server è un PostgreSQL 9.6.
- Ogni studente iscritto al III anno ha a disposizione una base di dati personale per poter svolgere le esercitazioni.
- La base di dati personale è accessibile usando le proprie credenziali GIA e deve essere attivata prima di poterla usare.
- Accedere al sito web `http://dbserver.scienze.univr.it` con le proprie credenziali GIA e seguire le istruzioni per attivare la propria base di dati.
- La base di dati personale è disponibile fino al 28/02/2019.

Ci si connette al server tramite l'applicazione `psql`:

```
psql -h <server> -U <username> [-d] <database>
```

Esempio 1 (Esempio inizio sessione con il client psql)

```
$ psql -h dbserver.scienze.univr.it -U id051563 id051563
Inserisci la password per l'utente id051563:
psql (9.6.1)
connessione SSL (protocollo: TLSv1.2, cifrario:
     ECDHE-RSA-AES256-GCM-SHA384, bit: 256, compressione:
     disattivato)
Digita "help" per avere un aiuto.

id051563=>
```

- `psql` è un interprete iterativo da terminale.
- `psql` accetta due tipi di comando: comandi di tipo *Structured Query Language (SQL)* 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.

Esempio 2 (Esempio comando help)

```
id051563=> help
Stai utilizzando psql, l'interfaccia a riga di comando di
     PostgreSQL.
Digita:  \copyright per le condizioni di distribuzione
         \h per la guida sui comandi SQL
         \? per la guida sui comandi psql
         \g o termina con punto e virgola per eseguire la query
         \q per uscire
id051563=>
```

Esempio 3 (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 4 (Comando SQL `CREATE`)

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

- Structured Query Language (SQL) è il linguaggio più diffuso per i RDBMS.
- SQL è stato definito negli anni 70 ed è stato standardizzato negli anni 80 e 90 da ISO e IEC. Vedere https://en.wikipedia.org/wiki/SQL per riferimenti.
- SQL è composto di diverse parti:
  - Linguaggio per la definizione delle strutture dati e dei vincoli di integrità (*Data Definition Language (DDL)*).
  - Linguaggio per manipolare i dati: inserimento, aggiornamento e cancellazione (*Data Manipulation Language (DML)*).
    - Linguaggio per interrogare: (*Data Query Language (DQL o QL)*).
  - Linguaggio per controllare la base di dati (*Data Control Language (DCL)*).

In questo modulo considereremo solo una parte di SQL, sufficiente per creare e gestire semplici basi di dati:

- Costrutti fondamentali del *Data Definition Language (DDL)*, come `CREATE TABLE`, `ALTER TABLE`,...
- Costrutti fondamentali del *Data Manipulation Language (DML)*, come `INSERT`, `UPDATE`, `DELETE`,...
  - Costrutti fondamentali del *Data Query Language (DQL o QL)*, come `SELECT`, `SELECT` con `JOIN`, `SELECT` **innestate**,...

Nota!

Si consiglia di leggere almeno una volta e di considerare come fonte principale di riferimento la sezione SQL del manuale utente del DBMS che si usa.
Per PostgreSQL, il manuale è alla pagina
http://www.postgresql.org/docs/9.6/interactive/

L'istruzione principale del DDL è `CREATE TABLE` [cap. 5.1 5.2]:

- Definisce lo schema di una relazione (o tabella) e ne crea un'istanza vuota.
- Specifica attributi, domini (tipo) e vincoli.

Sintassi

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

Notazione: [e] indicano parti opzionali; { e } con | indicano alternative da cui scegliere una; ... indicano che elemento precedente si può ripetere.

Esempio 5

```
-- Il seguente comando crea la tabella products!
CREATE TABLE products (
    productNo INTEGER PRIMARY KEY,
    nameId INTEGER NOT NULL,
    -- Si accettano solo prezzi positivi.
    price NUMERIC DEFAULT 9.99
        CONSTRAINT pPrice NOT NULL CHECK (price>0),
    FOREIGN KEY (nameId)
        REFERENCES productNames(id)
);
```

- Già con l'istruzione `CREATE TABLE` si devono conoscere i concetti fondamentali del DDL: identificatori, domini (tipo) e vincoli.
- **Identificatori SQL**: stringhe che iniziano con una lettera (UTF-8) o un underscore (_) e che possono contenere anche cifre (0-9).
- Gli identificatori SQL non sono sensibili al minuscolo/maiuscolo: `idPersona` e `idpersona` rappresentano un solo identificatore.
- Tutti i nomi di tabella, attributo, ecc. sono identificatori SQL.

- Domini elementari (predefiniti) [cap. 8]:
  - **carattere** (`CHAR`/`VARCHAR`): singoli caratteri o stringhe anche di lunghezza variabile.
  - **bit** (`BOOLEAN`/`BIT`): bit singoli (booleani) o stringhe di bit.
  - **numerici esatti** (`INTEGER`/`NUMERIC`/`serial`) e approssimati (`FLOAT`/`REAL`).
  - **date** (`DATE`), orario (`TIME`/`TIMESTAMP`) e intervalli di tempo (`INTERVAL`).
- Domini definiti dall'utente.

- Permette di rappresentare singoli caratteri oppure stringhe.
- Un carattere o stringa di caratteri sono rappresentati tra apici (’): `’a’`, `’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 assegna una stringa di lunghezza inferiore a 20, la stringa viene riempita di spazi fino a rendere la stringa lunga 20.
`CHARACTER VARYING(20)`: stringa di lunghezza variabile (max 20).
`TEXT`: stringa di lunghezza variabile senza limite fissato. Questa è un’estensione PostgreSQL.

- Forme abbreviate: `CHAR` e `VARCHAR(20)`.

- Tipicamente usato per rappresentare attributi, detti *flag*, che specificano se un oggetto rappresentato da una tupla possiede o meno una certa proprietà.
- I valori bit sono 0 e 1, rappresentati come `B'1'`/`B'0'`.
- Si può anche definire un dominio *stringa di bit*:
  `BIT [VARYING][(lunghezza)]`

`BIT`: bit singolo.
`BIT(20)`: stringa di bit lunghezza fissa.
`BIT VARYING(20)`: stringa di bit lunghezza variabile (max 20).

- Forme abbreviate: `VARBIT`.

- Permette di rappresentare i valori booleani `TRUE`/`FALSE`, rappresentati anche come `T`/`F`, `1`/`0`, `YES`/`NO`, più lo stato *unknown* (valore nullo), rappresentato come `NULL`.

`BOOLEAN`: valore booleano singolo.

- Forme abbreviate: `BOOL`.

Nota!

NON si possono fare stringhe di boolean, ma solo di bit.

- Permettono di rappresentare valori interi o valori decimali in virgola fissa.
- Valori interi:
  - `SMALLINT`: valori interi (2 bytes: da -32.768 a +32.767). Equivalente al tipo `short` in Java.
  - `INTEGER`: valori interi (4 bytes: da -2^{31} to $+2^{31} - 1$). Equivalente al tipo `int` in Java.
- Valori decimali:
  - `NUMERIC [(precisione [, scala])]`: precisione è il numero totale di cifre significative (cioè di cifre a sinistra e a destra della virgola), scala è il numero di cifre dopo la virgola.
  - `DECIMAL [(precisione [, scala])]`: equivalente al precedente.
  - I tipi `NUMERIC`/`DECIMAL` sono equivalenti al tipo `java.math.BigDecimal` in Java.
  - Esempio: `NUMERIC(5,2)` permette di rappresentare valori come 100.01, 100.999 (arrotondato a 101.00), ma non valori come 1000.01!

Permettono di rappresentare valori in virgola mobile:

- `REAL`: una precisione di 6 cifre decimali.
- `DOUBLE PRECISION`: una precisione di 15 cifre decimali.

Nota!

- `REAL` e `DOUBLE PRECISION` sono comunque valori approssimati.
- Sono implementati secondo lo standard IEEE Standard 754 for Binary Floating-Point Arithmetic.
- Se si devono rappresentare importi di denaro che contengono anche decimali, MAI usare questi tipi ma usare `NUMERIC(precisione)`!

Permettono di rappresentare istanti di tempo.

- `DATE`: istanti del tipo (anno, mese, giorno).
- Un valore `DATE` si rappresenta tra apici (') e lo standard ISO prescrive il formato `YYYY-MM-DD`.
- `TIME [(precisione)][WITH TIME ZONE]`: istanti del tipo (hour, minute, second).
  - precisione: numero cifre decimali per rappresentare frazioni del secondo;
  - `WITH TIME ZONE`: permette di specificare anche `[+-]hour:minute`, che rappresentano la differenza con l'ora di Greenwich, o `nomeFusoOrario`.
- `TIMESTAMP [(precisione)][WITH TIME ZONE]`: date+time.

`DATE`: '2016-01-15'
`TIME (3)`: '04:05:06.789'
`TIME WITH TIME ZONE`: '04:05:06-08:00' o '12:01:01 CET'
`TIMESTAMP WITH TIME ZONE`: '2016-01-24 00:00:00+01'

- È possibile definire dei domini specifici assegnando vincoli sui domini di base:

Sintassi

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

- *vincolo*: è un vincolo `CHECK(condizione)` dove la *condizione* è un'espressione booleana.

Esempio 6 (Definizione di un tipo enumerato)

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

- Vincoli di attributo/intrarelazionali specificano proprietà che devono essere soddisfatte da ogni tupla di una singola relazione della base di dati [cap. 5.3].

Sintassi vincoli di attributo

```
[ CONSTRAINT vincolo ]
{ NOT NULL |
  CHECK (espressione) [ NO INHERIT ] |
  DEFAULT valore |
  UNIQUE |
  PRIMARY KEY |
  REFERENCES tabella [ (attributo) ]
    [ ON DELETE azione ] [ ON UPDATE azione ] }
```

- Vincoli di tabella:

Sintassi vincoli di tabella

```
[ CONSTRAINT vincolo ]
{ CHECK (espressione) [ NO INHERIT ] |
  UNIQUE (attributo [, ... ]) |
  PRIMARY KEY (attributo [, ... ]) |
  FOREIGN KEY (attributo [, ... ])
     REFERENCES reftable [ (refcolumn [, ... ]) ]
    [ ON DELETE azione ] [ ON UPDATE azione ] }
```

- 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 7

```
nome VARCHAR(20) NOT NULL
cognome VARCHAR(20) NOT NULL DEFAULT ''
```

- Il vincolo `UNIQUE` impone che i valori di un attributo (o di un insieme di attributi) siano una superchiave.
- Si può definire su:
  - un solo attributo;
  - 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), ('n', NULL), (NULL, NULL), (NULL, NULL).

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

- Il vicolo `PRIMARY KEY` identifica l'attributo che rappresenta la chiave primaria della relazione:
- Si usa una sola volta per tabella.
- Implica il vincolo `NOT NULL`.
- Due forme di specifica:
  - nella definizione dell'attributo, se è l'unico componente della chiave primaria:

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

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

```
nome    VARCHAR(20),
cognome VARCHAR(20),
PRIMARY KEY(nome, cognome)
```

- 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.
- In molti casi, un'espressione è nulla se uno degli operandi è nullo.
- Conviene mettere sempre NOT NULL insieme al vincolo CHECK()!

Esempio 8

```
CREATE TABLE Impiegato (
    matricola CHAR(6) PRIMARY KEY,
    nome VARCHAR(20) NOT NULL,
    cognome VARCHAR(20) NOT NULL,
    qualifica VARCHAR(20),
    stipendio NUMERIC(8,2) DEFAULT 500.00 NOT NULL
      CHECK (stipendio >= 0.0),--check di attributo
    UNIQUE(cognome, nome),
    CHECK (nome <> cognome)--check di tabella
);
```

- La struttura di una tabella si può modificare dopo la sua creazione con il comando `ALTER TABLE` [cap. 5.5]. Di seguito, le modifiche più comuni:
- Aggiunta di un nuovo attributo (`ADD COLUMN`):

Sintassi

```
ALTER TABLE tabella ADD COLUMN attributo tipo;
```

```
ALTER TABLE impiegato ADD COLUMN stipendio NUMERIC(8,2);
```

- Rimozione di un attributo:

Sintassi

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

- Modifica valore di default di un attributo:

Sintassi

```
ALTER TABLE tabella ALTER COLUMN attributo
    { SET DEFAULT valore | DROP DEFAULT };
```

```
ALTER TABLE impiegato ALTER COLUMN stipendio
    SET DEFAULT 1000.00;
```

Una tabella viene popolata con il comando `INSERT INTO` [cap. 6.1]:

Sintassi

```
INSERT INTO tabella [(elencoAttributi)]
   VALUES (elencoValori) [, ...];
```

Ogni `INSERT` aggiunge una o più tuple (righe) in una tabella.

Esempio 9

```
INSERT INTO impiegato (matricola, nome, cognome)
   VALUES ('A00001', 'Mario', 'Rossi'),
          ('A00002', 'Luca', 'Bianchi');

INSERT INTO impiegato VALUES ('A00003','Ivo','Bo');
```

Il secondo INSERT è corretto ma poco robusto nella pratica!

Una tupla di una tabella può essere modificata con il comando `UPDATE`:

Sintassi [cap. 6.2]

```
UPDATE tabella
    SET attributo = espressione [, ... ]
    [ WHERE condizione ];
```

`condizione` è una espressione booleana che seleziona quali righe aggiornare.

Se `WHERE` non è presente, tutte le tuple saranno aggiornate.

```
UPDATE impiegato
    SET stipendio = stipendio * 1.10
    WHERE nomeDipartimento = 'Vendite';
UPDATE impiegato
    SET telefono = '+39' || telefono;
```

L'operatore '||' concatena due espressioni e ritorna la stringa corrisp.

Le tuple di una tabella vengono cancellate con il comando `DELETE` [cap. 6.3]:

Sintassi

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

`condizione` è una espressione booleana che seleziona quali tuple cancellare. Se `WHERE` non è presente, tutte le tuple saranno cancellate.

```
DELETE FROM impiegato WHERE matricola = 'A001';
```

Una tabella viene cancellata con il comando `DROP TABLE`:

Sintassi

```
DROP TABLE tabella;
```

Dati iniziali

Si consideri il seguente schema relazionale:

`INGREDIENTE(Id, Nome, Calorie, Grassi, Proteine, Carboidrati);`
`COMPOSIZIONE(Ricetta, Ingrediente, Quantità)`
`RICETTA(Id, Nome, Regione, Porzioni, TempoPreparazione)`

La quantità di grassi, proteine e carboidrati è in grammi su 100 grammi di ingrediente; la quantità nella tabella `COMPOSIZIONE` si assume espressa in grammi. La unità di misura di `INGREDIENTE.Calorie` è kcal e quello di `Ricetta.TempoPreparazione` è min. Vincoli di integrità: `COMPOSIZIONE.Ricetta`$\rightarrow$`RICETTA`, `COMPOSIZIONE.Ingrediente`$\rightarrow$`INGREDIENTE`.

Quesito

Scrivere il codice PostgreSQL che generi le tabelle rappresentanti le relazioni. Si inseriscano tutti i possibili controlli di integrità e di correttezza dei dati. Si giustifichi, dove necessario, la scelta del dominio.

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

- Un vincolo di integrità referenziale si dichiara nella tabella interna e ha due possibili sintassi [cap. 5.3.5]:
  - `REFERENCES`, vincolo di attributo, da usare quando il vincolo è su un singolo attributo della tabella interna, $|A| = 1$.
  - `FOREIGN KEY`, vincolo di tabella, da usare quando il vincolo coinvolge più attributi della tabella interna, $|A| > 1$.

Il costrutto REFERENCES è usato quando il vincolo è su un singolo attributo della tabella interna, $|A| = 1$.

Sintassi REFERENCES

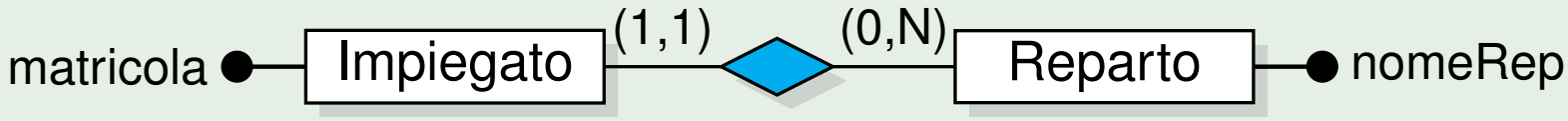
```
REFERENCES tab_esterna [ (attr_esterno) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE azione ] [ ON UPDATE azione ]
```

Esempio 10 (Dichiarazione dentro tabella)

```
CREATE TABLE Interna
    ...
    attributo VARCHAR(15)
        REFERENCES TabellaEsterna(chiave)
    ...
```

Esempio 11 (Completo)

1. Schema concettuale (solo con attributi chiave):



2. Schema relazionale:

   Impiegato(matricola, nome, cognome, nomeRep)

   Reparto(nomeRep, sede, telefono)

Esempio 12 (Completo)

3. Schema fisico:

```
CREATE TABLE Reparto (
    nomeRep    VARCHAR(20) PRIMARY KEY,
    sede       VARCHAR(20) NOT NULL,
    telefono   VARCHAR(20) NOT NULL
);
CREATE TABLE Impiegato (
    matricola  CHAR(6) PRIMARY KEY,
    nome       VARCHAR(20) NOT NULL,
    cognome    VARCHAR(20) NOT NULL,
    nomeRep VARCHAR(20) DEFAULT 'Acquisti'
        REFERENCES Reparto(nomeRep)
);
```

Tabella esterna

Vincolo obbl.: UNIQUE o PRIMARY KEY

Tabella interna

Chiave esportata

Tabella esterna

Chiave esterna

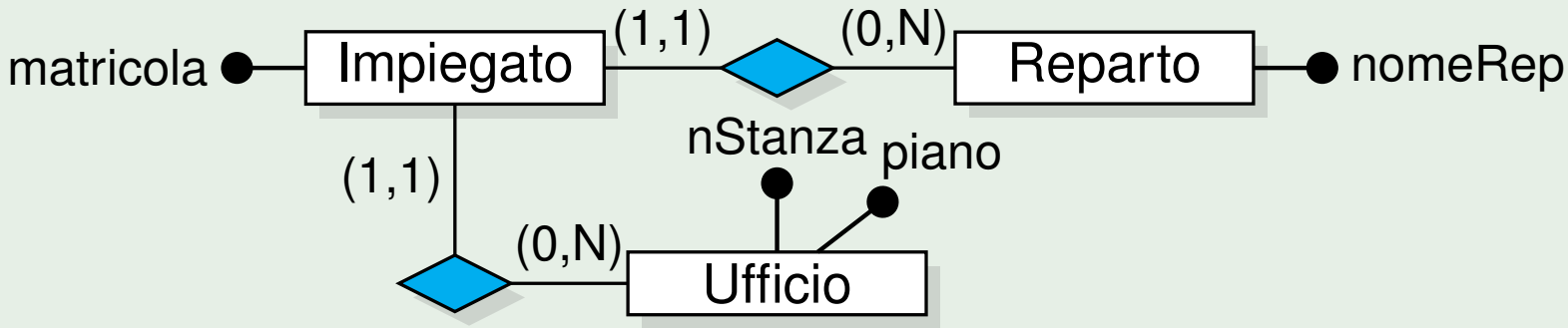
Il costrutto FOREIGN KEY è usato quando il vincolo di integrità referenziale è definito su un insieme di attributi della tabella interna.

Sintassi FOREIGN KEY

```
FOREIGN KEY (column_name [, ... ]) REFERENCES
     reftable [ (refcolumn [, ... ]) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [
     ON DELETE azione] [ ON UPDATE azione ]
```

Esempio 13 (Completo)

1. Schema concettuale (solo con attributi chiave):



2. Schema relazionale:

Impiegato(matricola, nome, cognome, piano, nStanza, nomeRep)
Ufficio(piano, nStanza, formato, finestre)
Reparto(nomeRep, sede, telefono)

Esempio 14 (Continuazione Esempio 3)

3. Schema fisico:

```
CREATE TABLE Reparto (
    nomeRep VARCHAR(20) PRIMARY KEY,...
);
CREATE TABLE Ufficio (
    piano VARCHAR(10) NOT NULL,
    nStanza INTEGER NOT NULL,...
     UNIQUE (piano, nStanza)
);
CREATE TABLE Impiegato (
    matricola CHAR(6) PRIMARY KEY,...
    piano VARCHAR(10),
    stanza INTEGER,
    nomeRep VARCHAR(20) DEFAULT 'Acquisti'
        REFERENCES Reparto(nomeRep),
    FOREIGN KEY (piano,stanza)
        REFERENCES Ufficio(piano, nStanza)
);
```

Tabella esterna

Vincolo obbl.: UNIQUE o PRIMARY KEY

Tabella interna

Chiave esportata

Tabella esterna

Chiave esterna (attributi ordinati)

... = ci sono altre dichiarazioni omesse per questioni di spazio.

- Un vincolo è *violato* quando la proprietà stabilita dal vincolo non è vera per una qualsiasi tupla.
- In generale, in un DBMS, gli inserimenti, gli aggiornamenti e le cancellazione vengono eseguite solo se nessun vincolo viene violato dall'operazione.

Esempio 15 (Blocco operazione per violazione vincolo intra-relazione)

Se si considera la dichiarazione di Reparto nell'Esempio12, l'operazione
INSERT INTO **Reparto** (**nomeRep**, **sede**) VALUES ('Vendita', 'Verona');
non è mai eseguita (=bloccata) perché l'inserimento di tale tupla violerebbe il vincolo di `NOT NULL` dell'attributo `telefono`.

- Caso a parte sono le violazioni dei vincoli di integrità referenziale.
- Per i vincoli di integrità referenziale è possibile associare una *politica di reazione (azione)* alle violazioni che permette di decidere cosa fare in caso di violazione per garantire che il vincolo torni a valere.
- La violazione di un vincolo di integrità referenziale può avvenire nella tabella interna (slave) e può essere originato:
  - o da operazioni di aggiornamento/inserimento nella tabella interna;
  - o da operazioni di aggiornamento/cancellazione nella tabella esterna.

Violazione nella tabella interna

- Una violazione di un vincolo di integrità referenziale può avvenire solo in due casi:
  - modificando il valore dell'attributo referente (chiave esportata) in una riga esistente.
  - inserendo una nuova tupla.

Politiche di reazione

In SQL queste violazioni sono considerate come violazioni di vincoli intra-relazione, quindi non è ammessa nessuna azione alternativa.

Le operazioni che possono violare i vincoli sono bloccate.

Esempio 16 (Violazione nella tabella interna. Vedi Esempio 6)

Impiegato			
matricola	**nome**	**cognome**	**NomeRep**
A0001	Mario	Rossi	Acquisti
A0002	Paolo	Verdi	Vendite

Reparto		
NomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000
Vendite	Verona	02 8027900

Se si tenta di inserire la tupla:

matricola	**nome**	**cognome**	**NomeRep**
A0003	Dante	Bianchi	Controllo

l'operazione fallisce perché il reparto Controllo non esiste nella tabella **Reparto**.

L'inserimento non viene effettuato!

Violazione originata dalla tabella esterna

- Una violazione di un vincolo di integrità referenziale può avvenire solo in due casi:
  - modificando il valore dell'attributo *riferito* (chiave esterna) in una riga esistente.
  - cancellando una tupla la cui chiave è usata nella tabella interna.

Fatto

La tabella interna (slave) deve essere adeguata alle modifiche che avvengono nella tabella esterna (master).

Politiche di reazione

In SQL si possono attivare diverse politiche di adeguamento della tabella interna.

Nota!

In sintesi: 1) operazioni sulla tabella slave NON possono violare i vincoli. 2) operazioni sulla tabella master possono violare dei vincoli che però sono

Politiche di reazione

Le politiche di reazione disponibili sono:

- `CASCADE`: la modifica del valore di un attributo riferito nella tabella master si propaga anche in tutte le righe corrispondenti nelle tabelle slave.
- `SET NULL`: la modifica del valore di un attributo riferito nella tabella master determina che in tutte le righe corrispondenti nelle tabelle slave il valore dell'attributo referente è posto a `NULL` (se ammesso).
- `SET DEFAULT`: la modifica del valore di un attributo riferito nella tabella master determina che in tutte le righe corrispondenti nelle tabelle slave il valore dell'attributo referente è posto al valore di default (se esiste).
- `NO ACTION`: indica che non si fa nessuna azione. Il vincolo però deve essere sempre valido. Quindi, la modifica del valore di un attributo riferito nella tabella master non viene effettuata.

Politica di reazione CASCADE

- La modifica del valore di un attributo riferito nella tabella master viene fatta automaticamente anche in tutte le righe corrispondenti nella tabella slave.
- La cancellazione della tupla con un attributo riferito nella tabella master comporta la cancellazione automatica di tutte le righe corrispondenti nella tabella slave.

Sintassi per attivare azione CASCADE

```
FOREIGN KEY (column_name [, ... ]) REFERENCES
    reftable [ (refcolumn [, ... ]) ]
  ON DELETE CASCADE ON UPDATE CASCADE
```

Esempio 17 (Modifica nella tabella esterna con azione CASCADE)

Reparto		
nomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000
Vendite	Verona	02 8027900

Impiegato			
matricola	**nome**	**cognome**	**NomeRep**
A0001	Mario	Rossi	Acquisti
A0002	Paolo	Verdi	Vendite

Si modifica il nome del reparto 'Vendite' in 'Controllo':

```
UPDATE Reparto SET nomeRep='Controllo' WHERE nomeRep='Vendite';
```

Le tabelle diventano:

Reparto		
nomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000
Controllo	Verona	02 8027900

Impiegato			
matricola	**nome**	**cognome**	**nomeRep**
A0001	Mario	Rossi	Acquisti
A0002	Paolo	Verdi	Controllo

Esempio 18 (Cancellazione nella tabella esterna con azione CASCADE)

Reparto		
nomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000
Vendite	Verona	02 8027900

Impiegato			
matricola	**nome**	**cognome**	**nomeRep**
A0001	Mario	Rossi	Acquisti
A0002	Paolo	Verdi	Vendite

Si elimina il reparto 'Vendite':

```
DELETE FROM Reparto WHERE nomeRep = 'Vendite';
```

Le tabelle diventano:

Reparto		
nomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000

Impiegato			
matricola	**nome**	**cognome**	**NomeDip.**
A0001	Mario	Rossi	Acquisti

Politica di reazione SET NULL

- La modifica del valore di un attributo riferito nella tabella master determina che in tutte le righe corrispondenti nella tabella slave il valore dell'attributo referente è posto a NULL (se ammesso).
- Lo stesso accade anche in caso di cancellazione della tupla con un attributo riferito nella tabella master.

Sintassi per attivare azione SET NULL

```
FOREIGN KEY (column_name [, ... ]) REFERENCES
    reftable [ (refcolumn [, ... ]) ]
  ON DELETE SET NULL ON UPDATE SET NULL
```

Esempio 19 (Modifica nella tabella esterna con azione SET NULL)

Reparto		
nomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000
Vendite	Verona	02 8027900

Impiegato			
matricola	**nome**	**cognome**	**NomeDip.**
A0001	Mario	Rossi	Acquisti
A0002	Paolo	Verdi	Vendite

Si modifica il nome del reparto 'Vendite' in 'Controllo':

```
UPDATE Reparto SET nomeRep='Controllo' WHERE nomeRep='Vendite';
```

Le tabelle diventano:

Reparto		
nomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000
Controllo	Verona	02 8027900

Impiegato			
matricola	**nome**	**cognome**	**nomeRep**
A0001	Mario	Rossi	Acquisti
A0002	Paolo	Verdi	

Esempio 20 (Cancellazione nella tabella esterna con azione SET NULL)

Reparto		
nomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000
Vendite	Verona	02 8027900

Impiegato			
matricola	**nome**	**cognome**	**NomeRep**
A0001	Mario	Rossi	Acquisti
A0002	Paolo	Verdi	Vendite

Si elimina il reparto 'Vendite':

```
DELETE FROM Reparto WHERE nomeRep = 'Vendite';
```

Le tabelle diventano:

Reparto		
nomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000

Impiegato			
matricola	**nome**	**cognome**	**NomeRep**
A0001	Mario	Rossi	Acquisti
A0002	Paolo	Verdi	

Politica di reazione SET DEFAULT

- La modifica del valore di un attributo riferito nella tabella master determina che in tutte le righe corrispondenti nella tabella slave il valore dell'attributo referente è posto al valore di default.
- Lo stesso accade anche in caso di cancellazione della tupla con un attributo riferito nella tabella master.

Sintassi per attivare azione SET DEFAULT

```
FOREIGN KEY (column_name [, ... ]) REFERENCES
    reftable [ (refcolumn [, ... ]) ]
  ON DELETE SET DEFAULT ON UPDATE SET DEFAULT
```

Esempio 21 (Modifica nella tabella esterna con azione SET DEFAULT)

Reparto		
nomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000
Vendite	Verona	02 8027900

Impiegato			
matricola	**nome**	**cognome**	**NomeRep**
A0001	Mario	Rossi	Acquisti
A0002	Paolo	Verdi	Vendite

Si modifica il nome del reparto 'Vendite' in 'Controllo':

```
UPDATE Reparto SET nomeRep = 'Controllo' WHERE nomeRep = 'Vendite';
```

Le tabelle diventano:

Reparto		
nomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000
Controllo	Verona	02 8027900

Impiegato			
matricola	**nome**	**cognome**	**NomeRep**
A0001	Mario	Rossi	Acquisti
A0002	Paolo	Verdi	Acquisti

Esempio 22 (Cancellazione nella tabella esterna con azione SET DEFAULT)

Reparto		
nomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000
Vendite	Verona	02 8027900

Impiegato			
matricola	**nome**	**cognome**	**NomeRep**
A0001	Mario	Rossi	Acquisti
A0002	Paolo	Verdi	Vendite

Si elimina il reparto 'Vendite':

DELETE FROM **Reparto** WHERE **nomeRep** = 'Vendite';

Le tabelle diventano:

Reparto		
nomeRep	**sede**	**telefono**
Acquisti	Verona	02 8020000

Impiegato			
matricola	**nome**	**cognome**	**NomeDip.**
A0001	Mario	Rossi	Acquisti
A0002	Paolo	Verdi	Acquisti

Politica di reazione NO ACTION

- Indica che non si fa nessuna azione. Il vincolo però deve essere sempre valido. Quindi, la modifica del valore di un attributo riferito nella tabella master non viene effettuata.
- Lo stesso accade anche in caso di cancellazione della tupla con un attributo riferito nella tabella master: non viene effettuata.

Sintassi per attivare azione SET NULL

```
FOREIGN KEY (column_name [, ... ]) REFERENCES
    reftable [ (refcolumn [, ... ]) ]
  ON DELETE NO ACTION ON UPDATE NO ACTION
```

Sommario 1/2

- Le azioni possono essere combinate: non è necessario usare la stessa azione sia con `ON UPDATE` sia con `ON DELETE`.
- Un'*azione* è: {`NO ACTION`|`CASCADE`|`SET NULL`|`SET DEFAULT`}
- Sintassi vincolo di attributo:

```
[ CONSTRAINT nome_vincolo ]
{ NOT NULL |
  CHECK (espressione) [ NO INHERIT ] |
  DEFAULT valore |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters |
  REFERENCES tabella [ (attributo) ]
    [ ON DELETE azione ] [ ON UPDATE azione ]
}
```

Sommario 2/2

- Le azioni possono essere combinate: non è necessario usare la stessa azione sia con `ON UPDATE` sia con `ON DELETE`.
- Un'*azione* è: {`NO ACTION`|`CASCADE`|`SET NULL`|`SET DEFAULT`}
- Sintassi vincolo di tabella:

```
[ CONSTRAINT nome_vincolo ]
{ CHECK (expression) [ NO INHERIT ] |
  UNIQUE (column_name [, ...]) index_parameters |
  PRIMARY KEY (column_name [, ... ])
     index_parameters |
  FOREIGN KEY (column_name [, ... ]) REFERENCES
     reftable [(refcolumn [, ... ])]
    [ ON DELETE azione ] [ ON UPDATE azione ]
}
```

```
DROP TABLE IF EXISTS Reparto CASCADE;
CREATE TABLE Reparto (
    nomeRep VARCHAR(20) PRIMARY KEY,...
);
DROP TABLE IF EXISTS Ufficio CASCADE;
CREATE TABLE Ufficio (
    piano   VARCHAR(10) NOT NULL,
    nStanza INTEGER NOT NULL,
    UNIQUE (piano, nStanza)
);
DROP TABLE IF EXISTS Impiegato CASCADE;
CREATE TABLE Impiegato (
    matricola CHAR(6) PRIMARY KEY,...
    piano  VARCHAR(10), stanza INTEGER,
    nomeRep VARCHAR(20) DEFAULT 'Vendite'
        REFERENCES Reparto(nomeRep) ON DELETE SET DEFAULT
        ON UPDATE CASCADE,
    FOREIGN KEY (piano, stanza)
        REFERENCES Ufficio(piano,nStanza) ON DELETE SET NULL
        ON UPDATE CASCADE
);
```

Si possono aggiungere/rimuovere anche dopo la creazione della tabella.

Sintassi aggiunta/rimozione vincolo di integrità referenziale

- Per aggiungere:

  `ALTER TABLE nome_tabella ADD CONSTRAINT` *def_vincolo*;

  def_vincolo = dichiarazione vincolo di tabella.
- Per rimovere:

  `ALTER TABLE nome_tabella DROP CONSTRAINT nome_vincolo;`

  Il `nome_vincolo` è il nome dato durante la dichiarazione. Se non si definisce un nome (come in questa lezione), il DBMS ne assegna uno.

In PostgreSQL, "`\d nome_tabella`" mostra la struttura della tabella con i nomi dei vincoli.

```
#\d impiegato
Table "public.impiegato"
      Column       |          Type          | Modifiers
-------------------+------------------------+-----------------
 matricola         | character(6)           | not null
 ...
 piano             | character varying(10)  | not null
 stanza            | integer                | not null
 nomeRep           | character varying(20)  | default 'Vendite'
Indexes:
    "impiegato_pkey" PRIMARY KEY, btree (matricola)
Foreign-key constraints:
    "impiegato_nomerep_fkey" FOREIGN KEY (nomerep) REFERENCES
     Reparto(nomerep) ON UPDATE...
    "impiegato_piano_fkey" FOREIGN KEY (piano, stanza) REFERENCES
     ufficio(piano, nstanza) ON UPDATE CASCADE ON DELETE SET NULL
#ALTER TABLE IMPIEGATO DROP CONSTRAINT impiegato_piano_fkey;
ALTER TABLE
#
```