C

DBMS open source: Postgres*

Lo sviluppo del mondo dell'*open source* rappresenta sicuramente uno dei fenomeni recenti di maggiore impatto nel mondo dell'informatica. Sfruttando le caratteristiche di facile riproduzione e trasmissione grazie a Internet dei programmi e lo spirito di partecipazione volontaria di molti esperti di tecnologia informatica, questo movimento ha portato allo sviluppo di un ampio insieme di componenti software e applicazioni con codice sorgente liberamente disponibile. Il frutto più significativo di questo movimento è sicuramente il sistema operativo Linux, il quale, anche beneficiando di iniziative preesistenti come l'ambiente GNU sviluppato sotto la spinta della Free Software Foundation, ha fornito una piattaforma completa e tecnicamente valida che ha dato un impulso fortissimo allo sviluppo di nuovi progetti. Le iniziative del mondo open source si sono sviluppate su numerosissimi fronti, con la realizzazione di browser come Firefox, client di posta elettronica come Thunderbird, server Web come Apache Server, ambienti completi di automazione di ufficio come OpenOffice. In tutti questi casi, i prodotti open source costituiscono delle valide alternative rispetto alle offerte dei produttori commerciali, con un seguito significativo e in continua crescita nella comunità degli utenti, producendo un impatto che travalica il dominio del sistema operativo Linux.
Il movimento open source ha coinvolto anche il mondo dei DBMS, si potrebbe dire *inevitabilmente* dato il ruolo centrale che i DBMS relazionali rivestono nell'architettura dei moderni sistemi informativi. Questo interesse ha dato luogo a diverse iniziative, spesso in concorrenza tra di loro. In parte come risposta a queste iniziative, si è anche assistito recentemente al rilascio come strumenti gratuiti di versioni limitate dei prodotti commerciali di maggiore diffusione. Inoltre sia IBM, sia Microsoft, sia Oracle, ossia ciascuno dei tre grandi produttori commerciali di DBMS attualmente presenti sul mercato, coordinano iniziative che consentono agli studenti di corsi universitari di disporre di versioni complete dei motori relazionali da loro prodotti a costo gratuito, spesso a condizione che sia attivata una particolare convenzione con l'università, con la limitazione che l'uso del sistema sia a fini educativi.
Vi sono poi prodotti che per varie ragioni, dopo essere stati sviluppati inizialmente come prodotti commerciali, sono poi evoluti in prodotti open source; per esempio, da Interbase è derivato Firebird, da Cloudscape deriva Apache Derby.
I database server open source che vedono attualmente maggior seguito sono MySQL e Postgres.
MySQL è ancora oggi il prodotto relazionale che presenta il maggior numero di adozioni all'interno della comunità open source. Si tratta di un prodotto che nasce con l'ambizione di offrire un supporto per la gestione relazionale di dati nel contesto di applicazioni che non generano scritture concorrenti, presentando come punto di forza la facilità di installazione e configurazione e le ottime prestazioni nell'ambito degli scenari applicativi per cui era pensato, tra i quali figura con particolare importanza la costruzione di siti Web. Dato il successo incontrato, gli

*Recentemente il sistema Postgres è stato ridenominato PostgreSQL

sviluppatori hanno di recente cercato di far evolvere il sistema con l'ambizione di offrire l'insieme di servizi che caratterizza tipicamente un DBMS relazionale, ma l'iniziativa ha incontrato diversi ostacoli, facilmente prevedibili dato il ruolo cruciale che riveste all'interno di un sistema relazionale la gestione di accessi concorrenti e il supporto per le transazioni.
PostgreSQL rappresenta invece la moderna incarnazione del sistema Postgres (questo è il nome che continua a essere adottato dalla comunità ed è il nome che useremo in questo testo), un progetto nato nella seconda metà degli anni '80 presso l'Università della California a Berkeley, sotto la supervisione di Mike Stonebraker. Il sistema era nato con l'obiettivo di indagare la frontiera della tecnologia delle basi di dati, costruendo un sistema in grado di offrire supporto a diverse innovazioni che sono state studiate negli anni. Aldilà di questi aspetti innovativi, il sistema presentava un supporto moderno per l'insieme dei servizi che caratterizza un pieno DBMS relazionale, riconoscendo un insieme ampio di costrutti SQL e gestendo i vari aspetti legati al supporto transazionale.
Nell'ambito della comunità open source Postgres è stato spesso penalizzato da una reputazione che lo considerava un sistema lento e pesante, rispetto alla struttura leggera ed efficiente di MySQL. Una parte di questa reputazione era legata ad alcune scelte di progetto che imponevano una certa rigidità e lentezza al sistema, in parte al necessario onere computazionale e conseguente rallentamento imposto dalla corretta gestione dei requisiti transazionali. Versioni recenti dimostrano un livello di prestazioni estremamente competitivo con quello offerto dalle versioni di MySQL che offrono il supporto transazionale. Data la maggiore flessibilità e il supporto naturale per tutti i servizi che caratterizzano un vero DBMS, è stato naturale nel contesto di questo libro scegliere Postgres come sistema di riferimento per illustrare le caratteristiche di un DBMS open source.

C.1 Caratteristiche del sistema

La versione corrente di Postgres (PostgreSQL 8.3.5 nel momento in cui si scrive) è disponibile presso il sito del progetto http://www.postgresql.org. Il progetto è coperto da una licenza open-source di tipo BSD, ovvero una licenza "non virale", meno restrittiva dalla licenza GPL che caratterizza molti prodotti open-source. Il principale vincolo posto dalla licenza è che, nel momento nel quale si intende utilizzare il sistema all'interno di un proprio prodotto, si deve dichiarare che il sistema ha al suo interno componenti che sono protetti da quella licenza. Vi è quindi una significativa possibilità di utilizzare il sistema anche all'interno di iniziative commerciali.

Il nucleo principale di Postgres è rappresentato dal motore relazionale, il server che gestisce le tabelle e accetta query dagli utenti del sistema. Il server è multipiattaforma e può essere eseguito sia in ambiente Linux, sia Windows, sia Mac OS X. Il server è nato inizialmente per sistemi Unix e nell'uso mostra chiaramente questa origine. In ambito Windows il sistema offre un modello di installazione, gestione e interazione che è coerente con quello delle applicazioni Windows native, ma appena si cercano di utilizzare funzionalità avanzate, la natura originale del sistema si rivela chiaramente. Dato l'obiettivo del sistema e il target di utenti cui il sistema tipicamente offre le proprie interfacce (gestori di sistema e programmatori di applicazioni), questa eterogeneità rispetto al consueto modello di interazione di Windows costituisce una limitazione abbastanza lieve. Data la migliore coe-

renza con il mondo Unix, tenendo inoltre conto che il mondo delle basi di dati per Windows è trattato nell'appendice dedicata a Microsoft Access, si è scelto di far riferimento in questa appendice a un'installazione su ambiente Linux. In particolare, le sessioni sono state realizzate su una distribuzione Fedora.

Lo scopo di questo capitolo non è ovviamente quello di fornire una guida dettagliata all'uso di Postgres. A questo fine, esiste una documentazione curata e di ampie dimensioni; per esempio, il manuale standard offerto assieme alla distribuzione consiste al momento di quasi 2000 pagine. L'obiettivo è bensì quello di illustrare sinteticamente quali sono le caratteristiche di base del sistema e le funzionalità offerte, per facilitare un'esperienza diretta d'uso che, comunque, avrà bisogno in molti casi di far riferimento alla documentazione originale per risolvere gli aspetti di dettaglio relativi alla configurazione e sviluppo di un'applicazione.

C.2 Installazione e prima configurazione

Mostriamo i passi necessari per installare il sistema. La maggior parte delle moderne distribuzioni Linux mette già a disposizione un'installazione completa di Postgres. Qualora questa non sia presente o si desideri aggiungere qualche componente, sarà necessario procedere all'installazione manualmente. L'installazione può fare uso degli strumenti di configurazione della distribuzione (`yum` per Fedora, invocando come *root* da shell il comando `yum install postgresql`), o può utilizzare strumenti di livello più basso quali `rpm`, o può addirittura partire dai sorgenti C del sistema compilandoli direttamente. Un vantaggio degli strumenti di più alto livello è la possibilità di mantenere aggiornato in modo facile il sistema in seguito al rilascio di nuove versioni, oltre a disporre di un meccanismo di verifica delle dipendenze tra i diversi componenti molto più robusto. Per quanto riguarda la fonte delle versioni compilate, ci si può appoggiare alla versione rilasciata dal gestore della distribuzione, o si può modificare la propria configurazione esplicitando come sorgente per Postgres il sito ufficiale del sistema, disponendo quindi di versioni più recenti e con un maggior numero di componenti a scelta, a scapito del rischio di incompatibilità con alcuni componenti standard della propria distribuzione Linux. Per seguire questa strada in Fedora è necessario intervenire sulla configurazione locale di `yum`.

Nella configurazione standard, l'installazione del database crea un account di nome `postgres` sul sistema. Il primo passo nell'uso del sistema consiste nell'usare questo account (via `su` o `sudo`) per inizializzare un database vuoto tramite la coppia di comandi `initdb` e `createdb`, di cui sarà proprietario l'utente `postgres`. Le norme di buona gestione di un sistema richiedono di fare attenzione a utilizzare account con troppi privilegi e di cercare di configurare i sistemi secondo il classico principio del "least privilege". In questa trattazione l'attenzione è rivolta all'apprendimento dei principi e funzionalità del sistema, piuttosto che alla installazione e configurazione di un sistema per un ambiente di produzione. Chiaramente nella realizzazione di un'applicazione reale, particolarmente se esposta verso Internet, è necessario adottare tutte le buone pratiche di costruzione

di sistemi sicuri, dedicando una particolare attenzione alla corretta configurazione degli aspetti di sicurezza. Il modo più semplice per gestire la protezione consiste nel creare dall'account `postgres` un nuovo ruolo/utente nel database e poi concedere al nuovo utente la possibilità di costruire con il comando `createdb` una propria base di dati, di cui l'utente sarà proprietario; in questo modo si protegge l'integrità del sistema, che è invece esposta a rischi maggiori qualora si operi utilizzando direttamente l'account `postgres`. La creazione del nuovo ruolo/utente può essere ottenuta invocando `create role` all'interno dell'interprete SQL o il comando `createuser` dalla shell.

Una volta che il database server è stato inizializzato, il motore relazionale può essere avviato; per esempio in Fedora si può attivare il server con il comando `service postgresql start`. Il sistema è quindi pronto per accettare le richieste degli utenti autorizzati.

C.2.1 Interazione testuale: psql

Una volta installato, il motore relazionale risponde a richieste che giungono dalle applicazioni aprendo un canale di comunicazione su una porta TCP. Oltre a questa modalità, il sistema consente anche di utilizzare una serie di strumenti che permettono di interagire direttamente con il sistema. Uno strumento di interazione molto utilizzato con Postgres è lo strumento `psql`, un programma che gestisce un'interfaccia di dialogo testuale, con un accesso diretto verso l'interprete SQL del motore relazionale. Per accedere al database di default, è sufficiente invocare `psql` dalla shell. Nella configurazione standard di Postgres è possibile accedere al DBMS senza fornire una password, basandosi su una corrispondenza diretta tra gli account del sistema operativo e gli account del DBMS. Si noti che i due sistemi sono indipendenti, ovvero gli account del sistema operativo non corrispondono agli account gestiti sul database server, ed è relativamente facile modificare la configurazione del sistema per far sì che a ogni accesso al DBMS sia richiesta una verifica esplicita della identità dell'utente.

In seguito all'invocazione del comando `psql`, il sistema presenta questa interfaccia:

```
Welcome to psql 8.3.5, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

postgres=#
```

Il sistema è quindi pronto per accettare le richieste dell'utente. Il comando `psql` può anche essere usato in altro modo, fornendo come parametro d'invocazione il nome di un file, tramite l'opzione `-f` oppure tramite il meccanismo di ridirezione dello standard input gestito dalla shell. Per esempio, si può invocare dalla

shell Linux il comando "`psql -f ./script.sql`", equivalente a "`psql < ./script.sql`". Il file `script.sql` conterrà di norma il testo di una sequenza di comandi SQL, che verranno eseguiti in ordine come se venissero immessi direttamente dall'utente al prompt. Al termine dell'esecuzione della sequenza di comandi, `psql` termina.

C.2.2 Interazione grafica: pgAdmin

La distribuzione Postgres presenta anche il componente `pgAdmin`, ora giunto alla terza versione, il quale offre un'interfaccia grafica in grado di gestire in modo sofisticato i compiti di gestione della base di dati. In ambiente Linux, il componente viene invocato dal comando `pgadmin3`; qualora non sia presente, l'installazione in Fedora può avvenire invocando come root il comando `yum install pgadmin` (a condizione che `yum` sia configurato per recuperare i componenti della distribuzione completa di Postgres). Grazie a questa applicazione, anche Postgres dispone di un'interfaccia grafica di gestione, in modo coerente con l'evoluzione degli ultimi anni di tutti i DBMS commerciali, i quali, a partire dai sistemi Microsoft, hanno negli ultimi anni spostato sempre più l'attenzione verso l'uso di

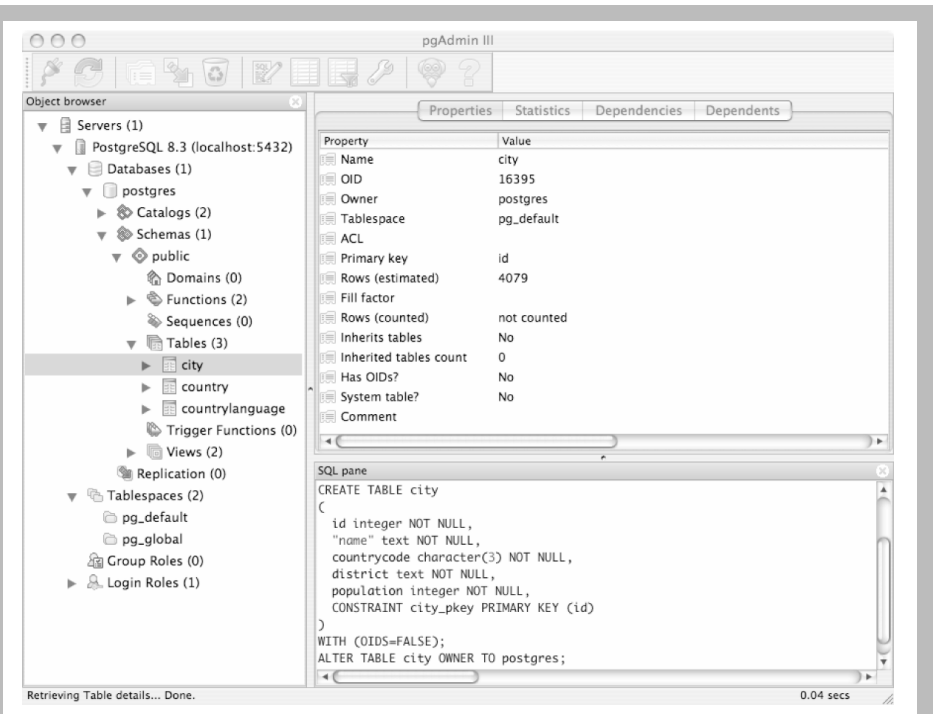

Figura C.1 La finestra di partenza di `pgAdmin`

interfacce di questo tipo, riducendo pian piano il ruolo delle classiche interfacce testuali. La schermata iniziale è mostrata nella Figura C.1.

Come avviene in molti di questi ambienti, l'interfaccia è strutturata in un insieme di pannelli.

- *Object browser*: sul lato sinistro in figura, presenta in un formato ad albero navigabile la struttura di componenti del sistema. A ogni componente è associato un menu contestuale, attivabile con il tasto destro del mouse, che mette a disposizione i consueti comandi che consentono di operare sull'oggetto (creando per esempio dei sotto componenti).
- *Information pane*: in alto a destra nella figura, presenta le proprietà dell'oggetto selezionato nell'Object browser. Questo pannello offre diverse prospettive sull'oggetto, rappresentate dai diversi tab: *Properties* descrive le proprietà dell'oggetto, recuperandole dal catalogo della base di dati; *Statistics* descrive dati sintetici relativi all'istanza dell'oggetto selezionato (i cosiddetti profili, che sono discussi nel Capitolo 1 del secondo volume [5]); `Dependencies` e `Dependents` descrivono i componenti dello schema che sono coinvolti in dipendenze con l'oggetto selezionato.
- *SQL pane*: in basso a destra nella figura, presenta in formato SQL le caratteristiche dell'oggetto selezionato. Il codice SQL presentato viene ricostruito a partire dal contenuto del catalogo; non coincide quindi necessariamente con il codice SQL che è stato utilizzato per generare l'oggetto, ma ne rappresenta una versione in grado di produrre per il motore relazionale esattamente lo stesso componente nello schema.

Lo strumento presenta poi un insieme abbastanza esteso di comandi, disponibili sia come bottoni sulla tool bar (nella figura vi sono 11 bottoni, appena sotto la barra dei menu), sia come voci all'interno dei vari menu. L'insieme è ricco e consente di gestire buona parte delle necessità di interazione con il DBMS. Diversi dei bottoni presenti nella tool bar aprono finestre dedicate. Per esempio, premendo il bottone etichettato "SQL" si apre una finestra per la gestione di query, rappresentata in Figura C.2.

La finestra ha un'area di testo editabile come componente principale e inoltre prevede un certo numero di pannelli, una tool bar e un menu. L'uso normale dello strumento prevede di scrivere il testo SQL della query nella componente principale. Il pannello di output (*Output pane*) è responsabile di mostrare il risultato dell'esecuzione del comando. Lo strumento permette anche di analizzare la struttura del piano di esecuzione della query generato internamente dall'ottimizzatore di Postgres, in diversi formati (si analizzano questi aspetti nel Capitolo 1 del secondo volume); queste funzioni sono particolarmente importanti per un uso avanzato di Postgres.

Un altro bottone della finestra principale consente invece di visualizzare il contenuto della tabella. Nella Figura C.3 si mostra la finestra con parte del contenuto di una tabella. La finestra permette di specificare nella visualizzazione dei criteri di filtraggio o ordinamento. È inoltre possibile limitare la visualizzazione a un numero limitato di tuple.

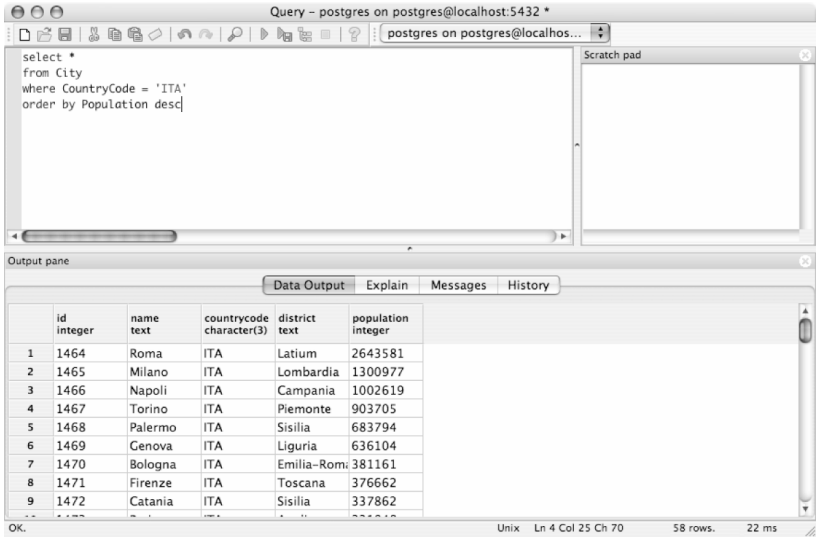

Figura C.2 L'interfaccia di `pgAdmin` per la definizione di query

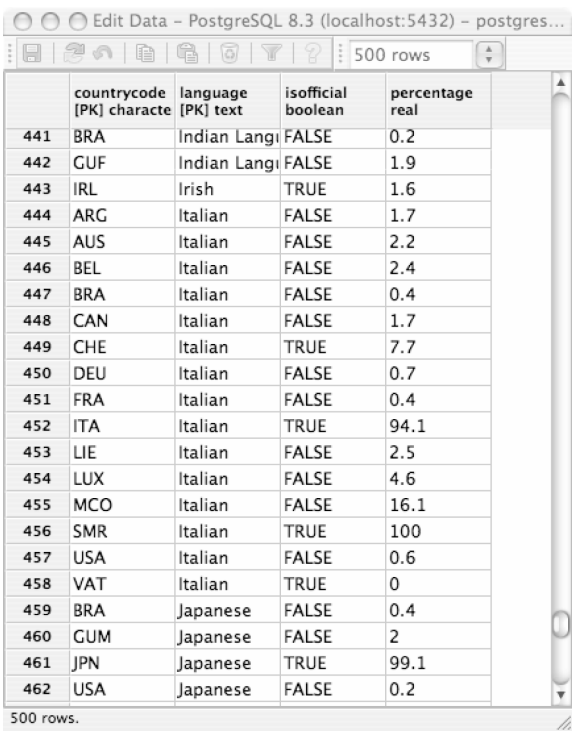

Figura C.3 L'interfaccia di `pgAdmin` per l'accesso al contenuto di una tabella

C.3 Costruzione di una base di dati d'esempio

Postgres rappresenta uno dei sistemi relazionali con il maggior grado di compatibilità con lo standard SQL. Oltre al supporto assai esteso per i costrutti definiti nello standard, Postgres presenta alcune forme sintattiche che non rientrano in SQL-3, la cui presenza è giustificata dalla necessità di offrire servizi di gestione dei dati coerenti con l'architettura interna del sistema.

Per quanto riguarda l'insieme di domini di base, Postgres offre una ampia varietà di scelta, che va ben al di là dei classici domini che fanno parte dello standard SQL. Inoltre, Postgres si caratterizza per essere un sistema espandibile, con alcune caratteristiche delle basi di dati a oggetti (vedi il Capitolo 3 del secondo volume) e il sistema di domini presenta una particolare flessibilità. Utilizzando l'interfaccia a finestre di `pgAdmin` per la costruzione di schemi di tabelle, è possibile osservare un elenco di alcune decine di domini come possibilità di scelta per il dominio di un singolo attributo.

In generale, la costruzione di uno schema di base di dati può fare uso dello strumento `psql`, con l'immissione da terminale o da file della sequenza di comandi SQL in grado di definire lo schema; in alternativa, si può usare l'interfaccia interattiva di `pgAdmin`, che permette di minimizzare l'immissione di testo ed è in grado di ricostruire internamente il codice SQL necessario per la definizione di una certa struttura dati.

In questa appendice, piuttosto che descrivere passo-passo la costruzione di una base di dati d'esempio, assumiamo che sul sistema venga installata una delle basi di dati pubbliche che fanno parte della distribuzione estesa di Postgres. Tra queste basi di dati, scegliamo la base di dati *World*, che consiste di tre tabelle, CITY(Id, Name, CountryCode, District, Population), COUNTRY(Code, Name, Continent, Region, SurfaceArea, IndepYear, Population, LifeExpectancy, Gnp, GnpOld, LocalName, GovernmentForm, HeadOfState, Capital, Code2), COUNTRYLANGUAGE (CountryCode,Language, IsOfficial, Percentage) che descrivono dati sintetici di tipo geo-economico. Per esempio, la tabella CITY contiene la descrizione di circa 4000 città del mondo, tra cui la descrizione delle prime 58 città italiane per numero di abitanti. La base di dati è ricostruibile invocando l'esecuzione dello script "world.sql", tramite lo strumento `psql` o pgAdmin.

C.4 Supporto dello standard SQL

Facendo riferimento alla trattazione del linguaggio SQL del Capitolo 4 e del Capitolo 6, Postgres offre il supporto per grandissima parte dei costrutti. Tra le funzioni non gestite, segnaliamo l'assenza del supporto per la clausola `check` nella definizione delle viste e il vincolo, non presente nello standard SQL, di non usare sottoquery nei vincoli costruiti con la clausola `check`. A parte queste lievi eccezioni, la definizione delle query e delle viste segue quindi la sintassi dello standard SQL.

Per esempio, moltiplicando la popolazione di una città per la percentuale di persone di madre lingua italiana nella nazione, si ottiene una stima del numero di persone italiane nella città. Si può quindi costruire la seguente query che estrae le città al di fuori dell'Italia con persone che parlano italiano, ordinate in base al loro numero:

```
select Name, City.CountryCode, District,
       Population*Percentage/100
from City join CountryLanguage
  on City.CountryCode = CountryLanguage.CountryCode
where Language = 'Italian'
  and City.CountryCode != 'ITA'
order by Population*Percentage desc
```

La seguente query invece estrae per ogni nazione il rapporto tra la popolazione della nazione e la somma degli abitanti delle sue città presenti nella base di dati, presentando i risultati in ordine decrescente del valore del rapporto.

```
select Country.Name, sum(City.Population)*100/Country.Population
from City join Country on City.CountryCode = Country.Code
group by Country.Name,Country.Code,Country.Population
order by sum(City.Population)*100/Country.Population desc
```

Infine, la definizione di una vista che rappresenti le città europee con almeno un milione di abitanti può avvenire con il seguente comando SQL:

```
create view LargeEuropeanCity as
 select City.*
 from City join Country
   on (City.CountryCode = Country.Code)
 where City.Population > 1000000
   and Continent = 'Europe'
```

C.5 Funzionalità evolute

Presentiamo ora sinteticamente alcune delle caratteristiche evolute di Postgres, che possono essere di interesse per la realizzazione di applicazioni sofisticate. Mostreremo le caratteristiche principali dell'estensione procedurale, l'integrazione con i linguaggi di programmazione di alto livello e la gestione di trigger. Alcuni dei principi che sono alla base della costruzione di questi servizi sono descritti nel secondo volume. Si ritiene comunque utile fornire un inquadramento di queste funzionalità, piuttosto che limitarsi ai soli aspetti trattati in questo primo volume.

C.5.1 Estensione procedurale

Come descritto nel Paragrafo 6.1, i sistemi relazionali spesso offrono il supporto per un'estensione procedurale di SQL, consentendo la costruzione di procedure

che possono essere gestite direttamente dal sistema, venendo incontro alle esigenze degli sviluppatori che debbano realizzare un'applicazione che necessita di svolgere un certo numero di passi di trasformazione relativamente semplici su dati che sono inseriti in ampie collezioni. In questi contesti, si osserva spesso un notevole vantaggio in termini di efficienza del sistema se si riesce a tenere la computazione vicina ai dati stessi, all'interno del motore relazionale.

L'estensione procedurale di SQL riconosciuta da Postgres va sotto il nome di *PL/pgSQL*. Il nome si ispira a PL/SQL, il nome dell'estensione procedurale di Oracle Server. PL/pgSQL si ispira non solo nel nome a PL/SQL, bensì anche nell'insieme di funzionalità e nella struttura sintattica degli arricchimenti procedurali. L'architettura di Postgres è fortemente modulare e per poter utilizzare il linguaggio PL/pgSQL è necessario averlo installato sul sistema, lanciando dalla shell il comando `createlang plpgsql` *NomeDatabase*. La necessità di caricare esplicitamente l'estensione procedurale è legata alla scelta di Postgres di mantenere un'elevata flessibilità, consentendo un facile arricchimento del sistema, che può infatti esteso in modo relativamente facile con il supporto per altri linguaggi.

Mostriamo alcuni semplici esempi del funzionamento di questa estensione, sulla base di dati di riferimento per questo capitolo. Il primo esempio mostra una procedura che estrae il nome della nazione in cui è parlato dal maggior numero di persone il linguaggio che viene passato come parametro.

```
create function
  NazionePiuParlanti(Linguaggio CountryLanguage.Language%type)
  returns Country.Name%type as
$corpo$
declare Nazione Country.Name%type;
begin
  select C.Name into Nazione
  from Country C join CountryLanguage CL
       on C.Code=CL.CountryCode
  where CL.Language = Linguaggio and
        C.Population * CL.Percentage >= all
            (select C1.Population * CL1.Percentage
             from Country C1 join CountryLanguage CL1
               on C1.Code=CL1.CountryCode
             where CL1.Language = Linguaggio);
  return Nazione;
end
$corpo$
language 'plpgsql';
```

Le prime righe della procedura `NazionePiuParlanti()` descrivono i parametri di ingresso e uscita; in questo caso la procedura riceve in ingresso un valore del dominio dell'attributo Language della tabella COUNTRYLANGUAGE e restituisce un valore dello stesso tipo dell'attributo Name della tabella COUNTRY. Viene poi il corpo della procedura, racchiuso da una coppia di stringhe identiche racchiuse dal carattere `$`; in questo caso il corpo della procedura è racchiuso tra due coppie della stringa `$corpo$`. Nel corpo della procedura si dichiara

inizialmente la variabile Nazione e si descrive poi la query SQL che dovrà essere eseguita all'esecuzione della funzione. Nella query compaiono riferimenti al parametro di ingresso Linguaggio, che assumerà al momento dell'esecuzione il valore attuale, secondo il classico paradigma di invocazione di procedure comune a tutti i linguaggi di programmazione. L'istruzione `return` restituisce il valore prodotto dalla funzione. Il codice della procedura termina con la specifica del linguaggio utilizzato per la scrittura del corpo della funzione, in questo caso PL/pgSQL. L'invocazione della procedura potrà avvenire nel contesto di una query SQL in ogni ambito nel quale è accettabile un valore del dominio. È una query corretta per esempio `select NazionePiuParlanti('Italian')`, così come la seguente, che restituisce per ogni lingua il numero totale di parlanti nelle diverse nazioni, il nome della nazione con il maggior numero di parlanti e la popolazione di questa nazione:

```
select Language, sum(Population*Percentage/100),
       NazionePiuParlanti(Language), C2.Population
from CountryLanguage join Country C1 on CountryCode=C1.Code,
     Country C2
where C2.Name=NazionePiuParlanti(Language)
group by Language, C2.Population
order by sum(Population*Percentage) desc
```

Il seguente frammento di codice introduce una funzione che recupera le città italiane e le restituisce all'ambiente chiamante. All'interno della procedura, l'insieme delle tuple viene considerato una tupla alla volta.

```
create or replace function cittaItaliane()
  returns setof city as
$$
declare c city%rowtype;
begin
  for c in select * from city
           where countryCode = 'ITA'
  loop
    if c.Population > 1000000
      return next c.Id | c.Name | c.District | 'Large'
    else
      return next c.Id | c.Name | c.District | 'Small'
  end loop;
  return;
end
$$
language 'plpgsql' ;
```

La procedura fa uso al suo interno di strutture di controllo, con una sintassi molto simile a quella usata in PL/SQL. L'invocazione della procedura può avvenire dovunque può essere valutata un'espressione che restituisce un insieme di tuple, come nell'ambito di una clausola `from`. Per esempio, si può scrivere:

```
select * from cittaItaliane();
```

C.5.2 Integrazione con altri linguaggi di programmazione

Come detto, Postgres si caratterizza per avere un'architettura esplicitamente aperta e configurabile. Uno degli obiettivi di disegno iniziale era proprio la costruzione di un sistema relazionale flessibile, in grado di interagire in molti modi con l'ambiente informatico esterno e con una forte apertura rispetto alla rigidità che caratterizzava i primi sistemi relazionali. Il disegno di Postgres si è dimostrato coerente con l'evoluzione dei sistemi relazionali i quali di norma offrono oggi questi servizi evoluti. Un aspetto rilevante, dal punto di vista della flessibilità del sistema e della sua capacità di interagire con il resto dei componenti di un sistema informatico, consiste nella capacità di interagire con i normali linguaggi di programmazione.

Postgres offre diverse modalità di integrazione. La modalità più semplice si basa sull'uso di SQL Embedded, che viene realizzato in Postgres rispettando in modo fedele lo standard SQL-2. La descrizione riportata nella Sezione 6.3 è direttamente applicabile in questo contesto. `ecpg` è lo strumento di preprocessing che converte un programma C con all'interno codice SQL embedded in un programma C direttamente compilabile. Postgres offre anche una libreria `libpq` che fornisce un'interfaccia diretta di dialogo tra un programma C/C++ e l'ambiente Postgres. Il sistema mette inoltre a disposizione librerie e moduli che consentono di interagire con il sistema tramite JDBC e ODBC. Infine, il sistema presenta anche alcuni componenti specifici che consentono di realizzare una stretta integrazione tra i servizi della base di dati e linguaggi quali Python, Perl e Tcl.

Al di là di questa varietà di servizi, che rientrano nelle caratteristiche di molti sistemi relazionali maturi, Postgres si caratterizza per la flessibilità che dimostra nell'integrazione, consentendo non solo di invocare i servizi del motore relazionale in modo sofisticato all'interno di applicazioni scritte in linguaggi di alto livello, ma anche di invocare direttamente dall'interno del sistema relazionale funzioni definite dall'utente utilizzando questi linguaggi di programmazione. Questo servizio offre quindi la possibilità a utenti esperti di estendere in modo relativamente facile il comportamento del sistema relazionale. Non è un caso che Postgres sia la piattaforma più utilizzata nell'ambito della ricerca per esplorare e dimostrare nuove funzionalità dei sistemi relazionali.

L'introduzione di funzioni esterne prevede di inserire nell'esecuzione di accessi ai dati il codice compilato di funzioni C che devono essere state preparate in modo tale da rispettare le convenzioni di gestione dei dati all'interno del motore relazionale. Gli aspetti tecnici da gestire sono relativamente complicati e vanno oltre il livello di dettaglio che caratterizza questa appendice. Al di là dell'intrinseca complessità associata allo sfruttamento di questi servizi, si può osservare che, grazie al suo disegno e alla sua natura di sistema open-source, Postgres presenta un grado di flessibilità non comune e può essere uno strumento interessante per la realizzazione di applicazioni sofisticate.

C.5.3 Gestione di trigger

Chiudiamo l'analisi delle funzioni evolute di Postgres descrivendo la modalità di gestione dei trigger. Ai trigger è dedicato il Capitolo 5 del secondo volume e sono stati descritti brevemente nella Sezione 6.2 di questo testo. In questa appendice mostriamo sinteticamente le caratteristiche specifiche della gestione dei trigger offerta da Postgres.

La sintassi è la seguente:

```
CREATE TRIGGER Nome
    { BEFORE | AFTER } { Evento [ OR ... ] }
    ON NomeTabella
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    EXECUTE PROCEDURE NomeFunzione ( Argomenti )
```

Le differenze principali rispetto ai trigger SQL standard sono l'assenza della condizione e la restrizione per cui l'azione del trigger consiste sempre nell'invocazione di una procedura, scritta in qualunque linguaggio, e non nell'esecuzione di istruzioni SQL. Nell'esempio si mostra una procedura di tipo `trigger` scritta in *PL/pgSQL*, ma Postgres ammette anche l'uso di funzioni C, opportunamente costruite. Le procedure C invocate nell'azione dei trigger devono essere progettate gestendo diversi parametri di interfaccia che devono essere tenuti in attenta considerazione, rendendo relativamente complicata la scrittura del codice. Un esempio di trigger Postgres è il seguente.

```
create function ControllaCitta()
  returns trigger as $ControllaCitta$
 begin
   -- Controlla che il valore di continente non sia nullo
     if new.Continent is null then
        raise exception 'Continente non puo` essere null';
     end if;
   -- Controlla se e` un valore accettabile
     if new.Population <= 0 then
        raise exception '% deve avere una popolazione positiva',
        new.Name;
     end if;
 end;
$ControllaCitta$ language plpgsql;

create trigger ControllaCitta
  before insert or update on City
  for each row
  execute procedure ControllaCitta();
```

I trigger possono essere sia *row-level*, con un'invocazione del trigger per ogni tupla distinta, sia *statement-level*, con un'invocazione unica del trigger in risposta al comando che ha generato l'evento, indipendentemente dal numero di tuple modificate dal comando. Il sistema permette anche di specificare sia trigger *before*, sia *after*, senza porre restrizioni nella sintassi all'azione dei trigger before (il manuale

del sistema consiglia giustamente prudenza). Per i trigger before row-level, Postgres assume che la procedura invocata nell'azione di un trigger insert o update restituisca una tupla, che sostituirà la tupla originariamente prodotta dal comando che ha attivato il trigger. Per quanto riguarda l'ordine di esecuzione dei trigger, qualora ve ne siano diversi pronti, Postgres segue l'ordine alfabetico sul nome del trigger. Postgres non prevede alcun limite esplicito al numero di invocazioni in cascata dei trigger; è quindi piena responsabilità del programmatore garantire la terminazione dei trigger.

Rispetto alla definizione dei trigger prevista dallo standard SQL-3, in Postgres è possibile fare riferimento alle variabili `old` e `new` nell'ambito dei trigger row-level, mentre non è ancora previsto il supporto per `old_table` e `new_table` nei trigger statement-level. Le variabili `old` e `new` inoltre non possono essere ridenominate. Non è poi possibile specificare trigger che reagiscono a eventi di update su specifici attributi. In SQL-3 l'azione può presentare più componenti da eseguire in sequenza, senza la necessità di introdurre un'apposita funzione. In generale, i limiti che presentano i trigger in Postgres sono di entità relativamente lieve e non pongono particolari ostacoli nello sviluppo di applicazioni. A causa dell'assenza di un meccanismo che limiti il numero di esecuzioni dei trigger, bisogna però prestare molta attenzione nella scrittura del codice. Alcune delle limitazioni dei trigger sono giustificate dalla presenza in Postgres di un meccanismo alternativo, le *rules*, che rappresentano uno strumento per la riscrittura di comandi SQL, che sono alla base di diversi meccanismi evoluti. Rimandiamo il lettore interessato alla consultazione della documentazione del sistema.