38. Indexování a optimalizace dotazů v relačních databázích, datové struktury, jejich výhody a nevýhody

Využití databázových indexů

Databázové indexy slouží ke zrychlení přístupu k datům a měly by se používat u všech sloupců, podle kterých se vyhledává, třídí nebo podle kterých se spojují tabulky.

Při ukládání dat do tabulek nejsou záznamy obvykle nijak tříděny a ukládají se většinou za sebe tak, jak byly postupně vloženy. V momentě, kdy chceme data z tabulky později vybrat podle nějakého kritéria, je nutné projít všechny záznamy a vybrat z nich ty, které kritériu vyhovují. K tomu, aby při výběru několika záznamů nebylo potřeba procházet všechny ostatní, slouží právě indexy, ve kterých jsou data organizována tak, aby bylo možné rychle vybrat pouze relevantní záznamy.

Indexy se vytvářejí nad jedním nebo několika sloupci tabulky, každá tabulka může mít indexů několik. Index definovaný nad sloupcem tabulky umožňuje rychlý přístup k záznamům podle hodnot tohoto sloupce.

Organizace dat v indexu umožňuje nejen přímé vybrání záznamů s určitou hodnotou, ale samozřejmě také záznamů v intervalu hodnot. Kromě toho jsou prvky v indexu provázané podle svého pořadí při řazení (ať už se jedná o číselné, nebo řetězcové sloupce), takže indexy umožňují také rychlé seřazení tabulky podle sloupců, nad kterými je index definován. Tím pádem umožňují i rychlé vybrání minima a maxima. Informaci o počtu hodnot a počtu různých hodnot (SQL funkce COUNT a COUNT DISTINCT) databáze obvykle uchovávají nezávisle na indexu ve statistikách tabulky, které používají například také při hledání strategie pro vyhodnocování složitějších dotazů.

Indexy nad řetězcovými sloupci umožňují také rychlejší vyhledávání pomocí operátoru LIKE, avšak pouze v případě, kdy je znám začátek hledaného výrazu – tedy např. X LIKE 'text%' využití indexu dovoluje, X LIKE '%text%' ne.

Použití indexů se často zanedbává a faktem je, že u malých tabulek obsahujících řádově desítky záznamů je jejich význam zanedbatelný. U větších tabulek indexy naopak výkon ovlivňují zásadně. Vzhledem k tomu, že správa indexu stojí určitou režii při každém vkládání záznamu nebo jeho mazání, měli bychom se vytváření indexů vyhnout u tabulek, do kterých se převážně vkládá a jen výjimečně se z nich čte, což jsou např. logy.

Kromě běžných indexů lze definovat také unikátní indexy, které do tabulky nedovolí vložit více záznamů se stejnou hodnotou sloupců, nad kterými je index definován (výjimku tvoří hodnoty NULL). Tato informace může databázovému serveru sloužit také k efektivnějšímu uspořádání dat. Speciálním typem indexu je primární klíč označující sloupce, které jednoznačně identifikují libovolný záznam v tabulce. Definování primárního klíče by mělo být samozřejmostí.

Po vytvoření indexů se o ně již dále nemusíme starat, databázový server sám zajišťuje jejich automatickou aktualizaci a sám rozhoduje o tom, jaké indexy využije při získávání dat. Pokud nás zajímá, jaké indexy server použije, můžeme v MySQL i u několika dalších serverů použít příkaz EXPLAIN.

Druhy indexů

Databázové indexy (či index sekvenční moduly) dělíme do různých druhů podle toho, co

chceme při přístupech k primárním datům příslušné databázové tabulky optimalizovat. Označení druhů indexů se může různit, nejčastěji se používají tyto hlavní druhy:

PRIMARY
Související informace lze nalézt také v článku Primární klíč.
Tento primární index tvoří sloupec (nebo kombinace více sloupců), které obsahují primární klíč (někdy označován také jako hlavní index). Jedná se o zvláštní druh indexu, který se v každé tabulce může vyskytovat nejvýše jednou. Je definován sloupcem (sloupci) tabulky, který svou hodnotou jednoznačně identifikují každý záznam. Ve většině případů dnes je dodržována zvyklost resp. existuje vžitá konvence tento sloupec nazvat ID (odvozeno od slova identifikovat) a jeho datový typ pak stanovit jako typ celočíselný (tedy typ INTEGER či SMALLINT, není-li třeba jinak - není to ale bezpodmínečně nutné). Databázový server musí být v tomto případě navržen tak, že není možné, aby do takto označeného sloupce (k němuž se tento primární index vztahuje) byla vložena duplicitní či multiplicitní hodnota klíče, tedy stejný klíč, který již v tabulce existuje resp. který již byl jednou vložen (takový pokus končí chybovým hlášením a zápisem do chybového logovacího souboru či do logovací tabulky). Klíč je tedy v tabulce unikátní, jedná se de facto o zvláštní případ druhu klíče UNIQUE.

UNIQUE
Tento unikátní index je tvořen ze sloupců obsahujících unikátní klíč, jedná se o speciální druh indexu, který je významově podobný předchozímu typu PRIMARY co do jednoznačnosti záznamu podle unikátní hodnoty klíče (typ PRIMARY je pouze zvláštní případ typu UNIQUE, jedná se vlastně o podtyp) v databázové tabulce (ostatně, jak naznačuje i sám jeho název) a v praktickém dopadu, který to pak na práci s příslušnou databází má. Na rozdíl od předchozího typu však nemusí být unikátní index jediný, ale může být definováno více. Například kromě ID záznamu o osobě můžeme požadovat i unikátnost sloupce s loginovým jménem osoby. Jednotlivý index může být i složen z více sloupců. Například v tabulce o biologických druzích budeme potřebovat unikátnost kombinace druhového a rodového jména. Potom opět nelze vložit záznam s hodnotou klíče, který by již v této kombinaci někde v tabulce existoval respektive byl již dříve do tabulky vložen (situace opět vede k chybovému hlášení vypisovanému na standardní výstup a zápisu do chybového logovacího souboru či tabulky).

INDEX
Index též zvaný **SECONDARY** je tvořen pro sloupce, které obsahují sekundární klíč čili druhotný klíč (někdy bývá též označován jako vedlejší index). Definicí jednoho či více indexů tohoto typu v tabulce zajišťujeme optimalizaci vyhledávání podle dalších sloupců, mimo primární nebo unikátní indexy. Databázový server vytvoří a nadále udržuje vnitřní konstrukci odkazů na řádky tabulky, jež poskytuje uspořádání podle příslušných hodnot ve sloupci, k němuž je index logicky vázán (podle hodnot sekundárního klíče). Udržování takto uspořádané konstrukce urychluje vyhledávání záznamů v databázi (je možno použít některé matematické interpolační numerické metody), logické či fyzické řazení záznamů jakož i jiné další datové operace s tabulkou, jež se mají provést na podmnožině záznamů z tabulky vymezené podmínkou položenou na hodnoty v sekundárním klíči. Na rozdíl od předchozích indexů PRIMARY a UNIQUE lze do tabulky vkládat záznamy, které nejsou v sekundárním indexu unikátní. U některých databázových systémů se může jednat i o sloupce tzv. fiktivní, tedy sloupce odvozené respektive vypočtené z hodnot sloupců fyzických resp. uložených.

FULLTEXT
Vytvořením indexu tohoto typu se databázový server bude snažit optimalizovat full-textové

vyhledávání v daném sloupci u dané tabulky. To, jakým způsobem to udělá, záleží na databázovém serveru samotném, průvodním jevem může být to, že na disku nebo v operační paměti bude udržovat například statistiku slov, které byly v tomto sloupci a tabulce zadány, nebo jiné pomocné hašovací funkce nebo vyhledávací tabulky.

Optimalizace dotazů v relačních databázích
Jedním z hlavních důvodů provádění optimalizace v databázových prostředcích je minimalizace nákladů.
Jedná se především o minimalizaci nákladů na:

- zdrojový čas
- kapacitu paměti (prostor)
- programátorskou práci

na ladění výkonu podílí

- Návrhář databáze (designer)
- Vývojář (developer)
- Správce databáze (DBA)
- Uživatel

Obecná pravidla pro psaní SQL dotazů

1. Vyjmenovat sloupce
   V SELECT dotazech nepoužívat v seznamu sloupců hvězdičku (*)
   Ve většině případů nepracujeme se všemi sloupci výsledku
2. Používat co nejméně klauzuli LIKE
   Nedoporučuje se používat pro vyhledávání ve velkých textových polích (můžou obsahovat až několik GB textu)
3. Zamyslet se, zda nejde vyhledávání provést jinou metodou
4. Používat co nejméně klauzule IN, NOT IN
   Vhodnější je použití příkazů WHERE a WHERE NOT EXISTS
5. Používat klauzule typu LIMIT
6. Na začátek dávat obecnější podmínky
   V klauzuli WHERE dávat na začátek podmínky, po kterých vypadne ze seznamu nejvíce záznamů
   Databáze nejprve vyhledá záznamy, vyhovující první podmínce, z nich pak vybírá záznamy vyhovující druhé podmínce

   Snažíme se, aby systém vyřadil na začátku co nejvíce řádků; ty se pak již při další podmínce nezkoumají…
7. Výběr vhodného pořadí spojení
   vyhnout se plnému prohledávání tabulky (pokud možno využít index)
   efektivně vybírat takové indexy, které načtou z tabulky co nejméně záznamů

   vybrat takové pořadí spojení ze všech možných pořadí, aby bylo spojeno co nejméně položek
8. Používat hinty
   Hint = podnět, kterým optimalizátoru určíme, jaký má použít plán vykonávání dotazu

   Hinty se aplikují na blok dotazu, ve kterém se vyskytují.
9. Nastavit indexy

Procházení tabulky pomocí indexu trvá mnohem kratší dobu než procházení tabulky bez jeho použití.

Změna indexů se zdá být nejlepším řešením pro optimalizaci, jelikož má větší sílu než změna SQL dotazu či změna dat.

Samotné vytvoření indexů však nelze brát v úvahu jako univerzální řešení problému.

Datové struktury, jejich výhody a nevýhody

Hierarchická databáze
Hierarchická databáze (také hierarchická nebo stromová datová struktura) je datový model, ve kterém jsou data uspořádána ve stromové struktuře. Je to první z datových modelů který byl v minulosti hojně využíván v praxi. Její vznik se datuje do 70. let minulého století a nejznámějším hierarchickým systémem řízení báze dat byl IMS od společnosti IBM. Hierarchická koncepce ovšem vykazuje jisté nedostatky při modelování reality, a proto bylo od jejího používání upuštěno, kdy byla prvně překonána koncepcí síťovou a později koncepcí relační.

Základní konstruktory

Věta - obdoba relace v relačním datovém modelu. Každá věta je definována svými atributy. Oproti relačnímu modelu neexistuje omezení při tvorbě atributu - atributy nemusí být atomické (není zde zaveden pojem normalizace datové základny), atributy mohou být i vícerozměrnými strukturami jako jsou například pole apod.
Vlastnicko-členský vztah (také vztah rodič - dítě) - vztah kardinality 1:N mezi dvěma větami; věta na straně jedna se nazývá vlastník, věta na straně N se nazývá člen.

Vlastnosti

hierarchické schéma má jeden kořen, který není člen v žádném vztahu
každá věta, kromě kořene, je členem právě v jenom vztahu
každá věta může být vlastníkem 1 až n počtu vět
věta, která není vlastníkem v žádném vztahu, se nazývá list

Z výše uvedených vlastností vyplývá, že hierarchický model je schopen bezproblémově modelovat pouze vztahy kardinality 1:N. Vztahy kardinality M:N je možné modelovat za využití dvou vztahů 1:N za pomocí tzv. směrníkových vět, jedná se o speciální případ "virtuální věty", která obsahuje klíče z obou vztahů 1:N a binární ukazatele na umístění vztahů v hierarchickém schématu.

Síťový model dat

Síťový model byl standardizován (1972) výborem Database Task Group (DBTG), který vznikl v rámci konference o jazycích datových systémů (Conference on Data Systems Languages - CODASYL). Příkladem síťového modelu je systém IDMS (Integrated Database Management System)

- Síťový model odstraňuje omezení kladená v hierarchickém modelu na vazby mezi prvky.
- Prvek může být propojen vazbou s více prvky z vyšší úrovně i s prvky na stejné úrovni.
- Vztahy jsou označovány jako C-množiny (Sets)
- Sety propojují záznamy různého či stejného typu, přičemž spojení může být realizováno na jeden nebo více záznamů.
- Správné nastavení vazeb mezi prvky je důležité pro efektivní využívání databáze
- Případná dodatečná modifikace vazeb je časově náročná

Relační model

Relační model je nejrozšířenějším způsobem uložení dat v databázi. Jedná se způsob uložení v logickém smyslu.

Popis

V roce 1969 přišel doktor E. F. Codd (A relational data model for large shared data banks) se svou představou o databázi založené na matematickém aparátu relačních množin a predikátové logiky. Databázová relace se od matematické poněkud liší. Má zavedený pomocný aparát nazvaný schéma relace. Schéma relace říká, jaký je název relace, kolik má sloupců a jaké jsou jejich názvy a domény (doména je množina přípustných hodnot pro daný sloupec). V databázích je schématem relace definice struktury tabulky. Ovšem relací nemusí být pouze tabulka, nýbrž jakákoliv struktura dělená do řádků a sloupců. Relací je například i výsledek jakéhokoliv dotazu, a podle toho s ním je možno i dále pracovat. Velmi rozšířeným omylem je, že relační model se nazývá podle vztahů mezi daty, pojem "relační" však vychází z relací matematických, na kterých je celý model založen.

Implementace

Relační databázový model sdružuje data do tzv. relací (tabulek), které obsahují n-tice (řádky). Tabulky (relace) tvoří základ relační databáze. Tabulka je struktura záznamů s pevně stanovenými položkami (sloupci - atributy). Každý sloupec má definován jednoznačný název, typ a rozsah, neboli doménu. Záznam se stává n-ticí (řádkem) tabulky. Pokud jsou v různých tabulkách sloupce stejného typu, pak tyto sloupce mohou vytvářet vazby mezi jednotlivými tabulkami. Tabulky se poté naplňují vlastním obsahem - konkrétními daty.
Kolekce více tabulek, jejich funkčních vztahů, indexů a dalších součástí tvoří relační databázi.
Relační model přináší celou řadu výhod, zejména mnohdy přirozenou reprezentaci zpracovávaných dat, možnost snadného definování a zpracování vazeb apod.
Relační model klade velký důraz na zachování integrity dat. Zavádí pojmy referenční integrita,

cizí klíč, primární klíč, normální tvar apod.
S relačními databázemi je úzce spojen pojem SQL (Structured Query Language), neboli strukturovaný dotazovací jazyk. Jeho základní model je obecně použitelný pro většinu relačních databází. Od svého vzniku prošel několika revizemi a poskytovatelé databázových produktů jej obohatili o různá lokální rozšíření. Tato rozšíření ale nejsou vzájemně kompatibilní.