JACK FELLERS

MySQL

LA GUIDA COMPLETA E AGGIORNATA PER RISPARMIARE TEMPO E DENARO NELLA GESTIONE E PROGETTAZIONE DATABASE SQL. INCLUSI ESEMPIDI CODICI E FUNZIONI AGGIORNATE

WEBHAWK

MYSQL

JACK FELLERS

INDICE

Caro lettore, per ringraziarti per la fiducia dimostratami acquistando il mio libro, ecco per te in **regalo**, una guida per fortificare ancora di più la tua conoscenza nella programmazione web!

Scansiona il codice o clicca sul link per riscattarlo in meno di un minuto:

Link alternativo al Qr code:

https://webhawk.tech/optin-it/

Buona lettura!

PREMESSA

Lo sviluppo di un'applicazione richiede competenze diverse: che si tratti di un'applicazione per smartphone, un'applicazione Web o solo un sito Web, è necessario conoscere alcuni linguaggi di programmazione e concetti diversi sulla rete e la comunicazione Web.

Questi framework e linguaggi si evolvono rapidamente e spesso i programmatori non riescono a tenere il passo con gli aggiornamenti suggeriti, ma in tutto questo oceano una certezza c’è. Ovviamente è necessario archiviare i dati in un database, quindi è necessario fare la scelta giusta in base al progetto che si sta sviluppando. Una buona struttura, unita a un pensiero chiaro, ti aiuterà sicuramente a ottenere risultati eccellenti.

MySQL può sicuramente aiutarti perché con la sua semplicità e le sue prestazioni, ti dà la possibilità di creare applicazioni o siti Web con tempi di risposta molto bassi, soprattutto se puoi sfruttare appieno tutte le possibilità che questo database offre. Nel corso di questo Ebook, discuteremo diversi aspetti: dai punti di forza di MySQL alla sua struttura, dalle tabelle alle relazioni tra tabelle.

1

DATABASE

I database, detti anche DB, consentono di archiviare i dati in modo persistente. Infatti ogni giorno nei database vengono conservate grandi quantità di dati. Pensa a un grande database di e-commerce come quello di Amazon e pensa a quanto sia grande ed efficiente il database di Facebook, in quanto memorizza ciascuno dei nostri post, foto e video.

Inizialmente bisogna fare una distinzione tra un database e un DBMS, e bisogna avere una buona comprensione di cosa stiamo parlando.

- Un DBMS è un programma (software) che gestisce uno o più database, quindi i suoi compiti sono gestire gli utenti e i loro privilegi, gestire la sicurezza e l'ottimizzazione dei database contenuti.
- I database, invece, sono responsabili della conservazione dei dati in modo facilmente recuperabile e del rispetto di tutti i vincoli imposti. Di quali limiti stiamo parlando? Per vincoli si intendono le regole da rispettare, come garantire che il numero di elementi archiviati sia un valore numerico o che gli identificatori per gli elementi siano di una certa lunghezza e unicità.

Ad oggi esistono diverse tipologie di DBMS, ma sostanzialmente si possono suddividere in due categorie: NoSQL e RDBMS.

I database NoSQL sono nati di recente e sono ampiamente utilizzati quando si hanno molti dati, come utenti Facebook o ordini Amazon. Questo tipo di database ha un focus particolare sulla scalabilità, infatti possono facilmente ospitare incrementi di potenza di calcolo. Immagina cosa è successo su Amazon durante il Black Friday, con più accessi, più ordini e più utenti in un breve periodo di tempo. I database NoSQL si adattano facilmente, garantendo elevate prestazioni e grande flessibilità perché non strettamente legati alle relazioni tra i dati.

Il database di tipo RDBMS si basa sulla relazione tra i dati (ovvero la sigla di R), infatti ogni dato memorizzato segue una struttura rigida. Questa struttura permette di impostare regole e vincoli, ad esempio possiamo

garantire ad ogni voce che il campo data_di_nascita sia effettivamente una data e non una stringa. In questo modo sarà più facile ordinare o filtrare i risultati in un secondo momento. Queste relazioni sono basate su righe e colonne, quindi i dati sono organizzati in tabelle con indici ed eventualmente chiavi primarie per definire valori univoci nella tabella.

Tuttavia, un'altra importante differenza tra questi tipi di database è la garanzia delle proprietà ACID. I database NoSQL non li garantiscono, quindi tieni d'occhio nel caso in cui sei interessato a usarli e fai una buona valutazione per tutti i tuoi casi d'uso.

Riferimento proprietà ACID:

- Atomicità o garanzia che la transazione venga eseguita in modo deterministico o per niente;
- Coerente, ovvero i dati saranno sicuramente coerenti tra loro e quindi conformi allo schema del database;
- Isolamento significa che ogni transazione è separata dalle altre transazioni;
- Durabilità consente di ripristinare i dati all'ultimo stato noto.

Bene, se stai considerando un database NoSQL per il software di fatturazione, probabilmente non stai facendo la scelta giusta. Se vuoi creare grafici, creare applicazioni in tempo reale come quelle relative alle borse, ecc., i database NoSQL possono aiutarti.

Se i database NoSQL sono nati intorno agli anni 2000, i database relazionali sono ancora più vecchi, infatti, sono ampiamente utilizzati dal 1970 e continuano ad evolversi per garantire prestazioni migliori e sempre più Function. Oggi questa evoluzione si concretizza in un prodotto così facile da usare da poterlo associare ad un normale foglio di lavoro Excel. Creare, popolare ed esportare tabelle è facile come collegare un database a un'applicazione sviluppata in Java, PHP, Perl o altri linguaggi.

Una tabella crea relazioni al suo interno, ad esempio possiamo creare una tabella con tutti gli smartphone Samsung, utilizzando le seguenti colonne: nome articolo, nome modello, anno di produzione, processore, ecc. Immaginiamo però che due o più smartphone abbiano lo stesso processore, che sarebbero dati ridondanti duplicati su due o più righe. Per fare ciò, è possibile creare relazioni tra le tabelle per evitare questa

ridondanza e mantenere le tabelle più compatte con dati altamente coesi. Esploreremo varie possibili relazioni nei capitoli seguenti.

Probabilmente hai sentito così tanto parlare di SQL che appare spesso in molti nomi di database (MySQL, PostgreSQL, Sqlite, ecc.). SQL è l'acronimo di Structured Query Language e rappresenta un linguaggio che i database relazionali possono comprendere e interpretare per creare database, archiviare dati, recuperare dati, modificarli e altro ancora. Va ricordato che SQL permette anche di gestire e gestire database, consentendo l'accesso a diversi utenti.

Le classiche operazioni eseguite sui database sono la creazione, la lettura, la modifica e l'eliminazione dei dati. Queste operazioni sono denominate operazioni CRUD, acronimo di Crea, Leggi, Aggiorna ed Elimina.

I punti di forza

Continuando la nostra analisi dei database relazionali, determinando che sono più adatti ai nostri scopi, cerchiamo di capire perché dovremmo usare MySQL invece di un altro database relazionale.

Innanzitutto, MySQL è nato nel lontano 1996, quindi stiamo parlando di un prodotto molto maturo che si è evoluto nel tempo. Il suo sviluppo è in gran parte dovuto alla comunità, in quanto è un progetto open source e gratuito, quindi chiunque può analizzare il codice sorgente, modificarlo o contribuire a migliorarlo.

MySQL ha molti vantaggi: il più importante è la sua popolarità. L'uso di software popolari, ti consentirà di trovare soluzioni a ogni problema perché qualcuno potrebbe aver posto la tua stessa domanda. Nella maggior parte dei casi, la Ricerca Google può aiutarti ed evitare di perdere molto tempo.

Un altro vantaggio è ovviamente la sua affidabilità: infatti è garantito un funzionamento 24 ore su 24, 7 giorni su 7, motivo per cui viene spesso utilizzato per siti web insieme a CMS come Wordpress.

MySQL può essere installato su qualsiasi piattaforma, infatti funziona bene su Windows, Linux e macOS.

Oltre a tutto quello che abbiamo detto, il modo in cui lavora è molto apprezzato. Questo è il punto chiave perché è caratterizzato da un'elevata efficienza nella gestione di piccole quantità di dati e di grandi quantità di dati. Puoi ottimizzare il database a tuo piacimento, ottimizzare le query per recuperare i dati più velocemente, inserire indici nelle tabelle per semplificarne la ricerca e altro ancora.

Ultimo ma non meno importante, è molto facile integrarlo con applicazioni nuove o esistenti scritte in linguaggi di programmazione come Java, Python, C, C++ o anche Node.js.

Non è tutto oro quel che luccica, e infatti MySQL non è esente da problemi. Sebbene fosse molto ben progettato, ha iniziato ad avere alcuni problemi, soprattutto se confrontato con i nuovi prodotti basati su MySQL. Esistono infatti anche Fork (progetti basati su MySQL ma sviluppati da terze parti) con prestazioni leggermente migliori, circa il 5%.

Tuttavia, il positivo è che è facile migrare in uno di questi rami, molte volte è sufficiente esportare il database ed eseguire nuovamente la query sul nuovo database per completare con successo la migrazione.

2

INSTALLAZIONE

Esistono diverse versioni di MySQL e in questo ebook preferiamo usare la *Community Edition* in quanto gratuita. Questa versione è sottoposta a licenza GPL perciò può essere scaricata in modo gratuito e mette a disposizione diverse funzionalità che possono tornare utili. In questo capitolo, vedremo come installare MySQL su Linux, Windows e MacOS.

Linux

L’installazione su piattaforme basate su Linux è davvero semplice infatti si può scaricare sottoforma di archivio *.tar.gz* o come pacchetto grazie ad *apt* o *yum*.

Prendiamo in considerazione il package manager *apt*, basterà digitare i seguenti comandi:

- Sudo apt update
- Sudo apt install mysql-server

Questo comando installerà MySQL, ma non ti chiederà di impostare una password o apportare altre modifiche alla configurazione.

Per le nuove installazioni, ti consigliamo di eseguire lo script di sicurezza. Questo comando modifica alcune delle opzioni predefinite che sono meno sicure per gli accessi root remoti e gli utenti di esempio. Nelle versioni precedenti di MySQL, era possibile inizializzare manualmente anche la directory dei dati ma questo passaggio viene eseguito automaticamente.

Digita il seguente comando nel terminale:

- Sudo mysql_secure_installation

Questo comando ti guiderà attraverso una serie di istruzioni in cui puoi apportare alcune modifiche alle opzioni di sicurezza della tua installazione MySQL. Innanzitutto, ti verrà chiesto se desideri configurare il plug-in *Validate Password*, che può essere utilizzato per testare la validità della tua password MySQL. Indipendentemente dalla tua scelta, il prossimo step sarà quello di impostare una password per l'utente *root* di MySQL. Immetti la password e conferma la tua scelta. Da adesso, è possibile premere il tasto Y e quindi INVIO per accettare le impostazioni predefinite per tutte le domande successive. Ciò rimuoverà alcuni utenti anonimi e il database di test, disabiliterà gli accessi root remoti e caricherà queste nuove regole in modo che MySQL rispetti le modifiche apportate. Se l’installazione è

andata a buon fine e non vedi alcun messaggio di errore puoi eseguire questo comando nel tuo terminale per l'autenticazione:

- Mysql -u root -p

Dopo aver digitato questo comando puoi creare un nuovo utente ed assegnargli una password sicura:

- mysql> CREATE USER 'pippo'@'localhost' IDENTIFIED BY 'password';
- mysql> GRANT ALL PRIVILEGES ON *.* TO 'pippo'@'localhost' WITH GRANT OPTION;
- mysql> exit

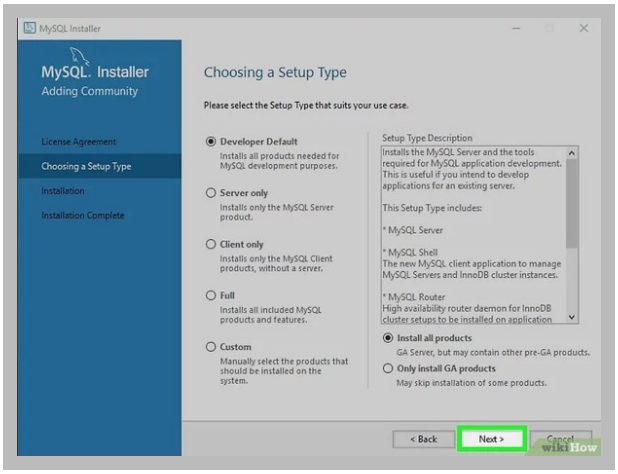
Il primo comando consente di creare l'utente pippo con la password scelta, il secondo concede al nuovo utente i privilegi appropriati. Nell'esempio sono stati concessi i privilegi a tutte le tabelle all'interno del database, nonché il potere di aggiungere, modificare e rimuovere i privilegi degli utenti. L'ultimo comando serve per uscire da MySQL.

Windows

In Windows il processo di installazione è completamente guidato da un'interfaccia grafica che spiega in maniera completa ed esaustiva quali sono gli step necessari per l'installazione e cosa si sta installando.

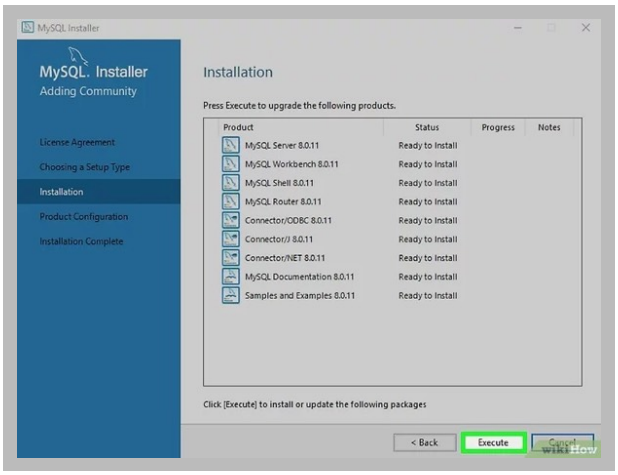
Per prima cosa è necessario collegarsi al sito ufficiale, raggiungibile all'indirizzo https://dev.mysql.com/downloads/windows/installer/8.0.html, per ottenere la Community Edition. Dopo aver cliccato su "Download" del pacchetto più piccolo in termini di dimensioni possiamo lanciare l'eseguibile scaricato. È anche possibile scaricare il pacchetto di dimensioni più grandi, risparmiando un po' di tempo in fase di installazione. Nella prima schermata viene chiesto cosa vogliamo installare: solo il server, solo il client, un'installazione completa ecc.

Selezioniamo la prima opzione (Developer Default) e installeremo tutti i prodotti che vediamo nella colonna di destra. Proseguiamo come nell'immagine:



Successivamente ci viene richiesto che tipo di architettura vogliamo usare pertanto selezioneremo *Standalone MySQL Server / Classic MySQL Replication*. Se vogliamo installare il database sul PC locale per motivi di sviluppo o per prendere confidenza con MySQL scegliamo *Development Machine*, se stiamo configurando un server dedicato scegliamo *Server Machine* altrimenti *Dedicated Machine* se si tratta di un server su cui sarà ospitato solo MySQL. Seleziona il protocollo TCP/IP e assicurati che sia impostata la porta di default per MySQL ovvero la 3306. Alla fine di queste configurazioni può iniziare l'installazione del software che richiederà diversi minuti a seconda dell'hardware su cui state installando.

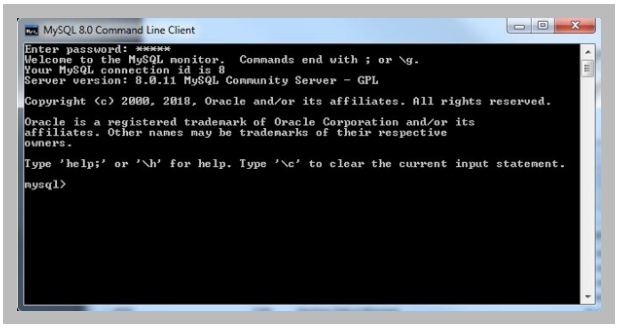
La pagina di installazione proporrà un riepilogo dei prodotti pronti per l'installazione:



Alla fine, selezioniamo *Strong Password* per specificare la password dell'utente principale, ti prego di custodire tale password in un luogo sicuro perché verrà spesso utilizzata. Questa password è fondamentale in quanto password dell'utente con i privilegi massimi.

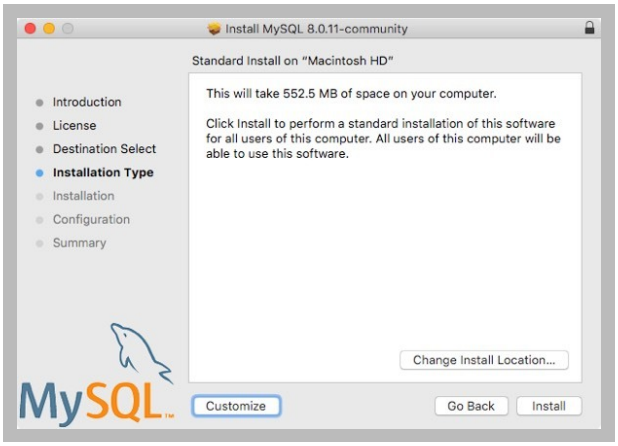
Quando richiesto sarà necessario impostare MySQL come servizio di Windows in modo che parta all'avvio del sistema operativo e non sarà necessario avviarlo manualmente. Proseguite con le impostazioni predefinite e successivamente vi consiglio di selezionare entrambe le voci per eseguire MySQL Workbench e MySQL Shell.

Se l'installazione è andata a buon fine vedrai un terminale simile al seguente:

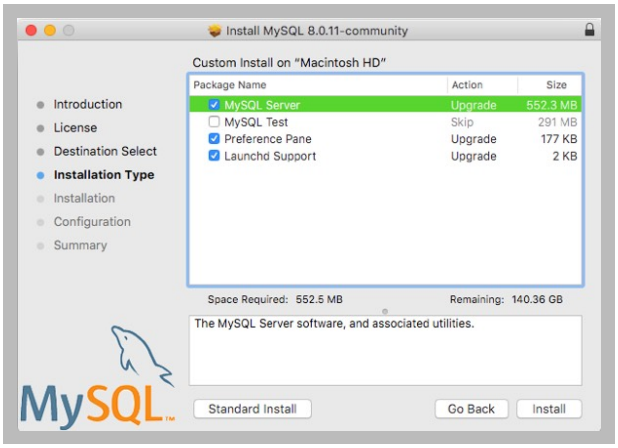


MacOS

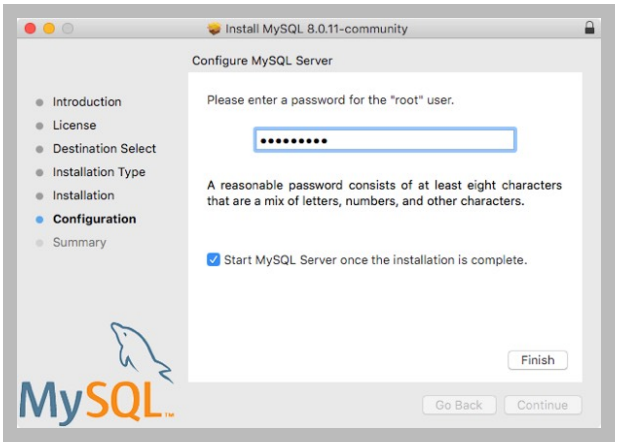
Puoi scaricare un archivio *.dmg* dal sito ufficiale di MySQL https://dev.mysql.com/downloads/mysql/ e seguire le istruzioni dell'interfaccia utente. Quando il download sarà completato potrai montare il disco immagine con un semplice doppio click per vedere il contenuto di quanto scaricato. La procedura guidata faciliterà l'installazione del software chiedendo la cartella dove si intende installare MySQL:



Successivamente è necessario selezionare tutte le opzioni come nell'immagine seguente:



Nei passi successivi è necessario definire il tipo di password da usare, selezioniamo *Use Strong Password Encryption* e digitiamo la password per l'utente root ovvero l'utente con i privilegi massimi:



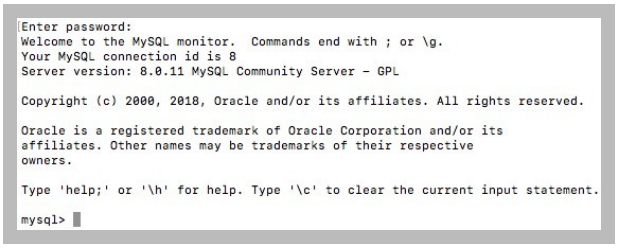
Se l'installazione è andata a buon fine saremo in grado di verificarlo con il comando:

mysql.server status

Se MySQL è avviato potremo entrare nella console tramite il comando:

mysql -u root -p

Il risultato, dopo aver inserito correttamente la password, sarà qualcosa simile alla seguente schermata:



3

COMANDI SQL

I comandi SQL sono utilizzati per comunicare con un database ed eseguire operazioni, funzioni e query sui dati. È possibile utilizzare i comandi SQL per effettuare ricerche nel database e per eseguire altre funzioni, come la creazione o l'eliminazione di tabelle e l'aggiunta o la modifica di dati. Ecco un elenco di comandi SQL di base (talvolta chiamati clausole) che è necessario conoscere se si ha bisogno di lavorare con SQL.

Database

In questo Ebook stiamo usando l’utente root per scopi dimostrativi ma è sempre consigliato usare un utente specifico con privilegi specifici su database o tabelle.

Dopo aver effettuato l’accesso alla console di MySQL con il comando:

mysql -u root -p

Il primo passaggio, nella gestione dei dati per qualsiasi database, è la creazione del database stesso. Questa attività può variare dall'elementare al complicato, a seconda delle tue esigenze. Molti sistemi includono strumenti grafici (come MySQL Workbench) che consentono di creare completamente il database con qualche click. Questa funzione è sicuramente utile per risparmiare tempo, ma dovresti comprendere le istruzioni SQL che vengono eseguite in risposta ai click del mouse. Attraverso l'esperienza personale, abbiamo imparato l'importanza di creare un buon script di installazione SQL. Un file di script contiene il codice SQL necessario per ricostruire completamente uno o più database; lo script spesso include elementi del database come indici, procedure e trigger.

La conoscenza della sintassi SQL è fondamentale in quanto ti consente di applicare le tue conoscenze ad altri sistemi di database relazionali. La prima considerazione riguarda il livello di autorizzazione infatti è necessario assicurarsi che si disponga delle impostazioni di autorizzazione a livello di amministratore di sistema o che l'amministratore di sistema abbia concesso l'autorizzazione per il comando **CREATE DATABASE**.

Per iniziare è fondamentale sapere quali sono i database già definiti di default durante l’installazione, MySQL consente di fare ciò tramite il comando:

mysql> show databases;

Probabilmente vedrai soltanto i database utili al funzionamento di MySQL se non ne hai ancora creati. Per creare un database di nome dbTEST sarà sufficiente digitare ed eseguire:

mysql> CREATE DATABASE dbTEST;

Dopo aver creato un database bisogna selezionarlo per i successivi comandi:

mysql> USE dbTEST;

A questo punto siamo pronti per usare il database appena creato, popolandolo di tabelle piene di dati. Prima di riempire le tabelle è importante **normalizzare** il database ovvero suddividere i dati in componenti separati per ridurre la ripetizione dei dati stessi. Esistono diversi livelli di normalizzazione ed ogni livello riduce la ripetizione dei dati. La normalizzazione dei dati può essere un processo estremamente complesso ma esistono numerosi strumenti di progettazione del database che possono esserti d’aiuto.

Ci sono diversi fattori che possono influenzare la progettazione del database, ad esempio lo spazio disponibile sul disco, la velocità con cui viene aggiornato il database o la velocità con cui vengono recuperati i dati.

Lo spazio sul disco è un fattore importante e da tenere in mente infatti anche se siamo in un’era dove gli smartphone possono archiviare Terabyte di dati, ricorda che più grande è il tuo database, più tempo ci vuole per recuperare i record. Se non hai ottimizzato la struttura del DB, è probabile che tu abbia ripetuto inutilmente gran parte dei tuoi dati. Spesso, però, può verificarsi il problema opposto infatti potresti aver cercato di normalizzare completamente la progettazione delle tue tabelle con il database e, in tal modo, hai creato molte tabelle. Anche in questo caso, qualsiasi operazione di query eseguita su questo database potrebbe richiedere molto tempo per essere eseguita. I database progettati in questo modo sono talvolta difficili da manutenere perché la struttura della tabella potrebbe oscurare l'intento del progettista. Questo problema sottolinea l'importanza di documentare sempre il codice o il design in modo che chi possa lavorare con te possa avere un'idea di cosa stavi pensando nel momento in cui hai creato la struttura del tuo database.

Tabelle

L'obiettivo di progettazione più importante che dovresti avere è quello di creare la struttura della tabella in modo tale che ognuna abbia una chiave primaria ed una chiave esterna. La chiave primaria serve per garantire che:

- Ogni record sia univoco all'interno di una tabella (nessun altro record all'interno della tabella ha tutte le sue colonne uguali a qualsiasi altro);
- i dati in una colonna non siano ripetuti in nessun altro punto della tabella.

Per quanto riguarda il secondo obiettivo, la colonna con dati completamente unici in tutta la tabella è nota come **chiave primaria**. Una chiave esterna è un campo che collega una tabella alla chiave primaria o alla chiave esterna di un'altra tabella. Facciamo un esempio: supponiamo di avere tre tabelle: BOLLETTE, CONTO_CORRENTE e AZIENDA.

La chiave primaria nella tabella BOLLETTE potrebbe essere il campo NOME o, sarebbe meglio, una combinazione di NOME con qualche altro campo. Il campo IBAN nella tabella CONTO_CORRENTE è la chiave primaria per quella tabella mentre il campo NOME è la chiave primaria per la tabella AZIENDA.

Le chiavi esterne in questo esempio sono probabilmente facili da individuare. Il campo IBAN nella tabella BOLLETTE unisce la tabella BOLLETTE alla tabella CONTO_CORRENTE. Il campo NOME nella tabella BOLLETTE unisce la tabella BOLLETTE alla tabella AZIENDA. Se si trattasse di un progetto di database completo, si otterrebbero molte più tabelle e suddivisioni dei dati. Ad esempio, il campo BANCA nella tabella CONTO_CORRENTE potrebbe puntare ad una tabella contenente tutte le informazioni bancarie come indirizzi e numeri di telefono. La tabella AZIENDA può essere collegata con un'altra tabella (o database) contenente informazioni sulla società e sui suoi prodotti.

CREATE

Vediamo come creare la tabella BOLLETTE in MySQL:

```
mysql> CREATE TABLE BOLLETTE (
-> NOME VARCHAR(30),
-> IMPORTO FLOAT,
-> IBAN VARCHAR(30));
Query OK, 0 rows affected (0.52 sec)
```

Abbiamo creato la tabella che conterrà il campo NOME di una lunghezza massima di 30 caratteri, IMPORTO e l'IBAN del conto corrente (che è univoco).

MySQL consente di identificare ciò che può essere archiviato in una colonna ed un valore NULL può sembrare quasi un ossimoro, perché avere un campo con un valore NULL vuol dire che il campo in realtà non ha alcun valore memorizzato in esso. Quando si crea una tabella, MySQL consente di indicare una colonna con le parole chiave NOT NULL in modo da indicare che la colonna non può contenere valori NULL per nessun record nella tabella. NOT NULL significa che ogni record deve avere un valore effettivo in questa colonna. Aggiorniamo l'esempio precedente con questa clausola:

```
mysql> CREATE TABLE BOLLETTE (
-> NOME VARCHAR(30) NOT NULL,
-> IMPORTO FLOAT,
-> IBAN VARCHAR(30) NOT NULL);
Query OK, 0 rows affected (0.52 sec)
```

In questa tabella vuoi salvare il nome dell'azienda a cui devi pagare una fattura, insieme all'importo della fattura. Se il campo NOME e/o IBAN non fosse memorizzato, il record sarebbe privo di significato.

INSERT

Adesso che abbiamo la tabella possiamo inserire i dati al suo interno:

mysql> INSERT INTO BOLLETTE VALUES("ENEL",22.5, IT000000001231231');

Query OK, 1 row affected (0.15 sec)

mysql> INSERT INTO BOLLETTE VALUES(NULL, 25000, 'IT0000000000012312331');

ERROR 1048 (23000): Column 'NOME' cannot be null

Puoi notare che il secondo record nell'esempio precedente non contiene un valore per il nome. Poiché la tabella è stata creata con NOT NULL per il campo NOME, è stato generato un errore. Una buona regola è che la chiave primaria e tutti i campi della chiave esterna non debbano mai contenere valori NULL.

Chiavi primarie

Uno dei tuoi obiettivi di progettazione dovrebbe essere quello di avere una colonna unica all'interno di ogni tabella. Questa colonna costituisce un campo per la chiave primaria in modo da impedire l'inserimento di valori di campo con chiave duplicati nel database. Dovresti notare diverse cose quando scegli un campo chiave infatti spesso viene usato un campo che viene incrementato per ogni riga aggiunta, il che rende questo campo per impostazione predefinita sempre una chiave univoca. Oltre ad essere davvero utile, è molto più veloce ritrovare un valore intero piuttosto che una stringa di 80 caratteri. Moltiplica questo concetto su centinaia di tabelle ed avrai dimensioni inferiori del database rispettando questa regola.

Ora possiamo creare le tabelle che abbiamo menzionato in precedenza:

```
mysql> CREATE TABLE CONTO_CORRENTE (
-> IBAN VARCHAR(30) NOT NULL,
-> SALDO FLOAT,
-> BANCA VARCHAR(30));
Query OK, 0 rows affected (0.51 sec)
```

```
mysql> CREATE TABLE AZIENDA (
-> NOME VARCHAR(30) NOT NULL,
-> INDIRIZZO VARCHAR(50),
-> CITTA VARCHAR(30),
-> STATO CHAR(2));
Query OK, 0 rows affected (0.74 sec)
```

Nella creazione della tabella ho usato diversi tipi di dati ma balza all'occhio la differenza tra CHAR e VARCHAR. Entrambi consentono di memorizzare delle stringhe di caratteri ma il primo ha una lunghezza fissa mentre il secondo ha lunghezza variabile. Entrambi consentono di definire una lunghezza massima ma con VARCHAR si occupa solo lo spazio necessario. Nel campo indirizzo potremmo memorizzare una stringa di 40 caratteri o di 10, consentendo un notevole risparmio rispetto all'uso di CHAR.

Inseriamo alcuni dati al loro interno:

```
mysql> INSERT INTO CONTO_CORRENTE VALUES('IT0000000000012312331', 105.22, 'BANCA 1');
Query OK, 1 row affected (0.20 sec)

mysql> INSERT INTO AZIENDA VALUES('ENEL', 'VIA PIPPO, 2', 'ROMA','IT');
Query OK, 1 row affected (0.34 sec)
```

Come avrai notato è possibile inserire i comandi SQL su un’unica riga come in questo caso o su più righe. È fondamentale, in entrambi i casi, terminare ogni singola istruzione con un punto e virgola (;) in modo che l’interprete possa capire dove finisce un’istruzione e ne inizia un’altra, soprattutto se si creano degli script con molteplici istruzioni.

ALTER TABLE

Molte volte la progettazione del database cambia perché cambiano i requisiti o magari abbiamo dimenticato qualcosa che va aggiunta. L'istruzione ALTER TABLE consente all'amministratore o al progettista del database di modificare la struttura di una tabella dopo che è stata creata. Il comando ALTER TABLE consente di eseguire sostanzialmente due operazioni:

- Aggiungere una colonna ad una tabella esistente
- Modificare una colonna già esistente
- Cancellare una colonna

La sintassi prevede rispettivamente:

- ALTER TABLE nome_tabella ADD nome_colonna tipo_colonna;
- ALTER TABLE nome_tabella MODIFY COLUMN nome_colonna tipo_colonna;
- ALTER TABLE nome_tabella DROP COLUMN nome_colonna tipo_colonna.

Ad esempio, proviamo a modificare il campo STATO della tabella AZIENDA:

mysql> ALTER TABLE AZIENDA MODIFY COLUMN STATO VARCHAR(3);

Query OK, 1 row affected (1.33 sec)

Records: 1 Duplicates: 0 Warnings: 0

Puoi aumentare o diminuire la lunghezza delle colonne a tuo piacere; comunque non è possibile ridurre la lunghezza di una colonna se la

dimensione di uno dei suoi valori è maggiore del valore che si desidera assegnare alla lunghezza della colonna.

Per l’istruzione ALTER TABLE, tuttavia, esistono alcune restrizioni infatti si può cambiare una colonna da NOT NULL a NULL, ma non viceversa. Una colonna può essere modificata da NULL a NOT NULL solo se la colonna non contiene alcun valore NULL.

DROP TABLE

MySQL fornisce ovviamente un comando per rimuovere completamente una tabella da un database. Il comando DROP TABLE elimina una tabella insieme a tutte le viste e gli indici associati. È necessario prestare attenzione con questo comando perché dopo averlo eseguito, non è più possibile tornare indietro.

L'uso più comune dell'istruzione DROP TABLE è legato alla creazione di tabelle temporanee. Dopo aver completato tutte le operazioni sulla tabella temporanea, si può digitare un’istruzione DROP TABLE simile alla seguente:

```
mysql> DROP TABLE TABELLA_TEMP;
```

DROP DATABASE

Qualora si intenda cancellare un intero database, per esempio, dopo una migrazione avvenuta con successo si può usare il seguente comando:

```
mysql> DROP DATABASE NOME_DB;
```

4

RECUPERARE I DATI

Spesso si parla di **query SQL** e spesso si pensa ad un'interrogazione del database per recuperare dati. In realtà una query SQL può essere un comando per eseguire una delle seguenti operazioni:

- Cercare in diverse tabelle informazioni specifiche e restituire i risultati in un ordine specifico;
- Creare o eliminare una tabella;
- Inserire, modificare o eliminare righe o campi.

La sintassi SQL è abbastanza flessibile e facile da capire, sebbene ci siano regole da seguire come in qualsiasi linguaggio di programmazione. Presta molta attenzione alle maiuscole e alle minuscole, alla spaziatura e alla separazione logica dei componenti di ciascuna query per parole chiave SQL. Scrivere bene delle query aiuterà te e chiunque altro a capire rapidamente cosa stai cercando di fare.

SELECT

Per recuperare i dati dalla tabella creata e popolata nel capitolo precedente puoi usare l’istruzione SELECT:

```
mysql> SELECT * FROM CONTO_CORRENTE;
+--------------------+--------+---------+
| IBAN | SALDO | BANCA |
+--------------------+--------+---------+
| IT000000000012312331 | 105.22 | BANCA 1 |
+--------------------+--------+---------+
1 row in set (0.01 sec)
```

L'asterisco (*) in SELECT * indica al database di restituire tutte le colonne associate alla tabella indicata descritta dalla clausola FROM. Il database determina l'ordine in cui restituire le colonne. Per specificare l'ordine delle colonne, è possibile digitare qualcosa del tipo:

```
mysql> SELECT BANCA, IBAN, SALDO FROM CONTO_CORRENTE;
+---------+--------------------+--------+
| BANCA | IBAN | SALDO |
+---------+--------------------+--------+
| BANCA 1 | IT000000000012312331 | 105.22 |
+---------+--------------------+--------+
1 row in set (0.00 sec)
```

Devi notare che il nome di ogni colonna è elencato nella clausola SELECT. L'ordine in cui sono elencate le colonne è l'ordine in cui appariranno nell'output. L’elenco delle colonne da mostrare è separato da virgole e la clausola FROM è separata da uno spazio. Se non hai intenzione di mostrare tutte le colonne presenti in una tabella puoi anche limitarti a selezionare una colonna come segue:

```
mysql> SELECT IBAN FROM CONTO_CORRENTE;
+--------------------+
```

```
| IBAN |
+---------------------+
| IT000000000012312331 |
+---------------------+
1 row in set (0.00 sec)
```

DISTINCT

Talvolta alcuni dati possono ripetersi all’interno di una colonna e, se abbiamo bisogno valori univoci, è necessario usare la parola chiave DISTINCT. Inseriamo altre righe nella tabella BOLLETTE:

```
mysql> INSERT INTO BOLLETTE VALUES('FASTWEB', 35, 'IT000000001231231');
Query OK, 1 row affected (0.20 sec)

mysql> INSERT INTO BOLLETTE VALUES('ENEL', 104.22, 'IT000000003333334');
Query OK, 1 row affected (0.10 sec)

mysql> INSERT INTO BOLLETTE VALUES('FASTWEB', 110, 'IT000000003333334');
Query OK, 1 row affected (0.18 sec)

mysql> SELECT * FROM BOLLETTE;
+------------+---------+------------------+
| NOME | IMPORTO | IBAN |
+------------+---------+------------------+
| Amazon AWS | 22.5 | IT000000001231231 |
| FASTWEB | 35 | IT000000001231231 |
| ENEL | 104.22 | IT000000003333334 |
| FASTWEB | 110 | IT000000003333334 |
+------------+---------+------------------+
4 rows in set (0.00 sec)
```

In questo caso abbiamo aggiunto le bollette dell’ufficio che paghiamo da un altro conto corrente, quindi, con un codice IBAN diverso. Come puoi

vedere nella SELECT vengono restituiti tutti i valori ma in questo caso vogliamo una lista di codici IBAN senza duplicati:

```
mysql> SELECT DISTINCT IBAN FROM BOLLETTE;
+------------------+
| IBAN |
+------------------+
| IT000000001231231 |
| IT000000003333334 |
+------------------+
2 rows in set (0.12 sec)
```

Se vuoi trovare un determinato articolo o gruppo di elementi nel tuo database, hai bisogno di una o più condizioni. Le condizioni sono contenute nella clausola WHERE, ad esempio cerchiamo tutte le bollette pagate relative a Fastweb:

```
mysql> SELECT * FROM BOLLETTE WHERE NOME='FASTWEB';
+---------+---------+------------------+
| NOME | IMPORTO | IBAN |
+---------+---------+------------------+
| FASTWEB | 35 | IT000000001231231 |
| FASTWEB | 110 | IT000000003333334 |
+---------+---------+------------------+
2 rows in set (0.00 sec)
```

SELECT, FROM e WHERE sono le tre clausole più utilizzate in MySQL. WHERE semplicemente fa in modo che le tue query siano più selettive infatti non usandola, la cosa più utile che potresti fare con una query, è visualizzare tutti i record nella tabella selezionata.

5

GLI OPERATORI

Gli operatori sono gli elementi usati all'interno di un'espressione per articolare il modo in cui desideri che determinate condizioni recuperino i dati. Gli operatori si dividono in diversi gruppi: aritmetici, di confronto, per caratteri, logici ecc.

Aritmetici

Gli operatori aritmetici sono addizione (+), sottrazione (-), moltiplicazione (*), divisione (/) e modulo (%). I primi quattro sono semplici da capire e li conosciamo tutti. L'operatore modulo, invece, restituisce il resto intero di una divisione. È possibile usare questi operatori nella selezione dei dati dalle tabelle. Supponiamo di dover aggiungere ad ogni valore nella tabella BOLLETTE il costo per effettuare il bonifico che è di € 1,50:

mysql> SELECT NOME, IMPORTO, IMPORTO+1.5 FROM BOLLETTE;

```
+---------+---------+-------------------+
| NOME | IMPORTO | IMPORTO+1.5 |
+---------+---------+-------------------+
| ENEL | 22.5 | 24 |
| FASTWEB | 35 | 36.5 |
| ENEL | 104.22 | 105.72 |
| FASTWEB | 110 | 111.5 |
+---------+---------+-------------------+
4 rows in set (0.00 sec)
```

In questo caso abbiamo aggiunto il costo del bonifico e MySQL assume come intestazione della colonna IMPORTO+1.5. Il nome di questa colonna potrebbe essere difficile da usare o ricordare, tuttavia, possiamo assegnare un nome diverso tramite un **alias**. Un alias viene definito tramite la parola chiave AS quindi proviamo a rinominare la colonna IMPORTO TOTALE:

mysql> SELECT NOME, IMPORTO, IMPORTO+1.5 AS 'IMPORTO TOTALE' FROM BOLLETTE;

```
+---------+---------+-------------------+
| NOME | IMPORTO | IMPORTO TOTALE |
+---------+---------+-------------------+
| ENEL | 22.5 | 24 |
| FASTWEB | 35 | 36.5 |
| ENEL | 104.22 | 105.72 |
| FASTWEB | 110 | 111.5 |
```

```
+---------+---------+-------------------+
4 rows in set (0.00 sec)
```

Attenzione! Tutto quello che vedi nella tabella è frutto di un’elaborazione successiva al recupero dei dati infatti se recuperi i dati della tabella vedrai che conterrà i dati originali. In sostanza, abbiamo creato una colonna virtuale o derivata, modificando i valori di una colonna esistente. Lo stesso può essere fatto per sottrazione, moltiplicazione e divisione:

```
mysql> SELECT NOME, IMPORTO, IMPORTO-1 AS 'IMPORTO SCONTATO' FROM BOLLETTE;
+---------+---------+-------------------+
| NOME | IMPORTO | IMPORTO SCONTATO |
+---------+---------+-------------------+
| ENEL | 22.5 | 21.5 |
| FASTWEB | 35 | 34 |
| ENEL | 104.22 | 103.22000122070312 |
| FASTWEB | 110 | 109 |
+---------+---------+-------------------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT NOME,IMPORTO, IMPORTO-(IMPORTO*0.1) AS 'SCONTATO del 10%' FROM BOLLETTE;
+---------+---------+------------------+
| NOME | IMPORTO | SCONTATO del 10% |
+---------+---------+------------------+
| ENEL | 22.5 | 20.25 |
| FASTWEB | 35 | 31.5 |
| ENEL | 104.22 | 93.79800109863281 |
| FASTWEB | 110 | 99 |
+---------+---------+------------------+
4 rows in set (0.00 sec)
```

mysql> SELECT NOME,IMPORTO, IMPORTO/2 AS 'SCONTATO del 50%' FROM BOLLETTE;

```
+---------+---------+------------------+
| NOME | IMPORTO | SCONTATO del 50% |
+---------+---------+------------------+
| ENEL | 22.5 | 11.25 |
| FASTWEB | 35 | 17.5 |
| ENEL | 104.22 | 52.11000061035156 |
| FASTWEB | 110 | 55 |
+---------+---------+------------------+
4 rows in set (0.00 sec)
```

Infine, l’operatore modulo restituisce il resto di una divisione perciò supponiamo di dividere 176 caramelle tra 6 bambini:

```
mysql> SELECT 176%6 AS 'CARAMELLE RESTANTI';
+--------------------+
| CARAMELLE RESTANTI |
+--------------------+
| 2 |
+--------------------+
1 row in set (0.00 sec)
```

Confronto

Fedeli al loro nome, questi operatori confrontano le espressioni e restituiscono un valore: TRUE, FALSE o NULL cioè vero, falso o sconosciuto. Qui bisogna prestare attenzione perché MySQL si comporta in modo inaspettato con i valori NULL. Confronti del tipo valore = NULL o valore <> NULL producono sempre un risultato NULL (che non è né vero né falso) perché non è possibile decidere se veri o falsi. Di conseguenza anche NULL = NULL restituisce NULL perché non è possibile determinare se un valore sconosciuto è uguale ad un altro valore sconosciuto. Per verificare davvero se un valore è NULL, bisogna utilizzare gli operatori IS NULL e IS NOT NULL.

mysql> SELECT NOME, IMPORTO FROM BOLLETTE WHERE IMPORTO IS NULL;

Empty set (0.00 sec)

mysql> SELECT NOME,IMPORTO FROM BOLLETTE WHERE IMPORTO IS NOT NULL;

```
+---------+---------+
| NOME | IMPORTO |
+---------+---------+
| ENEL | 22.5 |
| FASTWEB | 35 |
| ENEL | 104.22 |
| FASTWEB | 110 |
+---------+---------+
```

4 rows in set (0.00 sec)

Gli operatori di confronto sono sei:

- Maggiore (>)
- Maggiore o uguale (>=)
- Minore (<)

- Minore o uguale (<=)
- Uguale (=)
- Diverso (<>)

Facciamo qualche esempio con i dati inseriti nella tabella BOLLETTE:

mysql> SELECT NOME, IMPORTO FROM BOLLETTE WHERE IMPORTO < 40;

```
+---------+---------+
| NOME | IMPORTO |
+---------+---------+
| ENEL | 22.5 |
| FASTWEB | 35 |
+---------+---------+
2 rows in set (0.00 sec)
```

mysql> SELECT NOME, IMPORTO FROM BOLLETTE WHERE IMPORTO >= 35;

```
+---------+---------+
| NOME | IMPORTO |
+---------+---------+
| FASTWEB | 35 |
| ENEL | 104.22 |
| FASTWEB | 110 |
+---------+---------+
3 rows in set (0.00 sec)
```

mysql> SELECT NOME, IMPORTO FROM BOLLETTE WHERE IMPORTO = 110;

```
+---------+---------+
| NOME | IMPORTO |
+---------+---------+
| FASTWEB | 110 |
+---------+---------+
1 row in set (0.00 sec)
```

```
mysql> SELECT NOME, IMPORTO FROM BOLLETTE WHERE IMPORTO <> 110;
+---------+---------+
| NOME | IMPORTO |
+---------+---------+
| ENEL | 22.5 |
| FASTWEB | 35 |
| ENEL | 104.22 |
+---------+---------+
3 rows in set (0.00 sec)
```

Caratteri

È possibile utilizzare gli operatori dei caratteri per manipolare il modo in cui sono rappresentate le stringhe di caratteri, sia nell'output dei dati sia nel processo di posizionamento delle condizioni sui dati da recuperare. Se volessi selezionare parti di un database che si adattano ad un pattern ma non corrispondono esattamente? È possibile utilizzare il segno uguale ed eseguire tutti i casi possibili, ma tale processo sarebbe noioso e richiederebbe molto tempo.

La parola chiave in questo caso è LIKE:

mysql> SELECT NOME FROM BOLLETTE WHERE NOME LIKE 'FA%';

```
+---------+
| NOME |
+---------+
| FASTWEB |
| FASTWEB |
+---------+
```

2 rows in set (0.00 sec)

In questo caso abbiamo selezionato tutti i nomi che iniziano per “FA” perciò il database restituisce soltanto due righe. Se utilizzato all'interno di un'espressione LIKE, % è un carattere jolly. Puoi spostare il carattere jolly all’inizio della stringa in modo da cercare tutti i campi che terminano per “FA”:

mysql> SELECT NOME FROM BOLLETTE WHERE NOME LIKE '%FA';

Empty set (0.00 sec)

Un altro modo per cercare tra le stringhe, limitandoci ad un singolo carattere come jolly, consiste nell’usare il trattino basso (_):

mysql> SELECT DISTINCT NOME FROM BOLLETTE WHERE NOME LIKE 'ENE_';

```
+------+
| NOME |
```

```
+------+
| ENEL |
+------+
1 row in set (0.00 sec)
```

Puoi anche combinare i due elementi in modo da cercare tutte le stringhe che contengono una E seguita da una sola lettera ma preceduta da altre lettere:

```
mysql> SELECT NOME FROM BOLLETTE WHERE NOME LIKE '%E_';
+---------+
| NOME |
+---------+
| ENEL |
| FASTWEB |
| ENEL |
| FASTWEB |
+---------+
4 rows in set (0.00 sec)
```

Logici

Gli operatori logici separano due o più condizioni nella clausola WHERE di un'istruzione SQL. Vogliamo cercare una bolletta da pagare, sappiamo che è di Fastweb e che ammonta a 35 euro. Possiamo usare l'operatore logico AND per unire queste condizioni in una WHERE:

```
mysql> SELECT * FROM BOLLETTE WHERE NOME = 'FASTWEB' AND IMPORTO = 35;
+---------+---------+------------------+
| NOME | IMPORTO | IBAN |
+---------+---------+------------------+
| FASTWEB | 35 | IT000000001231231 |
+---------+---------+------------------+
1 row in set (0.00 sec)
```

AND indica che le espressioni su entrambi i lati devono essere vere per restituire TRUE. Se una delle espressioni è falsa, AND restituisce FALSE.

Puoi anche usare OR per riassumere una serie di condizioni. Se uno dei confronti è vero, OR restituisce TRUE:

```
mysql> SELECT * FROM BOLLETTE WHERE NOME = 'FASTWEB' OR IMPORTO < 40;
+---------+---------+------------------+
| NOME | IMPORTO | IBAN |
+---------+---------+------------------+
| ENEL | 22.5 | IT000000001231231 |
| FASTWEB | 35 | IT000000001231231 |
| FASTWEB | 110 | IT000000003333334 |
+---------+---------+------------------+
3 rows in set (0.00 sec)
```

Come vedi per questa query abbiamo un risultato composto da 3 righe perché vengono unite quelle con nome pari a "FASTWEB" con quelle che hanno importo inferiore a 40 euro.

Un altro operatore spesso utilizzato è il NOT che nega un’intera espressione. Spesso viene utilizzato per mantenere una certa leggibilità del codice dato che si potrebbe usare l’operatore <> per verificare la disuguaglianza. Se la condizione a cui viene applicato è TRUE, NOT la rende FALSE. Se la condizione dopo l’operatore NOT è FALSE, diventa TRUE:

```
mysql> SELECT NOME FROM BOLLETTE WHERE NOME NOT LIKE '%EL';
+---------+
| NOME |
+---------+
| FASTWEB |
| FASTWEB |
+---------+
2 rows in set (0.00 sec)
```

Insiemistici

Le tabelle possono essere viste come degli insiemi di elementi perciò esistono dei comandi SQL che ti consentono di unire le righe di due tabelle come se fossero degli insiemi. Uno di questi comandi è UNION che restituisce i risultati di due query tranne le righe duplicate:

mysql> SELECT NOME FROM AZIENDA UNION SELECT NOME FROM BOLLETTE;

```
+---------+
| NOME |
+---------+
| AMAZON |
| ENEL |
| FASTWEB |
+---------+
3 rows in set (0.00 sec)
```

Questo comando è molto utile quando si hanno valori duplicati o ripetuti in diverse tabelle perché magari importate da altri database o altri sistemi. UNION ALL ha lo stesso funzionamento di UNION, tranne per il fatto che non elimina i duplicati.

Un altro modo per selezionare gli elementi contenuti in un altro insieme consiste nell’usare la parola chiave IN:

mysql> SELECT * FROM BOLLETTE WHERE NOME IN ('ENEL', 'TEST');

```
+------+---------+------------------+
| NOME | IMPORTO | IBAN |
+------+---------+------------------+
| ENEL | 22.5 | IT000000001231231 |
| ENEL | 104.22 | IT000000003333334 |
+------+---------+------------------+
2 rows in set (0.00 sec)
```

La query appena scritta equivale ad una clausola WHERE con più condizioni in OR quindi corrisponde a:

```
mysql> SELECT * FROM BOLLETTE WHERE NOME = 'ENEL' OR NOME = 'TEST';
+------+---------+------------------+
| NOME | IMPORTO | IBAN |
+------+---------+------------------+
| ENEL | 22.5 | IT000000001231231 |
| ENEL | 104.22 | IT000000003333334 |
+------+---------+------------------+
2 rows in set (0.00 sec)
```

In questo caso risulta molto più leggibile e facile da usare la clausola IN rispetto a diverse condizioni in OR.

Un’altra clausola può essere molto utile quando si vuole selezionare una serie di valori compresi tra due estremi (inclusi). Con la parola chiave BETWEEN si indicano tutti gli elementi compresi tra il valore a sinistra (estremo inferiore) e l’elemento a destra (estremo superiore) della clausola:

```
mysql> SELECT * FROM BOLLETTE WHERE IMPORTO BETWEEN 22 AND 105;
+---------+---------+------------------+
| NOME | IMPORTO | IBAN |
+---------+---------+------------------+
| ENEL | 22.5 | IT000000001231231 |
| FASTWEB | 35 | IT000000001231231 |
| ENEL | 104.22 | IT000000003333334 |
+---------+---------+------------------+
3 rows in set (0.00 sec)
```

6

FUNZIONI

Le funzioni in SQL consentono di eseguire delle azioni come determinare la somma di una colonna o la conversione in maiuscolo di tutti i caratteri di una stringa.

Esistono diversi tipi di funzioni:

- Funzioni aggregate
- Funzioni per data e ora
- Funzioni aritmetiche
- Funzioni per i caratteri
- Funzioni di conversione
- Funzioni varie

Le funzioni aumentano notevolmente la tua capacità di manipolare le informazioni recuperate utilizzando le funzioni di base di SQL descritte in precedenza.

Funzioni aggregate

Le prime cinque funzioni aggregate, COUNT, SUM, AVG, MAX e MIN, definite nello standard ANSI.

Le funzioni di aggregazione sono anche chiamate funzioni di gruppo e restituiscono un valore basato sui valori in una colonna. La funzione COUNT restituisce il numero di righe che soddisfano una condizione nella clausola WHERE. Potresti anche voler sapere quante righe sono contenute nella tua tabella quindi digiterai:

```
mysql> SELECT COUNT(*) AS 'NUMERO BOLLETTE' FROM BOLLETTE;
+-----------------+
| NUMERO BOLLETTE |
+-----------------+
| 4 |
+-----------------+
1 row in set (0.00 sec)
```

Mentre per sapere quante bollette sono di Fastweb la query sarà:

```
mysql> SELECT COUNT(*) AS 'BOLLETTE FASTWEB' FROM BOLLETTE WHERE NOME = 'FASTWEB';
+------------------+
| BOLLETTE FASTWEB |
+------------------+
| 2 |
+------------------+
1 row in set (0.12 sec)
```

La funziona SUM addiziona tutti i valori in una colonna, per esempio se volessimo calcolare il totale di tutte le bollette:

```
mysql> SELECT SUM(IMPORTO) AS TOTALE FROM BOLLETTE;
+------------------+
| TOTALE |
```

```
+------------------+
| 271.7200012207031 |
+------------------+
1 row in set (0.00 sec)
```

La funzione SUM prende in input solo valori numerici quindi non restituisce alcun risultato sui caratteri. Inoltre, puoi usare le condizioni WHERE per filtrare le righe da sommare.

La funzione AVG consente di creare la media dei valori contenuti in una colonna. Ad esempio, la media delle nostre bollette sarà pari a:

```
mysql> SELECT AVG(IMPORTO) AS MEDIA FROM BOLLETTE;
+------------------+
| MEDIA |
+------------------+
| 67.93000030517578 |
+------------------+
1 row in set (0.00 sec)
```

Molto semplici ed intuitive sono le funzioni MAX e MIN che consentono di recuperare rispettivamente il valore massimo ed il valore minimo contenuti in una colonna:

```
mysql> SELECT MAX(IMPORTO) AS 'IMPORTO MASSIMO' FROM BOLLETTE;
+-----------------+
| IMPORTO MASSIMO |
+-----------------+
| 110 |
+-----------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT MIN(IMPORTO) AS 'IMPORTO MINIMO' FROM BOLLETTE;
+---------------+
```

```
| IMPORTO MINIMO |
+----------------+
| 22.5 |
+----------------+
1 row in set (0.00 sec)
```

A fini statistici possono aiutare anche la varianza e la deviazione standard. La varianza fornisce una misura della variabilità dei valori assunti dalla variabile stessa; in particolare, la misura di quanto essi si discostino in modo quadratico rispetto alla media aritmetica o del valore atteso. La deviazione standard, invece, è un indice di dispersione statistico, vale a dire una stima della variabilità di una popolazione di dati. Useremo la parola chiave VARIANCE per calcolare la varianza e STD_DEV per calcolare la deviazione standard:

```
mysql> SELECT VARIANCE(IMPORTO) AS 'VARIANZA' FROM BOLLETTE;
+--------------------+
| VARIANZA |
+--------------------+
| 1558.7797221496585 |
+--------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT STDDEV(IMPORTO) AS 'DEVIAZIONE STANDARD' FROM BOLLETTE;
+---------------------+
| DEVIAZIONE STANDARD |
+---------------------+
| 39.48138450142875 |
+---------------------+
1 row in set (0.00 sec)
```

Funzioni per data e ora

Spesso è fondamentale usare delle date in un database per memorizzare il giorno, il mese e l'anno di un evento oppure per memorizzare l'orario e la data di accesso di un utente.

La funzione ADDDATE consente di aggiungere un determinato valore ad una determinata unità temporale, ad esempio giorni, mesi, anni ecc.

```
mysql> SELECT NOW() AS ADESSO;
+---------------------+
| ADESSO |
+---------------------+
| 2020-06-27 12:48:27 |
+---------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT ADDDATE(NOW(),INTERVAL 1 MONTH) AS 'TRA UN MESE';
+---------------------+
| TRA UN MESE |
+---------------------+
| 2020-07-27 12:48:35 |
+---------------------+
1 row in set (0.00 sec)
```

I valori che possibile specificare sono molti ma i più usati sono:

- MICROSECOND
- SECOND
- MINUTE
- HOUR
- DAY
- WEEK
- MONTH

- YEAR
- MINUTE_SECOND
- HOUR_MINUTE
- DAY_HOUR
- YEAR_MONTH

Come puoi notare sono tutti abbastanza semplici da capire ma gli ultimi possono trarre in inganno. Se volessimo aggiungere ore e minuti oppure minuti e secondi ad una data, non serve invocare due volte la funzione. Useremo rispettivamente HOUR_MINUTE e MINUTE_SECOND per risparmiare tempo:

```
mysql> SELECT ADDDATE(NOW(),INTERVAL '1:30' HOUR_MINUTE) AS 'TRA UN\'ORA E MEZZO';
+--------------------+
| TRA UN'ORA E MEZZO |
+--------------------+
| 2020-06-27 14:24:32 |
+--------------------+
1 row in set (0.00 sec)
```

Personalmente dimentico sempre quanti giorni hanno i mesi e, se non fosse per la famosa filastrocca, non saprei proprio come fare. Possiamo usare la funzione LAST_DAY() per recuperare l’ultimo giorno di un mese:

```
mysql> SELECT LAST_DAY(NOW());
+-----------------+
| LAST_DAY(NOW()) |
+-----------------+
| 2020-06-30 |
+-----------------+
1 row in set (0.01 sec)
```

MySQL è davvero utile con le date, infatti, dando in input la tua data di nascita può calcolare quanti giorni fa sei nato. Non ci credi? Verifichiamo insieme:

```
mysql> SELECT DATEDIFF(NOW(), "1980-05-11") AS 'QUANTI GIORNI FA\' SONO NATO?';
+-----------------------------+
| QUANTI GIORNI FA' SONO NATO? |
+-----------------------------+
| 14657 |
+-----------------------------+
1 row in set (0.00 sec)
```

Immagina quanto è potente MySQL e quanto può semplificarti la vita. Potresti fare lo stesso calcolo nel tuo linguaggio di programmazione ma probabilmente avresti bisogno di più di una riga.

Funzioni aritmetiche

Quando si parla di funzioni si pensa sempre alle funzioni matematiche come valore assoluto, seno, coseno e radice quadrata. Tutte queste funzioni e molte altre sono disponibili in MySQL, vediamo come usarle:

```
mysql> SELECT ABS(-10);
+----------+
| ABS(-10) |
+----------+
| 10 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT SIN(0);
+--------+
| SIN(0) |
+--------+
| 0 |
+--------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COS(0);
+--------+
| COS(0) |
+--------+
| 1 |
+--------+
1 row in set (0.00 sec)
mysql> SELECT SQRT(16);
+----------+
| SQRT(16) |
+----------+
| 4 |
```

```
+----------+
1 row in set (0.00 sec)
```

Funzioni per caratteri

Per i caratteri ci sono davvero tante funzioni disponibili infatti possiamo concatenare stringhe, rendere i caratteri di una stringa tutti minuscoli o maiuscoli, eseguire un riempimento a sinistra o a destra fino a raggiungere una determinata lunghezza e tanto altro.

Per raggiungere gli obiettivi appena descritti useremo rispettivamente le funzioni CONCAT, LOWER, UPPER, LPAD, RPAD.

```
mysql> SELECT CONCAT('test', 'concat');
+------------------------+
| CONCAT('test', 'concat') |
+------------------------+
| testconcat |
+------------------------+
1 row in set (0.00 sec)


mysql> SELECT LOWER('TESTdiLOwer');
+---------------------+
| LOWER('TESTdiLOwer') |
+---------------------+
| testdilower |
+---------------------+
1 row in set (0.12 sec)


mysql> SELECT UPPER('TESTdiLOwer');
+---------------------+
| UPPER('TESTdiLOwer') |
+---------------------+
| TESTDILOWER |
+---------------------+
1 row in set (0.00 sec)
mysql> SELECT LPAD('TEST', 10, "-");
+---------------------+
```

```
| LPAD('TEST', 10, "-") |
+----------------------+
| ------TEST |
+----------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT RPAD('TEST', 10, "-");
+----------------------+
| RPAD('TEST', 10, "-") |
+----------------------+
| TEST------ |
+----------------------+
1 row in set (0.00 sec)
```

Un’altra funzione molto usata è TRIM che consente di eliminare gli spazi ad inizio e fine stringa. È possibile anche eliminare solo gli spazi ad inizio stringa usando LTRIM e solo a fine stringa usando RTRIM.

```
mysql> SELECT LTRIM(' |a|');
+-----------------+
| LTRIM(' |a|') |
+-----------------+
| |a| |
+-----------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT RTRIM(' |a| ');
+--------------------+
| RTRIM(' |a| ') |
+--------------------+
| |a| |
+--------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT TRIM(' |a| ');
    +-------------------+
    | TRIM(' |a| ') |
    +-------------------+
    | |a| |
    +-------------------+
    1 row in set (0.12 sec)
```

La funziona SUBSTR (abbreviazione di substring) accetta tre argomenti e consente di estrarre delle parti da una stringa di input. Il primo argomento indica la stringa da cui estrarre, il secondo argomento è la posizione del primo carattere da considerare mentre il terzo argomento è il numero di caratteri da mostrare.

```
    mysql> SELECT SUBSTR('PIPPO', 3);
    +-------------------+
    | SUBSTR('PIPPO', 3) |
    +-------------------+
    | PPO |
    +-------------------+
    1 row in set (0.00 sec)
```

```
mysql> SELECT SUBSTR('PIPPO', 3, 2);
    +----------------------+
    | SUBSTR('PIPPO', 3, 2) |
    +----------------------+
    | PP |
    +----------------------+
    1 row in set (0.00 sec)
```

Infine, ma non meno importante, troviamo la funzione LENGTH che restituisce il numero di caratteri contenuti in una stringa:

```
    mysql> SELECT LENGTH('PIPPO');
    +----------------+
```

```
| LENGTH('PIPPO') |
+-----------------+
| 5 |
+-----------------+
1 row in set (0.00 sec)
```

7

JOIN DELLE TABELLE

Una delle funzionalità più potenti di MySQL è la sua capacità di raccogliere e manipolare dati da più tabelle. Senza questa funzione dovresti archiviare tutti gli elementi di dati necessari per ogni applicazione in una tabella. Senza la possibilità di selezionare i dati da più tabelle sarebbe necessario archiviare gli stessi dati in più tabelle. Immagina di dover riprogettare, ricostruire e ripopolare tabelle e database ogni volta hai bisogno di una query con una nuova informazione. L'istruzione JOIN di MySQL consente di progettare tabelle più piccole, più specifiche e più facili da gestire rispetto alle tabelle più grandi.

Il modo più semplice per unire delle tabelle consiste nell'unirle in una FROM. In questo modo si effettua un prodotto cartesiano ovvero si avrà una tabella di NxM righe dove N sono le righe della prima tabella, M le righe della seconda. Il prodotto cartesiano può essere creato unendo le due tabelle nella clausola FROM, separandole dalla virgola o, in alternativa, con una CROSS JOIN.

```
mysql> SELECT * FROM BOLLETTE,CONTO_CORRENTE;
+---------+---------+------------------+------------------+--------+---------+
| NOME | IMPORTO | IBAN | IBAN | SALDO | BANCA |
+---------+---------+------------------+------------------+--------+---------+
| ENEL | 22.5 | IT000000001231231 | IT000000001231231 | 105.22 | BANCA 1 |
| FASTWEB | 35 | IT000000001231231 | IT000000001231231 | 105.22 | BANCA 1 |
| ENEL | 104.22 | IT000000003333334 | IT000000001231231 | 105.22 | BANCA 1 |
| FASTWEB | 110 | IT000000003333334 | IT000000001231231 | 105.22 | BANCA 1 |
+---------+---------+------------------+------------------+--------+---------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM BOLLETTE CROSS JOIN CONTO_CORRENTE;
```

+---------+---------+------------------+------------------+--------+---------+

| NOME | IMPORTO | IBAN | IBAN | SALDO | BANCA |

+---------+---------+------------------+------------------+--------+---------+

| ENEL | 22.5 | IT000000001231231 | IT000000001231231 | 105.22 | BANCA 1 |

| FASTWEB | 35 | IT000000001231231 | IT000000001231231 | 105.22 | BANCA 1 |

| ENEL | 104.22 | IT000000003333334 | IT000000001231231 | 105.22 | BANCA 1 |

| FASTWEB | 110 | IT000000003333334 | IT000000001231231 | 105.22 | BANCA 1 |

+---------+---------+------------------+------------------+--------+---------+

4 rows in set (0.00 sec)

Come vedi ogni riga della tabella BOLLETTE è legata ad ogni riga della tabella CONTO_CORRENTE. Ti ricordo che finora nella prima tabella ci sono solo 4 righe, nella seconda esiste solo 1 riga.

INNER JOIN

In MySQL una INNER JOIN seleziona tutte le righe da due tabelle per mostrare un risultato se e solo se entrambe le tabelle soddisfano le condizioni specificate nella clausola ON. La clausola ON specifica su quali campi si vuole restringere la selezione.

mysql> SELECT * FROM CONTO_CORRENTE CC INNER JOIN BOLLETTE B ON CC.IBAN = B.IBAN;

```
+------------------+--------+---------+---------+---------+------------------+
| IBAN | SALDO | BANCA | NOME | IMPORTO | IBAN |
+------------------+--------+---------+---------+---------+------------------+
| IT000000001231231 | 105.22 | BANCA 1 | ENEL | 22.5 | IT000000001231231 |
| IT000000001231231 | 105.22 | BANCA 1 | FASTWEB | 35 | IT000000001231231 |
+------------------+--------+---------+---------+---------+------------------+
2 rows in set (0.00 sec)
```

In questo caso abbiamo effettuato una INNER JOIN sul campo IBAN che è presente in entrambe le tabelle.

Inoltre, abbiamo dato un alias alle tabelle in modo da poter capire su quale stiamo operando con maggiore facilità, in particolare, la tabella CONTO_CORRENTE è stata ridenominata CC mentre BOLLETTE è stata ridenominata B.

LEFT JOIN

Se pensiamo due tabelle come due insiemi che contengono elementi (ovvero righe) possiamo unire i due insiemi considerando la parte in comune oltre a tutti gli elementi della tabella a sinistra.

```
mysql> SELECT * FROM CONTO_CORRENTE CC LEFT JOIN BOLLETTE B ON CC.IBAN = B.IBAN;
+------------------+--------+---------+---------+---------+------------------+
| IBAN | SALDO | BANCA | NOME | IMPORTO | IBAN |
+------------------+--------+---------+---------+---------+------------------+
| IT000000001231231 | 105.22 | BANCA 1 | ENEL | 22.5 | IT000000001231231 |
| IT000000001231231 | 105.22 | BANCA 1 | FASTWEB | 35 | IT000000001231231 |
+------------------+--------+---------+---------+---------+------------------+
2 rows in set (0.00 sec)
```

RIGHT JOIN

Proprio come nella LEFT JOIN, nella RIGHT JOIN il concetto è uguale ma cambia il verso infatti si recuperano tutti gli elementi in comune oltre a quelli della tabella di destra.

mysql> SELECT * FROM CONTO_CORRENTE CC RIGHT JOIN BOLLETTE B ON CC.IBAN = B.IBAN;

```
+------------------+--------+---------+---------+---------+------------------+
| IBAN | SALDO | BANCA | NOME | IMPORTO | IBAN |
+------------------+--------+---------+---------+---------+------------------+
| IT000000001231231 | 105.22 | BANCA 1 | ENEL | 22.5 | IT000000001231231 |
| IT000000001231231 | 105.22 | BANCA 1 | FASTWEB | 35 | IT000000001231231 |
| NULL | NULL | NULL | ENEL | 104.22 | IT000000003333334 |
| NULL | NULL | NULL | FASTWEB | 110 | IT000000003333334 |
+------------------+--------+---------+---------+---------+------------------+
4 rows in set (0.00 sec)
```

SELF JOIN

È anche possibile creare qualsiasi tipo di JOIN sulla stessa tabella, infatti, stiamo per creare una INNER JOIN della tabella BOLLETTE con sé stessa:

```
mysql> SELECT CONCAT(BB.NOME, ' - ', BB.IMPORTO) AS 'NOME - IMPORTO'
    -> FROM BOLLETTE B
    -> INNER JOIN BOLLETTE BB ON
    -> BB.NOME = B.NOME
    -> WHERE BB.NOME = 'ENEL';
    +----------------+
    | NOME - IMPORTO |
    +----------------+
    | ENEL - 22.5 |
    | ENEL - 22.5 |
    | ENEL - 104.22 |
    | ENEL - 104.22 |
    +----------------+
    4 rows in set (0.00 sec)
```

In questo caso abbiamo creato una INNER JOIN della tabella con sé stessa assegnando un alias (B e BB) per distinguerle. Puoi sostituire alla INNER JOIN una LEFT JOIN o una RIGHT JOIN per recuperare quello di cui hai bisogno dalla tua tabella. Forse ti stai chiedendo a cosa serve esattamente la SELF JOIN. Effettivamente in questo caso avremmo potuto selezionare le righe con una semplice condizione WHERE ma non tutte le tabelle sono così concise e soprattutto i dati contenuti, a volte, sono molto più complessi. Immagina di “ereditare” il database creato e popolato da qualcun altro e scopri che ogni tabella contiene almeno 15 colonne. In questo caso una SELF JOIN potrebbe davvero tornarti utile.

8

STORED ROUTINE

In questo capitolo vengono descritti gli oggetti archiviati in database, in termini di codice SQL archiviato sul server per un'esecuzione successiva. Gli oggetti archiviati includono diverse forme:

- Stored procedure: un oggetto creato con CREATE PROCEDURE e richiamato utilizzando l'istruzione CALL. Una procedura non ha un valore di ritorno ma può modificare i suoi parametri per un'ispezione successiva da parte del chiamante. Può anche generare set di risultati da restituire al programma client.
- Stored function: un oggetto creato con CREATE FUNCTION e utilizzato in modo molto simile a una funzione integrata. Lo invochi in un'espressione e restituisce un valore durante la valutazione dell'espressione.
- Trigger: un oggetto creato con CREATE TRIGGER associato a una tabella. Un trigger viene attivato quando si verifica un evento particolare per la tabella, ad esempio un inserimento o un aggiornamento.
- Evento: un oggetto creato con CREATE EVENT e richiamato dal server in base alla pianificazione.
- Vista: un oggetto creato con CREATE VIEW che, se referenziato, produce un set di risultati. Una vista funge da tabella virtuale.

Ciascun programma memorizzato contiene un corpo costituito da un'istruzione SQL. Questo corpo può essere un'istruzione composta da più statement separati da punti e virgola (;). Ad esempio, la seguente stored procedure ha un corpo costituito da un blocco BEGIN ... END che contiene un'istruzione SET e un ciclo REPEAT che contiene un'altra istruzione SET:

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
SET @x = 0;
REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
```

```
END;
```

Se si utilizza il programma client mysql per definire un programma memorizzato contenente caratteri punto e virgola, sorge un problema. Per impostazione predefinita, mysql stesso riconosce il punto e virgola come delimitatore di istruzione; quindi, è necessario ridefinire temporaneamente il delimitatore per fare in modo che mysql passi l'intera definizione del programma memorizzato al server. Per ridefinire il delimitatore mysql, bisogna utilizzare il comando delimiter. L'esempio seguente mostra come eseguire questa operazione per la procedura dorepeat() appena mostrata. Il delimitatore viene modificato in // per consentire che l'intera definizione venga passata al server come una singola istruzione e quindi ripristinato in ; prima di avviare la procedura. In questo modo si abilita il delimitatore ; utilizzato nel corpo della procedura per poter essere passato al server anziché essere interpretato da mysql stesso.

```
mysql> delimiter //
mysql> CREATE PROCEDURE dorepeat(p1 INT)
-> BEGIN
-> SET @x = 0;
-> REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
-> END
-> //
Query OK, 0 rows affected (0.00 sec)
mysql> delimiter ;
mysql> CALL dorepeat(1000);
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @x;
+------+
| @x |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

È possibile ridefinire il delimitatore in una stringa diversa da // e il delimitatore può essere costituito da un singolo carattere o più caratteri. Dovresti evitare l'uso del carattere backslash (\) perché questo è il carattere di escape per MySQL. Di seguito è riportato un esempio di una funzione che accetta un parametro, esegue un'operazione utilizzando una funzione SQL e restituisce il risultato. In questo caso, non è necessario utilizzare delimiter perché la definizione della funzione non contiene delimitatori con il carattere punto e virgola (;):

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
-> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world! |
+----------------+
1 row in set (0.00 sec)
```

MySQL supporta le stored routine (procedure e funzioni). Una stored routine è un insieme di istruzioni SQL che possono essere archiviate nel server. Una volta che questo è stato fatto, i client non hanno bisogno di continuare a riemettere le singole dichiarazioni, ma possono invece fare riferimento alla stored routine. Possono essere particolarmente utili in determinate situazioni:

- Quando più applicazioni client sono scritte in linguaggi diversi o funzionano su piattaforme diverse, ma devono eseguire le stesse operazioni sul database.
- Quando la sicurezza è fondamentale. Le banche, ad esempio, utilizzano procedure e funzioni per tutte le operazioni comuni. Ciò fornisce un ambiente coerente e sicuro e le routine possono garantire che ogni operazione sia registrata correttamente. In una tale configurazione, le applicazioni e gli utenti non hanno

accesso diretto alle tabelle del database, ma possono solo eseguire specifiche stored routine.

Le stored routine possono anche fornire prestazioni migliori poiché è necessario inviare meno informazioni tra il server e il client. Il compromesso è che ciò aumenta il carico sul server di database perché viene svolto più lavoro lato server e meno lato client (applicazione). Considera questo aspetto perché se molte macchine client (come i server Web) sono servite da un solo o da pochi server di database. Le stored routine consentono inoltre di avere librerie di funzioni nel server di database. Questa è una funzionalità condivisa dai moderni linguaggi applicativi che abilitano tale progettazione internamente (ad esempio, utilizzando le classi). L'utilizzo di queste funzionalità del linguaggio dell'applicazione client è vantaggioso per il programmatore anche al di fuori dell'ambito dell'utilizzo del database.

Una stored routine è una procedura o una funzione e vengono create, rispettivamente, con le istruzioni CREATE PROCEDURE e CREATE FUNCTION. Una procedura viene invocata utilizzando un'istruzione CALL e può restituire valori solo utilizzando variabili di output. Una funzione può essere chiamata dall'interno di un'istruzione proprio come qualsiasi altra funzione (ovvero invocando il nome della funzione) e può restituire un valore scalare. Il corpo di una routine può utilizzare istruzioni composte e possono essere eliminate con le istruzioni DROP PROCEDURE e DROP FUNCTION, modificate con le istruzioni ALTER PROCEDURE e ALTER FUNCTION. Una stored procedure o function è associata a un determinato database. Ciò ha diverse implicazioni:

- Quando la routine viene invocata, viene eseguito un implicito USE db_name (e annullato quando la routine termina). Le istruzioni USE all'interno delle stored routine non sono consentite.
- È possibile qualificare i nomi di routine con il nome del database. Questo può essere utilizzato per fare riferimento a una routine che non è nel database corrente. Ad esempio, per richiamare una procedura p o una funzione f associata al database di test, è possibile digitare CALL test.p() o test.f().

- Quando un database viene eliminato, vengono eliminate anche tutte le routine memorizzate ad esso associate.

Le stored function non possono essere ricorsive. La ricorsione nelle stored procedure è consentita ma disabilitata per impostazione predefinita. Per abilitare la ricorsione, bisogna impostare la variabile di sistema del server max_sp_recursion_depth su un valore maggiore di zero. La ricorsione nella stored procedure aumenta la richiesta di spazio nello stack dei thread e se si aumenta il valore di max_sp_recursion_depth, potrebbe essere necessario aumentare la dimensione dello stack di thread aumentando il valore di thread_stack all'avvio del server.

MySQL, inoltre, supporta un'estensione molto utile che consente l'uso di normali istruzioni SELECT (ovvero, senza utilizzare cursori o variabili locali) all'interno di una stored procedure. Il set di risultati di tale query viene semplicemente inviato direttamente al client. Più istruzioni SELECT generano più set di risultati, quindi il client deve utilizzare una libreria client MySQL che supporti più set di risultati. Ciò significa che il client deve utilizzare una libreria client da una versione di MySQL recente. Il client deve anche specificare l'opzione CLIENT_MULTI_RESULTS quando si connette. Per i programmi scritti in linguaggio C, questo può essere fatto con la funzione API C mysql_real_connect().

In MySQL 8.0.22 e versioni successive, una variabile utente a cui fa riferimento un'istruzione in una stored procedure ha il suo tipo determinato la prima volta che la procedura viene invocata e mantiene questo tipo ogni volta che la procedura viene richiamata in seguito. Il sistema di MySQL tiene conto delle stored routine come segue:

- Il privilegio CREATE ROUTINE è necessario per creare stored routine
- Il privilegio ALTER ROUTINE è necessario per modificare o eliminare le stored routine. Questo privilegio viene concesso automaticamente al creatore di una routine, se necessario, e viene eliminato dal creatore quando la routine viene eliminata.
- Il privilegio EXECUTE è necessario per eseguire le stored routine. Tuttavia, questo privilegio viene concesso automaticamente al creatore di una routine, se necessario (e viene eliminato dal creatore quando la routine viene eliminata).

Inoltre, la caratteristica di SQL SECURITY predefinita per una routine è DEFINER, che consente agli utenti che hanno accesso al database a cui è associata la routine di eseguire la routine.

- Se la variabile di sistema automatic_sp_privileges è 0, i privilegi EXECUTE e ALTER ROUTINE non vengono concessi e verranno eliminati automaticamente dal creatore della routine.
- Il creatore di una routine è l'account utilizzato per eseguire l'istruzione CREATE per essa. Potrebbe non corrispondere all'account denominato DEFINER nella definizione della routine.
- L'account denominato come DEFINER di routine può visualizzare tutte le proprietà di routine, inclusa la sua definizione.

L'account ha quindi accesso completo all'output di routine prodotto da:

- Il contenuto della tabella INFORMATION_SCHEMA.ROUTINE.
- Le istruzioni SHOW CREATE FUNCTION e SHOW CREATE PROCEDURE.
- Le istruzioni SHOW FUNCTION CODE e SHOW PROCEDURE CODE.
- Le istruzioni SHOW FUNCTION STATUS e SHOW PROCEDURE STATUS.

Per un account diverso dall'account denominato DEFINER di routine, l'accesso alle proprietà della routine dipende dai privilegi concessi all'account:

- Con il privilegio SHOW_ROUTINE o il privilegio SELECT globale, l'account può vedere tutte le proprietà della routine, inclusa la sua definizione.
- Con il privilegio CREATE ROUTINE, ALTER ROUTINE o EXECUTE concesso in un ambito che include la routine, l'account può visualizzare tutte le proprietà della routine tranne la sua definizione.

All'interno del corpo di una stored routine (procedura o funzione) o di un trigger, il valore di LAST_INSERT_ID() cambia allo stesso modo delle istruzioni eseguite al di fuori del corpo di questo tipo di oggetti. L'effetto di una stored routine o di un trigger sul valore di LAST_INSERT_ID() visualizzato dalle seguenti istruzioni dipende dal tipo di routine. Se una stored procedure esegue istruzioni che modificano il valore di LAST_INSERT_ID(), il valore modificato viene visualizzato da dichiarazioni che seguono la procedura call. Per le funzioni e i trigger che modificano il valore, il valore viene ripristinato al termine della funzione o del trigger; quindi, le istruzioni seguenti non vedono un valore modificato.

Trigger

Un trigger è un oggetto database con nome associato a una tabella e si attiva quando si verifica un evento particolare per la tabella. Vengono usati per eseguire controlli sui valori da inserire in una tabella o per eseguire calcoli sui valori coinvolti in un aggiornamento. Viene definito un trigger da attivare quando un'istruzione inserisce, aggiorna o elimina delle righe nella tabella associata. Queste operazioni di riga sono eventi trigger. Ad esempio, le righe possono essere inserite dalle istruzioni INSERT o LOAD DATA e un trigger di inserimento si attiva per ogni riga inserita. È possibile impostare un trigger per l'attivazione prima o dopo l'evento trigger, ad esempio, è possibile attivare un trigger prima di ogni riga inserita in una tabella o dopo ogni riga aggiornata.

Ricorda: i trigger MySQL si attivano solo per le modifiche apportate alle tabelle dalle istruzioni SQL. Ciò include le modifiche alle tabelle che sono alla base delle viste aggiornabili. I trigger non si attivano per le modifiche alle tabelle effettuate da API che non trasmettono istruzioni SQL al server MySQL. I trigger non vengono attivati dalle modifiche nelle tabelle INFORMATION_SCHEMA o performance_schema. Tali tabelle sono in realtà delle viste e i trigger non sono consentiti nelle viste.

Vediamo ora la sintassi per la creazione e l'eliminazione dei trigger, mostrando alcuni esempi di come utilizzarli e come ottenere i metadati dei trigger. Per creare un trigger o eliminare un trigger, utilizza l'istruzione CREATE TRIGGER o DROP TRIGGER. Ecco un semplice esempio che associa un trigger ad una tabella, da attivare per le operazioni di INSERT. Il trigger funge da accumulatore, sommando i valori inseriti in una delle colonne della tabella.

mysql> **CREATE TABLE** account **(**acct_num INT**,** amount DECIMAL**(**10**,**2**));**

Query OK**,** 0 **rows** affected **(**0.03 sec**)**

mysql> **CREATE TRIGGER** ins_sum **BEFORE INSERT ON** account
FOR EACH ROW SET @**sum** = @**sum** + **NEW.**amount**;**

Query OK**,** 0 **rows** affected **(**0.01 sec**)**

L'istruzione CREATE TRIGGER crea un trigger denominato ins_sum associato alla tabella account. Include anche delle clausole che specificano il tempo di azione del trigger, l'evento di trigger e cosa fare quando il trigger si attiva:

- La parola chiave BEFORE indica il tempo di azione del trigger. In questo caso, il trigger si attiva prima di ogni riga inserita nella tabella. L'altra parola chiave consentita in questo caso è AFTER.
- La parola chiave INSERT indica l'evento trigger; ovvero il tipo di operazione che attiva il trigger. Nell'esempio, le operazioni INSERT causano l'attivazione del trigger. Puoi anche creare trigger per le operazioni DELETE e UPDATE.
- L'istruzione seguente FOR EACH ROW definisce il corpo del trigger; ovvero l'istruzione da eseguire ogni volta che si attiva il trigger, che si verifica una volta per ogni riga interessata dall'evento di trigger. Nell'esempio, il corpo del trigger è un semplice SET che accumula in una variabile utente i valori inseriti nella colonna dell'importo. La dichiarazione si riferisce alla colonna come NEW.amount che significa "il valore della colonna dell'importo da inserire nella nuova riga".

Per utilizzare il trigger, imposta la variabile accumulatore su zero, esegui un'istruzione INSERT e quindi vedrai quale valore ha la variabile in seguito:

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+----------------------+
| Total amount inserted |
+----------------------+
| 1852.48 |
+----------------------+
```

In questo caso, il valore di @sum dopo l'esecuzione dell'istruzione INSERT è 14,98 + 1937,50 - 100 ovvero 1852,48.

Per distruggere il trigger, utilizza un'istruzione DROP TRIGGER, è necessario specificare il nome dello schema se il trigger non è nello schema predefinito:

mysql> **DROP TRIGGER** test**.**ins_sum**;**

Se elimini una tabella, vengono eliminati anche tutti i trigger per la tabella. I nomi dei trigger esistono nel namespace dello schema, il che significa che tutti i trigger devono avere nomi univoci all'interno di uno schema. I trigger in schemi diversi possono avere lo stesso nome. È possibile definire più trigger per una determinata tabella che hanno lo stesso evento di trigger e lo stesso tempo di azione. Ad esempio, puoi avere due trigger BEFORE UPDATE per una tabella. Per impostazione predefinita, i trigger che hanno lo stesso evento trigger e lo stesso tempo di azione si attivano nell'ordine in cui sono stati creati. Per influenzare l'ordine del trigger, specifica una clausola dopo FOR EACH ROW che indichi FOLLOWS o PRECEDES e il nome di un trigger esistente che ha anche lo stesso evento trigger e lo stesso tempo di azione. Con FOLLOWS, il nuovo trigger si attiva dopo il trigger esistente, con PRECEDES, il nuovo trigger si attiva prima del trigger esistente. Ad esempio, la seguente definizione di trigger definisce un altro trigger BEFORE INSERT per la tabella account:

mysql> **CREATE TRIGGER** ins_transaction **BEFORE INSERT ON** account
FOR EACH ROW PRECEDES ins_sum
SET
@deposits = @deposits + **IF(NEW.**amount>0**,NEW.**amount**,**0**),**
@withdrawals = @withdrawals + **IF(NEW.**amount<0**,-NEW.**amount**,**0**);**
Query OK**,** 0 **rows** affected **(**0.01 sec**)**

Questo trigger, ins_transaction, è simile a ins_sum ma accumula depositi e prelievi separatamente. Ha una clausola PRECEDES che lo fa attivare

prima di ins_sum; senza quella clausola si attiverebbe dopo ins_sum perché viene creato dopo ins_sum.

All'interno del corpo del trigger, le parole chiave OLD e NEW ti consentono di accedere alle colonne nelle righe interessate da un trigger. OLD e NEW sono estensioni MySQL ai trigger e non fanno distinzione tra maiuscole e minuscole. In un trigger INSERT, è possibile utilizzare solo NEW.col_name; in un trigger DELETE, è possibile utilizzare solo OLD.col_name; non c'è una nuova riga. In un trigger UPDATE, puoi utilizzare OLD.col_name per fare riferimento alle colonne di una riga prima che venga aggiornata e NEW.col_name per fare riferimento alle colonne della riga dopo che è stata aggiornata. Una colonna denominata con OLD è di sola lettura. Puoi fare riferimento ad essa (se hai il privilegio SELECT), ma non modificarla. Puoi fare riferimento a una colonna denominata con NEW se hai il privilegio SELECT per essa.

In un trigger BEFORE, puoi anche cambiarne il valore con SET NEW.col_name = value se hai il privilegio UPDATE, ciò significa che puoi utilizzare un trigger per modificare i valori da inserire in una nuova riga o da utilizzare per aggiornare una riga. Un'istruzione SET di questo tipo non ha alcun effetto in un trigger AFTER perché la modifica della riga è già avvenuta.

In un trigger BEFORE, il valore NEW per una colonna AUTO_INCREMENT è pari a 0 e non è pari al numero di sequenza che viene generato automaticamente quando la nuova riga è effettivamente inserito. Utilizzando il costrutto BEGIN ... END, è possibile definire un trigger che esegua più istruzioni. All'interno del blocco BEGIN, puoi anche usare altra sintassi valida come istruzioni condizionali e loop. Tuttavia, proprio come per le stored routine, se si utilizza il programma mysql per definire un trigger che esegua più istruzioni, è necessario ridefinire il delimitatore dell'istruzione mysql in modo da poter utilizzare il delimitatore dell'istruzione ; all'interno della definizione del trigger. L'esempio seguente illustra questi punti. Definisce un trigger UPDATE che controlla il nuovo valore da utilizzare per aggiornare ogni riga e modifica il valore in modo che rientri nell'intervallo da 0 a 100. Questo deve essere un trigger BEFORE perché il valore deve essere verificato prima di essere utilizzato per l'aggiornamento la riga:

```
mysql> delimiter //
```

```
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
    FOR EACH ROW
    BEGIN
    IF NEW.amount < 0 THEN
    SET NEW.amount = 0;
    ELSEIF NEW.amount > 100 THEN
    SET NEW.amount = 100;
    END IF;
    END;//
    mysql> delimiter ;
```

Può essere più semplice definire una stored procedure separatamente e quindi richiamarla dal trigger utilizzando una semplice istruzione CALL. Questo è anche vantaggioso se si desidera eseguire lo stesso codice da più trigger. Tuttavia, esistono limitazioni su ciò che può apparire nelle istruzioni che un trigger esegue quando attivato:

- Il trigger non può utilizzare l'istruzione CALL per richiamare stored procedure che restituiscono dati al client o che utilizzano SQL dinamico. Le stored procedure possono restituire dati al trigger tramite parametri OUT o INOUT.
- Il trigger non può utilizzare istruzioni che iniziano o terminano esplicitamente o implicitamente una transazione, come START TRANSACTION, COMMIT o ROLLBACK. È consentito ROLLBACK su SAVEPOINT perché non termina una transazione.

MySQL gestisce gli errori durante l'esecuzione del trigger come segue:

- Se un trigger BEFORE fallisce, l'operazione sulla riga corrispondente non viene eseguita.
- Un trigger BEFORE viene attivato dal tentativo di inserire o modificare la riga, indipendentemente dal fatto che il tentativo successivamente abbia esito positivo.

- Un trigger AFTER viene eseguito solo se eventuali trigger BEFORE e l'operazione sulla riga vengono eseguiti correttamente.
- Un errore durante un trigger BEFORE o AFTER provoca un errore dell'intera istruzione che ha causato l'invocazione del trigger.
- Per le tabelle transazionali, l'errore di un'istruzione dovrebbe causare il rollback di tutte le modifiche eseguite dall'istruzione. L'errore di un trigger provoca il fallimento dell'istruzione, quindi anche l'errore del trigger provoca il rollback. Per le tabelle non transazionali, tale rollback non può essere eseguito, quindi anche se l'istruzione non riesce, tutte le modifiche eseguite prima del punto dell'errore rimangono valide.

I trigger possono contenere riferimenti diretti a tabelle per nome, come il trigger denominato testref mostrato in questo esempio:

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
b4 INT DEFAULT 0
);
delimiter |
CREATE TRIGGER testref BEFORE INSERT ON test1
FOR EACH ROW
BEGIN
INSERT INTO test2 SET a2 = NEW.a1;
DELETE FROM test3 WHERE a3 = NEW.a1;
UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
END;
|
delimiter ;
INSERT INTO test3 (a3) VALUES
(NULL), (NULL), (NULL), (NULL), (NULL),
(NULL), (NULL), (NULL), (NULL), (NULL);
```

```
INSERT INTO test4 (a4) VALUES
(0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
```

Si supponga di inserire i seguenti valori nella tabella test1 come mostrato qui:

```
mysql> INSERT INTO test1 VALUES
(1), (3), (1), (7), (1), (8), (4), (4);
Query OK, 8 rows affected (0.01 sec)
Records: 8 Duplicates: 0 Warnings: 0
```

Di conseguenza, le quattro tabelle contengono i seguenti dati:

```
mysql> SELECT * FROM test1;
+------+
| a1 |
+------+
| 1 |
| 3 |
| 1 |
| 7 |
| 1 |
| 8 |
| 4 |
| 4 |
+------+
8 rows in set (0.00 sec)
mysql> SELECT * FROM test2;
+------+
| a2 |
+------+
| 1 |
| 3 |
| 1 |
| 7 |
| 1 |
```

| 8 |
| 4 |
| 4 |
+------+
8 **rows in set (**0.00 sec**)**
mysql> **SELECT * FROM** test3**;**
+----+
| a3 |
+----+
| 2 |
| 5 |
| 6 |
| 9 |
| 10 |
+----+
5 **rows in set (**0.00 sec**)**
mysql> **SELECT * FROM** test4**;**
+----+------+
| a4 | b4 |
+----+------+
1	3
2	0
3	1
4	2
5	0
6	0
7	1
8	1
9	0
10	0
+----+------+
10 **rows in set (**0.00 sec**)**

Eventi schedulati

MySQL Event Scheduler gestisce la pianificazione e l'esecuzione di eventi, ovvero le attività che vengono eseguite in base a una pianificazione. Le stored routine richiedono la tabella del dizionario dei dati degli eventi nel database di sistema mysql. Questa tabella viene creata durante la procedura di installazione di MySQL 8.0. Se stai eseguendo l'aggiornamento a MySQL 8.0 da una versione precedente, assicurati di eseguire la procedura di aggiornamento per poter usare tali funzionalità.

Gli eventi MySQL sono attività che vengono eseguite in base a una pianificazione, pertanto, a volte li chiamiamo eventi programmati. Quando si crea un evento, si crea un oggetto del database con nome contenente una o più istruzioni SQL da eseguire a uno o più intervalli regolari, con inizio e fine in una data e ora specifica. Concettualmente, questo è simile all'idea del crontab Unix (noto anche come "cron job") o dell'Utilità di pianificazione di Windows. Le attività pianificate di questo tipo sono talvolta note anche come "trigger temporali", il che implica che si tratta di oggetti attivati dal passare del tempo. Sebbene ciò sia sostanzialmente corretto, preferiamo usare il termine eventi per evitare confusione con i trigger discussi nella sezione precedente.

Gli eventi, in modo più specifico, non dovrebbero essere confusi con i "trigger temporanei". Mentre un trigger è un oggetto di database le cui istruzioni vengono eseguite in risposta a un tipo specifico di evento che si verifica su una determinata tabella, un evento (programmato) è un oggetto le cui istruzioni vengono eseguite in risposta al trascorrere di un intervallo di tempo specificato. Sebbene non vi sia alcuna disposizione nello standard SQL per la pianificazione degli eventi, esistono precedenti in altri sistemi di database e potresti notare alcune somiglianze tra queste implementazioni e quella che si trova in MySQL Server.

Gli eventi MySQL hanno le seguenti caratteristiche e proprietà principali:

- In MySQL, un evento è identificato in modo univoco dal suo nome e dallo schema a cui è assegnato.

- Un evento esegue un'azione specifica in base a una pianificazione. Questa azione consiste in un'istruzione SQL, che può essere un'istruzione composta in un blocco BEGIN ... END se lo si desidera. La tempistica di un evento può essere una tantum o ricorrente. Un evento occasionale viene eseguito una sola volta. Un evento ricorrente ripete la sua azione a intervalli regolari e alla pianificazione di un evento ricorrente possono essere assegnati un giorno e un'ora di inizio specifici, un giorno e un'ora di fine, entrambi o nessuno dei due (per impostazione predefinita, la pianificazione di un evento ricorrente inizia non appena viene creato e continua indefinitamente, finché non viene disabilitata o eliminata). Se un evento ricorrente non dovesse terminare entro il suo intervallo di pianificazione, il risultato potrebbe essere l'esecuzione di più istanze dell'evento contemporaneamente. Qualora tale comportamento non fosse utile, è necessario istituire un meccanismo per prevenire istanze simultanee. Ad esempio, puoi utilizzare la funzione GET_LOCK() o il blocco di righe o tabelle. • Gli utenti possono creare, modificare ed eliminare eventi pianificati utilizzando istruzioni SQL destinate a tali scopi. Le istruzioni di creazione e modifica di eventi sintatticamente non valide restituiscono un messaggio di errore appropriato. Un utente può includere istruzioni nell'azione di un evento che richiedono privilegi che l'utente in realtà non ha. L'istruzione di creazione o modifica dell'evento ha esito positivo ma l'azione dell'evento ha esito negativo.
- Molte delle proprietà di un evento possono essere impostate o modificate utilizzando le istruzioni SQL. Queste proprietà includono il nome dell'evento, l'orario, la persistenza (cioè se viene conservato dopo la scadenza della pianificazione), lo stato (abilitato o disabilitato), l'azione da eseguire e lo schema a cui è assegnato. Il “creatore” predefinito (definer) di un evento è l'utente che ha creato l'evento, a meno che l'evento non sia stato modificato, nel qual caso il definer è l'utente che ha emesso l'ultima istruzione ALTER EVENT che interessa quell'evento. Un evento può essere modificato da qualsiasi utente che

disponga del privilegio EVENT sul database per il quale è definito l'evento.

Gli eventi vengono eseguiti da uno speciale thread di pianificazione degli eventi; quando ci riferiamo all'Event Scheduler, in realtà ci riferiamo a questo thread. Durante l'esecuzione, il thread dell'utilità di pianificazione eventi e il suo stato corrente possono essere visualizzati dagli utenti che dispongono del privilegio PROCESS nell'output di SHOW PROCESSLIST. La variabile di sistema globale event_scheduler determina se l'Utilità di pianificazione eventi è abilitata e in esecuzione sul server. Ha uno dei seguenti valori, che influiscono sulla pianificazione degli eventi come descritto: ON, OFF, DISABLED.

Con il valore ON, viene avviata l'Utilità di pianificazione eventi; il thread dell'utilità di pianificazione degli eventi è attivo ed esegue tutti gli eventi pianificati. ON è il valore di predefinito per event_scheduler e quando l'utilità di pianificazione degli eventi è attiva, il thread dell'utilità di pianificazione degli eventi è mostrato nell'output di SHOW PROCESSLIST come un processo daemon e il suo stato è rappresentato come segue:

mysql> SHOW PROCESSLIST\G
*************************** 1. **row** ***************************
Id**:** 1
User: root
Host: localhost
db**: NULL**
Command**:** Query
Time**:** 0
State: NULL
Info**:** show processlist
*************************** 2. **row** ***************************
Id**:** 2
User: event_scheduler
Host: localhost
db**: NULL**
Command**:** Daemon
Time**:** 3

State: Waiting **for next** activation
Info**: NULL**
2 **rows in set (**0.00 sec**)**

La pianificazione degli eventi può essere interrotta impostando il valore di event_scheduler su OFF. In tal caso l'Utilità di pianificazione eventi viene arrestata. Il thread dell'utilità di pianificazione degli eventi non viene eseguito, non viene visualizzato nell'output di SHOW PROCESSLIST e non viene eseguito alcun evento pianificato. Quando l'Event Scheduler viene arrestato (quindi event_scheduler è OFF), può essere avviato impostando il valore di event_scheduler su ON.

Il valore DISABLED rende l'Utilità di pianificazione eventi non operativa. Quando l'utilità di pianificazione eventi è DISABLED, il thread dell'utilità di pianificazione eventi non viene eseguito (e quindi non viene visualizzato nell'output di SHOW PROCESSLIST). Inoltre, lo stato dell'Utilità di pianificazione eventi non può essere modificato in fase di esecuzione. Se lo stato dell'Event Scheduler non è stato impostato su DISABLED, event_scheduler può essere commutato tra ON e OFF (usando SET). È anche possibile utilizzare 0 per OFF e 1 per ON quando si imposta questa variabile. Pertanto, una qualsiasi delle seguenti 4 istruzioni può essere utilizzata nel client mysql per attivare l'utilità di pianificazione degli eventi:

SET GLOBAL event_scheduler = **ON;**
SET @@**GLOBAL.**event_scheduler = **ON;**
SET GLOBAL event_scheduler = 1**;**
SET @@**GLOBAL.**event_scheduler = 1**;**

In modo simile, è possibile disattivarla con una delle seguenti 4 istruzioni:

SET GLOBAL event_scheduler = **OFF;**
SET @@**GLOBAL.**event_scheduler = **OFF;**
SET GLOBAL event_scheduler = 0**;**
SET @@**GLOBAL.**event_scheduler = 0**;**

Se l’Event Scheduler è abilitato, l'abilitazione della variabile di sistema super_read_only impedisce l'aggiornamento dei timestamp dell'evento (last executed), cioè "ultima esecuzione", nella tabella del dizionario dei dati degli eventi. Ciò provoca l'arresto dell'Utilità di pianificazione eventi la volta successiva che tenterà di eseguire un evento pianificato, con un messaggio nel registro degli errori del server.

In questa situazione la variabile di sistema event_scheduler non cambia da ON a OFF. Un'implicazione è che questa variabile riflette l'intento dei DBA con cui l'Event Scheduler sia abilitato o disabilitato, dove il suo stato effettivo di avvio o arresto potrebbe essere distinto. Se super_read_only viene successivamente disabilitato dopo essere stato abilitato, il server riavvia automaticamente Event Scheduler secondo necessità, a partire da MySQL 8.0.26. Prima di MySQL 8.0.26, è necessario riavviare manualmente Event Scheduler, abilitandolo nuovamente.

Sebbene ON e OFF abbiano equivalenti numerici, il valore visualizzato per event_scheduler da SELECT o SHOW VARIABLES è sempre uno tra OFF, ON o DISABLED (che non ha un equivalente numerico). Per questo motivo, quando si imposta questa variabile, di solito si preferisce ON e OFF su 1 e 0. Nota che il tentativo di impostare event_scheduler senza specificarlo come variabile globale causa un errore:

mysql< **SET** @@event_scheduler = **OFF;**
ERROR 1229 **(**HY000**): Variable** 'event_scheduler' **is** a **GLOBAL variable and** should be **set with SET GLOBAL**

È possibile impostare l'Utilità di pianificazione eventi su DISABLED solo all'avvio del server. Se event_scheduler è ON o OFF, non è possibile impostarlo su DISABLED in fase di esecuzione. Inoltre, se Event Scheduler è impostato su DISABLED all'avvio, non è possibile modificare il valore di event_scheduler in fase di esecuzione. Per disabilitare l'utilità di pianificazione degli eventi, utilizzare uno dei due metodi seguenti:

1. Come opzione della riga di comando all'avvio del server:

--event-scheduler=DISABLED

1. Nel file di configurazione del server (my.cnf o my.ini su sistemi Windows), includendo la riga in cui può essere letto dal server (ad esempio, in una sezione [mysqld]):

event_scheduler=DISABLED

Per abilitare l'Utilità di pianificazione eventi, riavvia il server senza l'opzione della riga di comando --event-scheduler=DISABLED o dopo aver rimosso o commentato la riga contenente eventscheduler=DISABLED nel file di configurazione del server, a seconda dei casi. In alternativa, è possibile utilizzare ON (valore 1) o OFF (valore 0) al posto del valore DISABLED all'avvio del server. Nota È possibile emettere istruzioni di manipolazione degli eventi quando event_scheduler è impostato su DISABLED. In questi casi non vengono generati avvisi o errori (a condizione che le affermazioni stesse siano valide). Tuttavia, gli eventi pianificati non possono essere eseguiti finché questa variabile non è impostata su ON (o 1). Una volta eseguita questa operazione, il thread dell'utilità di pianificazione degli eventi esegue tutti gli eventi le cui condizioni di pianificazione sono soddisfatte.

L'avvio del server MySQL con l'opzione --skip-grant-tables fa sì che event_scheduler venga impostato su DISABLED, sovrascrivendo qualsiasi altro valore impostato sulla riga di comando o nel file my.cnf o my.ini.

Per completare, parliamo della sintassi degli eventi. MySQL fornisce diverse istruzioni SQL per lavorare con gli eventi pianificati:

- I nuovi eventi vengono definiti utilizzando l'istruzione CREATE EVENT.
- La definizione di un evento esistente può essere modificata tramite l'istruzione ALTER EVENT.
- Quando un evento pianificato non è più utile o necessario, può essere eliminato dal server dal suo definer utilizzando l'istruzione DROP EVENT. Il fatto che un evento persista oltre la fine della sua pianificazione dipende anche dalla sua clausola ON COMPLETION, se ne ha una. Un evento può essere

eliminato da qualsiasi utente che dispone del privilegio EVENT per il database in cui si trova l'evento definito.

Per abilitare o disabilitare l'esecuzione di eventi programmati, è necessario impostare il valore della variabile di sistema globale event_scheduler. Ciò richiede privilegi sufficienti per impostare le variabili di sistema globali. Il privilegio EVENT regola la creazione, la modifica e l'eliminazione degli eventi, questo privilegio può essere conferito utilizzando GRANT. Ad esempio, questa istruzione GRANT conferisce il privilegio EVENT per lo schema denominato myschema all'utente jon@ghidora:

GRANT EVENT **ON** myschema**.* TO** jon@ghidora**;**

Presumiamo che questo account utente esista già e desideriamo che rimanga invariato altrimenti. Per concedere allo stesso utente il privilegio EVENT su tutti gli schemi, utilizza la seguente istruzione:

GRANT EVENT **ON *.* TO** jon@ghidora**;**

Il privilegio EVENT ha un ambito globale o relativo al livello di schema. Pertanto, il tentativo di concederlo su una singola tabella genera un errore come mostrato:

mysql> **GRANT** EVENT **ON** myschema**.**mytable **TO** jon@ghidora**;**
ERROR 1144 **(**42000**):** Illegal **GRANT/REVOKE** command**;** please consult the manual **to** see which **privileges** can be used

È importante comprendere che un evento viene eseguito con i privilegi del suo definer e che non può eseguire azioni per le quali il suo definer non dispone dei privilegi richiesti.

Ad esempio, supponiamo che jon@ghidora abbia il privilegio EVENT per myschema. Si supponga inoltre che questo utente abbia il privilegio SELECT per myschema, ma nessun altro privilegio per questo schema. È possibile per jon@ghidora creare un nuovo evento come questo:

CREATE EVENT e_store_ts

ON SCHEDULE
EVERY 10 **SECOND**
DO
INSERT INTO myschema**.**mytable **VALUES (**UNIX_TIMESTAMP**());**

L'utente attende circa un minuto, quindi esegue la query SELECT * FROM mytable; aspettandosi di vedere diverse nuove righe nella tabella. Invece, la tabella è vuota.

Poiché l'utente non dispone del privilegio INSERT per la tabella in questione, l'evento non ha effetto. Se ispezioni il log degli errori MySQL (hostname.err), puoi vedere che l'evento è in esecuzione, ma l'azione che sta tentando di eseguire non va a buon fine:

2022-09-24T12**:**41**:**31.261992Z 25 **[**ERROR**]** Event Scheduler**:**
[jon@ghidora**][**cookbook**.**e_store_ts**] INSERT** command denied **to user**
'jon'@'ghidora' **for table** 'mytable'
2022-09-24T12**:**41**:**31.262022Z 25 **[**Note**]** Event Scheduler**:**
[jon@ghidora**].[**myschema**.**e_store_ts**]** event execution failed**.**
2022-09-24T12**:**41**:**41.271796Z 26 **[**ERROR**]** Event Scheduler**:**
[jon@ghidora**][**cookbook**.**e_store_ts**] INSERT** command denied **to user**
'jon'@'ghidora' **for table** 'mytable'
2022-09-24T12**:**41**:**41.272761Z 26 **[**Note**]** Event Scheduler**:**
[jon@ghidora**].[**myschema**.**e_store_ts**]** event execution failed**.**

Poiché molto probabilmente questo utente non ha accesso al log degli errori, è possibile verificare se l'istruzione di azione dell'evento è valida eseguendola direttamente:

mysql> **INSERT INTO** myschema**.**mytable **VALUES (**UNIX_TIMESTAMP**());**
ERROR 1142 **(**42000**): INSERT** command denied **to user**
'jon'@'ghidora' **for table** 'mytable'

L'ispezione della tabella INFORMATION_SCHEMA.EVENTS mostra che e_store_ts esiste ed è abilitato, ma la sua colonna LAST_EXECUTED è NULL:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
> WHERE EVENT_NAME='e_store_ts'
> AND EVENT_SCHEMA='myschema'\G
*************************** 1. row ***************************
EVENT_CATALOG: NULL
EVENT_SCHEMA: myschema
EVENT_NAME: e_store_ts
DEFINER: jon@ghidora
EVENT_BODY: SQL
EVENT_DEFINITION: INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP())
EVENT_TYPE: RECURRING
EXECUTE_AT: NULL
INTERVAL_VALUE: 5
INTERVAL_FIELD: SECOND
SQL_MODE: NULL
STARTS: 0000-00-00 00:00:00
ENDS: 0000-00-00 00:00:00
STATUS: ENABLED
ON_COMPLETION: NOT PRESERVE
CREATED: 2022-02-09 22:36:06
LAST_ALTERED: 2022-02-09 22:36:06
LAST_EXECUTED: NULL
EVENT_COMMENT:
1 row in set (0.00 sec)
```

Per annullare il privilegio EVENT, utilizza l'istruzione REVOKE. In questo esempio, il privilegio EVENT sullo schema myschema viene rimosso dall'account utente jon@ghidora:

```
REVOKE EVENT ON myschema.* FROM jon@ghidora;
```

La revoca del privilegio EVENT a un utente non elimina o disabilita gli eventi che potrebbero essere stati creati da quell'utente. Un evento non viene migrato o eliminato a seguito della ridenominazione o dell'eliminazione dell'utente che lo ha creato. Si supponga che all'utente jon@ghidora siano stati concessi i privilegi EVENT e INSERT sullo schema myschema. Questo utente crea quindi il seguente evento:

```
CREATE EVENT e_insert
ON SCHEDULE
EVERY 7 SECOND
DO
INSERT INTO myschema.mytable;
```

Dopo che questo evento è stato creato, root revoca il privilegio EVENT per jon@ghidora. Tuttavia, e_insert continua a essere eseguito, inserendo una nuova riga in mytable ogni sette secondi. Lo stesso sarebbe vero se root avesse rilasciato una di queste affermazioni:

- **DROP USER** jon@ghidora**;**
- **RENAME USER** jon@ghidora **TO** someotherguy@ghidora**;**

È possibile verificare che ciò sia vero esaminando la tabella INFORMATION_SCHEMA.EVENTS prima e dopo l'emissione di un'istruzione DROP USER o RENAME USER. Le definizioni degli eventi sono archiviate nel dizionario dei dati. Per eliminare un evento creato da un altro account utente, devi essere l'utente root MySQL o un altro utente con i privilegi necessari. I privilegi EVENT degli utenti sono archiviati nelle colonne Event_priv delle tabelle mysql.user e mysql.db. In entrambi i casi, questa colonna contiene uno dei valori 'Y' o 'N' dove 'N' è l'impostazione predefinita. La proprietà mysql.user.Event_priv è impostata su 'Y' per un determinato utente solo se tale utente dispone del privilegio globale EVENT (ovvero, se il privilegio è stato concesso utilizzando GRANT EVENT ON *.*). Per un privilegio EVENT a livello di schema, GRANT crea una riga in mysql.db e imposta la colonna Db di quella riga sul nome

dello schema, la colonna User sul nome dell'utente e la colonna Event_priv su 'Y'. Non dovrebbe mai essere necessario manipolare direttamente queste tabelle, poiché le istruzioni GRANT EVENT e REVOKE EVENT eseguono le operazioni richieste su di esse. Esistono cinque variabili di stato che forniscono il conteggio delle operazioni relative agli eventi (ma non delle istruzioni eseguite dagli eventi):

- Com_create_event: il numero di istruzioni CREATE EVENT eseguite dall'ultimo riavvio del server.
- Com_alter_event: il numero di istruzioni ALTER EVENT eseguite dall'ultimo riavvio del server.
- Com_drop_event: il numero di istruzioni DROP EVENT eseguite dall'ultimo riavvio del server.
- Com_show_create_event: il numero di istruzioni SHOW CREATE EVENT eseguite dall'ultimo riavvio del server.
- Com_show_events: il numero di istruzioni SHOW EVENTS eseguite dall'ultimo riavvio del server.

È possibile visualizzare i valori correnti per tutti questi valori contemporaneamente eseguendo l'istruzione SHOW STATUS LIKE '%event%';.

INFORMATION_SCHEMA

INFORMATION_SCHEMA fornisce l'accesso ai metadati del database, alle informazioni sul server MySQL (come il nome di un database o una tabella), il tipo di dati di una colonna o i privilegi di accesso. Altri termini che vengono talvolta utilizzati per queste informazioni sono il dizionario dei dati (data dictionary) e il catalogo di sistema (system catalog). INFORMATION_SCHEMA è un database all'interno di ciascuna istanza MySQL, il luogo in cui vengono archiviate le informazioni su tutti gli altri database gestiti dal server MySQL. Il database INFORMATION_SCHEMA contiene diverse tabelle di sola lettura, che in realtà sono viste, non tabelle di base, quindi non ci sono file ad esse associati e non puoi impostare trigger su di esse. Inoltre, non esiste una directory di database con quel nome. Sebbene sia possibile selezionare INFORMATION_SCHEMA come database predefinito con un'istruzione USE, è possibile solo leggere il contenuto delle tabelle, non eseguire operazioni INSERT, UPDATE o DELETE su di esse. Ecco un esempio di un'istruzione che recupera informazioni da INFORMATION_SCHEMA:

```
mysql> SELECT table_name, table_type, engine
FROM information_schema.tables
WHERE table_schema = 'db5'
ORDER BY table_name;
+------------+------------+--------+
| table_name | table_type | engine |
+------------+------------+--------+
| fk | BASE TABLE | InnoDB |
| fk2 | BASE TABLE | InnoDB |
| goto | BASE TABLE | MyISAM |
| into | BASE TABLE | MyISAM |
| k | BASE TABLE | MyISAM |
| kurs | BASE TABLE | MyISAM |
| loop | BASE TABLE | MyISAM |
| pk | BASE TABLE | InnoDB |
| t | BASE TABLE | MyISAM |
| t2 | BASE TABLE | MyISAM |
| t3 | BASE TABLE | MyISAM |
```

t7	BASE **TABLE**	MyISAM
tables	BASE **TABLE**	MyISAM
v	**VIEW**	**NULL**
v2	**VIEW**	**NULL**
v3	**VIEW**	**NULL**
v56	**VIEW**	**NULL**
+------------+------------+--------+
17 **rows in set (**0.01 sec**)**

Spiegazione: l'istruzione richiede un elenco di tutte le tabelle nel database db5, mostrando solo tre informazioni: il nome della tabella, il suo tipo e il suo motore di archiviazione. A partire da MySQL 8.0.30, le informazioni sulle chiavi primarie invisibili generate sono visibili per impostazione predefinita in tutte le tabelle INFORMATION_SCHEMA che descrivono le colonne delle tabelle, le chiavi o entrambe, come le tabelle COLUMNS e STATISTICS. Se desideri nascondere tali informazioni alle query che selezionano i dati da queste tabelle, puoi farlo impostando il valore della variabile di sistema del server show_gipk_in_create_table_and_information_schema su OFF.

La definizione per le colonne di caratteri (ad esempio, TABLES.TABLE_NAME) è generalmente VARCHAR(N) CHARACTER SET utf8mb3 dove N è almeno pari a 64. MySQL utilizza le regole di confronto predefinite per questo set di caratteri (utf8mb3_general_ci) per tutte le ricerche, gli ordinamenti, i confronti e altre operazioni sulle stringhe su tali colonne. Poiché alcuni oggetti MySQL sono rappresentati come file, le ricerche nelle colonne di tipo stringa in INFORMATION_SCHEMA possono essere influenzate dalla distinzione tra maiuscole e minuscole del file system.

L'istruzione SELECT ... FROM INFORMATION_SCHEMA è intesa come un modo più coerente per fornire l'accesso alle informazioni fornite dalle varie istruzioni SHOW supportate da MySQL (SHOW DATABASES, SHOW TABLES e così via). L'utilizzo di SELECT ha questi vantaggi rispetto a SHOW:

- È conforme alle regole di Codd, perché tutti gli accessi avvengono sui tavoli.
- È possibile utilizzare la sintassi familiare dell'istruzione SELECT e occorre solo imparare alcuni nomi di tabelle e colonne.
- L'implementatore non deve preoccuparsi di aggiungere parole chiave.
- È possibile filtrare, ordinare, concatenare e trasformare i risultati delle query INFORMATION_SCHEMA in qualsiasi formato richiesto dall'applicazione, ad esempio una struttura dati o una rappresentazione di testo da analizzare.
- Questa tecnica è interoperabile con altri sistemi di database. Ad esempio, gli utenti di database Oracle hanno familiarità con le tabelle di query nel dizionario di dati Oracle.

Per la maggior parte delle tabelle INFORMATION_SCHEMA, ogni utente MySQL ha il diritto di accedervi, ma può vedere solo le righe nelle tabelle che corrispondono agli oggetti per i quali l'utente ha i privilegi di accesso appropriati. In alcuni casi (ad esempio, la colonna ROUTINE_DEFINITION nella tabella INFORMATION_SCHEMA ROUTINE), gli utenti che dispongono di privilegi insufficienti vedranno NULL. Alcune tabelle hanno requisiti di privilegio diversi; per questi i requisiti sono riportati nelle descrizioni delle tabelle applicabili. Ad esempio, le tabelle InnoDB (tabelle con nomi che iniziano con INNODB_) richiedono il privilegio PROCESS. Gli stessi privilegi si applicano alla selezione delle informazioni da INFORMATION_SCHEMA e alla visualizzazione delle stesse informazioni tramite le istruzioni SHOW. In entrambi i casi, è necessario disporre di alcuni privilegi su un oggetto per visualizzare le informazioni su di esso. Le query INFORMATION_SCHEMA che cercano informazioni da più di un database potrebbero richiedere più tempo per l’elaborazione, influenzando negativamente le prestazioni. Per verificare l'efficienza di una query, puoi utilizzare EXPLAIN.

10

REPLICA DEL DB

La replica integrata di MySQL è la base per la creazione di applicazioni grandi e ad alte prestazioni su MySQL, utilizzando la cosiddetta architettura "scale-out". La replica consente di configurare uno o più server come repliche di un altro server, mantenendo i relativi dati sincronizzati con la copia master. Questo non è utile solo per le applicazioni ad alte prestazioni, ma anche alta affidabilità, scalabilità, ripristino di emergenza, backup, analisi, data warehousing e molte altre attività. In effetti, la scalabilità e l'alta affidabilità sono argomenti correlati e intrecceremo questi temi in questo capitolo. In questo capitolo esaminiamo tutti gli aspetti della replica. Sebbene le prestazioni siano importanti, siamo ugualmente interessati alla correttezza e all'affidabilità quando si tratta di una replica, quindi ti mostreremo come la replica può fallire e come aiutarti nelle tue applicazioni. Il problema di base risolto dalla replica è mantenere sincronizzati i dati di un server con quelli di un altro. Molte repliche possono connettersi a un singolo master e rimanere sincronizzate con esso e una replica può, a sua volta, fungere da master. Puoi organizzare master e repliche in molti modi diversi ma MySQL supporta due tipi di replica: replica basata su istruzioni e replica basata su riga. La replica basata su istruzioni (o "logica") è disponibile da MySQL 3.23. La replica basata su righe è stata aggiunta in MySQL 5.1. Entrambi i tipi funzionano registrando le modifiche nel registro binario del master e riproducendo il registro sulla replica, entrambi sono asincroni ovvero non è garantito che la copia dei dati della replica sia aggiornata in un dato istante. Non ci sono garanzie su quanto potrebbe essere grande la latenza sulla replica, infatti, query di grandi dimensioni possono far sì che la replica resti indietro di secondi, minuti o persino ore rispetto al master.

La replica di MySQL è per lo più compatibile con le versioni precedenti. Un server più recente può in genere essere una replica di un server più vecchio senza problemi. Tuttavia, le versioni precedenti del server spesso non sono in grado di fungere da repliche delle versioni più recenti: potrebbero non comprendere le nuove funzionalità o la sintassi SQL utilizzata dal server più recente e potrebbero esserci differenze nei formati di file utilizzati dalla replica. Ad esempio, non è possibile replicare da un master MySQL 5.1 a una replica MySQL 4.0. È una buona idea

testare la configurazione della replica prima di eseguire l'aggiornamento da una versione principale o secondaria a un'altra, ad esempio dalla 4.1 alla 5.0 o dalla 5.1 alla 5.5. Gli aggiornamenti all'interno di una versione minore, ad esempio dalla 5.1.51 alla 5.1.58, sono generalmente compatibili: ti consiglio comunque di leggere il log delle modifiche per scoprire esattamente cosa è cambiato da una versione all'altra.

La replica generalmente non aggiunge molto sovraccarico al master. Richiede l'abilitazione della registrazione binaria (binary logging) sul master, che può avere un sovraccarico significativo, ma è comunque necessaria per backup adeguati e ripristino point-in-time. Oltre alla registrazione binaria, ogni replica collegata aggiunge anche un piccolo carico (principalmente I/O di rete) sul master durante il normale funzionamento. Se le repliche stanno leggendo i vecchi log binari dal master, invece di seguire semplicemente gli eventi più recenti, l'overhead può essere molto più alto a causa dell'I/O richiesto per leggere i vecchi log. Questo processo può anche causare alcune contese mutex che ostacolano i commit delle transazioni. Infine, se stai replicando un carico di lavoro a velocità effettiva molto elevata (ad esempio, 5.000 o più transazioni al secondo) su molte repliche, l'overhead di riattivare tutti i thread di replica per inviarli agli eventi può aumentare. La replica è relativamente buona per ridimensionare le letture, che puoi indirizzare a una replica, ma non è un buon modo per ridimensionare le scritture a meno che non lo progetti correttamente. L'associazione di molte repliche a un master fa sì che le scritture vengano eseguite più volte, una volta su ciascuna replica. L'intero sistema è limitato al numero di scritture che la parte più debole può eseguire. La replica è anche uno spreco con più di poche repliche, perché essenzialmente duplica molti dati inutilmente. Ad esempio, un singolo master con 10 repliche ha 11 copie degli stessi dati e duplica la maggior parte degli stessi dati in 11 diverse cache. Questo non è un uso economico dell'hardware, ma è sorprendentemente comune vedere questo tipo di configurazione di replica.

Problemi risolti dalla replica

Ecco alcuni degli usi più comuni per la replica:

1. Distribuzione dei dati

La replica di MySQL di solito non richiede molta larghezza di banda, sebbene, come vedremo in seguito, la replica basata su righe introdotta in MySQL 5.1 può utilizzare molta più larghezza di banda rispetto alla più tradizionale replica basata su istruzioni. Pertanto, è utile per conservare una copia dei dati in una posizione geograficamente distante, ad esempio un data center diverso. La replica distante può anche funzionare con una connessione intermittente (che sia volontaria o no). Tuttavia, se desideri che le tue repliche abbiano un ritardo di replica molto basso, avrai bisogno di un collegamento stabile a bassa latenza.

1. Bilanciamento del carico

La replica MySQL può aiutarti a distribuire le query di lettura su diversi server, il che funziona molto bene per le applicazioni ad alta intensità di lettura. Puoi eseguire il bilanciamento del carico di base con alcune semplici modifiche al codice. Su piccola scala, puoi utilizzare approcci semplicistici come hostname hardcoded o DNS round-robin (che puntano un singolo nome host a più indirizzi IP). Puoi anche adottare approcci più sofisticati. Le soluzioni standard di bilanciamento del carico, come i prodotti per il bilanciamento del carico di rete, possono funzionare bene per distribuire il carico tra i server MySQL. Anche il progetto Linux Virtual Server (LVS) funziona bene.

1. Backup

La replica è una tecnica preziosa per aiutarti a gestire i backup. Tuttavia, una replica non è né un backup né un sostituto dei backup.

1. Alta affidabilità e failover

La replica può aiutare a evitare di rendere MySQL un singolo punto di errore nell'applicazione. Un buon sistema di failover che prevede la replica può aiutare a ridurre significativamente i tempi di inattività.

1. Testare gli aggiornamenti di MySQL

È prassi comune configurare una replica con una versione aggiornata di MySQL e utilizzarla per garantire che le query funzionino come previsto, prima di aggiornare ogni singola istanza.

Come funziona la replica

Prima di entrare nei dettagli della configurazione della replica, diamo un'occhiata a come MySQL replica effettivamente i dati. Ad alto livello, la replica è un semplice processo in tre parti:

1. Il master registra le modifiche ai suoi dati nel suo log binario (questi record sono chiamati eventi di log binari)
2. La replica copia gli eventi di log binari del master nel suo log di inoltro.
3. La replica riproduce gli eventi nel registro di inoltro, applicando le modifiche ai propri dati.

Questa è solo la panoramica: ognuno di questi passaggi è piuttosto complesso. La prima parte del processo è la registrazione binaria sul master. Appena prima del completamento di ogni transazione che aggiorna i dati sul master, il master registra le modifiche nel proprio registro binario. MySQL scrive le transazioni in serie nel log binario, anche se le istruzioni nelle transazioni sono state intercalate durante l'esecuzione. Dopo aver scritto gli eventi nel registro binario, il master dice ai motori di archiviazione di eseguire il commit delle transazioni. Il passaggio successivo prevede che la replica copi il registro binario del master sul proprio disco rigido, nel cosiddetto registro di inoltro.

Per iniziare, avvia un thread di lavoro, chiamato thread slave di I/O. Il thread I/O apre una normale connessione client al master; quindi, avvia uno speciale processo di dump binlog (non esiste un comando SQL corrispondente). Il processo di dump binlog legge gli eventi dal registro binario del master. Non esegue il polling di eventi, se raggiunge il master, va in standby e aspetta che il master segnali quando ci sono nuovi eventi. Il thread I/O scrive gli eventi nel registro di inoltro della replica.

Prima di MySQL 4.0, la replica funzionava in modo molto diverso. Ad esempio, la prima funzionalità di replica di MySQL non utilizzava un registro di inoltro, quindi la replica utilizzava solo due thread, non tre. La maggior parte delle persone utilizza versioni più recenti del server; quindi,

in questo capitolo non menzioneremo ulteriori dettagli sulle versioni molto vecchie di MySQL.

Il thread slave SQL gestisce l'ultima parte del processo. Questo thread legge e riproduce gli eventi dal registro di inoltro, aggiornando così i dati della replica in modo che corrispondano a quelli del master. Finché questo thread tiene il passo con il thread I/O, il registro di inoltro di solito rimane nella cache del sistema operativo; quindi, i registri di inoltro hanno un sovraccarico molto basso. Gli eventi eseguiti dal thread SQL possono facoltativamente entrare nel log binario della replica, utile per gli scenari che menzioniamo più avanti in questo capitolo. Quindi ci sono due thread di replica che vengono eseguiti sulla replica, ma c'è anche un thread sul master: come qualsiasi connessione a un server MySQL, la connessione che la replica apre al master avvia un thread sul master. Questa architettura di replica disaccoppia i processi di recupero e riproduzione degli eventi sulla replica, consentendo loro di essere asincroni. Il thread di I/O può funzionare indipendentemente dal thread SQL. Inoltre, pone vincoli al processo di replica, il più importante dei quali è che la replica sia serializzata sulla replica. Ciò significa che gli aggiornamenti che potrebbero essere stati eseguiti in parallelo (in thread diversi) sul master non possono essere parallelizzati nella replica, perché vengono eseguiti in un singolo thread. Come vedremo più avanti, questo è un collo di bottiglia delle prestazioni per molti carichi di lavoro. Ci sono alcune soluzioni a questo, ma la maggior parte degli utenti è ancora soggetta al vincolo a thread singolo.

Configurare la replica

L'impostazione della replica è un processo abbastanza semplice in MySQL, ma ci sono molte variazioni sui passaggi di base, a seconda dello scenario. Lo scenario più semplice è un master e una replica appena installati. A un livello elevato, il processo è il seguente:

1. Configurare gli account di replica su ogni server.
2. Configurare il master e la replica.
3. Indicare alla replica di connettersi e replicarsi dal master.

Ciò presuppone che molte impostazioni predefinite siano sufficienti, il che è vero se hai appena installato il master e la replica e hanno gli stessi dati (il database mysql predefinito). Ti mostriamo come eseguire ogni passaggio, supponendo che i tuoi server siano chiamati server1 (indirizzo IP 192.168.0.1) e server2 (indirizzo IP 192.168.0.2). Spiegheremo quindi come inizializzare una replica da un server già attivo e in esecuzione ed esploreremo la configurazione di replica consigliata. MySQL ha alcuni privilegi speciali che consentono l'esecuzione dei processi di replica. Il thread di I/O slave, che viene eseguito sulla replica, stabilisce una connessione TCP/IP al master. Ciò significa che devi creare un account utente sul master e assegnargli i privilegi appropriati, in modo che il thread I/O possa connettersi come quell'utente e leggere il log binario del master. Ecco come creare quell'account utente, che chiameremo repl:

mysql> **GRANT REPLICATION** SLAVE**, REPLICATION** CLIENT **ON *.***

-> **TO** repl@'192.168.0.%' **IDENTIFIED BY** 'p4ssword'**;;**

Creiamo questo account utente sia sul master che sulla replica. Si noti che abbiamo limitato l'utente alla rete locale, perché l'account di replica ha la capacità di leggere tutte le modifiche al server, il che lo rende un account privilegiato (anche se non ha la possibilità di eseguire SELECT o modificare i dati, può comunque vedere alcuni dei dati nei registri binari). L'utente di replica ha effettivamente bisogno solo del privilegio

REPLICATION SLAVE sul master e non ha davvero bisogno del privilegio REPLICATION CLIENT su entrambi i server. Allora perché abbiamo concesso questi privilegi su entrambi i server? Stiamo mantenendo le cose semplici, in realtà. I motivi sono due:

- L'account che utilizzi per monitorare e gestire la replica avrà bisogno del privilegio REPLICATION CLIENT ed è più facile utilizzare lo stesso account per entrambi gli scopi (piuttosto che creare un account utente separato per questo scopo).
- Se si configura l'account sul master e quindi si clona la replica da esso, la replica verrà configurata correttamente per fungere da master, nel caso in cui si desideri che la replica e il master si scambino i ruoli.

Il prossimo passo è abilitare alcune impostazioni sul master, che assumiamo sia chiamato server1. È necessario abilitare la registrazione binaria e specificare un ID server. Immettere (o verificare la presenza di) le seguenti righe nel file my.cnf del master:

log_bin = mysql-**bin**
server_id = 10

I valori esatti dipendono da te. Stiamo prendendo la strada più semplice in questo caso, ma puoi fare qualcosa di più elaborato. È necessario specificare in modo esplicito un ID server univoco. Abbiamo scelto di utilizzare 10 invece di 1, perché 1 è il valore predefinito che un server sceglierà in genere quando non viene specificato alcun valore (questo dipende dalla versione; alcune versioni di MySQL semplicemente non funzioneranno affatto). Pertanto, l'utilizzo di 1 può facilmente causare confusione e conflitti con server che non hanno ID server espliciti. Una pratica comune consiste nell'utilizzare l'ottetto finale dell'indirizzo IP del server, supponendo che non cambi e sia univoco (cioè, i server appartengono a una sola sottorete). Dovresti scegliere una convenzione che abbia senso per te e seguirla. Se la registrazione binaria non era già stata specificata nel file di configurazione del master, dovrai riavviare MySQL. Per verificare che il file di registro binario sia stato creato sul master, esegui

SHOW MASTER STATUS e verificare di ottenere un output simile al seguente. MySQL aggiungerà alcune cifre al nome del file, quindi non vedrai un file con il nome esatto che hai specificato:

```
mysql> SHOW MASTER STATUS;
+-----------------+----------+--------------+------------------+
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----------------+----------+--------------+------------------+
| mysql-bin.000001 | 98 | | |
+-----------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

La replica richiede una configurazione nel suo file my.cnf simile al master e dovrai anche riavviare MySQL sulla replica:

```
log_bin = mysql-bin
server_id = 2
relay_log = /var/lib/mysql/mysql-relay-bin
log_slave_updates = 1
read_only = 1
```

Molte di queste opzioni non sono tecnicamente necessarie e per alcune stiamo solo rendendo esplicite le impostazioni predefinite. In realtà, su una replica è richiesto solo il parametro server_id, ma abbiamo abilitato anche log_bin e abbiamo assegnato al file di log binario un nome esplicito. Per impostazione predefinita, prende il nome dall'hostname del server, ma ciò può causare problemi se l'hostname dovesse cambiare. Stiamo usando lo stesso nome per il master e le repliche per semplificare le cose, ma puoi scegliere diversamente se lo desideri. Abbiamo anche aggiunto altri due parametri di configurazione opzionali: relay_log (per specificare la posizione e il nome del log di inoltro) e log_slave_updates (per fare in modo che la replica registri gli eventi replicati nel proprio log binario). Quest'ultima opzione comporta un po' di lavoro extra per le repliche, ma come vedrai più avanti, abbiamo buone ragioni per aggiungere queste impostazioni opzionali su ogni replica. Alcune persone abilitano solo il log binario e non log_slave_updates, in modo che possano vedere se qualcosa,

come un'applicazione configurata in modo errato, stia modificando i dati sulla replica.

Se possibile, è meglio usare l'impostazione di configurazione di sola lettura, che impedisce a qualsiasi cosa tranne che ai thread con privilegi speciali di modificare i dati (non concedere ai tuoi utenti più privilegi di quelli di cui hanno bisogno!). Tuttavia, read_only spesso non è pratico, specialmente per le applicazioni che devono essere in grado di creare tabelle su repliche.

Non inserire opzioni di configurazione della replica come master_host e master_port nel file my.cnf della replica. Questo è un vecchio modo deprecato per configurare una replica. Può causare problemi e non ha benefici. Il passaggio successivo consiste nel dire alla replica come connettersi al master e iniziare a riprodurre i suoi log binari. Non dovresti usare il file my.cnf per questo; utilizza invece l'istruzione CHANGE MASTER TO. Questa istruzione sostituisce completamente le impostazioni my.cnf corrispondenti. Consente inoltre di puntare la replica a un master diverso in futuro, senza dover arrestare il server. Ecco l'istruzione di base che dovrai eseguire sulla replica per avviare la replica:

```
mysql> CHANGE MASTER TO MASTER_HOST='server1',
-> MASTER_USER='repl',
-> MASTER_PASSWORD='p4ssword',
-> MASTER_LOG_FILE='mysql-bin.000001',
-> MASTER_LOG_POS=0;
```

Il parametro MASTER_LOG_POS è impostato su 0 perché questo è l'inizio del registro. Dopo averlo eseguito, dovresti essere in grado di ispezionare l'output di SHOW SLAVE STATUS e vedere che le impostazioni della replica sono corrette:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
Slave_IO_State:
Master_Host: server1
Master_User: repl
Master_Port: 3306
Connect_Retry: 60
```

Master_Log_File**:** mysql-**bin**.000001
Read_Master_Log_Pos**:** 4
Relay_Log_File**:** mysql-relay-**bin**.000001
Relay_Log_Pos**:** 4
Relay_Master_Log_File**:** mysql-**bin**.000001
Slave_IO_Running**: No**
Slave_SQL_Running**: No**
...omitted**...**
Seconds_Behind_Master**: NULL**

Le colonne Slave_IO_State, Slave_IO_Running e Slave_SQL_Running mostrano che i processi di replica non sono in esecuzione. I lettori più astuti noteranno anche che la posizione del registro è 4 anziché 0. Questo perché 0 non è davvero una posizione del registro; significa semplicemente "all'inizio del file di registro". MySQL sa che il primo evento si trova davvero nella posizione 4. Per avviare la replica, esegui il comando seguente:

mysql> **START** SLAVE**;**

Questo comando non dovrebbe produrre errori o output. Ora controlla di nuovo SHOW SLAVE STATUS:

mysql> SHOW SLAVE STATUS\G
****************************** 1. **row** *****************************
Slave_IO_State**:** Waiting **for** master **to** send event
Master_Host**:** server1
Master_User**:** repl
Master_Port**:** 3306
Connect_Retry**:** 60
Master_Log_File**:** mysql-**bin**.000001
Read_Master_Log_Pos**:** 164
Relay_Log_File**:** mysql-relay-**bin**.000001
Relay_Log_Pos**:** 164
Relay_Master_Log_File**:** mysql-**bin**.000001

Slave_IO_Running**:** Yes
Slave_SQL_Running**:** Yes
...omitted**...**
Seconds_Behind_Master**:** 0

Si noti che i thread slave I/O e SQL sono entrambi in esecuzione e Seconds_Behind _Master non è più NULL. Il thread I/O è in attesa di un evento dal master, il che significa che ha recuperato tutti i log binari del master. Le posizioni del registro sono aumentate, il che significa che alcuni eventi sono stati recuperati ed eseguiti (i risultati varieranno). Se apporti una modifica al master, dovresti vedere aumentare le varie impostazioni di file e posizione sulla replica. Dovresti anche vedere le modifiche nei database sulla replica! Potrai anche vedere i thread di replica nell'elenco dei processi sia sul master che sulla replica. Sul master, dovresti vedere una connessione creata dal thread I/O della replica:

mysql> SHOW PROCESSLIST\G
******************************* 1. **row** *******************************
Id**:** 55
User: repl
Host: replica1**.**webcluster_1**:**54813
db**: NULL**
Command**:** Binlog **Dump**
Time**:** 610237
State: Has sent **all** binlog **to** slave**;** waiting **for** binlog **to** be updated
Info**: NULL**

Sulla replica, dovresti vedere due thread. Uno è il thread I/O e l'altro è il thread SQL:

mysql> SHOW PROCESSLIST\G
******************************* 1. **row** *******************************
Id**:** 1
User: system **user**
Host:
db**: NULL**
Command**: Connect**

Time**:** 611116
State: Waiting **for** master **to** send event
Info**: NULL**
****************************** 2. **row ***************************
Id**:** 2
User: system **user**
Host:
db**: NULL**
Command**: Connect**
Time**:** 33
State: Has **read all** relay **log;** waiting **for** the slave I/O thread **to update** it
Info**: NULL**

L'output di esempio mostrato proviene da server in esecuzione da molto tempo, motivo per cui la colonna Time del thread di I/O sul master e sulla replica ha un valore elevato. Il thread SQL è stato inattivo per 33 secondi nella replica, il che significa che nessun evento è stato riprodotto per 33 secondi. Questi processi verranno sempre eseguiti con l'account utente "system user", ma gli altri valori di colonna potrebbero variare. Ad esempio, quando il thread SQL sta riproducendo un evento sulla replica, la colonna Info mostrerà la query che sta eseguendo.

Inizializzare la replica da un altro server

Le istruzioni di configurazione precedenti presupponevano che tu avessi avviato il master e la replica con i dati iniziali predefiniti dopo una nuova installazione; quindi, avevi implicitamente gli stessi dati su entrambi i server e conoscevi la posizione del registro binario del master. Questo non è in genere il caso, di solito hai un master che è attivo e funzionante da un po' di tempo e ti consigliamo di sincronizzare una replica appena installata con il master, anche se non ha i dati del master. Esistono diversi modi per inizializzare o "clonare" una replica da un altro server. Questi includono la copia dei dati dal master, la clonazione di una replica da un'altra replica e l'avvio di una replica da un backup recente. Sono necessari tre elementi per sincronizzare una replica con un master:

- Uno snapshot dei dati del master in un determinato momento.
- Il file di registro corrente del master e l'offset di byte all'interno di quel registro nel momento esatto in cui è stata scattata l'istantanea. Ci riferiamo a questi due valori come coordinate del file di registro, perché insieme identificano una posizione di registro binario. Puoi trovare le coordinate del file di registro del master con il comando SHOW MASTER STATUS.
- I file di registro binari del master da quel momento ad oggi.

Ecco alcuni modi per clonare una replica da un altro server:

1. Con una copia a freddo (cold copy)

Uno dei modi più semplici per avviare una replica è spegnere il futuro master e copiarne i file nella replica. È quindi possibile riavviare il master, che avvia un nuovo log binario, e utilizzare CHANGE MASTER TO per avviare la replica all'inizio di quel log binario. Lo svantaggio di questa tecnica è evidente: è necessario spegnere il master mentre si esegue la copia.

1. Con una copia a caldo

Se utilizzi solo tabelle MyISAM, puoi usare mysqlhotcopy o rsync per copiare i file mentre il server è ancora in esecuzione.

1. Usando mysqldump

Se utilizzi solo tabelle InnoDB, puoi utilizzare il comando seguente per eseguire il dump di tutto dal master, caricarlo tutto nella replica e modificare le coordinate della replica nella posizione corrispondente nel registro binario del master:

$ mysqldump --single-transaction --all-databases --master-data=1--host=server1 \

| mysql --host=server2

L'opzione --single-transaction fa sì che il dump legga i dati così come esistevano all'inizio della transazione. Se non utilizzi le tabelle transazionali, puoi utilizzare l'opzione --lock-all-tables per ottenere un dump coerente di tutte le tabelle.

1. Con uno snapshot o backup

Finché conosci le coordinate del log binario corrispondente, puoi utilizzare uno snapshot dal master o un backup per inizializzare la replica (se usi un backup, questo metodo richiede che tu abbia conservato tutti i log binari del master dal momento del backup). È sufficiente ripristinare il backup o lo snapshot sulla replica; quindi, utilizzare le coordinate del registro binario appropriate in CHANGE MASTER TO. Puoi usare snapshot LVM, snapshot SAN, snapshot EBS: qualsiasi snapshot andrà bene.

1. Con Percona XtraBackup

Percona XtraBackup è uno strumento di backup a caldo open source introdotto diversi anni fa. Può eseguire backup senza bloccare il funzionamento del server, il che lo rende perfetto per la creazione di repliche. È possibile creare repliche clonando il master o clonando una replica esistente. Basta creare il backup (dal master o da una replica

esistente) e ripristinarlo sulla macchina di destinazione. Quindi cerca nel backup la posizione corretta per avviare la replica:

- Se hai preso il backup dal master della nuova replica, puoi avviare la replica dalla posizione menzionata nel file xtrabackup_binlog_pos_innodb.
- Se hai eseguito il backup da un'altra replica, puoi avviare la replica dalla posizione indicata nel file xtrabackup_slave_info.

L’uso di InnoDB Hot Backup o MySQL Enterprise Backup è un altro buon modo per inizializzare una replica.

1. Da un’altra replica

È possibile utilizzare un qualsiasi snapshot o copia appena menzionate per clonare una replica da un'altra. Tuttavia, se usi mysqldump, l'opzione --master-data non funziona. Inoltre, invece di usare SHOW MASTER STATUS per ottenere le coordinate del registro binario del master, dovrai usare SHOW SLAVE STATUS per trovare la posizione in cui la replica era in esecuzione sul master quando l'hai catturata. Il più grande svantaggio della clonazione di una replica da un'altra è che se la tua replica non è più sincronizzata con il master, clonerai dati errati.

Non importa quale tecnica scegli, mettiti a tuo agio con essa e documentala o creane uno script. Probabilmente lo farai più di una volta e devi essere in grado di farlo in poco tempo se qualcosa dovesse andare storto.

11

SCALABILITÀ

Le persone usano spesso termini come "scalabilità", "alta affidabilità" e "prestazioni" come sinonimi in conversazioni casuali, ma sono completamente diversi. Definiamo le prestazioni come tempo di risposta, anche la scalabilità può essere definita con precisione; lo esploreremo più a fondo tra un momento, ma in poche parole è la capacità del sistema di offrire lo stesso rapporto qualità-prezzo man mano che aggiungi risorse per eseguire più lavoro. I sistemi poco scalabili raggiungono un punto di rendimento decrescente e non possono crescere ulteriormente.

La capacità è un concetto correlato. La capacità del sistema è la quantità di lavoro che può svolgere in un dato periodo di tempo. Tuttavia, la capacità deve essere qualificata. Il throughput massimo del sistema non è uguale alla sua capacità. La maggior parte dei benchmark misura il throughput massimo di un sistema, ma non puoi spingere i sistemi reali così tanto. Se lo fai, le prestazioni si degraderanno e i tempi di risposta diventeranno inaccettabilmente troppo alti e variabili. Definiamo la capacità effettiva del sistema come il throughput che può raggiungere pur offrendo prestazioni accettabili. Questo è il motivo per cui i risultati del benchmark di solito non dovrebbero essere ridotti a un solo numero. Questo non vuol dire che le prestazioni non contino, perché in realtà sono importanti. Stiamo solo sottolineando che i sistemi possono essere scalabili anche se non sono ad alte prestazioni. La scalabilità è la capacità di aggiungere capacità aggiungendo risorse. Anche se la tua architettura MySQL è scalabile, la tua applicazione potrebbe non esserlo. Se è difficile aumentare la capacità per qualsiasi motivo, la tua applicazione non è scalabile, in generale. Posizionare tutti i dati della tua applicazione in una singola istanza MySQL semplicemente non risulterà scalabile. Prima o poi incontrerai colli di bottiglia delle prestazioni. La soluzione tradizionale in molti tipi di applicazioni consiste nell'acquistare server più potenti. Questo è ciò che è noto come "ridimensionamento verticale" o "ridimensionamento". L'approccio opposto consiste nel dividere il tuo lavoro su molti computer, che di solito viene chiamato "ridimensionamento orizzontale". Discuteremo come combinare soluzioni scale-out e scale-up con il consolidamento e come scalare con soluzioni di clustering. Infine, la maggior parte delle

applicazioni dispone anche di alcuni dati che sono raramente o mai necessari e che possono essere eliminati o archiviati. Chiamiamo questo approccio "ridimensionamento", solo per dargli un nome che corrisponda alle altre strategie.

Di solito si inizia a pensare alla scalabilità quando il server ha difficoltà a tenere il passo con l'aumento del carico. Questo di solito si manifesta come uno spostamento del carico di lavoro da CPU a I/O, contesa tra query simultanee e aumento della latenza. I colpevoli sono l'aumento della complessità delle query o di una parte dei dati o dell'indice che prima rientravano nella memoria e che ora sono troppo grandi o complessi per farlo. Potresti notare un cambiamento in alcuni tipi di query e non in altri. Ad esempio, le query lunghe o complesse spesso mostrano il fianco prima delle query più piccole. Se la tua applicazione è altamente scalabile, puoi semplicemente collegare più server per gestire il carico e i problemi di prestazioni scompariranno. Se non fosse scalabile, potresti ritrovarti a combattere gli incendi all'infinito ma puoi evitarlo pianificando la scalabilità. La parte più difficile della pianificazione della scalabilità è stimare la quantità di carico che dovrai gestire. Non è necessario farlo esattamente nel modo giusto, ma è necessario rientrare in un ordine di grandezza. Se sopravvaluti, sprecherai risorse per lo sviluppo, ma se sottovaluti, sarai impreparato al carico. Devi anche stimare il tuo programma in modo approssimativamente corretto, ovvero devi sapere dove si trova "l'orizzonte". Per alcune applicazioni, un semplice prototipo potrebbe funzionare bene per alcuni mesi, dandoti la possibilità di raccogliere capitali e costruire un'architettura più scalabile. Per altre applicazioni, potresti aver bisogno della tua attuale architettura per fornire una capacità sufficiente per due anni.

Ecco alcune domande che puoi porre a te stesso per pianificare la scalabilità:

- Quanto è completa la funzionalità della tua applicazione? Molte delle soluzioni di ridimensionamento che suggeriamo possono rendere più difficile l'implementazione di determinate funzionalità. Se non hai ancora implementato alcune delle funzionalità principali della tua applicazione, potrebbe essere difficile vedere come crearle in un'applicazione ridimensionata. Allo stesso modo, potrebbe essere difficile decidere una

soluzione di ridimensionamento prima di aver visto come funzioneranno davvero queste funzionalità.

- Qual è il carico di picco previsto? La tua applicazione dovrebbe funzionare anche con questo carico. Cosa accadrebbe se il tuo sito venisse linkato dalla prima pagina di Google? Anche se la tua applicazione non è un sito Web popolare, puoi comunque avere picchi di carico. Ad esempio, se sei un rivenditore online, le festività natalizie, in particolare i famigerati giorni di shopping online nelle poche settimane prima di Natale, sono spesso un periodo di picco di carico. Negli Stati Uniti, anche il giorno di San Valentino e il fine settimana prima della Festa della Mamma sono periodi di punta per i fiorai online.
- Se fai affidamento su ogni parte del tuo sistema per gestire il carico, cosa accadrà se una parte di esso si guasta? Ad esempio, se ti affidi alle repliche per distribuire il carico di lettura, puoi comunque tenere il passo se una di esse fallisce? Dovrai disabilitare alcune funzionalità per farlo? Puoi creare una capacità di riserva per alleviare queste preoccupazioni.

Scale up

Scale up (aumentare) significa acquistare hardware più potente e per molte applicazioni questo è sufficiente. Ci sono molti vantaggi in questa strategia, è molto più facile mantenere e sviluppare un singolo server rispetto a più server, con notevoli risparmi sui costi. Anche il backup e il ripristino della tua applicazione su un singolo server è molto più semplice perché non ci sono mai dubbi sulla coerenza o su quale set di dati sia quello valido. Ci sarebbero altre mille ragioni valide con cui continuare ma ci sono anche degli svantaggi. Il costo è complessità e l'aumento è più semplice del ridimensionamento. Oggi i server sono disponibili facilmente con mezzo terabyte di memoria, 32 o più core CPU e più potenza di I/O di quella che puoi persino usare per MySQL (storage flash su schede PCIe, ad esempio). Con la progettazione intelligente di applicazioni e database ottimizzati, puoi creare applicazioni molto grandi con MySQL su tali server. Ma quanto può scalare MySQL su hardware moderno? Sebbene sia possibile eseguirlo su server molto potenti, si scopre che, come la maggior parte dei server di database, MySQL non si adatta perfettamente (sorpresa!) man mano che si aggiungono risorse hardware.

Per eseguire MySQL su server di grandi dimensioni, avrai sicuramente bisogno di una versione recente del server. Le versioni di MySQL 5.0 e 5.1 si strozzeranno molto su hardware così grande, a causa di problemi di scalabilità interna. Avrai bisogno di MySQL 5.5 o successive versioni, o Percona Server dalla versione 5.1 in poi. Anche così, il "punto di rendimento decrescente" attualmente ragionevole è probabilmente intorno a 256 GB di RAM, 32 core e un'unità flash PCIe. MySQL continuerà a fornire prestazioni migliori su hardware più grandi di quello, ma il rapporto prezzo-prestazioni non sarà ottimale, e infatti anche su questi sistemi è spesso possibile ottenere prestazioni molto migliori eseguendo diverse istanze più piccole di MySQL invece di una grande istanza che utilizza tutte le risorse del server.

Il ridimensionamento può funzionare per un po' e molte applicazioni non supereranno questa strategia, ma se l'applicazione diventa estremamente grande alla fine non funzionerà più così bene. Il primo motivo sono i costi. Indipendentemente dal software in esecuzione sul server, a un certo punto l'aumento delle dimensioni diventerà una pessima

decisione finanziaria. Al di fuori della gamma di hardware che offre il miglior rapporto qualità-prezzo, l'hardware tende a diventare più proprietario e insolito e, di conseguenza, più costoso. Ciò significa che esiste un limite pratico su quanto puoi permetterti di scalare. Se usi la replica e aggiorni il tuo master a hardware di fascia alta, ci sono anche poche possibilità che tu possa costruire un server di replica abbastanza potente per tenere il passo. Un master abbastanza carico può facilmente svolgere più lavoro di quanto possa gestire un server di replica con lo stesso hardware, perché il thread di replica non può utilizzare più CPU e dischi in modo efficiente. Infine, non puoi scalare all'infinito, perché anche i computer più potenti hanno dei limiti. Le applicazioni a server singolo di solito incontrano prima i limiti di lettura, soprattutto se eseguono query di lettura complicate. Tali query sono a thread singolo all'interno di MySQL, quindi utilizzeranno solo una CPU. Le CPU server-grade più veloci che puoi acquistare sono solo un paio di volte più veloci delle CPU di fascia base. L'aggiunta di molte CPU o core della CPU non aiuterà le query lente a essere eseguite più velocemente. Il server inizierà anche a raggiungere dei limiti di memoria poiché i tuoi dati diventano troppo grandi per essere memorizzati nella cache in modo efficace. Questo di solito si presenta come un utilizzo intenso del disco e i dischi sono le parti più lente dei moderni computer. Il luogo più ovvio in cui non è possibile aumentare è nel cloud. In genere non è possibile ottenere server molto potenti nella maggior parte dei cloud pubblici; quindi, il ridimensionamento non è un'opzione se la tua applicazione deve crescere di molto. Di conseguenza, ti consigliamo di non pianificare l'aumento indefinito se la prospettiva di raggiungere un limite massimo di scalabilità è reale e rappresenterebbe un serio problema aziendale. Se sai che la tua applicazione crescerà di molto, va bene acquistare un server più potente per il breve termine mentre lavori su un'altra soluzione. Tuttavia, in generale, alla fine dovrai ridimensionare, il che ci porta al nostro prossimo argomento.

Scaling Out

Possiamo raggruppare le tattiche di scale-out in tre grandi gruppi: replica, partizionamento e partizionamento orizzontale. Il modo più semplice e comune per eseguire la scalabilità orizzontale consiste nel distribuire i dati su più server con la replica, quindi utilizzare le repliche per le query di lettura. Questa tecnica può funzionare bene per un'applicazione che ha molte query per la lettura dei dati. Tuttavia, presenta degli svantaggi come la duplicazione della cache, ma anche questo potrebbe non essere un problema grave se la dimensione dei dati è limitata.

L'altro modo comune per scalare consiste nel partizionare il carico di lavoro su più "nodi". Il modo esatto in cui partizionare il carico di lavoro è una decisione complicata. La maggior parte delle grandi applicazioni MySQL non automatizza il partizionamento, almeno non completamente. In questa sezione, diamo un'occhiata ad alcune delle possibilità di partizionamento ed esploriamo i loro punti di forza e di svantaggio. Un nodo è l'unità funzionale nella tua architettura MySQL. Se non stai pianificando ridondanza e alta affidabilità, un nodo potrebbe essere un server. Se stai progettando un sistema ridondante con failover, un nodo è generalmente uno dei seguenti:

- Una coppia di replica master-master, con un server attivo e una replica passiva • Un master e molte repliche
- Un server attivo che utilizza un dispositivo a blocchi replicato (DRBD) per uno standby
- Un "cluster" basato su SAN

Nella maggior parte dei casi, tutti i server all'interno di un nodo dovrebbero avere gli stessi dati. Ci piace l'architettura di replica master-master per nodi attivi-passivi a due server.

La partizione funzionale, o divisione dei compiti, significa dedicare nodi diversi a compiti diversi e abbiamo già menzionato alcuni approcci simili. Il partizionamento funzionale di solito sfrutta questa strategia dedicando singoli server o nodi a diverse applicazioni, quindi ognuna contiene solo i dati necessari alla sua particolare applicazione. Stiamo

usando la parola "applicazione" in modo un po' ampio qui. Non intendiamo un singolo programma per computer, ma un insieme di programmi correlati facilmente separabili da altri programmi non correlati. Ad esempio, se disponi di un sito Web con sezioni distinte che non richiedono la condivisione dei dati, puoi partizionare per area funzionale sul sito Web. È comune vedere portali che legano insieme le diverse aree; dal portale è possibile navigare nella sezione news del sito, nei forum, nell'area supporto e knowledge base, e così via. I dati per ciascuna di queste aree funzionali potrebbero trovarsi su un server MySQL dedicato.

Se l'applicazione è enorme, ogni area funzionale può anche avere il proprio server web dedicato, ma è una strategia meno comune. Un altro possibile approccio al partizionamento funzionale consiste nel dividere i dati di una singola applicazione determinando insiemi di tabelle che non si uniscono mai tra loro. Se necessario, di solito puoi eseguire alcuni di questi join nell'applicazione se non sono critici per le prestazioni. Esistono alcune varianti di questo approccio, ma hanno la proprietà comune che ogni tipo di dati può essere trovato solo su un singolo nodo. Questo non è un modo comune per partizionare i dati, perché è molto difficile da attuare in modo efficace e non offre alcun vantaggio rispetto ad altri metodi. In ultima analisi, non è ancora possibile ridimensionare il partizionamento funzionale indefinitamente, perché ogni area funzionale deve ridimensionarsi verticalmente se è collegata a un singolo nodo MySQL. È probabile che una delle applicazioni o aree funzionali alla fine diventi troppo grande, costringendoti a trovare una strategia diversa. E se si spinge troppo oltre il partizionamento funzionale, può essere più difficile passare a un design più scalabile in un secondo momento.

Sharding dei dati

Il data sharding è l'approccio più comune e di successo per il ridimensionamento delle applicazioni MySQL di grandi dimensioni. Dividi i dati suddividendoli in parti più piccole, o frammenti, e archiviandoli su nodi diversi. Il partizionamento orizzontale funziona bene se combinato con un certo tipo di partizionamento funzionale. La maggior parte dei sistemi partizionati ha anche alcuni dati "globali" che non sono affatto partizionati (ad esempio, elenchi di città o dati di accesso). Questi dati globali sono generalmente archiviati su un singolo nodo, spesso dietro una cache come memcached. In effetti, la maggior parte delle applicazioni divide solo i dati che necessitano di partizionamento orizzontale, in genere le parti del set di dati che diventeranno molto grandi.

Supponiamo che tu stia costruendo un servizio di blogging. Prevendo 10 milioni di utenti, potresti non aver bisogno di partizionare le informazioni di registrazione dell'utente perché potresti essere in grado di adattare tutti gli utenti (o il loro sottoinsieme attivo) interamente in memoria. Se ti aspetti 500 milioni di utenti, d'altra parte, dovresti probabilmente frammentare questi dati. Il contenuto generato dagli utenti, come post e commenti, richiederà quasi sicuramente lo sharding in entrambi i casi, perché questi record sono molto più grandi e sono molti di più. Le applicazioni di grandi dimensioni potrebbero avere diversi set di dati logici che è possibile partizionare in modo diverso. Puoi archiviarli su diversi set di server, ma non è necessario. Puoi anche dividere gli stessi dati in più modi, a seconda di come accedi.

Il partizionamento orizzontale è notevolmente diverso dal modo in cui la maggior parte delle applicazioni viene progettata inizialmente e può essere difficile modificare un'applicazione da un archivio dati monolitico a un'architettura partizionata. Ecco perché è molto più semplice creare un'applicazione con un archivio dati frammentato fin dall'inizio se prevedi che alla fine ne avrai bisogno. La maggior parte delle applicazioni che non integrano lo sharding dall'inizio attraversano fasi man mano che diventano più grandi. Ad esempio, puoi utilizzare la replica per ridimensionare le query di lettura sul tuo servizio di blogging finché non funziona più. Quindi puoi dividere il servizio in tre parti: utenti, post e commenti. Puoi posizionarli su server diversi (partizionamento funzionale), magari con

un'architettura orientata ai servizi, ed eseguire i join nell'applicazione. Infine, puoi dividere i post e i commenti in base all'ID utente e mantenere le informazioni sull'utente su un singolo nodo.

Se sai in anticipo che dovrai ridimensionare molto e conosci i limiti del partizionamento funzionale, potresti scegliere di saltare i passaggi intermedi e passare direttamente da un singolo nodo a un archivio dati frammentato. In effetti, la previsione può spesso aiutarti a evitare brutti schemi di sharding che potrebbero derivare dall'affrontare ogni sfida man mano che si presenta. Le applicazioni partizionate hanno spesso una libreria di astrazione del database che facilita la comunicazione tra l'applicazione e l'archivio dati partizionato. Tali librerie di solito non nascondono completamente il partizionamento orizzontale, perché l'applicazione di solito sa qualcosa su una query che l'archivio dati non conosce. Troppa astrazione può causare inefficienze, ad esempio, porterebbe ad interrogare tutti i nodi per i dati che risiedono su un singolo nodo.

Un archivio dati frammentato potrebbe sembrare una soluzione elegante, ma è difficile da costruire. Allora perché scegliere questa architettura? La risposta è semplice: se vuoi scalare la tua capacità di scrittura, devi partizionare i tuoi dati. Non è possibile ridimensionare la capacità di scrittura se si dispone di un solo master, indipendentemente dal numero di repliche di cui disponi. Lo sharding, nonostante tutti i suoi inconvenienti, è la soluzione preferita a questo problema.

Allora ci si potrebbe chiedere: sharding, si o no? È questa la domanda, vero? Ecco la semplice risposta: non effettuare lo shard a meno che non sia necessario. Verifica se puoi ritardarlo tramite l'ottimizzazione delle prestazioni o una migliore progettazione dell'applicazione o del database. Se riesci a rimandare lo sharding abbastanza a lungo, potresti essere in grado di acquistare un server più grande, aggiornare MySQL a una nuova versione con prestazioni più elevate e continuare ad usare un singolo server, magari insieme alla replica. In poche parole, il partizionamento orizzontale è inevitabile quando la dimensione dei dati o il carico di lavoro di scrittura diventano eccessivi per un singolo server. Sareste sorpresi nel sapere di quanto i sistemi possono essere scalati senza partizionamento orizzontale, utilizzando la progettazione di applicazioni intelligenti. Alcune applicazioni molto popolari che probabilmente penseresti siano state “sharded” dal primo giorno; infatti, sono cresciute fino a valutazioni multimiliardarie e

quantità folli di traffico senza sharding. Non è l'unica alternativa ed è un modo complesso per creare un'applicazione se non è necessaria.

Ridimensionare correttamente MySQL è un po' meno affascinante di quanto possa sembrare. Il modo giusto per scalare non è costruire la prossima architettura di Facebook dal primo giorno. La strategia migliore è fare ciò che è chiaramente necessario per la tua applicazione e pianificare in anticipo in modo che, se cresci molto rapidamente, il tuo successo finanzierà tutti i passaggi necessari per soddisfare la domanda. È prezioso avere una definizione matematica di scalabilità, così come è utile avere un concetto preciso di prestazione. Sapere che i sistemi non riescono a scalare in modo lineare a causa di costi come la serializzazione e la diafonia può aiutarti a evitare di creare questi problemi nella tua applicazione. Allo stesso tempo, molti problemi di scalabilità non sono matematici; possono essere dovuti a problemi all'interno dell'organizzazione, come la mancanza di lavoro di squadra o altri problemi meno concreti. In termini di strategie di scalabilità MySQL, l'applicazione tipica che cresce molto di solito si sposta da un singolo server, a un'architettura scale-out con repliche di lettura, allo sharding e/o al partizionamento funzionale. Non siamo d'accordo con coloro che sostengono un approccio "shard early, shard many" per ogni applicazione, è complicato, costoso e molte applicazioni non ne avranno mai bisogno. È perfettamente legittimo aspettare il proprio tempo e vedere cosa succede con il nuovo hardware, le nuove versioni di MySQL o i nuovi sviluppi in MySQL Cluster e persino valutare un sistema proprietario come Clustrix. Lo sharding è un sistema di clustering costruito a mano, dopotutto, è una buona idea non reinventare la ruota se non è necessario.

Dove sono presenti più server, ci sono problemi di coerenza e atomicità. I problemi più comuni che vediamo sono la mancanza di coerenza della sessione (pubblicazione di un commento su un sito Web, aggiornamento della pagina e mancata visualizzazione del commento appena pubblicato) ed errori nel comunicare all'applicazione quali server sono leggibili e scrivibili.

12

MYSQL E IL CLOUD

Non dovrebbe sorprendere che molte persone utilizzino MySQL nel cloud, a volte su larga scala. Nella nostra esperienza, la maggior parte di loro utilizza la piattaforma Amazon Web Services (AWS): in particolare Elastic Compute Cloud (EC2) di Amazon, i volumi Elastic Block Store (EBS) e, in misura minore, il Relational Database Service (RDS). Un modo per inquadrare la discussione su MySQL nel cloud è dividerla in due categorie approssimative:

- IaaS (Infrastructure as a Service) è un'infrastruttura cloud per l'hosting del tuo server MySQL. Puoi acquistare una risorsa del server virtuale nel cloud e utilizzarla per installare ed eseguire la tua istanza MySQL. Puoi configurare MySQL e il sistema operativo come desideri, ma non hai accesso o informazioni dettagliate sull'hardware fisico sottostante.
- DBaaS (Database as a Service) dove MySQL stesso è la risorsa gestita dal cloud. Ricevi le credenziali di accesso a un server MySQL e puoi configurare alcune delle impostazioni di MySQL, ma non hai accesso o informazioni dettagliate sul sistema operativo sottostante o sull'istanza del server virtuale. Un esempio è Amazon RDS che esegue MySQL. Alcuni di questi servizi non sono realmente MySQL di serie, ma sono compatibili con il protocollo MySQL e il linguaggio di query.

Ci sono molte buone risorse per imparare a distribuire e gestire MySQL e le risorse necessarie per eseguirlo, e ci sono troppe piattaforme per poterle coprire tutte; quindi, non mostriamo esempi di codice o discutiamo di tecniche operative. Invece, questo capitolo si concentra sulle differenze chiave tra l'esecuzione di MySQL nel cloud e l'implementazione tradizionale del server e sulle caratteristiche economiche e prestazionali risultanti. Partiamo dal presupposto che tu abbia un minimo di familiarità con il cloud computing. Questa non è un'introduzione all'argomento; il nostro obiettivo è solo quello di aiutarti a evitare alcune insidie che potresti incontrare se non sei un esperto di MySQL in cloud.

In generale, MySQL funziona bene nel cloud. L'esecuzione di MySQL nel cloud non è molto diversa dall'esecuzione di MySQL su qualsiasi altra piattaforma, ma ci sono diverse distinzioni molto importanti. Devi esserne consapevole e progettare in modo appropriato la tua applicazione e architettura, per ottenere buoni risultati. In alcune circostanze l'hosting di MySQL nel cloud non è un'ottima soluzione, e talvolta è la cosa migliore, ma nella maggior parte dei casi è solo un'altra piattaforma di distribuzione. È importante comprendere che il cloud è una piattaforma di distribuzione, non un'architettura. La tua architettura è influenzata dalla piattaforma, ma la piattaforma e l'architettura sono distinte. Se confondessi le due cose, potresti essere più propenso a fare scelte sbagliate che possono causare problemi in futuro.

Ecco perché passeremo un po’ di tempo a discutere quali differenze contano per MySQL nel cloud. Il cloud computing ha molti vantaggi, alcuni dei quali sono specifici dell'utilizzo con MySQL. Ci sono libri scritti su questo argomento, e non vogliamo dedicarci troppo tempo. Ma elencheremo alcuni elementi importanti da tenere in considerazione, perché discuteremo degli svantaggi tra un momento e non vogliamo che tu pensi che siamo eccessivamente critici nei confronti del cloud:

- Il cloud è un modo per esternalizzare parte della tua infrastruttura in modo da non doverla gestire. Non è necessario acquistare hardware e predisporre uffici ad hoc, non è necessario sostituire dischi rigidi guasti e così via.
- Il cloud ha generalmente un prezzo in base al consumo, convertendo le spese di capitale anticipate in spese operative correnti.
- Il cloud offre un valore crescente nel tempo poiché i provider offrono nuovi servizi e costi inferiori. Non devi fare nulla (come aggiornare i tuoi server) per sfruttare questi miglioramenti; hai semplicemente a tua disposizione più opzioni ad un costo inferiore con il passare del tempo.
- Il cloud ti consente di effettuare facilmente il provisioning di server e altre risorse e di spegnerli quando hai finito, senza doveri smaltire o senza doverli rivendere per recuperare i costi.
- Il cloud rappresenta un modo diverso di pensare all'infrastruttura, in quanto risorse definite e controllate tramite

API, questo consente molta più automazione. Puoi anche ottenere questi vantaggi in un "cloud privato".

Naturalmente, non tutto ciò che riguarda il cloud è buono. Ecco alcuni inconvenienti che possono porre delle sfide:

- Le risorse sono condivise e imprevedibili, perciò, potresti effettivamente ottenere più di quanto stai pagando. Questo potrebbe sembrarti un aspetto positivo ma può rendere difficile la pianificazione della capacità. Se stai ricevendo più della tua quota di risorse informatiche e non lo sai, c'è il rischio che qualcun altro rivendichi la giusta quota di risorse, riportando le tue prestazioni a quelle che dovrebbero essere. In generale, può essere difficile sapere con certezza cosa dovresti ottenere e la maggior parte dei provider di hosting cloud non fornisce risposte concrete su tali domande.
- Non ci sono garanzie sulla capacità o sulla disponibilità.
- Le risorse virtualizzate e condivise possono essere più difficili da gestire, soprattutto perché non si ha accesso all'hardware fisico sottostante per ispezionare e misurare ciò che sta accadendo. Ad esempio, abbiamo visto sistemi in cui iostat affermava che l'I/O andava bene o vmstat mostrava che la CPU andava bene, eppure quando abbiamo effettivamente misurato il tempo trascorso per completare le attività, le risorse erano chiaramente sovraccaricate da qualcos'altro sul sistema. Se si verificano problemi di prestazioni su una piattaforma cloud, è ancora più importante del solito misurare con attenzione. Potresti non essere in grado di identificare se il sistema sottostante sta funzionando male o se hai fatto qualcosa che sta causando all'applicazione richieste non valide per il sistema.

Possiamo riassumere i punti precedenti dicendo che c'è una ridotta trasparenza e controllo su prestazioni, disponibilità e capacità nel cloud. Infine, ecco alcuni miti del cloud da tenere a mente. Grazie al cloud, le applicazioni, le loro architetture e le organizzazioni che le gestiscono sono scalabili. Il cloud non è intrinsecamente scalabile solo perché è un cloud e

la scelta di una piattaforma scalabile non rende automaticamente scalabile la tua applicazione. È vero che se il provider di hosting cloud non ha sottoscrizioni in più, ci sono risorse che puoi acquistare su richiesta, ma la disponibilità delle risorse quando ne hai bisogno è un tipico aspetto solo della scalabilità.

I singoli server ospitati nel cloud hanno in realtà maggiori probabilità di fallire o avere interruzioni, in generale, rispetto a un'infrastruttura dedicata ben progettata. Molte persone non se ne rendono conto, tuttavia. Ad esempio, una persona ha scritto "stiamo aggiornando la nostra infrastruttura a un sistema basato su cloud per darci il 100% di uptime e scalabilità". Questo è successo subito dopo che AWS ha subito due enormi interruzioni che hanno colpito gran parte dei suoi utenti. Un buon architetto può progettare sistemi affidabili con componenti inaffidabili, ma in generale un'infrastruttura più affidabile contribuisce a una maggiore disponibilità. Non esiste un tempo di attività del 100%, ovviamente. D'altra parte, abbonandoti a un servizio di cloud computing, stai acquistando una piattaforma creata da esperti, si sono presi cura di molte cose di basso livello per conto tuo e ciò significa che puoi concentrarti su compiti di livello superiore. Se costruisci la tua piattaforma e non sei un esperto di tutte quegli aspetti, è probabile che tu commetta degli errori da principiante, che probabilmente causeranno dei tempi di inattività prima o poi. In questo modo, il cloud computing può aiutarti a migliorare i tuoi tempi di attività.

Molti dei vantaggi del cloud sono ereditati dalle tecnologie utilizzate per creare piattaforme cloud e possono essere ottenuti con o senza il cloud. Con una virtualizzazione e una pianificazione della capacità gestite correttamente, ad esempio, puoi avviare una nuova macchina nel modo più semplice e rapido possibile in qualsiasi cloud. Non hai bisogno del cloud per questo.

Il cloud computing offre vantaggi unici, indubbiamente, e nel tempo svilupperemo una maggiore comprensione condivisa di cosa sono e quando sono utili. Una cosa è certa: questo è qualcosa del tutto nuovo e in continua evoluzione.

CONCLUSIONI

In questa guida abbiamo cercato di mostrare la flessibilità e la potenza di MySQL e di spiegare come applicare queste caratteristiche ai problemi del mondo reale.

Avete imparato a gestire funzioni, query, tabelle e database, quindi fate del vostro meglio per rendere il codice più leggibile ed evitare query troppo lunghe o complesse.

Molti dei concetti e delle funzionalità che abbiamo analizzato sono uguali o simili in altri database, in quanto possono essere utilizzati in DB2, PostgreSQL e molti altri. È necessario comprendere chiaramente quali sono le parole chiave, come e quali dati possono essere memorizzati, come strutturare il database e come estrarre informazioni utili da esso. Si sono verificati problemi o MySQL restituisce errori? Non lasciatevi scoraggiare, spesso ci sono errori di sintassi, quindi fate riferimento alla documentazione ufficiale per risolvere il problema. Se si impostano le query su più righe, leggere attentamente il messaggio di MySQL per scoprire quale riga presenta l'errore. Non sarete mai al sicuro da errori di sintassi o di logica, ma con una maggiore esperienza con MySQL imparerete a evitare molti problemi. Gli errori possono essere una grande opportunità di apprendimento, perché "*si impara sbagliando*".