Indici e chiavi

Effettuare una query su un database è in genere abbastanza semplice, grazie alla sintassi intuitiva del linguaggio SQL. Più complesso può essere, invece, ottimizzarne le performance. Per questo scopo esistono degli strumenti appositi: gli **indici**.

Un indice è una struttura dati ausiliaria che consente di recuperare più velocemente i dati di una tabella, evitandone la lettura dell'intero contenuto (*full table scan*), tramite una selezione più mirata.

Gli indici vanno usati consapevolmente, verificando quando sono effettivamente necessari e scegliendo con cura su quali campi della tabella applicarli. Un loro abuso, infatti, potrebbe avere addirittura l'effetto di ridurre le performance di interfacciamento con il database. Infatti, tali strutture dati vanno aggiornate ad ogni modifica apportata alla tabella, e quindi se da un lato gli indici agevolano le operazioni di lettura, dall'altro rendono più onerose tutte le altre.

Con MySQL si possono creare indici su qualunque tipo di dato, ma i loro dettagli applicativi dipendono dallo **Storage Engine** scelto per la tabella. Gli Storage Engine verranno trattati più avanti in questa guida, e per il momento è sufficiente sapere che si tratta dei gestori del salvataggio e reperimento dei dati su disco.

Nel seguito della lezione, si illustreranno le azioni collegate alla **creazione e gestione degli indici**. Tutti i comandi, espressi nella sintassi SQL, possono essere verificati tramite il client *mysql* avendo a disposizione un'istanza in funzione del DBMS ed un database già creato.

Tipi di indici

Esistono diversi tipi di indici:

- **PRIMARY KEY**: applicato ad uno o più campi di una tabella permette di distinguere univocamente ogni riga. Il campo sottoposto all'indice primary key non ammette duplicati né campi nulli;
- **UNIQUE**: simile alla primary key, con la differenza che tollera valori nulli, mentre i duplicati restano vietati;
- **COLUMN INDEX**: sono gli indici più comuni. Applicati ad un campo di una tabella, hanno puramente lo scopo di velocizzarne l'accesso permettendo valori duplicati e nulli. Come variante, possono esistere indici "multicolonna", che includono quindi più campi della tabella, oppure i cosiddetti **PREFIX INDEX** che, nei campi stringa, permettono di indicizzare non tutto il campo ma solo una porzione iniziale di caratteri, appunto il prefisso;

- **FULLTEXT**: sono indici che permettono di accelerare operazioni onerose, come la ricerca testuale su un intero campo.

Creare chiavi primarie

La chiave primaria viene solitamente creata in fase di definizione di una tabella. L'abbiamo già visto in una lezione precedente: si sceglie un campo indicato per tale ruolo e lo si designa ufficialmente come tale usando la keyword *PRIMARY KEY*.

```
CREATE TABLE `Persone` (
  `id` INT NOT NULL AUTO_INCREMENT,
  ...
  ...
  PRIMARY KEY (`id`))
```

Dovendo possedere caratteristiche di unicità, il campo ideale da scegliere come chiave primaria dovrebbe essere univocamente distintivo del record in cui si trova (possono andare bene dati come partite IVA, codici fiscali, numeri di documenti di identità, eccetera, in quanto univoci per definizione). Nel caso non ve ne siano, viene adottato – e non succede di rado – un numero intero progressivo, incrementato automaticamente dal DBMS ogni volta che si inserisce una nuova riga. L'esempio che vedremo più avanti utilizza questa opzione.

Analogamente, possono essere scelte più colonne per costituire una chiave primaria. In questo caso, i campi nel complesso costituiranno un'identificazione univoca per la riga.

Una chiave primaria può anche essere creata successivamente alla creazione della tabella:

```
CREATE PRIMARY KEY ON nome_tabella (elenco campi);
```

Creare indici

Al pari delle chiavi primarie, gli indici possono essere creati contestualmente alla tabella o aggiunti successivamente.

Nel primo caso, la dichiarazione avverrà all'interno del costruttore *CREATE TABLE*:

```
CREATE TABLE `Persone` (
  ...
  `eta` INT NOT NULL,
  ...
  INDEX eta_ind (`eta`) );
```

Nello stralcio di codice precedente, viene creato un indice sul campo *eta*. Ciò agevolerà selezioni di righe in base a questo valore. Si noti, inoltre, che l'indice possiede un nome che lo distingue, in questo caso *eta_ind*.

In alternativa, l'indice può essere creato successivamente alla definizione della tabella:

```
CREATE INDEX eta_ind ON Persone (`eta`) ;
```

L'**eliminazione di un indice** avviene tramite comando *DROP*:

```
DROP INDEX eta_ind ON Persone;
```

Altro aspetto interessante è che si possono creare, come accennato, dei **PREFIX INDEX**, basati su un prefisso di un campo stringa, ossia solo sui primi caratteri del valore del campo. Ciò può essere utile in campi testuali, dove realizzare un PREFIX INDEX può comunque agevolare le ricerche senza rendere troppo onerosa la gestione dell'indice:

```
CREATE TABLE `Persone` (
  ...
  `cognome` VARCHAR(20) NOT NULL,
  ...
  INDEX cognome_ind (cognome(6)) );
```

Un *PREFIX INDEX* viene dichiarato nella stessa maniera di un indice normale, l'unica differenza consiste nel numero tra parentesi tonde, che indica quanti byte (e non quanti caratteri) verranno catalogati.

FOREIGN KEYS e vincoli

Le *FOREIGN KEYS* sono costrutti che sfruttano gli indici per collegare due tabelle mediante l'associazione di campi. Applicare vincoli ai vari campi di diverse tabelle consente di mantenere la **consistenza dei dati**.

L'uso di questa tecnica è parte integrante dello Storage Engine **InnoDB**, che dalla versione 5.5 del DBMS è impostato di default nelle tabelle.

Sin dalle prime lezioni di questa guida, si è spiegato che in un database relazionale le tabelle sono collegate tra loro tramite relazioni logiche, facendo sì che righe di una tabella contengano riferimenti a valori chiave di un'altra tabella.

Tipicamente, in queste relazioni, esistono delle tabelle principali ed altre secondarie. Ad esempio, in un database che gestisce utenti ed i loro contatti, potremmo avere una

tabella principale per catalogare i dati personali ed una secondaria per contenere uno o più riferimenti telefonici per utente:

```
CREATE TABLE utenti (`id` INT NOT NULL AUTO_INCREMENT,
        nome VARCHAR(50),
        ...
        ...
        PRIMARY KEY (`id`)
) ;

CREATE TABLE contatti(`id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
             utente_id INT,
             telefono VARCHAR(20),
           INDEX utente_ind (utente_id),
           FOREIGN KEY (utente_id) REFERENCES utenti(`id`)
);
```

L'esempio è schematico ma mette in luce i principali aspetti di questo tipo di legame:

- le tabelle devono avere lo stesso Storage Engine, InnoDB, ed i campi coinvolti nella relazione devono essere di tipi simili;
- la clausola *FOREIGN KEY* deve essere contenuta all'interno della definizione della tabella secondaria;
- i campi coinvolti nella relazione devono essere sottoposti ad un indice di qualche tipo, sia nella tabella principale che in quella secondaria, in modo da renderli accessibili in maniera efficiente;
- non possono essere coinvolti indici di tipo *PREFIX*, quindi non si possono utilizzare campi *BLOB* o *TEXT*.

Il mantenimento dell'**integrità referenziale** (ciò che consente di mantenere la consistenza dei dati logicamente connessi, su tabelle distinte) segue due regole.

La prima vieta di inserire o modificare dati nella tabella secondaria che non abbiano collegamento con quelli della tabella principale. Ad esempio, nella tabella *contatti*, ogni numero di telefono inserito deve corrispondere ad un ID di un utente esistente nella tabella principale. Operazioni che violano questa regola verranno respinte da MySQL che addurrà come motivazione la violazione del vincolo imposto dalla foreign key.

La seconda regola dispone che, in caso di cancellazione o modifica di dati nella tabella principale, un apposito vincolo debba specificare quale azione dovrà essere applicata nella tabella secondaria.

Un vincolo di tale genere può essere specificato in fase di definizione della foreign key. Per gestire la cancellazione possiamo procedere come segue:

```
FOREIGN KEY (utente_id) REFERENCES utenti(`id`)
ON DELETE azione da attivare
```

Se invece vogliamo gestire il caso dell'aggiornamento:

```
FOREIGN KEY (utente_id) REFERENCES utenti(`id`)
ON UPDATE azione da attivare
```

Le azioni attivabili sono cinque:

- **CASCADE**: la cancellazione o la modifica di una riga nella tabella principale causerà, a cascata, la medesima modifica nella tabella secondaria;
- **SET NULL**: il campo oggetto della relazione nella tabella secondaria verrà impostato a *NULL*. In questo caso, è necessario che tale campo non sia stato qualificato come *NOT NULL* in fase di creazione;
- **RESTRICT**: impedisce che la modifica o la cancellazione nella tabella principale venga eseguita. Equivale a non specificare alcun vincolo, in pratica è l'azione di default;
- **NO ACTION**: in MySQL è un sinonimo di *RESTRICT*, quindi vale quanto detto al punto che precede;
- **SET DEFAULT**: nonostante il parser di MySQL la riconosca come valida, questa impostazione è vietata dal motore InnoDB.