

# KAPITOLA 4

## Optimalizace výkonu dotazu

V předchozích kapitolách jsme vám vysvětlili, jak se má optimalizovat schéma, což je jedna z nutných podmínek vysokého výkonu. Nestačí ale pracovat pouze se schématem – musíte také dobře navrhovat dotazy. Pokud dotazy vaší aplikace nebudou dobře navrženy, ani optimálně navržené schéma s nimi mnoho nesvede – výkon rozhodně nebude ideální.

Optimalizace dotazu, optimalizace indexu a optimalizace schématu fungují ruku v ruce. Jak budete postupně získávat stále více zkušeností s psaním dotazů v MySQL, budete stále lépe vědět, jak navrhovat schémata, která budou podporovat efektivní dotazy. A obdobně – to, co se dozvíte o optimálním návrhu schématu, bude ovlivňovat, jaký druh dotazů napíšete. Protože se jedná o proces, který nějaký čas trvá, doporučujeme vám, abyste se průběžně vraceli k této a předchozí kapitole.

Tuto kapitolu začínáme všeobecnými úvahami o návrhu dotazu, což jsou první věci, nad kterými byste se měli zamyslet, pokud nějaký dotaz nemá příliš dobrý výkon. Poté se ponoříme hodně hluboko do optimalizace dotazů a do "vnitřních záležitostí" serveru. Předvedeme vám, jak zjistit, jakým způsobem MySQL vykonává konkrétní dotaz. Dozvíte se, jak změnit vykonávací plán dotazu. A nakonec se podíváme na několik míst, kde MySQL neoptimalizuje dotazy dobře a prozkoumáme vzory pro optimalizaci dotazu, které napomáhají MySQL vykonávat dotazy efektivněji.

Naším cílem je, abyste důkladně porozuměli tomu, jak MySQL skutečně vykonává dotazy, takže byste měli být schopni posoudit, co je efektivní a co je neefektivní, maximálně těžit ze síly MySQL a vyhýbat se jeho slabinám.

### Základní příčiny pomalého dotazu: optimalizujte přístup k datům

Nejzákladnější příčinou nedobrého výkonu dotazu bývá to, že pracuje s příliš mnoha daty. Existují dotazy, které musejí projít obrovské množství dat. S nimi toho mnoho nenaděláme. Jedná se ovšem o dost neobvyklý případ, protože většina špatných dotazů se dá změnit tak, aby přistupovaly k men-

šímu objemu dat. V praxi jsme zjistili, že dotaz s nevalným výkonem je vhodné analyzovat ve dvou krocích.

1. Zjistit, zda aplikace nezískává víc dat, než je nezbytně nutné. Ačkoli to obvykle znamená, že přistupuje k příliš mnoha řádkům, může přistupovat i k příliš mnoha sloupcům.
2. Zjistit, zda server MySQL neanalyzuje více řádků, než je potřeba.

## Nepožadujete z databáze data, která nepotřebujete?

Některé dotazy požadují víc dat, než potřebují, takže většinu z nich poté bez užitku odhodí. To znamená více práce pro server MySQL, vyšší režii na síti<sup>1</sup>, a samozřejmě také vyšší spotřebu paměti a zdrojů CPU na aplikačním serveru.

Následuje přehled několika typických chyb:

- **Získává se víc řádků, než je nezbytně nutné.** Jedním z běžných omylů je předpoklad, že MySQL poskytuje výsledky na přání, nikoliv že vypočítává a vrací úplnou výslednou sadu. Často to vidíme v aplikacích, které byly navrženy lidmi, již mají zkušenosti s jinými databázovými systémy. Tito vývojáři používají techniky, jako je například tato – vydají příkaz `SELECT`, který vrátí spoustu řádků, pak načtou prvních `N` řádků a uzavrou výslednou sadu (například načtou z databáze 100 nejnovějších článků pro nějaký zpravodajský web, ačkoliv na úvodní stránce se zobrazuje pouze 10 z nich). Domnívají se, že MySQL jim poskytne oněch 10 řádků a pak pozastaví vykonávání dotazu. MySQL ovšem ve skutečnosti vygeneruje kompletní výslednou sadu. Klientská knihovna pak načte všechna data a většinu z nich zahodí. Zde je nejlepším řešením přidat do dotazu klauzuli `LIMIT`.
- **Získávají se všechny sloupce z několika spojených tabulek.** Chcete-li získat všechny herce účinkující v *Academy Dinosaur*, napište dotaz v tomto stylu:

```
mysql> SELECT * FROM sakila.actor
-> INNER JOIN sakila.film_actor USING(actor_id)
-> INNER JOIN sakila.film USING(film_id)
-> WHERE sakila.film.title = 'Academy Dinosaur';
```

Takový dotaz totiž vrátí všechny sloupce ze všech tří tabulek. Napište dotaz raději takto:

```
mysql> SELECT sakila.actor.* FROM sakila.actor...;
```

- **Získávají se všechny sloupce.** Vždy byste se měli zarazit, pokud uvidíte příkaz `SELECT *`. Opravdu potřebujete úplně všechny sloupce? Patrně ne. Když se získávají všechny sloupce, mohou se tím potlačit různé optimalizace, jako jsou pokrývající indexy. O zvyšujících se nárocích na I/O, paměť a režii CPU pro server snad nemusíme ani hovořit.

---

1 Režie na síti je nejhorší, pokud je aplikace umístěna na jiném hostiteli než je server. Nicméně si uvědomte, že ani tehdy, když se aplikace a MySQL nachází na stejném serveru, není přenos dat mezi nimi zadarmo.

Databázoví administrátoři kvůli těmto skutečnostem často zakazují použití příkazu `SELECT *`. Také i z toho důvodu, aby snížili riziko, že budou vznikat různé potíže v důsledku toho, že někdo změnil seznam sloupců tabulky.

Samozřejmě, když požadujete více dat, než momentálně potřebujete, není to vždycky na závadu. V mnoha případech, které jsme prošetřovali, nám lidé říkali, že jejich "marnotratný" přístup zjednodušuje vývoj, a že vývojářům umožňuje použít stejný úsek kódu na více místech. To jsou rozumné argumenty, pokud ovšem víte, jak se to odrazí na nákladech z hlediska výkonu. Získávat více dat než je nutné může být také vhodné v situacích, kdy vaše aplikace používá nějaký druh cachování, nebo pokud z toho těžíte jiným způsobem. Získávání a následné cachování úplných objektů může být výhodné například tehdy, kdy spouštíte velké množství samostatných dotazů, které získávají pouze části těchto objektů.

## Nezkoumá MySQL příliš mnoho dat?

Pokud jste si jisti, že vaše dotazy získávají pouze ta data, která opravdu potřebujete, hledejte dotazy, jež zkoumají příliš mnoho dat, zatímco generují výsledky. V MySQL jsou nejjednoduššími metrikami nákladnosti dotazů:

- Doba vykonávání.
- Počet zkoumaných řádků.
- Počet vrácených řádků.

Ačkoliv žádná z těchto metrik není perfektní mírou nákladnosti dotazu, alespoň orientačně odrážejí, k jakému množství dat musí MySQL interně přistoupit, aby vykonal dotaz. Můžete si je rovněž přeložit na "jak rychle dotaz poběží". Všechny tři metriky se zaznamenávají do logu pomalých dotazů, takže dívat se do tohoto logu je jeden z nejlepších způsobů, jak najít dotazy, které zkoumají příliš mnoho dat.

### Doba vykonávání

V kapitole 2 jsme diskutovali o tom, že standardní log pomalých dotazů má v MySQL 5.0 a předchozích verzích několik závažných omezení – například v něm postrádáme podporu pro jemnější zaznamenávání do logu. Naštěstí existují záplaty, s nimiž můžete zaznamenávat a měřit pomalé dotazy s rozlišením na mikrosekundy. Ačkoliv tyto záplaty byly zařazeny až do MySQL verze 5.1, není problém je v případě potřeby aplikovat i na starší verze serveru. Nenechte se ale strhnout přílišným nadšením pro dobu vykonávání dotazu. Ačkoliv je to hezká míra, protože je objektivní, není konzistentní při různých úrovních zátěže. Jiné faktory – jako jsou zámky úložného enginu (zámky na tabulky a na řádky tabulek), vysoká souběžnost, nebo použitý hardware – rovněž mohou mít značný dopad na dobu vykonávání dotazů. Tato metrika je užitečná, protože s ní dokážete najít dotazy, které mají největší dopad na rychlost odpovědi aplikace, nebo dotazy, jež nejvíce zatěžují server. Už vám ovšem neřekne, zdali to náhodou není tak, že skutečná doba vykonávání dotazu

jednoduše odpovídá jeho složitosti. (Doba vykonávání může být jak symptomem, tak i současně příčinou problémů, takže nemusí být vždy zřejmé, co vlastně indikuje.)

## Počet zkoumaných řádků a počet vrácených řádků

Když analyzujete dotazy, vyplatí se popřemýšlet o tom, kolik řádků musí dotaz prozkoumat, protože z toho se dá usoudit, jak efektivně dotaz hledá data, která potřebujete.

Podobně jako doba vykonávání dotazu, ani toto není perfektní metrika pro zjišťování špatných dotazů. Všechny přístupy k řádkům nejsou rovnocenné. Ke kratším řádkům se přistupuje rychleji a řádky z paměti se získávají mnohem rychleji, než když se data čtou z disku.

Ideální by bylo, kdyby počet zkoumaných řádků byl roven počtu vrácených řádků, ale v praxi je tohle uskutečnitelné jen výjimečně. Pokud například sestrojíte řádky pomocí dotazů, v nichž se spojují tabulky, musíte přistoupit k několika řádkům, abyste vygenerovali jeden řádek výsledné sady. Ačkoliv poměr počtu zkoumaných řádků k počtu vrácených řádků je obvykle malý, řekněme od 1:1 k 10:1, v některých situacích může být i řádově větší.

## Počet zkoumaných řádků a přístupové typy

Když přemýšlíte o nákladnosti dotazu, posuďte, jaké jsou náklady na nalezení jediného řádku z tabulky. MySQL obsahuje několik přístupových metod pro nalezení a vrácení řádku. Některé metody požadují, aby se prozkoumalo mnoho řádků, v jiných se dají vygenerovat výsledky, aniž by se musel prozkoumat byť jen jediný řádek.

Přístupové metody se uvádějí ve sloupci type výstupu příkazu EXPLAIN. Přístupové typy mohou být různé – od kompletního průchodu celou tabulkou přes průchod indexy tabulky, průchod nějakým rozsahem, vyhledávání podle hodnot jedinečného indexu, až ke konstantám. Každý ze zde uvedených přístupových typů je rychlejší než ten, který je v seznamu před ním, protože čte méně dat. Ačkoliv se přístupové typy nemusíte učit nazpaměť, měli byste mít alespoň všeobecné povědomí o tom, co znamená průchod celou tabulkou, průchod jedním indexem, přístup přes rozsah a přístup přes jedinou hodnotu.

Pokud se vám zdá, že použitý přístupový typ není dobrý, nejlepší řešení obvykle spočívá v přidání vhodného indexu. Protože indexy jsme důkladně probírali v předchozí kapitole, předpokládáme, že už víte, proč jsou tak důležité pro optimalizaci dotazu. Indexy umožňují MySQL nacházet řádky s mnohem efektivnějším přístupovým typem, kdy se zkoumá mnohem méně dat.

Podívejte se například na jednoduchý dotaz obracející se na ukázkovou databázi Sakila:

```
mysql> SELECT * FROM sakila.film_actor WHERE film_id = 1;
```

Tento dotaz vrátí 10 řádků a příkaz EXPLAIN nám ukáže, že MySQL pro vykonání dotazu použije přístupový typ ref na index idx\_fk\_film\_id:

```
mysql> EXPLAIN SELECT * FROM sakila.film_actor WHERE film_id = 1\G
***** 1. row *****
```

```

        id: 1
select_type: SIMPLE
  table: film_actor
    type: ref
possible_keys: idx_fk_film_id
      key: idx_fk_film_id
    key_len: 2
      ref: const
    rows: 10
Extra:

```

EXPLAIN ukazuje, že MySQL odhadl, že bude muset přistoupit pouze k 10 řádkům. Řečeno jinými slovy – optimalizátor dotazu věděl, že zvolený přístupový typ bude schopen vykonat dotaz efektivně. Co by se ovšem stalo, kdybychom neměli pro dotaz příhodný index? MySQL by musel použít méně optimální přístupový typ, což uvidíte, když odstraníte index a spustíte dotaz znovu:

```

mysql> ALTER TABLE sakila.film_actor DROP FOREIGN KEY fk_film_actor_film;
mysql> ALTER TABLE sakila.film_actor DROP KEY idx_fk_film_id;
mysql> EXPLAIN SELECT * FROM sakila.film_actor WHERE film_id = 1\G
***** 1. row *****

        id: 1
select_type: SIMPLE
  table: film_actor
    type: ALL
possible_keys: NULL
      key: NULL
    key_len: NULL
      ref: NULL
    rows: 5073
Extra: Using where

```

Jak se dalo očekávat, přístupový typ se změnil na průchod celou tabulkou (ALL), přičemž MySQL nově odhadl, že bude muset prozkoumat celkem 5 073 řádků, aby splnil požadavky dotazu. Text "Using where" ve sloupci Extra sděluje, že server MySQL bude prostřednictvím klauzule WHERE odhazovat řádky až poté, co je načte úložný engine.

MySQL může aplikovat klauzuli WHERE jedním ze tří způsobů, od nejlepšího k nejhoršímu:

- Aplikuje podmínky na operaci indexového vyhledávání podle hodnot klíče, aby eliminoval nevyhovující řádky. Tohle se děje na vrstvě úložného enginu.
- Použije pokrývající index, aby zamezil přístupu k řádkům a odfiltroval nevyhovující řádky po získání jednotlivých výsledků z indexu (ve sloupci Extra bude uveden text "Using index"). Ačkoliv tohle se děje na vrstvě serveru, nevyžaduje to číst řádky z tabulky.

- Získá řádky z tabulky a poté odfiltruje nevyhovující řádky (ve sloupci `Extra` bude uveden text "Using where"). Tohle se děje na vrstvě serveru a vyžaduje, aby server načetl řádky z tabulky předtím, než je bude schopen odfiltrovat.

Výše uvedený příklad ilustruje, jak je důležité mít dobré indexy. Dobré indexy pomáhají dotazům získat dobrý přístupový typ, takže jim stačí prozkoumat pouze ty řádky, které potřebují. Když ovšem přidáte nějaký index, neznamená to vždy, že bude stejný počet řádků, k nimž bude MySQL přistupovat i které bude vracet. Podívejte se na dotaz, ve kterém se volá agregační funkce `COUNT()`<sup>1</sup>:

```
mysql> SELECT actor_id, COUNT(*) FROM sakila.film_actor GROUP BY actor_id;
```

Ačkoliv tento dotaz vrátil pouze 200 řádků, potřeboval načíst řádově tisíce řádků, aby mohl vybudovat výslednou sadu. V dotazech tohoto druhu není možné prostřednictvím nějakého indexu zredukovat počet zkoumaných řádků.

MySQL vám bohužel nesdělí kolik z řádků, k nimž přistoupil, bylo potřeba pro vybudování výsledné sady – sdělí pouze celkový počet řádků, k nimž přistupoval. Mnohé z těchto řádků lze eliminovat klauzulí `WHERE`, protože ničím nepřispívají do výsledné sady. Poté, co jsme v předchozím příkladu odstranili index z tabulky `sakila.film_actor`, dotaz postupně prošel všechny záznamy tabulky, ale klauzule `WHERE` téměř všechny odstranila, zbylo jich pouze 10. Pouze těchto zbylých 10 řádků se podílelo na budování výsledné sady. Pokud chcete porozumět tomu, ke kolika řádkům server přistoupil, a kolik z nich skutečně použil, musíte se nad dotazem zamyslet a zkusit to vydedukovat. Pokud zjistíte, že pro vyprodukování relativně malého počtu řádků výsledné sady bylo nutné prozkoumat obrovský počet řádků, můžete vyzkoušet tyto sofistikovanější opravy:

- Použijte pokrývající indexy, které ukládají data tak, aby úložný engine nemusel získávat kompletní řádky. (Detailně jsme je probrali v předchozí kapitole.)
- Změňte schéma. Jako příklad mohou posloužit souhrnné tabulky. (Detailně jsme je probrali v předchozí kapitole.)
- Přepište komplikovaný dotaz tak, aby ho optimalizátor MySQL mohl vykonat optimálnějšíм způsobem. (Tohle téma probereme později v této kapitole.)

## Restrukturalizace dotazu

Když optimalizujete problematické dotazy, vaším cílem by mělo být nalézt alternativní způsoby pro získání těch výsledků, které potřebujete – což nutně neznamená, že se musí jednat o stejnou výslednou sadu získanou z MySQL. Někdy získáte lepší výkon tím, že transformujete dotazy do nějaké jiné ekvivalentní formy. Měli byste také pouvažovat nad tím, zdali by nebylo lepší dotazy přepsat, aby vám poskytovaly odlišné výsledky, pokud to bude mít příznivý dopad na efektivitu. Je možné, že dosáhnete stejného cíle, když místo změn v dotazech změníte kód samotné aplikace. V tom-

---

<sup>1</sup> Další informace k tomuto tématu viz "Optimalizace dotazu s `COUNT()`" na straně 215.