Quando usare *ID auto increment* non è una buona idea

I programmatori PHP hanno spesso a che fare con il database MySQL e con il suo successore, MariaDB, mantenuto dagli stessi creatori di MySQL che crearono il fork mantenendolo compatibile (nonché migliorandolo) temendo che Oracle, dopo l'acquisizione di MySQL, avrebbe mantenuto un "basso profilo" di sviluppo per quest'ultimo, al fine di favorire i propri database commerciali. A dirla tutta, pare proprio che abbiano avuto ragione!

Una delle funzionalità più comode ed usate è quella fornita dall'attributo AUTO_INCREMENT dei campi numerici, spesso usata per creare facilmente degli ID numerici univoci, in quanto il numero aumenta in automatico di 1 ad ogni INSERT, resta invariato ad ogni UPDATE e l'ultimo valore generato viene considerato anche in caso di DELETE.

In pratica, se ad esempio inserite 4 record con un campo "id" con attributo AUTO_INCREMENT senza specificarne il valore, essi avranno id = 1, 2, 3, 4. Se eliminate il record con id = 4 e poi inserite un nuovo record, questo avrà id = 5. Ci togliamo dunque il pensiero di dover identificare i record! Fantastico!

Ecco quindi che possiamo sviluppare le nostre applicazioni in grado di individuare prodotti, utenti e quant'altro inserendo semplicemente l'ID nell'URL, come nei seguenti esempi:

http://www.esempio.com/prodotti.php?id=16
http://www.esempio.com/prodotti/16-moka-bialetti/
http://www.esempio.com/utenti.php?id=23
http://www.esempio.com/utenti/23-tuonome/

Se 6 secoli non bastano

In informatica tutto ha un limite e la domanda sorge spontanea: qual è il numero massimo che posso raggiungere usando AUTO_INCREMENT?

Può sembrare che MAX_ROWS (valore di default = 32 bit) indichi il numero massimo di record che può contenere una tabella e che quindi possa entrare in conflitto con AUTO_INCREMENT: di fatto può succedere soltanto se si usa lo storage engine NDB, perciò per chi non lo usa, cioè la stragrande maggioranza degli sviluppatori, si tratta soltanto del "numero minimo di record che le tabelle devono poter contenere".

La risposta alla domanda è quindi 2^64, cioè un numero a 64 bit, raggiungibile assegnando al campo "id" il tipo BIGINT. Ciò significa che, a seconda della frequenza degli inserimenti, sarete coperti per:

- Google (54.000 ricerche al secondo) = 10 milioni di anni
- 1 miliardo di INSERT al secondo = 585 anni
- 100 miliardi di INSERT al secondo = 5,85 anni

E non mi basta mai (voglio di più)

Può sembrare assurdo che qualcuno possa aver bisogno di valori più grandi di questi, ma di questi tempi, in cui una mole di Big data può sbucare in qualsiasi momento da dietro l'angolo (Social network, e-commerce, geolocalizzazione, ecc.), può capitare di imbattersi in casi in cui si hanno esigenze particolari, tra le quali c'è anche il bisogno di un maggior numero di ID possibile.

Ecco, quindi, che si cercano delle alternative e ci si imbatte ben presto in UUID (Universal Unique Identifier). Solitamente viene usata la variante RFC 4122 la quale ha 5 versioni, che determinano il modo in cui viene generato un valore a 128 bit (in pratica il quadrato del valore massimo di BIGINT) che per leggibilità viene scritto in esadecimale e separato con dei trattini in cinque gruppi (es. 123e4567-e89b-12d3-a456-426655440000).

È molto usata la versione 4, che genera l'ID in maniera casuale. Il fatto però di usare questa variante implica che alcuni bit vengano riservati e consente, quindi, di generare "soltanto" 2^122 valori unici.

È possibile individuare la versione nel valore stesso:

xxxxxxxx-xxxx-Vxxx-xxxx-xxxxxxxxxxxx
V = versione

Grazie al caso

Può sembrare assurdo lasciare la sicurezza dei valori unici AUTO_INCREMENT in favore di quelli random di UUID4. Ma, dopotutto, che probabilità ci sono di generare due valori uguali? Inferiori a quella di essere colpiti da un meteorite: solo dopo aver generato 1 miliardo di UUID al secondo per 100 anni la probabilità di creare un duplicato arriva al 50%.

Può sembrare comunque assurdo abbandonare la sicurezza di unicità dei valori AUTO_INCREMENT per abbracciare la casualità dei valori UUID4. Eppure, grazie al caso otteniamo alcuni vantaggi:

- rendere ragionevolmente probabile la generazione di un ID univoco in assenza di un generatore di ID centralizzato (che è il motivo per il quale è stato creato UUID)
- mascherare gli ID
    - BASE64 non è in grado di farlo ed MD5 e SHA vari nemmeno, dato che non generano valori univoci (MD5 dimostrato, gli altri quasi) e che esistono ormai diversi

dizionari per decodificare le stringhe
- impedisce di predire quale sarà il prossimo ID generato, e quindi migliora la sicurezza. Ad esempio, evita che gli utenti possano cambiare facilmente ID nell'URL per vedere altri contenuti: un conto è aggiungere o togliere 1, un conto è indovinare un ID a 32 cifre generato casualmente
- evita figuracce e abbandoni… Immaginate di iscrivervi ad un nuovo social network o a un sito di annunci e di scoprire che il vostro post è il 1033esimo: vi verrebbe voglia di sottoscrivere un account premium? Certo che no, cerchereste immediatamente un sito più serio
- può essere compresso in un campo BINARY(16)
- evitare che si verifichino problemi quando si replicano id AUTO_INCREMENT
- è molto comodo che gli UUID siano unici anche tra tabelle

Il grande svantaggio è ovviamente che è possibile che si verifichi un duplicato. Per ovviare a questo inconveniente, può venire la tentazione di applicare una chiave UNIQUE al campo "id", ma applicare chiavi o indici a questo campo alla lunga rallenta gli inserimenti e di conseguenza si tratta di un intervento che va valutato. Piuttosto, si possono usare uno o più server dedicati che grazie a dei *cron job* si occupano di generare un po' di valori per volta e di verificare che siano unici, così quando ce ne sarà bisogno l'applicazione potrà andarseli a pescare tra quelli già generati e verificati anziché generarli e verificarli al volo. Inoltre, ogni tot di tempo si possono spostare gli ultimi i record creati in un'altra tabella, in modo da potervi applicare un indice e renderli più rapidamente consultabili.

Come generare UUID

MySQL ha un comando dedicato, ma genera un poco efficiente e poco sicuro UUID1. Ecco delle alternative migliori e già pronte all'uso!

PHP: https://github.com/ramsey/uuid
JS: https://github.com/broofa/node-uuid
WEB SERVICE: https://www.uuidgenerator.net/api

Conclusioni

Gli UUID danno il meglio di sé nei sistemi distribuiti, dove non esiste un generatore di ID univoci centralizzato come può esserlo un campo AUTO_INCREMENT. Nonostante l'esigua possibilità di ID duplicati (comunque gestibile), se ne possono apprezzare molte qualità anche nei lavori di sviluppo web più comuni. Le più interessanti riguardano l'anonimato: ad esempio, si può tracciare un utente usando un imperscrutabile UUID4 anziché un ormai decifrabile MD5 dell'indirizzo email, aumentando così la privacy sua e nostra.