Breve Introduzione a SQL (con Postgres)

Marco Bernasocchi

[marco@opengis.ch](mailto:marco@opengis.ch)

Partial update 2018

Claudio Rocchini

rockini@tele2.it

Gennaio 2010

Indice

|  |  |  |  |
| --- | --- | --- | --- |
| [1](#page2) | [Introduzione](#page2) | | 2 |
| [2](#page2) | [Postgres](#page2) | | 2 |
| [3](#page3) | [Prepararsi al Lavoro](#page3) | | 3 |
| [4](#page6) | [La nestra di comandi SQL](#page6) | | 6 |
| [5](#page8) | [Pre-Introduzione al comando SELECT](#page8) | | 8 |
| [6](#page8) | [Valori letterali](#page8) | | 8 |
| [7](#page10) | [Tipi di dato](#page10) | | 10 |
| [8](#page10) | [De nizione dei Dati](#page10) | | 10 |
|  | [8.1](#page11) | [Creazione di una tabella](#page11) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 11 |
|  | [8.2](#page11) | [Analisi di una tabella](#page11) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 11 |
|  | [8.3](#page12) | [Distruzione di una tabella](#page12) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 12 |
|  | [8.4](#page12) | [Commenti al codice](#page12) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 12 |
|  | [8.5](#page13) | [Creazione avanzata di una tabella](#page13) . . . . . . . . . . . . . . . . . . . . . . . . . | 13 |
|  | [8.6](#page13) | [Modi ca della struttura di una tabella](#page13) . . . . . . . . . . . . . . . . . . . . . . | 13 |
| [9](#page14) | [Manipolazione dei dati](#page14) | | 14 |
|  | [9.1](#page14) | [Inserimento di dati](#page14) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 14 |
|  | [9.2](#page15) | [Il valore NULL](#page15) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 15 |
|  | [9.3](#page15) | [Test dei vincoli](#page15) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 15 |
|  | [9.4](#page16) | [Cancellazione di dati](#page16) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 16 |
|  | [9.5](#page17) | [Modi ca dei dati](#page17) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 17 |
| [10](#page18) | [Le relazioni](#page18) | | 18 |
|  | [10.1](#page18) | [De nizione di una relazione](#page18) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 18 |

1

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

|  |  |  |  |
| --- | --- | --- | --- |
| [11](#page19) | [Indici](#page19) | | 19 |
| [12](#page20) | [Le interrogazioni: SELECT](#page20) | | 20 |
|  | [12.1](#page20) | [Forma semplice di SELECT](#page20) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 20 |
|  | [12.2](#page22) | [Aggregazioni di righe](#page22) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 22 |
|  | [12.3](#page23) | [Join](#page23) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 23 |
| [13](#page24) | [Viste](#page24) | | 24 |
| [14](#page25) | [Editor gra ci di query](#page25) | | 25 |
| [15](#page26) | [Basi di dati reali](#page26) | | 26 |
| [16](#page26) | [Conclusioni](#page26) | | 26 |

* Introduzione

SQL (sigla che sta per Structured Query Language) e un linguaggio testuale standard per ope-rare con le basi di dati. Standard vuol dire che e (quasi) indipendente la particolare database scelto (Oracle, Microsoft SQL Server, Postgres, Mysql, etc.). Il linguaggio e funzionale (un solo costrutto esegue le operazioni speci cate), non imperativo (non ci sono variabili o elenchi di operazioni), anche se una sua estensione (il PL-SQL) permette di dichiarare funzioni in modo imperativo. Un'introduzione al linguaggio richiederebbe un corso annuale: in questa breve nota si vuole dare una breve descrizione alla struttura del linguaggio, in modo che poi sia possibile introdurne la parte propriamente spaziale. Inizieremo col vedere gli elementi di base (tipi di dato: numeri e parole), passeremo quindi alla de nizione dei dati (schemi, colon-ne e tabelle), alle operazioni di inserimento e modi ca dei dati, quindi all'interrogazione degli stessi. Per nire faremo un breve accenno agli elementi avanzati: indici, chiavi e relazioni.

* Postgres

Tutte le esercitazioni verranno e ettuate sul database Postgres, un database gratuito che ha un supporto spaziale molto ben sviluppato. Un altro database che ha un ottimo supporto spaziale e Oracle, che pero e un prodotto commerciale ed anche piuttosto caro.

Saltiamo la fase di installazione del software che non ci interessa e passiamo direttamente al suo utilizzo. Per accedere al database utilizzeremo l'interfaccia base, denominata pgAdmin III, mentre per visualizzare i dati geogra ci in modo grafico occorrera utilizzare un qualche GIS (es. QGis, UDig, Grass, etc.).

Nota: Per scopi dididattici, e opportuno che ciascuno abbia installato il proprio server Postgres personale. In realt l'utilizzo tipico di Postgres come database enterprise e quello di installare un server unico, in modo tale che tutti gli utenti operino su dati condivisi ed in modo concorrente. Si e deciso un'installazione separata per ciascuno per evitare di dover fare una serie di utenti diversi, questo per evitare con itti con la creazione delle tabelle (per ogni database deve esistere una sola tabella per ogni nome).

2

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

* Prepararsi al Lavoro

Lanciate Postgres - pgAdmin III: vi apparir l a nestra in Fig. [1](#page3). Se non avete la con - gurazione del server che vi serve, selezionate il menu File-Add Server, altrimenti cliccate su PostgreSQL 8.4, che e il vostro server locale. Se avete impostato una password durante l'in-stallazione, questa vi verra richiesta. Nella parte sinistra della nestra viene visualizzato un

Figura 1: La schermata di avvio di pgAdmin III.

albero con tutti gli oggetti presenti sul server. Per prima cosa l'elenco dei database presenti (ogni server ne puo contenere molti), un elenco di tablespace ed un elenco di Group e Login. I tablespace servono per gestire la memorizzazione sica dei dati; ad esempio grandi moli di dati possono essere partizionate in piu tablespace per aumerare l'e cienza. Questo argo-mento esula dagli scopi di questa esercitazione. La gestione dei gruppi e dei login permette di regolare i livelli di accesso ad ogni risorsa, in presenza di molti utilizzatori; anche questo argomento esula dai nostri scopi.

Come abbiamo visto, il nostro server contiene gia alcuni database, ma per i nostri scopi ne creiamo uno ad hoc. Cliccate col bottone destro sulla casella databases nell'albero e selezionate

3

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

il menu New Database... . Nel diaologo che si apre, scrivere il nome del database da creare (corso), controllate che l'encoding sia UTF8; questo vuol dire che il nostro db memorizzera i testi utilizzando questa codi ca di carattere (non abbiamo tempo per so ermarci su questo argomento). Inoltre selezionate nella casella template il valore postgis template. Se questa scelta non e presente vuol dire che non avete installato il supporto spaziale. Per ora questa scelta e un poco oscura; in seguito si vedra che questa opzione abilita il supporto geografico al database appena creato. Una volta creato il nostro database, questo apparir nell'albero;

Figura 2: Dialogo di creazione di un nuovo db.

cliccateci sopra ed aprite gli oggetti contenuti. C'e un sacco di roba... ma nieste paura, a noi interessano solo poche cose. Innazzitutto vedete la casella Schemas; questi sono gli schemi in cui un db puo essere suddiviso. Corrispondolo alle cartelle di un disco. Il nostro db contiene lo schema di default, che si chiama public. Lo schema public a sua volta contiene tra l'altro le Tables (le tabelle) e la Views (le viste) ovvero le interrogazioni salvate con un nome.

Cliccando col bottone destro sui vari oggetti dell'albero e possibile e ettuare delle opera-zioni tramite interfaccia gra ca; ad esempio cliccando su tables e possibile creare nuove tabelle (Fig. [3](#page5)). Noi non utilizzeremo mai questa opzione durante l'esercitazione, ma e ettueremo

4

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

ogni operazione tramite il linguaggio SQL. Questo perch tale linguaggio e standard per tutti gli altri software di database.

Figura 3: Contenuto del db ed interfaccia.

5

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

* La nestra di comandi SQL

Una volta che avete selezionato il database corso cliccate sullo strumento SQL, rappresentato dall'icona con la matita e il foglio con su scritto SQL. Si aprir la nestra col editor SQL (Fig. [4](#page6)) Nella parte sinistra della nistra c'e l'editor vero e proprio, nella parte destra c'e una

Figura 4: Finestra con editor SQL.

specie di notepad, utile per copiare ed incollare testi. Nella parte sottostante invece vengono visulizzati i risultati delle operazioni ed altre cose.

L'utilizzo della nestra e il seguente: si scrive la query SQL nella nestra di sinistra, si preme il tasto Execute (icona play) e si legge il risultato nella parte sottostate.

Per chi non ha dimestichezza con la rigidita di un linguaggio formale per computer, l'ap-proccio iniziale sar molto duro. La sintassi SQL deve essere esatta: ricordatevi di non inserire spazi all'interno dei comandi, di non confondere zero con la lettera o, di non confondere l'api-cetto singolo con le doppie virgolette o di non confondere le vigole, punti e punti e virgola. Per fortuna, SQL non e mai case sensitive, vale a dire che non si distingue maiuscole e minuscole (potete scrivere indi erentemente SELECT, select o Select).

6

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

Nota sullo stile di questa dispensa: i comandi SQL saranno scritti con il font courier; negli esempi, per chiarezza, scriveremo sempre i comandi di SQL in maiuscolo, mentre scriveremo in minuscolo i valori ed i nomi de niti dall'utente. Si ricorda che SQL non distingue in ge-nere le maiuscole dalle minuscole: se si vuole speci care un nome (di tabella o di colonna) in maiuscolo/minuscolo in modo speci co, e necessario racchiudere il nome fra doppie virgo-lette. In generale i comandi SQL saranno scritti su piu righe e con opportuna indentatura: questa suddivisione viene fatta solo per chiarezza, dato che in SQL la divisione in righe non e signi cativa ai ni dell'interpretazione della query.

7

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

* Pre-Introduzione al comando SELECT

Il comando fondamentale di SQL e SELECT e verra spiegato in dettaglio piu avanti. Dobbia-mo pero introdurlo per e ettuare alcune prove sui dati: questo comando infatti ci permettera di visualizzare le operazioni e ettuate sui tipi di dati di base. La sintassi minima del comando SELECT e:

SELECT {valori};

dove valori e la cosa che ci interessa selezionare.

Ad esempio provate a scrivere nella vostra nestra SQL il seguente comando:

SELECT 42;

quindi premete il pulsante Execute Il risultato visualizzato in basso sar 42. La query che abbiamo scritto richiede infatti al sistema il numero 42; una query non molto intelligente per ora, ma miglioreremo in futuro (nota curiosa: in Oracle e obbligatorio speci care sempre una tabella anche se non serve, a questo scopo esiste sempre una tabella ttizia che si chiama DUAL da cui e possibile selezionare tutto).

Il risultato di una query e sempre (anche in questo caso) una tabella, il numero 1 a sinistra del risultato sta a signi care che questa e la prima riga della tabella risultato, mentre la scritta sopra il numero 42 e il nome della prima colonna. Per rendere piu chiara la di erenza fra il valore ed il nome di una colonna dell risultato provate a scrivere il comando:

SELECT 42 AS valore;

Scrivendo dopo il valore desiderato l'istruzine AS seguita da un nome, e possibile dare il nome speci cato alla colonna del risultato.

* Valori letterali

Per valori letterali si intendono valori costanti dati, come numeri o parole. Come in molti linguaggi di programmazione, in SQL e possibile operare con i numeri interi e con la virgola (che si scrive punto); e inoltre possibile calcolare espressioni o chiamare funzioni. Provate ad eseguire:

SELECT 21\*2;

Oppure per i matematici:

SELECT cos(3.1415926);

Provate ad indovinare quali sono i risultati di queste query.

Oltre che con i numeri, e possibile operare con le parole (stringhe di caratteri). Per distiguere le parole intese come valori dai nomi di colonne e tabelle, e necessario racchiudere le parole fra apicetti singoli (non doppie virgolette!). Ad esempio provate ad eseguire:

SELECT 'Buongiorno';

8

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

Il risultato sar la parola Buongiorno. Se invece avessi scritto la query nella forma:

SELECT Buongiorno;

avrei ottenuto un errore da Postgres: il sistema infatti non riesce a trovare una colonna che si chiama Buongiorno. Per concludere la descrizione delle parole, bisogna dire che nel caso in cui io voglia inserire nella mia parola un apostrofo, lo devo scrivere due volta di la dentro la stringa, ad esempio:

SELECT 'L''area dell''edificio';

produce il risultato:

L'area dell'edificio

Come per i numeri, anche le parole possono avere le loro espressioni e le loro chiamate di funzione, ad esempio la funzione LENGTH calcola la lunghezza in caratteri di una parola, provate:

SELECT LENGTH('casa');

produce il risultato di 4 (la lunghezza in caratteri della parola). Un esempio di operazione fra parole molto utile e la concatenazione di due parole, che si ottiene con l'operatore doppia barra jj, provate ad indovinare quale sia il risultato della query:

SELECT 'pesce' || 'cane';

Attenzione a non confondere il numero 1984 (senza apicetti) dalla parola '1984' (fra apicetti). Nel secondo caso il valore e una parola. La confuzione fra parole e numeri puo portare a risultati sorprendenti: provate la query

SELECT 99 < 100;

(si vuole sapere se 99 e minore di 100) la risposta e t (che sta per true = vero), cioe 99 e minore di 100. Provate ora la query

SELECT '99' < '100';

la risposta e falso questo perch 99 viene DOPO in ordine alfabetico (o come si dice lessico-grafico) di 100.

Esistono comunque una serie di funzione per convertire un tipo di dato in un altro, ad esempio TO NUMBER trasforma un qualcosa in un numero, la query seguente da il risultato atteso (il secondo parametro della funzione speci ca la formattazione del numero):

SELECT SELECT TO\_NUMBER('42','9999')+1;

Esistono anche altre centinaia di funzioni che operano sui dati, per e ettuare tute le operazioni che servono; sarebbe troppo lungho elencarle in questa sede. Quando serve una certa funzione, basta cercarla nel manuale.

9

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

* Tipi di dato

I dati memorizzati nelle tabelle di un database appartengono ad un tipo. Il concetto di tipo di dato e alla base di molti concetti dell'informatica. Quando de nite un attributo di una feature class di Arcgis ad esempio, dovete sempre speci care il tipo di dato associato. Quindi i valori con cui si opera nelle basi di dati (e in quasi tutti i linguaggi di programmazione) sono classi cati in tipi. Tipi di dato sono: numeri interi, numeri con la virgola, parole (stringhe di caratteri), ore e date, valori di verita (vero o falso), BLOB (dati binari generici). Nei database abilitati ai dati geogra ci ci sono inotre tipi di dato spaziali.

In Postgres ogni tipo di dato ha un nome ben preciso, che andr speci cato nel comando di creazione di una tabella. I principali tipi di dato sono:

INTEGER: numero intero;

REAL oppure DOUBLE PRECISION: numero con la virgola in singola o doppia preci-sione;

CHARACTER(n): stringhe di lunghezza n

CHARACTER VARYING: stringhe di lunghezza variabile

BOOLEAN: valori di verita (vero o falso), curiosamente Oracle non ha questo tipo di dato;

DATE : data e ora.

Spesso i tipi di dato hanno dei parametri numerici, ad esempio il tipo stringa ha bisogno della de nizione del massimo numero di caratteri memorizzabili.

I tipi di dato di base sono moltissimi e non abbiamo il tempo di elencarli, ma non solo: nel corso degli anni i sistemi informatici hanno seguito un evoluzione: i tipi di dato di base si sono prima trasformati in tipi complessi (strutture) e poi in oggetti. Anche se non e questo il luogo per approfondire l'argomento dovremmo introdurre parzialmente questo argomento per poter descrivere la componente spaziale di Postgres: infatti il tipo di dato GEOMETRY di Postgres, che di nisce la componente spaziale di un'entita, e un oggetto vale a dire che e un tipo di dato complesso (nel caso di Oracle invece e addirittura un oggetto.

* De nizione dei Dati

Iniziamo adesso il corso vero e proprio di SQL. Vediamo per prima cosa la serie di comandi che permette di de nire la struttura dei dati (vale a dire la forma delle tabelle). I comandi SQL per de nire i dati sono 4:

CREATE TABLE : crea una tabella

DROP TABLE: distrugge una tabella

ALTER TABLE: modi ca una tabella

10

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

8.1 Creazione di una tabella

Creiamo adesso la nostra prima tabella: Il comando di creazione di una tabella CREATE TABLE ha la seguente struttura generale:

CREATE TABLE nome\_tabella

(

nome\_colonna1 TIPO1,

nome\_colonna2 TIPO2,

...

nome\_colonnan TIPOn

);

All'interno delle parentesi tonde che seguono il nome della tabella bisogna speci care la lista delle colonne della tabella stessa, separate da virgola (l'ultima colonna non e seguita da virgola). Le colonne sono speci cate dal loro nome e dal nome del tipo (attenzione! Tutti i nomi di colonna e tabella devono essere un'unica parola senza spazi: al massimo si puo usare la barra di sottolineato . Non c'e di erenza fra maiuscole e minuscole. Nei nomi si possono usare lettere, cifre e la barra sopra detta, anche se il nome non puo iniziare con una cifra). Provate adesso a creare la nostra prima tabella, con il comando:

CREATE TABLE studenti

(

nome CHARACTER(128),

eta INTEGER,

codice\_corso INTEGER

);

Questo comando creera la tabella studenti, formata da tre colonne: il nome dello studente (parola), la sua eta (numero intero) e il codice numerico del corso che lo studente segue (numero intero). Notate le 2 virgole che separano le 3 colonne e il fatto che i nomi di colonna non possano contenere spazi ne tanto meno lettere accentate. A questo punto salvate la query di creazione (se non l'avete gia fatto), tramite il bottone SAVE, per poterla recuperare in seguito.

Nota: si ricorda la formattazione (i ritorni a capo e gli spazi) non conta nulla. Scriveremo i comandi in una certo modo solo per renderli piu chiari. Potevamo scrivere anche (in modo molto meno chiaro):

CREATE TABLE studenti(nome CHARACTER(128), eta INTEGER, codice\_corso INTEGER);

8.2 Analisi di una tabella

Una volta che abbiamo creato una tabella possiamo analizzare la sua struttura tramite l'in-terfaccia gra ca. Torniamo alla nestra princiapale, aggiorniamo l'elenco delle tabelle e se-lezioniamo la tabella studenti; l'interfaccia ad albero ci mostra gli elementi contenuti nella

11

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

Figura 5: Visualizzazione della struttura di una tabella.

struttura della tabella (ma non i dati). A destra, dentro SQL Pane e possibile visualizzare il comando SQL che ha generato la tabella stessa (Fig. [7](#page21)). L'analisi di struttura delle tabelle puo essere utile per studiare le tabelle non create da noi ma dal sistema stesso: ad esempio le tabelle geometry columns e spatial ref sys che fanno parte del sistema geografico.

8.3 Distruzione di una tabella

Per distruggere de nitivamente una tabella, si utilizza il comando DROP. Adesso l'esercita-zione prevede la distruzione della tabella appena creata (la rifaremo meglio dopo), prima di tutto pero copiate il testo della query di creazione e incollatelo nello Scratch Pad, in modo da poterlo riutilizzare in seguito. Quindi provate adesso a distruggere la tabella studenti, con il comando:

DROP TABLE studenti;

Il comando distrugge per sempre la tabella (attenzione ad usarlo con cognizione di causa), non c'e l'annulla.

Abbiamo usato il verbo italiano (piuttosto desueto) distruggere e non cancellare per non confondere le due operazioni: diremo cancellare (in inglese DELETE) nel caso in cui vogliamo cancellare i dati di una tabella senza distruggerne la struttura. Mentre diremo distruggere (in inglese DROP) per eliminare una tabella completamente. Attenzione! Non c'e modo di recuperare una tabella distrutta, a meno che non si sia iniziata una TRANSAZIONE (di cui non parleremo in questo corso).

8.4 Un inciso: commenti al codice

Come in tutti i linguaggi per computer, in SQL e possibile inserire un testo di commento che viene ignorato dal database: la sintassi per inserire commenti nei comandi SQL e di due tipi: commenti a ne riga; tutto quello che segue il simbolo - - (due simboli meno consecutivi) viene ignorato. Commenti multi riga (in stile C++): tutto quello che e compreso fra i simboli /\* e \*/. Ad esempio la creazione della nostra tabella puo essere scritta nel seguente modo:

12

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

CREATE TABLE studenti

(

nome CHARACTER(128), -- Nome dipendente (questo testo viene ignorato) /\* Altri dati dello

Studente (questo testo viene ignorato) \*/ eta INTEGER,

codice\_corso INTEGER

);

A cosa servono i commenti? Servono per inserire note e spiegazioni al codice SQL, in modo tale le la documentazione sia compresa nel codice stesso ad uso dell'uomo e non della macchina.

8.5 Creazione avanzata di una tabella

Le colonne di una tabella possono avere molte speci che aggiuntive, oltre il nome ed il tipo di ogni colonna. ne vediamo due:

chiave primaria;

campo obbligatorio

Ricordiamo che la colonna chiave primaria speci ca il dato (o i dati) che identi cano uni-vocamente ogni riga della tabella, mentre un campo e obbligatorio se il suo valore e sempre non nullo. Di norma invece le caselle di una tabella possono essere anche vuote (avere valore nullo). Il nuovo comando di creazione della tabella studenti rivisto e il seguente:

CREATE TABLE studenti

(

nome CHARACTER (128) PRIMARY KEY,

eta INTEGER,

codice\_corso INTEGER NOT NULL

);

La speci ca PRIMARY KEY indica che il campo nome e quello che identi ca univocamente le righe della tabella. La speci ca NOT NULL indica il campo codice corso e obbligatorio e non puo rimanere vuoto durante l'inserimento delle righe della tabella (vale a dire che non puo assumere il valore speciale NULL). Provate adesso ad eseguire la nuova query di creazione della tabella (se vi siete dimenticati di distruggerla, il nuovo comando di creazione

1. segnaler un errore). Provate anche a visualizzare la tabella nell'albero grafico; vedrete che adesso vengono riportate tutte le informazioni riguardati i campi, compresa la presenza della chiave primaria e dei campi obbligatori.

8.6 Modi ca della struttura di una tabella

La struttura delle tabelle puo essere modi cata dinamicamente. Ad esempio possiamo ag-giungere o togliere colonne, oppure modi care le speci che dei campi (chiavi primarie, campi obbligatori) senza dover distruggere o ricreare la tabella. Una volta che una tabella e stata

13

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

creata, le modi che dinamiche alla sua struttura sono possibili tramite il comando ALTER TABLE, ad esempio aggiungiamo la colonna professione alla nostra tabella:

ALTER TABLE studenti

ADD professione CHARACTER(128);

Di solito i comandi SQL sono molto chiari (sono auto-esplicativi): questo comando modi ca la tabella studenti aggiungendo il campo professione, che e una parola di (al massimo) 128 caratteri. Provate a visualizzare la tabella per controllare l'e etivo cambio di struttura.

Allo stesso modo possiamo cancellare una colonna con il comando DROP COLUMN (non necessariamente l'ultima), provate ad eseguire:

ALTER TABLE studenti

DROP COLUMN professione;

il comando avr l'ovvio e etto che vi aspettate. Se la tabella contenesse gia dei dati, le operazioni di modi ca della struttura possono essere eseguite comunque. I dati delle colonne non interessati dalle modi che di struttura verranno conservati.

* Manipolazione dei dati

Abbiamo imparato a creare, distruggere e modi care le nostre tabelle. Adesso vediamo come si manipolano i dati. I principali comandi di manipolazione dei dati sono 3:

INSERT : inserisce nuove righe in una tabella (quindi inserisce nuovi dati);

DELETE: cancella righe di una tabella;

UPDATE: modi ca i dati esistenti delle righe di una tabella.

9.1 Inserimento di dati

Il comando INSERT server per inserire nuove righe in una tabella. La struttura del comando

INSERT e la seguente:

INSERT INTO nome\_tabella

(nome\_colonna1, nome\_colonna2, ... , nome\_colonnaN)

VALUES (valore1, valore2, ... , valoreN);

Per inserire righe in una tabella bisogna quindi speci care la tabella, l'elenco dei nomi delle colonne che vogliamo inserire, quindi l'elenco corrispondente dei valori.

Proviamo adesso ad inserire alcune righe nella nostra tabella studenti; come dati dobbiamo speci care per ogni studente il nome, l'eta e il codice numerico del corso:

INSERT INTO studenti

(nome,eta,codice\_corso)

VALUES ('Claudio Rocchini', 43, 1);

14

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

Notate che il nome e una parola, quindi va fra apici, mentre l'eta (43) e il codice del corso

1. sono numeri, quindi sono senza apicetti. I termini studenti,nome,eta,... sono i nomi delle tabelle e delle colonne SQL e quindi vanno scritti anche loro senza apicetti. Notate anche le virgole, che separano colonne e valori: ovviamente dopo l'ultimo valore (il numero 1) la virgola non ci vuole. Niente panico: la sintassi a una brutta bestia, che si doma con l'esperienza.

Proviamo adesso ad inserire altri valori nella tabella (potete anche provare ad inserire i vostri dati, mettendo dei codici di corso ttizzi). In particolare proviamo ad inserire un dato incompleto:

INSERT INTO studenti

(nome, codice\_corso)

VALUES ('Margherita Azzariti', 1);

Notate che in questo caso non abbiamo inserito l'eta (per cavalleria), che comunque non e obbligatoria (non possiede l'opzione NOT NULL; il campo codice corso invece e obbligatorio e va sempre speci cato. Nel caso in cui invece inseriamo dati per tutte le colonne, la sintassi del comando INSERT puo essere sempli cata omettendo la lista dei campi da inserire e speci cando solo i valori, nell'ordine con cui devono essere inseriti, ad esempio:

INSERT INTO studenti

VALUES ('Gianfranco Amadio', 86, 2);

9.2 Un'altro inciso: il valore NULL

Abbiamo visto che il campo eta non e obbligatorio. Quando un dato di una riga non e inserito, la relativa casella nella tabella e vuota. Il valore vuoto ha in SQL un nome: NULL. Ad esempio potevamo scrivere il comando di inserimento parziale nel seguente modo:

INSERT INTO studenti

VALUES ('Salvatore Arca', NULL, 2);

Intendendo che il campo eta (il secondo valore) deve rimanere nullo e quindi vuoto. Il valore NULL sar particolarmente utile nei controlli, che vedremo in seguito.

9.3 Test dei vincoli

Nella tabella che abbiamo inserito ci sono due vincoli: la chiave primaria e l'obbligatoriet del campo codice corso. Se proviamo ad inserire una nuova riga con un nome di studente du-plicato, violiamo il vincolo di chiave primaria ed il database ci comunichera l'errore; proviamo ad eseguire il comando:

INSERT INTO studenti

(nome,eta,codice\_corso)

VALUES ('Claudio Rocchini', 16, 1);

15

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

Otteniamo un errore del tipo (scritto in inglese): una chiave duplicata viola il vincolo di unicita. La chiave primaria infatti deve essere unica, mentre noi abbiamo tentato di inserire due studenti diversi con lo stesso nome.

Ricordiamo che il campo eta non e obbligatorio, mentre e obbligatorio il campo codi-ce corso (vale a dire che possiede l'opzione NOT NULL). Proviamo ad eseguire il seguente comando, per inserire uno studente di cui non conosciamo il codice del corso:

INSERT INTO studenti

(nome, eta)

VALUES ('Fantomas', 42);

Otteniamo un errore del tipo (in inglese): un valore nullo nella colonna codice corso viola il vincolo not-null. Inserire dei controlli nelle tabelle e molto importante per controllare a monte la correttezza e la completezza dei dati.

9.4 Cancellazione di dati

Il comando DELETE permette di cancellare righe da una tabella. La sua forma piu semplice sarebbe (NON ESEGUITELO!):

DELETE FROM studenti;

il comando sopra citato cancella TUTTE le righe della tabella studenti (ma non cancella la tabella stessa) senza possibilita di recupero (a meno che non utilizziate le transazioni). La forma del comando DELETE che invece di solito si utilizza e la seguente:

DELETE FROM studenti

WHERE {condizioni};

dove le condizioni speci cate dopo il termine WHERE ltrano le righe da cancellare e ettiva-mente. La speci ca di una condizione permette di eliminare solo quelle righe che rispettano la condizione speci cata, ad esempio se vogliamo eliminare dalla tabella gli studenti che hanno 20 anni scriviamo:

DELETE FROM studenti

WHERE eta=20;

Il comando canceller (se ci sono) tutte gli studenti che hanno l'eta uguale a 20. Nella condi-zione e possibile scrivere espressioni, controllare le colonne, eseguire confronti di uguaglianza (=), diversita (<>), confronti di quantit (< e >), ed usare i connettivi logici AND, OR, NOT (che stanno per e, o, non). Una descrizione accurata di tutte le forme di controllo esula dagli scopi di questo corso, facciamo solo alcuni esempi, il ltro:

...

WHERE eta<40 AND codice\_corso=1

identi ca tutti gli studenti che hanno meno di 40 anni E seguono il corso numero 1. Il ltro:

16

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

...

WHERE eta>40 OR NOT codice\_corso=2

identi ca tutti gli studenti che hanno piu di 40 anni OPPURE NON seguono il corso numero due. Per le parole si possono usare i confronti di uguaglianza, ma anche < e >, intesi come ordine alfabetico (es. 'abaco' < 'zuzzurellone'). L'operatore LIKE invece permette di eseguire confronti fra parole facendo utilizzo di caratteri jolly, il ltro:

...

WHERE nome LIKE 'Cla%';

identi ca tutti gli studenti il cui nome inizia per Cla: il simbolo % sta ad indicare qualsiasi sequenza di lettere.

9.5 Modi ca dei dati

I dati presistenti di una tabella si modi cano con il comando UPDATE. La struttura generale del comando UPDATE e:

UPDATE nome\_tabella

SET nome\_colonna = valore

WHERE {condizioni};

dove la de nizione delle condizioni e del tutto uguale a quella del comando DELETE. Pro-viamo adesso a cambiare il codice corso di qualcuno, eseguiamo la query:

UPDATE studenti

SET codice\_corso = 2

WHERE nome='Claudio Rocchini';

* Il valore della colonna speci cata viene cambiato per tutte le righe che rispettano la con-dizione impostata. In questo caso quindi alla riga che contiene lo studente indicato, verra cambiato dil codice del corso da 1 a 2. Se non si speci ca la condizione, il comando UPDATE modi ca TUTTE le righe della tabella, settando un valore costante sulla colonna indicata.

La dicitura IS NULL puo essere utilizzato nei controlli per determinare la presenza di valori nulli; se vogliamo ad esempio impostare l'eta di 60 anni a tutti gli studenti che non hanno indicazione di eta, scriviamo:

UPDATE studenti

SET eta=60

WHERE eta IS NULL;

In questo modo, tutte le caselle eta vuote (con valore NULL) vengono riempite con 60. Fino ad adesso abbiamo operato su di una sola tabella. Ovviamente le basi di dati

possono contenere molte tabelle. Prima di passare all'interrogazione dei dati, per rendere piu interessante il nostro database, creiamo una tabella corsi, che ci servira per fare degli esempi di interconnessione fra tabelle, eseguiamo la query:

17

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

CREATE TABLE corsi

(

codice\_corso INTEGER PRIMARY KEY,

descrizione CHARACTER(128) NOT NULL

);

Ormai siamo esperti: il codice del corso e la sua chiave primaria, segue una descrizione testuale obbligatoria (testo di al massimo 128 caratteri). Popoliamo adesso la tabella dei corsi:

INSERT INTO corsi

(codice\_corso,descrizione)

VALUES (1,'Basi di Dati');

Ed in ne un altro corso:

INSERT INTO corsi

(codice\_corso,descrizione)

VALUES (2,'Sistemi Informativi Territ.');

Provate ad inserire altri corsi di fantasia. Ovviamente, dato che il codice numerico e la chiave primaria, questo deve essere unico. Perch gli esempi successivi funzionino, e importante che siano presenti tutti i codici di corso che abbiamo inserito nella tabella studenti studenti.

10 Le relazioni

Un database non e fatto solo di entita (tabelle) ma anche di relazioni. Le relazioni sono importanti tanto quanto lo sono i dati. Due oggetti sono in relazione se esiste un dato che li mette in collegamento. Gli studenti sono in relazione con i corsi, dato che per ogni studente abbiamo speci cato un codice di corso. Le relazioni possono anche essere speci cate esplicitamente con l'aggiunta di un vincolo (constraint) alla tabella.

10.1 De nizione di una relazione

La tabella studenti contiene logicamente una relazione con la tabella corsi: infatti il cam-po codice corso della prima tabella riferisce lo stesso campo della seconda tabella. Questa relazione logica fra tabelle puo essere esplicitata tramite il seguente comando:

ALTER TABLE studenti

ADD CONSTRAINT studenti\_corso\_fk

FOREIGN KEY (codice\_corso) REFERENCES corsi(codice\_corso);

Il comando esplicita la relazione fra studenti e corsi, ed e formato da un vincolo sulla tabella studenti. Analizziamo la struttura del comando: la prima riga indica che vogliamo modi-care la tabella studenti (come nel caso di aggiunta di una nuova colonna), in quest caso pero vogliamo aggiungere un vincolo (constraint in inglese): studenti corso fk e il nome di questo vincolo (fk sta per foreign key = chiave straniera e si aggiunge per convenzione). Il

18

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

vincolo a erma (nell'ultima riga del comando) che la chiave straniera formata dalla colonna codice corso della tabella studenti DEVE riferire una valore (vale a dire contenere un numero di codice) della colonna codice corso nella tabella corsi.

Se nella tabella corsi non ci sono tutti i codici necessari, la creazione della relazione sar impossibile, dato che il sistema controlla la congruenza dei dati anche durante la creazione del vincolo. Una volta che il vincolo di relazione e impostato, siamo sicuri che tutti i codici di corso seguiti dagli utenti sono presenti nella tabella dei corsi.

Ai lettori piu attenti puo dar fastidio che la relazione studenti-corsi sia rappresentata da un vincolo sulla sola tabella studenti. Perch non c'e un vincolo anche in corsi? Perch questo tipo di relazione e asimettrica (1:n). E' lo studente che segue un particolare corso, mentre ad un corso possono partecipare ovviamente molti studenti.

Tentiamo adesso di inserire uno studente che segue un corso inesistente, eseguiamo:

INSERT INTO studenti

(nome,codice\_corso)

VALUES ('Fantomas',999);

Se il corso numero 999 non esiste, otteniamo un errore del tipo (in inglese) che ci informa della violazione del vincolo che si chiama studenti corso fk. La congruenza delle relazioni viene controllata dinamicamente in ogni momento, in particolare durante la modi ca dei dati delle tabelle studenti e corsi. Ad esempio non e piu possibile cancellare un corso se e seguito da almeno uno studente.

11 Indici

Accenniamo adesso alla gestione degli indici. Una descrizione dettagliata degli indici esula pero dagli scopi di questo corso.

Supponiamo di prevedere molte ricerche sull'eta degli studenti; inoltre supponiamo che gli studenti siano tanti (non e il caso di questo esempio). Normalmente il sistema deve scorrere l'intera tabella degli studenti per eseguire tale ricerca: se gli studenti sono tanti questa ricerca puo richiedere del tempo. Per velocizzare una ricerca del genere e possibile creare un indice. Gli indici servono per velocizzare le ricerche di valori su una (o piu) colonne di una tabella; il loro funzionamento e simile agli indici (o megli agli indici analitici) dei libri. Per creare un indice sulla colonna eta della tabella studenti, eseguiamo il semplice comando:

CREATE INDEX studenti\_eta\_idx ON studenti(eta);

Al solito, studenti eta idx e il nome dell'indice (idx sta per index), mentre la dicitura stu-denti(eta) indica che l'indice va creato nella tabella studenti ed in particolare sulla colonna eta.

Apparentemente la presenza di un indice non cambia il funzionamento del database: il risultato delle interrogazioni e lo stesso. Quello che cambia e la velocita di funzionamento. In realt vedremo che nel caso di dati spaziali, l'indice e fondamentale per la ricerca veloce dei dati. Gli indici non vengono mai creati automaticamente (eccetto che per gli indici sulle colonne chiave primaria, che vengono create implicitamente, come ci avverte il messaggio di

19

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

Postgres): devono essere progettati con cura da chi crea la struttura del database, in funzione del tipo di ricerche da e ettuare e dal tipo (e dalla quantita) dei dati presenti: la presenza di un indice su di una colonna velocizza sempre le operazioni di ricerca, mentre ne puo rallentare leggermente le operazioni di modi ca (dato che in questo caso e necessario aggiornare anche l'indice). Inoltre la creazione di un indice richiede un utilizzo aggiuntivo di spazio disco.

Nota: non e mai necessario creare indici per le colonne chiave primaria: in questo caso un indice e creato automaticamente.

Uno sguardo all'albero nella nestra principale ci mostra la struttura corrente del db, con le nostre tabelle, colonne, vincoli e indici (Fig. [6](#page20)).

Figura 6: pgAdmin III; struttura del db con vincoli ed indici.

12 Le interrogazioni: SELECT

Siamo arrivati (ovvero ritornati) nalmente alla parte nale di SQL: l'interrogazioni dei dati. Sebbene le interrogazioni siano eseguito dall'unico comando SELECT, questo e il comando piu complesso. Le forme del comando SELECT sono moltissime, quindi ne vedremo alcuni brevissimi esempi. Inoltre l'apparente semplicita di tale comando nasconde la notevole dif-colta di tradurre la richiesta che abbiamo in mente nella dicitura SQL; questo richiede un notevole livello di esperienza.

12.1 Forma semplice di SELECT

La forma piu semplice di SELECT e la seguente:

20

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

SELECT colonna1,colonna2,...,colonnaN

FROM tabella

WHERE {condizioni}

ORDER BY colonna1,colonna2;

Nella forma semplice di SELECT bisogna speci care: l'elenco delle colonne da visualizzare, la tabella sorgente, eventuali condizioni uguali a quelle dei comandi UPDATE e DELETE, ed un eventuale ordine (l'ordine non e mai de nito per default! Questo e un concetto fondamentale delle basi di dati, l'ordine delle righe e inde nito, ad esempio non centra niente con l'ordine di inserimento).

Invece di scrivere un elenco di colonne e possibile scrivere il simbolo \* (asterisco) che indica tutte le colonne della tabella.

La condizione WHERE e l'ordine ORDER BY possono essere anche omessi: ad esempio per visualizzare un'intera tabella possiamo scrivere il comando:

SELECT \*

FROM studenti;

Oppure:

SELECT \*

FROM corsi

ORDER BY descrizione;

Si ricorda che \* sta per tutte le colonne; inoltre, non essendoci ltro, vengono estratte tutte le righe della tabella indicata. Nel secondo caso i corsi saranno ordinati per descrizione (ordine alfabetico), mentre nel primo caso l'ordine e casuale. Il risultato della prima query e mostrato in Fig. [6](#page20).

Figura 7: Visualizzazione del risultato di una query.

Per visualizzare un sottoinsieme di colonne di una tabella, basta indicarne la lista dopo la parola SELECT; ad esempio possiamo scrivere:

SELECT nome,eta

FROM studenti

ORDER BY eta;

In questo caso si visualizza solo il nome e l'eta degli studenti, mentre l'ordine e per eta crescente.

Se invece vogliamo vedere un sottoinsieme delle righe di una tabella, possiamo speci care una condizione di ltro, in modo del tutto analogo ai comandi UPDATE e DELETE:

21

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

SELECT nome

FROM studenti

WHERE eta < 60 AND codice\_corso=2;

che in italiano si legge: selezionare il nome dalla degli studenti dove l'eta e minore di 60 (anni) e il codice del corso seguito e uguale ad 2.

12.2 Aggregazioni di righe

Un secondo tipo di SELECT, e quella del tipo aggregante, in cui piu linee di una tabella possono essere aggregate insieme, da particolari funzioni di aggregazione, lo schema della query diventa:

SELECT {funzioni\_aggreganti}

FROM tabella

WHERE {condizione sulle righe}

GROUP BY {colonne che discriminano l'aggretazione}

HAVING (condizione sul risultato aggregregato)

Niente paura, e piu complicato da dire che da fare. In generale l'aggregazione produce un dato totale, che riguarda l'intera tabella o insiemi di righe raggruppate. Quali sono le funzioni di aggregazione? Le principali funzioni di aggregazione sono:

Min: minimo dei valori

Max: massimo dei valori

Avg: media dei valori

Sum: somma dei valori

Count: numero di valori

Vedremo che ci sono anche funzioni di aggregazione spaziale (es. il baricentro di insieme di oggetti).

Supponiamo ad esempio di voler sapere l'eta minima, media e massima degli studenti presenti nella nostra tabella. Possiamo scrivere:

SELECT min(eta),max(eta),avg(eta)

FROM studenti;

Il risultato saranno i valori minimo, massimo e medio di tutte le eta della tabella studenti. Si noti che il risultato in questo caso e una sola riga: tutte le righe della tabella studenti sono state aggregate in una sola. Le funzioni di aggregazione (come le funzioni matematiche) hanno bisogno della speci ca dei parametri (nel nostro caso la colonna eta) su cui operare, i quali vanno speci cati fra parentesi tonde.

Vediamo adesso come si possono raggruppare le aggregazioni di valori. Vogliamo sapere l'eta minima e massima degli studenti, ma suddivisa secondo il corso seguito; proviamo ad eseguire:

22

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

SELECT codice\_corso, min(eta),max(eta),count(eta)

FROM studenti

GROUP BY codice\_corso;

La query e simile alla precedente: in questo caso pero le righe non sono aggregate tutte insieme, ma secondo il codice del corso. Questo raggruppamento e dovuto all'aggiunta della riga GROUP BY codice corso. Il risultato e l'analisi dell'eta degli studenti al variare del corso a cui appartengono. In questo caso il risultato e formato da piu righe: una per ogni corso presente: per ogni codice viene stampata la minima e massima eta degli studenti, limitatamente al corso in questione. La funzione count(...) conta semplicemente il numero di righe corrispondenti, vale a dire il numero di partecipanti ad un corso.

Per sapere quante righe contiene una tabella basta scrivere:

SELECT count(\*)

FROM corsi;

Dato che alla funzione count non interessa la particolare colonna (conta solo il numero di righe), e possibile scrivere il carattere \* (che sta per tutte le colonne) al posto del nome della colonna.

12.3 Join

Nel nostro database abbiamo due tabelle: studenti e corsi. Nella tabella studenti e presente il nome dello stesso e il codice del corso. Nella tabella corsi e presente la descrizione. Inoltre le due tabelle sono collegate da una relazione esplicita (FOREIGN KEY). Vogliamo adesso visualizzare il nome di ogni studente con associata la descrizione del corso seguito. Per fare questo e necessario utilizzare la relazione che intercorre fra le due tabelle: il termine tecnico di questa operazione e JOIN (uni cazione). L'esecuzione di una SELECT con JOIN implicata l'utilizzo di piu tabelle conteporaneamente, quindi la clausola FROM della nostra query avr una forma del tipo

...

FROM studenti, corsi

...

L'utilizzo di piu tabelle in una query comporta alcune complicazioni. Ad esempio dato che entrambe le tabelle in gioco hanno la colonna codice corso, indicandone solo il nome il sistema non saprebbe a quale tabella ci vogliamo riferire, se quella degli studenti o quella dei corsi. Per togliere ogni ambiguit bisogna speci care il nome di colonna completa del nome della tabella: i due nomi devono essere serapati da un punto. Inoltre dobbiamo speci care quale sia la regola di uni cazione delle due tabelle: nel nostro caso la regole di uni cazione e che il codice del corso seguito da uno studente (colonna studenti.codice corso) deve essere uguale al codice del corso della tabella corsi (colonna corsi.codice corso). Colleghiamo le due tabelle con la query:

SELECT studenti.nome, corsi.descrizione

FROM studenti,corsi

WHERE studenti.codice\_corso = corsi.codice\_corso;

23

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

|  |  |  |
| --- | --- | --- |
| Il risultato e il seguente: |  |  |
| "Margherita Azzariti ";"Basi di Dati | | " |
| "Salvatore Arca | ";"Sistemi Informativi Territ." | |
| "Claudio Rocchini | ";"Sistemi Informativi Territ." | |
| "Gianfranco Amadio | ";"Sistemi Informativi Territ." | |

Ci sono alcuni particolari da notare: per prima cosa in questa query facciamo utilizzo di DUE tabelle: dopo FROM infatti possiamo utilizzare quante tabelle vogliamo. In secondo luogo vediamo che le colonne dopo la SELECT sono speci cate nella forma NOME TABELLA. NOME COLONNA: questa speci ca e necessaria in presenza di piu tabelle per chiarire da quale tabella si pesca la colonna. Codice corso ad esempio e presente in entrambe le tabelle ed Postgres non sa decidere di quale tabella fa parte. In ne analizziamo la clausola WHERE: in questo caso la clausola non ha una funzione di ltro sul risultato, e invece questa che mette in relazione concretamente le due tabelle. Senza la clausola WHERE (provate a cancellarla ed eseguire la query), Postgres esegue quello che si chiama prodotto cartesiano dei valori, vale a dire tutte le combinazioni possibili fra studenti e corsi, senza nessun nesso fra le coppie studente-corsi. La clausola WHERE invece, fra tutte le combinazioni, seleziona solo quelle in relazione.

Le join fra tabelle sono molto importanti nel campo spaziale: vedremo che lo stesso meccanismo puo essere utilizzato per creare relazioni spaziali (es. relazionare gli edi ci con le strade a seconda della minima distanza relativa).

La join e un'operazione molto importante e complessa; per questo nasconde alcune di - colta. In particolare la gestione dei campi vuoti o che non hanno corrispondenza nella join deve essere gestita speci cando il tipo di comportamento da tenere.

13 Viste

Una volta che abbiamo creato una SELECT interessante (come quella fra studenti e corsi), e possibile che ci serva piu volte. Oltre al meccanismo di salvataggio delle query presente nell'interfaccia di Postgres (File- save), e possibile dare un nome ad una query importante, ed in questo modo salvarla permanentemente nella base di dati. Le query salvate con nome

prendono il nome di viste. E possibile salvare le query come viste, aggiungendo al codice della query il comando CREATE VIEW nome vista AS, provate ad esempio ad eseguire:

CREATE VIEW stud\_corsi AS

SELECT studenti.nome,

corsi.descrizione

FROM studenti,corsi

WHERE studenti.codice\_corso = corsi.codice\_corso;

Dalla seconda riga in poi la query e identica a quella della sezione precedente. In questo caso pero la query non viene eseguita: invece le viene dato il nome stud corsi e salvata nella base di dati come vista. Le viste in pratica sono query con nome: una volta create si utilizzano come se fossero tabelle. Provate adesso ad eseguire:

24

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

SELECT \* FROM stud\_corsi;

I dati delle viste variano al variare delle tabelle sottostanti (studenti e corsi), vale a dire che il risultato della query non e salvato al momento della creazione della vista, ma varia al variare delle tabelle originali. Se i dati della tabella studenti o corsi vengono cambiati, il risultato dell'interrogazione della vista cambia in modo conforme.

Ripetiamo che le viste si usano esattamente come se fossero tabelle: e quindi possibile aggiungere ltri, ordinamenti, etc. alla SELECT su viste.

14 Editor gra ci di query

SQL e (a nostro parere) un linguaggio molto elegante, inoltre spesso e chiaro di per se ed e au-toesplicativo. Molti sistemi (anche Postgres) prevedono pero un ausilio grafico alla costruzione della query.

Vediamo ad esempio il Query Builder di ArcGis in Fig. [8](#page25). Questa interfaccia permette di

Figura 8: Schermata di creazione guidata di un ltro SQL (Query Builder di ArcGIS).

costruire in modo guidato una clausola WHERE per la nostra query SQL, anche se rimane possibile digitarla manualmente. Altri sistemi di basi di dati (es. Access) hanno strumenti simili di costruzione.

25

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

15 Basi di dati reali

I database reali (ovviamente) possono essere molto complessi o contenere grandi moli di dati. Il database geografico del catalogo IGM on line contiene 728,625 Toponimi (ricerca spaziale-testuale); 213,906 Fotogrammi Aerei georeferenziati (raster+scheda monogra ca); 44,582 Punti Trigonometrici, IGM95 e Capisaldi di Livellazione; 27,414 Schede foto storiche (immagine + monogra a), 384040 attributi; 22,051 Carte storiche (immagine + monogra a), 72,834 relazioni spaziali; 8,274 Limiti Amministrativi Geogra ci; 6,825 Prodotti in catalogo con stato produzione; 5,000 Raster, preview dei prodotti cartogra ci; Per avere un'idea una basi di dati reale, vediamo lo schema grafico del programma per la gestione del personale IGM.

Figura 9: Gestione personale IGM (milioni di fatti registrati).

16 Conclusioni

Questa e solo una brevissima introduzione a SQL. La struttura del comando SELECT ha molte altre possibilita, che richiederebbero almeno un anno di corso. Il linguaggio SQL e di per se molto semplice, ma la creazione di un comando SELECT non banale, richiede, in alcuni casi, una certa esperienza. Es. banale: come si scrive una SELECT che mostra il nome dello studente piu giovane?

26

Master in Sistemi Informativi Territoriali - Breve Introduzione a SQL

Riferimenti bibliogra ci

1. Renzo Sprugnoli, Libri di base: le basi di dati, Editori Riuniti (www.editoririuniti.it), 1987.
2. PostGIS , PostGIS Online Documentatio, (postgis.refractions.net/documentation), December 2005.
3. PostgreSQL Global Development Group, PostgreSQL 8.4.1 Documentation, (www.postgresql.org/docs/manuals), 2010.

Elenco delle gure

|  |  |  |
| --- | --- | --- |
| [1](#page3) | [phAdmin III](#page3) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 3 |
| [2](#page4) | [Creazione di un db](#page4) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| [3](#page5) | [Contenuto del db](#page5) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [4](#page6) | [Finestra SQL](#page6) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 6 |
| [5](#page12) | [Descrizione Tabella](#page12) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 12 |
| [6](#page20) | [Struttura DB](#page20) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 20 |

* [Risultato SELECT](#page21) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

|  |  |  |
| --- | --- | --- |
| [8](#page25) | [ArcGis Query Builder](#page25) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 25 |
| [9](#page26) | [Schema ER Personale](#page26) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 26 |

27