



ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

Databázové systémy 1

KIV/DB1

Celá kniha je exportem z wikipedie, ale spolehlivě pokrývá rozsah znalostí pro tento předmět.

Obsah

Články

System řízení báze dat	1
Databáze	2
Relační databáze	5
Relační model	9
Trigger (databáze)	10

Integritní omezení **12**

Referenční integrita	12
Databázová integrita	13

Klíče **14**

Kandidátní klíč	14
Primární klíč	14
Cizí klíč	15
Index (databáze)	17

Normalizace **21**

Normalizace databáze	21
Třetí normální forma	22

SQL **25**

SQL	25
Příkazy jazyka SQL	26
MySQL	28
SELECT	32
Agregační funkce	34

Reference

Zdroje článků a příspěvatelé	37
Zdroje obrázků, licence a příspěvatelé	38

Licence článků

Licence	39
---------	----

System řízení báze dat

Systém řízení báze dat (zkracováno na **SŘBD** či **DBMS** podle anglického *database management system*) je softwarové vybavení, které zajišťuje práci s databází, tzn. tvoří rozhraní mezi aplikačními programy a uloženými daty. Občas se pojem zaměňuje s pojmem databázový systém. Databázový systém však je SŘBD dohromady s bází dat.

Schopnosti

Aby mohl být nějaký programový systém označený za SŘBD, musí být jednak schopen efektivně pracovat s velkým množstvím dat, ale také musí být schopný řídit (vkládat, modifikovat, mazat) a definovat strukturu těchto perzistentních dat (čímž se liší od prostého souborového systému).

V současnosti používané databázové systémy mají i mnoho dalších charakteristických vlastností:

- podporu pro definici datových modelů (například relační, logický, objektový)
- správa klíčů: vlastní (interně implementované) indexování, dodržování unikátnosti hodnot ve sloupcích, nad kterými je definován unikátní nebo primární klíč; implementace fulltextového vyhledávání pro fulltextové klíče; implementace cizích klíčů
- využití některého jazyka vyšší úrovně pro manipulaci a definici dat (např. SQL, QBE, datalog, *Common English Query*) a vyřešení komunikačního kanálu mezi uživatelem či skriptem a SŘBD v tomto jazyku,
- autentizaci uživatelů a jejich autorizaci k operacím nad daty (u každého uživatele může být definováno, jaký typ příkazů je oprávněn spouštět)
- správu transakcí, atomicitu jednotlivých příkazů
- robustnost a zotavitelnost po chybách bez ztráty dat
- uložené procedury
- trigger
- integritu dat; například nepovolením vložení duplicitního řádku s unikátním klíčem nebo řádku s hodnotami NULL u sloupců, které NULL být nesmějí
- kanály pro hlášení zpráv po úspěšně vykonaných dotazech, chybových hláškách, varování
- pokročilé funkce jako např. *Common Table Expressions*, zpožděné zápisy, a jiné
- profilování, statistické informace o běhu dotazů, procesů, přístupu uživatelů atd.

Seznam systémů řízení báze dat

Následující seznam obsahuje příklady některých systémů řízení báze dat.

- Oracle
 - DB2
 - Sybase Adaptive Server Enterprise
 - FileMaker
 - Firebird
 - Ingres
 - Informix
 - MariaDB
 - Microsoft Access
 - Microsoft SQL Server
 - Microsoft Visual FoxPro
 - MySQL
 - PostgreSQL
-

- Progress
- SQLite
- Teradata
- CSQL
- OpenLink Virtuoso

Databáze

Databáze (neboli **datová základna**) je určitá uspořádaná množina informací (dat), uložená na paměťovém médiu. V širším smyslu jsou součástí databáze i softwarové prostředky, které umožňují manipulaci s uloženými daty a přístup k nim. Tento software se v české odborné literatuře nazývá systém řízení báze dat (SŘBD). Běžně se označením *databáze* – v závislosti na kontextu – myslí jak uložená data, tak i software (SŘBD).

Historie

Předchůdcem databází byly papírové kartotéky. Umožňovaly uspořádávání dat podle různých kritérií a zařídování nových položek. Veškeré operace s nimi prováděl přímo člověk. Správa takových kartoték byla v mnohém podobná správě dnešních databází.

Dalším krokem bylo převedení zpracování dat na stroje. Za první velké strojové zpracování dat lze asi považovat sčítání lidu ve Spojených státech v roce 1890. Paměťovým médiem byl děrný štítek a zpracování sebraných informací probíhalo na elektromechanických strojích. Elektromechanické stroje se využívaly pro účely zpracování dat další půlstoletí.

Velkým impulsem pro další rozvoj databází byl překotný vývoj počítačů v padesátých letech 20. století. Ukázalo se, že původně univerzální používání strojového kódu procesorů je (nejen) pro databázové úlohy neefektivní, a proto se objevil požadavek na vyšší jazyk pro zpracování dat.

V roce 1959 se konala konference zástupců firem, uživatelů a amerického ministerstva obrany, jejímž závěrem byl požadavek na univerzální databázový jazyk. Výsledkem byla o rok později na konferenci CODASYL publikovaná první verze jazyka COBOL, který byl po mnoho dalších let nejrozšířenějším jazykem pro hromadné zpracování dat.

V roce 1965 na konferenci CODASYL byl vytvořen výbor *Database Task Group (DBTG)*, který měl za úkol vytvořit koncepci databázových systémů. Začaly vznikat první **síťové** SŘBD na sálových počítačích. Jedním z prvních průkopníků databází byl Charles Bachman.

V roce 1971 vydal výbor zprávu *The DBTG April 1971 Report*, kde se objevily pojmy jako **schéma databáze**, **jazyk pro definici schématu**, **subschéma** a podobně. Byla zde popsána celá architektura **síťového** databázového systému.

Ve stejné době byly vyvíjeny i **hierarchické** databáze. Jedním z prvních SŘBD byl **IMS**, který byl vyvinut firmou IBM pro program letu na Měsíc Program Apollo. Systém IMS patří stále k nejrozšířenějším na sálových počítačích.

V roce 1970 začínají zveřejněním článku E. F. Codda první **relační databáze**, které pohlížejí na data jako na tabulky. Kolem roku 1974 se vyvíjí první verze dotazovacího jazyka SQL. Vývoj této technologie po 10 letech přinesl výkonově použitelné systémy, srovnatelné se síťovými a hierarchickými databázemi.

V 90. letech 20. století se začínaly objevovat první **objektově** orientované databáze, jejichž filozofie byla přebírána z objektově orientovaných jazyků. Tyto databáze měly podle předpokladů vytlačit relační systémy. Původní předpoklady se však nenaplnily a vznikla kompromisní **objektově-relační** technologie.

Databázové modely

Z hlediska způsobu ukládání dat a vazeb mezi nimi můžeme rozdělit databáze do základních typů:

- Hierarchická databáze
- Síťová databáze
- Relační databáze
- Objektová databáze
- Objektově relační databáze
- Dokumentově orientovaná databáze

Databázové objekty

Pojem „databáze“ je často zjednodušován na to, co je ve skutečnosti databázový systém (databázový stroj) nebo též systém řízení báze dat. Ten neobsahuje pouze **tabulky** – ty jsou jedny z mnoha tzv. databázových objektů (někdy též databázových entit). Pokročilejší databázové systémy dále obsahují:

- **pohledy** neboli **views** – SQL příkazy, pojmenované a uložené v databázovém systému. Lze z nich vybírat (aplikovat na ně příkaz SELECT) jako na ostatní tabulky.
- **indexy** neboli **klíče** pro každou tabulku. Klíče jsou definovány nad jednotlivými sloupci tabulek (jeden klíč jich může zahrnovat i více) a jejich funkce je vést si v tabulkách rychlé LUT (*look-up tables* – „pořadníky“) na sloupce, nad nimiž byly definovány, vyloučit duplicitu v záznamech nebo zajišťovat fulltextové vyhledávání.
- **Trigger** neboli **spouště** – mechanismus mezi řádky dvou tabulek, který se v databázovém systému dá definovat jako jeden z několika úkonů, který se vyvolá po změně nebo smazání rodičovské tabulky.
- **uživatelem definované procedury a funkce** – některé databázové stroje podporují ukládání pojmenovaných kusů kódu, které provedou v databázi nad danými tabulkami určitou sekvenci příkazů (procedury) nebo navíc vrátí nějaký výsledek (uživatelské funkce). Mohou mít parametry, které se většinou dělí na vstupní (*IN*), výstupní (*OUT*) a vstupně-výstupní (*INOUT*).
- **události**, též (počeštěně) „eventy“ – de facto procedury, spouštěné v určitý (uživatelem definovaný) datum a čas nebo opakovaně s definovatelnou periodou. Mohou sloužit k údržbě, promazávání dočasných dat či kontrolování referenční integrity.
- **formuláře** – některé databázové systémy jako např. Microsoft Access umožňují uživatelům vytvářet vstupní formuláře pro vizuálně přívětivé zadávání hodnot. Uživatel si může např. nadefinovat rozložení jednotlivých vstupních polí z dané tabulky, popisky atd.
- **sestavy** nebo též **reporty** – podobně jako u formulářů sestavy umožňují uživateli definovat layout s políčky dané tabulky, do kterého se při použití doplní aktuální hodnoty. Používají se pro výstup dat (tisk, prezentaci nebo pouhé zobrazení). Sestavy mohou být např. doplněny o filtry, které vyfiltrují jen kýžené záznamy.
- **uživatelská oprávnění** – u lepších databázových systémů je samozřejmostí nabídnout možnosti, jak oddělit jednotlivé úrovně přístupu k ostatním objektům databáze jejich uživatelům. Možností bývají desítky, s rozlišením na jednotlivé typy příkazů, které ten který uživatel bude nebo nebude mít oprávnění spustit.
- **partitioning** – způsob, jak rozdělit data v tabulce na více pevných disků a tím rozložit zátěž na ni kladenou
- **procesy** – databázové stroje umí podat přehled o procesech, které jejich služeb aktuálně využívají.
- **proměnné nastavení** – typicky desítky proměnných, které lze přenastavovat a tím ovlivňovat chování a výkon databázového stroje jako takového.
- **collation** – MySQL má pokročilé možnosti pro nastavení několika desítek znakových sad a porovnávání, souhrnně nazývané *collation*. Nastavení collation může být provedeno na jednotlivé textové sloupce, celé tabulky i celé databáze (s kaskádovitou dědičností). Collation ovlivňuje i řazení, například hodnota `utf8_czech_ci` zajistí správné řazení podle češtiny (tedy včetně diakritiky a včetně `ch`).
- **vizuální E-R schéma** – (v MySQL `INFORMATION.SCHEMA`). Vizualizace vztahů (relací) na sobě závislých polí (cizích klíčů) mezi tabulkami.

- a další

Databázová integrita


Související informace naleznete také v článku Databázová integrita.

Databázová integrita je takový stav, při němž záznamy v celé databázi vyhovují soustavě určitých definovaných pravidel. Tato pravidla obvykle odpovídají vybraným pravidlům z té části světa, pro kterou databáze slouží. Může se jednat například o pravidla stanovující rozsah uložených hodnot, jejich typ nebo vazby mezi nimi.

Související články

- MySQL
- PostgreSQL
- Oracle
- Microsoft SQL Server
- SQLite
- Firebird
- Integritní omezení
- Normální forma
- Multimediální databáze
- Relační schéma databáze

Externí odkazy

-  Databáze na Wikicitátech
- <http://www.dbsvet.cz> – Databázový svět, informační portál o databázích
- [Seriól Embedded databáze](http://www.root.cz/serialy/embedded-database/) ^[1]

Reference

[1] <http://www.root.cz/serialy/embedded-database/>

Relační databáze



Tento článek potřebuje úpravy.

Můžete Wikipedii pomoci tím, že ho vylepšíte ^[1]. Jak by měly články vypadat, popisují stránky Vzhled a styl, Encyklopedický styl a Odkazy.

Relační databáze je databáze založená na relačním modelu. Často se tímto pojmem označuje nejen databáze samotná, ale i její konkrétní softwarové řešení.

Relační databáze je založena na tabulkách, jejichž řádky obvykle chápeme jako záznamy a eventuálně některé sloupce v nich (tzv. cizí klíče) chápeme tak, že uchovávají informace o relacích mezi jednotlivými záznamy v matematickém slova smyslu.

Termín *relační databáze* definoval Edgar F. Codd v roce 1970.

Historie

V roce 1880 vznikl na objednávku státních úřadů v USA první automat na bázi děrných štítků. U jeho zrodu stál Herman Hollerith, jehož firma při fúzi několika firem dává vzniknout IBM. Ta stojí v popředí i v roce 1951, kdy vznikl první digitální počítač pro komerční využití UNIVAC I. V roce 1960 vznikl předchůdce dnešních databázových jazyků COBOL. V 60. letech založil Charles Bachman spolu s dalšími výzkumníky seskupení Codasyl, které publikovalo základní specifikaci pro programovací jazyky, především pro COBOL. Většina Codasyl kompatibilních databází byla postavena na síťovém modelu, zatímco firma IBM se vydala cestou hierarchického modelu. V roce 1970 přišel Ted Codd s novým návrhem datového modelu, relačním modelem. Dle relační teorie lze pomocí základních operací (sjednocení, kartézský součin, rozdíl, selekce, projekce a spojení) uskutečnit veškeré operace s daty a ostatní operace jsou již jen kombinacemi těchto šesti. Zavádí tedy použití relačního kalkulu a algebry. Databáze mají být nezávislé na fyzickém uložení dat i na použitém jazyce. Pod tlakem událostí se do projektu vkládá i IBM, která přichází s jazykem SQL. První SQL databázi se v roce 1980 stal Oracle pro počítače VAX. Druhá v řadě přichází i firma IBM s produktem DB2. Osmdesátá léta lze považovat za zlatý věk databází^[zdroj?].

Terminologie

Základním konstruktorem relačních databází jsou relace (databázové tabulky), což jsou dvourozměrné struktury tvořené záhlavím a tělem. Jejich sloupce se nazývají atributy, řádky tabulky jsou pak záznamy. Atributy mají určen svůj konkrétní datový typ a doménu, což je množina přípustných hodnot daného atributu. Řádek je řezem přes sloupce tabulky a slouží k vlastnímu uložení dat.

Pojem „relační databáze“ souvisí s teorií množin. Každá konkrétní tabulka totiž realizuje podmnožinu kartézského součinu množin přípustných hodnot všech sloupců – relaci.

Kandidátní klíč

Kandidátní klíč je atribut nebo skupina atributů, které jednoznačně identifikují záznam v relační tabulce. Kandidátní klíč se může stát primárním klíčem; ty, které se primárním klíčem nestanou, jsou označovány jako alternativní klíče. Např. v relaci Zaměstnanec, která má atributy číslo_zaměstnance, rodné_číslo, jméno a příjmení, jsou kandidátními klíči atributy číslo_zaměstnance a rodné_číslo. Pokud primárním klíčem zvolíme číslo_zaměstnance, alternativním klíčem bude rodné_číslo a naopak.

Primární klíč

Primární klíč je jednoznačný identifikátor záznamu, řádku tabulky. Primárním klíčem může být jediný sloupec či kombinace více sloupců tak, aby byla zaručena jeho jednoznačnost. Pole klíče musí obsahovat hodnotu, tzn. nesmí se zde vyskytovat nedefinovaná prázdná hodnota NULL. V praxi se dnes často používají umělé klíče, což jsou číselné či písmenné identifikátory – každý nový záznam dostává identifikátor odlišný od identifikátorů všech předchozích záznamů (požadavek na unikátnost klíče), obvykle se jedná o celočíselné řady a každý nový záznam dostává číslo vždy o jednotku vyšší (zpravidla zcela automatizovaně) než je číslo u posledního vloženého záznamu (číselné označení záznamů s časem stoupá).

Cizí klíč

Dalším důležitým pojmem jsou nevlastní/cizí klíče. Slouží pro vyjádření vztahů, relací, mezi databázovými tabulkami. Jedná se o pole či skupinu polí, která nám umožní identifikovat, které záznamy z různých tabulek spolu navzájem souvisí.

Integrita databáze

Integrita databáze znamená, že data v ní uložená jsou konzistentní vůči definovaným pravidlům. Lze zadávat pouze data, která vyhovují předem definovaným kritériím (např. musí respektovat datový typ nastavený pro daný sloupec tabulky, či další omezení hodnot přípustných pro daný sloupec). K zajištění integrity slouží integritní omezení. Jedná se o nástroje, které zabrání vložení nesprávných dat či ztrátě nebo poškození stávajících záznamů v průběhu práce s databází. Například je možné zajistit mazání dat, která již ztratila svůj význam - například smažeme-li uživatele, odstraní se i zbytek jeho záznamů v ostatních databázových tabulkách.

Druhy integritních omezení

- *Entitní integritní omezení* – povinné integritní omezení, které zajišťuje úplnost primárního klíče tabulky; zamezí uložení dat, která neobsahují všechna pole sdružená do klíče, nebo data, jež by v těchto polích byla stejná jako v nějakém jiném, již zapsaném, řádku tabulky
- *Doménová integritní omezení* – zajišťují dodržování datových typů/domén definovaných u sloupců databázové tabulky
- *Referenční integritní omezení* – zabývají se vztahy dvou tabulek, kde jejich relace je určena vazbou primárního a cizího klíče
- *Aktivní referenční integrita* – definuje činnosti, které databázový systém provede, pokud jsou porušena některá pravidla

Dodržování integritních omezení

V zásadě existují tři způsoby, jak zajistit dodržování integritních omezení.

1. umístění jednoduchých mechanismů pro dodržování integritních omezení na straně databázového serveru. Jedná se o nejlepší způsob z hlediska ochrany dat. Uživateli však obvykle přináší delší odezvu systému a nelze vždy zajistit jejich přenositelnost na jiný databázový systém.
2. umístění ochranných mechanismů na straně klienta. Pro komfort a nezávislost na databázovém systému je nejlepší volbou. Nutnost kontrolních mechanismů pro každou operaci může způsobit chyby u aplikací a v případě většího počtu aplikací je potřeba je opravit na více místech.
3. samostatné programové moduly na straně serveru. V moderních databázových systémech jsou pro tento účel implementovány tzv. triggerů. Jedná se o samostatné procedury, které lze spouštět automatizovaně před a po operacích manipulujících s daty. Tento způsob umožňuje implementaci i složitých integritních omezení. Nevýhody opět přináší provádění na serveru, i velmi omezená možnost přenesení na jiný databázový systém.

Ideálním řešením je kombinace předchozích v závislosti na konkrétních podmínkách.

Kontroly integritních omezení se zpravidla provádějí po každé provedené operaci, což snižuje nároky na server. Není nutno nijak zaznamenávat, které kontroly mají být provedeny později. Složitější integritní omezení však vždy nelze takto ověřit, proto je možné kontrolovat dodržení pravidel až po dokončení celé transakce.

Vztahy mezi tabulkami

Vztahy (relationships) slouží ke svázání dat, která spolu souvisejí a jsou umístěny v různých databázových tabulkách. Každý vztah je charakterizován třemi základními vlastnostmi:

- stupněm,
- kardinalitou,
- volitelností účasti.

Stupeň vztahu

1. Unární vztah - relace je spojena sama se sebou. Typickým příkladem je vztah Zaměstnanec - Nadřízený, kdy nadřízený je také jedním ze zaměstnanců a může mít také nadřízeného. Vztah se realizuje vložením primárního klíče relace Zaměstnanec ve formě cizího klíče opět do relace Zaměstnanec.
2. Binární vztah - klasický vztah mezi dvěma relacemi.
3. Ternární vztah - jedná se o vztah mezi třemi relacemi najednou.
4. Enární vztah - jedná se o vztah mezi n-relacemi zároveň.

Ternární a enární vztahy se nesnadno modelují a v praxi se objevují velice zřídka.

Kardinalita vztahu

1. mezi daty v tabulkách není žádná spojitost, proto nedefinujeme žádný vztah.
2. 1:1 používáme, pokud záznamu odpovídá právě jeden záznam v jiné databázové tabulce a naopak. Takovýto vztah je používán pouze ojediněle, protože většinou není pádný důvod, proč takovéto záznamy neumístit do jedné databázové tabulky. Jedno z mála využití je zpřehlednění rozsáhlých tabulek. Jako ilustraci je možné použít vztah řidič - automobil. V jednu chvíli (diskrétní časový okamžik) řídí jedno auto právě jeden řidič a zároveň jedno auto je řízeno právě jedním řidičem.
3. 1:N přiřazuje jednomu záznamu více záznamů z jiné tabulky. Jedná se o nejpoužívanější typ relace, jelikož odpovídá mnoha situacím v reálném životě. Jako reálný příklad může posloužit vztah autobus - cestující. V jednu chvíli cestující jede právě jedním autobusem a v jednom autobuse může zároveň cestovat více cestujících.
4. M:N je méně častým. Umožňuje několika záznamům z jedné tabulky přiřadit několik záznamů z tabulky druhé. V databázové praxi bývá tento vztah z praktických důvodů nejčastěji realizován kombinací dvou vztahů 1:N a 1:M, které ukazují do pomocné tabulky složené z kombinace obou použitých klíčů (třetí resp. tzv. vazební tabulka). Příkladem z reálného života by mohl být vztah výrobek - vlastnost. Výrobek může mít více vlastností a jednu vlastnost může mít více výrobků. Dalším příkladem je vztah herec - film. Jeden herec může hrát ve více filmech, a v jednom filmu může hrát více herců. V reálném životě nicméně existuje velké množství vztahů M : N, mimo jiné také proto, že často existuje praktická potřeba zachovávat i údaje o historii těchto vztahů z časového hlediska (jeden řidič v delším časovém období řídí více rozličných aut a jedno auto v delším časovém období může mít více různých řidičů).

Volitelnost účasti ve vztahu

Volitelnost účasti ve vztahu vyjadřuje, zda je účast relace ve vztahu povinná nebo volitelná.

Normální formy

Pod pojmem *normalizace* rozumíme proces zjednodušování a optimalizace navržených struktur databázových tabulek. Hlavním cílem je navrhnout databázové tabulky tak, aby obsahovaly minimální počet redundantních dat. Správnost navržení struktur lze ohodnotit některou z následujících normálních forem.

1. Nultá normální forma (0NF) - tabulka v nulté normální formě obsahuje alespoň jeden sloupec (atribut), který může obsahovat více druhů hodnot.
2. První normální forma (1NF) - tabulka je v první normální formě, pokud všechny sloupce (atributy) nelze dále dělit na části nesoucí nějakou informaci neboli prvky musí být atomické. Jeden sloupec neobsahuje složené hodnoty.
3. Druhá normální forma (2NF) - tabulka je v druhé normální formě, pokud obsahuje pouze atributy (sloupce), které jsou závislé na celém klíči.
4. Třetí normální forma (3NF) - tabulka je ve třetí normální formě, pokud neexistují žádné závislosti mezi neklíčovými atributy (sloupci).
5. Čtvrtá normální forma (4NF) - tabulka je ve čtvrté normální formě, pokud sloupce (atributy) v ní obsažené popisují pouze jeden fakt nebo jednu souvislost.
6. Pátá normální forma (5NF) - tabulka je v páté normální formě, pokud by se přidáním libovolného nového sloupce (atributu) rozpadla na více tabulek.

Externí odkazy

- Článek na Root.cz - historie databází ^[2]
- Článek na Programujte.com - Úvod do relačních databází ^[3]
- Jiří Kosek: Co je to databáze ^[4]
- **(anglicky)** en:Edgar F. Codd
- **(anglicky)** en:Charles Bachman – en:Navigational database

Reference

- [1] http://cs.wikipedia.org/w/index.php?title=Rela%C4%8Dn%C3%AD_datab%C3%A1ze&action=edit
[2] <http://www.root.cz/clanky/historie-relacnich-databazi/>
[3] <http://programujte.com/index.php?akce=clanek&cl=2007110801-teoreticky-uvod-do-relacnich-databazi>
[4] <http://www.kosek.cz/clanky/iweb/12.html>

Relační model



Tento článek potřebuje úpravy.

Můžete Wikipedii pomoci tím, že ho vylepšíte ^[1]. Jak by měly články vypadat, popisují stránky Vzhled a styl, Encyklopedický styl a Odkazy.

Relační model je nejrozšířenějším způsobem uložení dat v databázi. Jedná se o 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 (sloupce - 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 integrity, 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.

Reference

[1] http://cs.wikipedia.org/w/index.php?title=Rela%C4%8Dn%C3%AD_model&action=edit

Trigger (databáze)

Trigger (česky **spoušť**) v databázi definuje činnosti, které se mají provést v případě definované události nad databázovou tabulkou. Definovanou událostí může být například vložení nebo smazání dat. Jednoduchá spoušť, která se má provést před vložení nového záznamu do tabulky může mít v SQL zápis:

```
CREATE TRIGGER jmeno_triggeru BEFORE INSERT
  ON jmeno_tabulky
BEGIN
  -- samotný kód spouště
END;
```

Tato spoušť před vložení dat do *jmeno_tabulky* provedou příkazy vložené mezi `BEGIN` a `END`. Kód triggerů lze spouštět i v jiných okamžicích (viz dále). Pokud je spouštěný příkaz jen jeden, mohou být klíčová slova `BEGIN` a `END` vynechána.

Parametry syntaxe

- Klíčové slovo `INSERT` indikuje, že trigger se spustí při vkládání do databáze. Místo něj může být též `UPDATE` nebo `DELETE`.
- Klíčové slovo `BEFORE` indikuje, že se trigger spustí *před* vkládáním, úpravou nebo mazáním záznamu. Pokud by bylo místo něj `AFTER`, spustil by se trigger poté.

Kdy se spustí který trigger

	BEFORE	AFTER
INSERT	Před vložení nového řádku. V příkazu <code>INSERT</code> a druhé části příkazu <code>REPLACE</code> .	Po vložení nového řádku.
UPDATE	Před úpravou existujícího řádku. V příkazu <code>UPDATE</code> nebo v druhé části příkazu <code>INSERT ... ON DUPLICATE KEY UPDATE</code> .	Po úpravě existujícího řádku.
DELETE	Před smazáním řádku. V příkazu <code>DELETE</code> nebo první části příkazu <code>REPLACE</code> , kdy je nalezen záznam se stejným primárním klíčem, jež <code>REPLACE</code> maže. Příkazy <code>DROP TABLE</code> ani <code>TRUNCATE TABLE</code> trigger nespouštějí, protože nedochází k doslovnému mazání jednotlivých záznamů.	Po smazání řádku.

OLD. a NEW.

Uvnitř kódu triggeru lze použít dva speciální prefixy, platící jenom v něm a nikde jinde. Jsou to:

- `OLD.` symbolizující „staré“ hodnoty měněného řádku (před přepsáním). Například `OLD.nazev` tak zpřístupňuje původní hodnotu sloupce `nazev` před úpravou.
- `NEW.` symbolizující „nové“ hodnoty měněného řádku (po přepsání – výsledek dané úpravy). Díky tomu lze tvořit elegantní konstrukce jako např.: `NEW.pocet_pristupu = OLD.pocet_pristupu + 1` apod. Časté je též získání právě vytvořeného unikátního klíče záznamu a manipulace se záznamem prostřednictvím něj. Databázové systémy nově vytvořený unikátní klíč většinou zpřístupňují přes klíčové slovo `LAST_INSERT_ID`, ale například pro vkládání více záznamů (zřetězené seznamy hodnot za klauzulí `VALUES`) to není možné a trigger je tak jediným prostředkem, který může kýženou manipulaci realizovat.

Manipulace s triggery

- Chceme-li trigger smazat, použijeme příkaz: `DROP TRIGGER jmeno_triggeru;`
- Pokud již trigger daného jména existuje a chceme jeho kód změnit, místo `CREATE` použijeme `REPLACE`.
Některé databázové systémy nicméně nemusejí `REPLACE` podporovat, a tak je místo toho potřeba trigger smazat a následně ho vytvořit.

Sémantika triggerů

Triggery jako takové jsou definovány ve většině moderních databázových systémů, ovšem mírně se liší v sémantice svého provedení. Klíčové rozdíly jsou zejména v tom...

- kdy přesně se trigger spustí
- jak proběhne, resp. co ho může přerušit
- jakým způsobem se řeší vzájemné volání triggerů
- jak (a jestli vůbec) jsou ošetřeny nekonečné zacyklování vzájemně se volajících triggerů

Externí Odkazy

- SQL – jak na triggery ^[1]
- prezentace o aktivních databázích a triggerech ^[2] - vysvětlení pojmů a porovnání nejvýznamnějších implementací triggerů (Starburst DB, Oracle, DB2, Chimera, SQL Server)
- Mysql (46) – Triggery ^[3]

Reference

[1] <http://interval.cz/clanky/sql-jak-na-triggeru/>

[2] http://www.tydlin.cz/__static/clanky-a-prace/triggery-aktivni-databaze/aktivni-databaze-triggery.pdf

[3] http://www.linuxsoft.cz/article.php?id_article=1019

Integritní omezení

Referenční integrita

Referenční integrita je nástroj databázového stroje, který pomáhá udržovat vztahy v relačně propojených databázových tabulkách.

Referenční integrita se definuje cizím klíčem, a to pro dvojici tabulek, nebo nad jednou tabulkou, která obsahuje na sobě závislá data (například stromové struktury). Tabulka, v níž je pravidlo uvedeno, se nazývá podřízená tabulka (používá se také anglický termín *slave*). Tabulka, jejíž jméno je v omezení uvedeno, je nadřízená tabulka (*master*). Pravidlo referenční integrity vyžaduje, aby pro každý záznam v podřízené tabulce, pokud tento obsahuje data vztahující se k nadřízené tabulce, odpovídající záznam v nadřízené tabulce existoval. To znamená, že každý záznam v podřízené tabulce musí v cizím klíči obsahovat hodnoty odpovídající primárnímu klíči nějakého záznamu v nadřízené tabulce, nebo NULL.

Jak se projevuje referenční integrita

- při přidání či změně záznamu v podřízené tabulce se kontroluje, zda stejná hodnota klíče existuje v nadřízené tabulce – porušení pravidla vyvolá chybu
- při mazání nebo úpravě záznamů v nadřízené tabulce se kontroluje, zda v podřízené tabulce není záznam se stejnou hodnotou klíče – porušení pravidla může vyvolat chybu nebo úpravu dat v podřízené tabulky v souladu s definovanými akcemi.

Příklad

- Při vkládání záznamu do tabulky, která obsahuje adresy podniků, se kontroluje, zda vložené poštovní směrovací číslo existuje v tabulce poštovních směrovacích čísel (směrovací číslo je v tabulce směrovacích čísel primárním klíčem).
- z tabulky poštovních směrovacích čísel nelze záznam s konkrétním poštovním směrovacím číslem odstranit, pokud existuje alespoň jeden podnik, který toto poštovní směrovací číslo používá

Související články

- Relační databáze
- Primární klíč
- Cizí klíč
- Databázový sirotek

Externí odkazy

- Krokodýlovy databáze ^[1] o referenční integritě

Reference

[1] <http://krokodata.vse.cz/DM/RefInt>

Databázová integrita

Integrita databáze znamená, že databáze vyhovuje zadaným pravidlům – integritním omezením. Tato integritní omezení jsou součástí definice databáze, a za jejich splnění zodpovídá systém řízení báze dat.

Integritní omezení se mohou týkat jednotlivých hodnot vkládaných do polí databáze (například známka z předmětu musí být v rozsahu 1 až 5), či může jít o podmínku na kombinaci hodnot v některých polích jednoho záznamu (například datum narození nesmí být pozdější než datum úmrtí). Integritní omezení se může týkat i celé množiny záznamů daného typu – může jít o požadavek na unikátnost hodnot daného pole či kombinace polí v rámci celé množiny záznamů daného typu, které se v databázi vyskytují (například číslo průkazu v záznamech o osobách).

Velmi často používaným integritním omezením v relačních databázích je tzv. referenční integrita. Jedná se o požadavek, aby pro pole záznamu, jež má obsahovat odkaz na jiný záznam někde v databázi, takový odkazovaný záznam skutečně existoval, tedy aby takový odkaz nevedl „do prázdna“ a nejednalo se o tzv. databázového sirotka.

Další integritní omezení lze definovat za pomoci tzv. triggerů. Jde o komplexnější definice kontrol, jež se budou provádět při každém pokusu o zápis záznamu do databáze.

Související články

- Databázový sirotek
- Doménová integrita
- Entitní integrita
- Referenční integrita

Klíče

Kandidátní klíč

Kandidátní klíč v relačním modelování označuje sloupec nebo kombinaci sloupců, ve kterých mají všechny řádky tabulky své hodnoty unikátní. Každý kandidátní klíč tak umožňuje jednoznačně identifikovat každý řádek tabulky. Jeden z kandidátních klíčů slouží jako primární klíč. Ostatní kandidátní klíče se pak označují také jako alternativní klíče.

Kandidátní klíč musí splňovat tyto časově nezávislé vlastnosti:

1. Hodnota ve sloupci (nebo kombinaci sloupců) kandidátního klíče musí být v rámci tabulky unikátní.
2. Množina sloupců vytvářejících kombinaci pro kandidátní klíč musí být v tabulce minimální. (Její nadmnožina by již nebyla kandidátním klíčem, přestože by každý řádek jednoznačně idenfikovala - nebyla by už ale minimálním klíčem.)

Primární klíč

Primární klíč je pole nebo kombinace polí, jednoznačně identifikující každý záznam v databázové tabulce. Žádné pole, které je součástí primárního klíče, nesmí obsahovat hodnotu NULL. Každá tabulka má mít definovaný právě jeden primární klíč (**entitní integrita**).

Primární klíč má dvě základní vlastnosti: jedinečnost v rámci tabulky a ne-NULL-ovou hodnotu.

Databázový systém by měl být navržen a udržován tak, aby se primární klíč založeného záznamu nikdy nemusel měnit.

Typickým příkladem primárního klíče je například katalogové číslo u výrobků, identifikační číslo v seznamu podniků apod. Pokud u záznamu neexistuje žádný přirozený primární klíč, nebo je takový primární klíč příliš složitý, používá se obvykle jako primární klíč číslo, které záznamu přidělí automaticky sama databáze. Takové číslo může být pořadové číslo nebo pseudonáhodné číslo. Většinou se pak nazývá ID.

Související články

- Cizí klíč
 - SQL
 - Relační databáze
 - Doménová integrita
 - Referenční integrita
 - Index (databáze)
 - Kandidátní klíč
-

Cizí klíč

Cizí klíč (FOREIGN KEY) v prostředí relačních databází definuje vztah mezi dvěma tabulkami takový, že hodnota v určeném sloupci musí existovat v jiné (primární) tabulce. Tím je definováno integritní omezení, které do tabulky položky umožní vložit jen povolené hodnoty. Je tím vlastně vytvořeno spojení jednoho nebo více sloupců se sloupcem nebo více sloupci jiné („cizí“) tabulky. Tomu se též říká reference nebo odkaz.

Cizí klíč umožňuje definovat akce, které mají nastat při pokusu o změnu nebo mazání záznamů v cizí tabulce. Například, po smazání záznamu z cizí tabulky budou ve zdrojové tabulce řádky s odpovídající hodnotou cizího klíče taktéž smazány, nebo budou jejich odkazy nastaveny na určitou (neutrální) hodnotu, nebo se smazání řádků v cizí tabulce zabrání. Omezení cizích klíčů tak představuje mechanismus pro udržení referenční integrity databáze.

Syntaxe

```
[CONSTRAINT [název omezení]] FOREIGN KEY
    [jméno klíče] (název sloupce tabulky nebo jejich seznam oddělený
čárkami)
    REFERENCES jméno_tabulky (název sloupce nebo jejich seznam oddělený
čárkami)
    [ON DELETE akce]
    [ON UPDATE akce]
```

Definice cizího klíče platí jen v jednom směru – často se pro účely lepší představy "cizí" tabulce říká *rodičovská tabulka* a tabulce, v níž se aplikují změny v závislosti na nastavení cizího klíče, říká *dceřiná tabulka*.

Akce

Jako akce může být:

- **CASCADE** – pro klauzuli **ON UPDATE** se v dceřiné tabulce adekvátně změní odkazy na cizí klíč; v případě **ON DELETE** se záznamy odkazující na smazaný klíč smažou také. Většina vyspělejších databázových systémů umožňuje tuto akci aplikovat zřetězeně (pokud je nastaveno, že dceřiná tabulka je z hlediska cizího klíče současně rodičovskou tabulkou pro tabulku jinou atd).
- **SET DEFAULT** – při změně nebo smazání rodičovského klíče se reference v dceřiné tabulce nastaví na hodnotu, kterou má definovanou jako výchozí
- **SET NULL** – jako **SET DEFAULT**, kde defaultní hodnotou je zde hodnota **NULL**
- **RESTRICT** – pokud v dceřiné tabulce existují záznamy odkazující na cizí klíč, pak odpovídající záznam v rodičovské tabulce nepůjde změnit nebo smazat (SŘBD tomu zabrání)
- **NO ACTION** – v konečném efektu stejné jako **RESTRICT**; rozdíly interpretace se liší podle jednotlivých databázových systémů: v některých akce **NO ACTION** na rozdíl od **RESTRICT** nehodí chybovou hlášku; v jiných implementacích se tyto dvě akce liší v „opožděnosti,“ s kterou databázový systém cizí klíče kontroluje. Např. v případě MySQL se kontrola provádí implicitně, pokud není vypnuta direktivou **SET FOREIGN_KEY_CHECKS = 0;**

Příklad

V databázi spolku přátel psů máme následující tabulky:

- `osoby` se sloupci `osoba_id` a `jméno`
- `psi` se sloupci `pes_id`, `majitel` a `rasa`

Aby byla data v databázi korektní, je třeba, aby každý záznam psa měl uvedeného platného majitele. Proto označíme v tabulce `psi` sloupec `majitel` jako cizí klíč, vztažený k sloupci `osoba_id` v tabulce `osoby`. Když je poté přidán záznam pro psa, databázový engine bude vyžadovat, aby hodnota v poli `majitel` nabývala některé z existujících hodnot `id` tabulky `osoby`. Zároveň můžeme určit, zda se při smazání osoby smažou i záznamy všech psů, kterými je majitelem, nebo zda má pokus o smazání osoby vlastnící alespoň jednoho psa selhat.

Související články

- SQL
- relační databáze
- referenční integrita

Externí odkazy

- **(anglicky)** Foreign Key Constraints ^[1] – popis implementace cizích klíčů v databázi MySQL
- **(anglicky)** FOREIGN KEY Constraints ^[2] – popis implementace cizích klíčů v databázi Microsoft SQL Server

Reference

[1] <http://dev.mysql.com/doc/refman/5.0/en/innodb-foreign-key-constraints.html>

[2] <http://msdn2.microsoft.com/en-us/library/ms175464.aspx>

Index (databáze)

INDEX (někdy též označovaný jako klíč - **KEY**) je databázová konstrukce, sloužící ke zrychlení vyhledávacích a dotazovacích procesů v databázi, definování unikátní hodnoty sloupce tabulky nebo optimalizaci fulltextového vyhledávání.

Databázové indexy

Definice indexu

Index je obvykle definován výběrem tabulky a jednoho konkrétního sloupce (nebo více sloupců), nad kterými si analytik nebo designér databáze přeje dotazování urychlit; dále pak technickým určením datového typu a typu indexu. Chování a způsoby uložení indexů se mohou výrazně i výrazně lišit podle použité databázové technologie.

Výjimku mohou tvořit například fulltextové indexy, které jsou v některých případech (nerelační databáze typu Lotus Notes) definovány nad celou databází, nikoliv nad konkrétní tabulkou.

Vytvoření indexu způsobí zvýšení nároků databázového serveru na operační paměť a diskový prostor (o vyhledávací struktury, pokud se ukládají ve formě souboru). Velikost samotných základních dat to ale neovlivní. Ovlivní se ale práce databáze. Například, při importu dat (exportovaných z databáze na jiném počítači) definuje cílový databázový server strukturu tabulek a indexů, inicializuje pro ně potřebnou paměť (a diskový prostor) a během importu bude krom samotných dat průběžně aktualizovat i struktury indexů. Jinými slovy, ukládání do db je zpomaleno těmito procesy (vytvoření indexu a zapsání), ale čtení je značně urychleno (právě díky indexům).

Funkce indexu

Vytvořením indexu (například příslušným příkazem jazyku SQL - CREATE) databázový server (též zvaný Systém řízení báze dat - SŘBD) zarezervuje pro požadovaný index určitou část paměťového prostoru a uloží do něj informace o rozmístění hodnot indexovaných sloupců v tabulce (obvykle ve strojové a pro člověka nečitelné podobě, která závisí na použitých vnitřních algoritmech indexace). Pokud později dojde k dotazu (například pomocí příkazu SELECT nebo použitím pohledu), který se týká indexovaných sloupců, není tabulka prohledávána podle toho, jak jsou za sebou řádky uloženy, ale pomocí informací uložených v paměťovém prostoru indexu je přístupováno přímo k relevantním řádkům tabulky - dalo by se říct, že index funguje trochu podobně jako rejstřík v knize, kde místo odkazu na příslušnou stránku knihy je ukazatel na paměťové místo s kýženými daty.

Použití indexu

Na první pohled by se mohlo zdát, že čím víc indexů, tím lepší chování databáze a že po vytvoření indexů pro všechny sloupce všech tabulkách dosáhneme maximálního zrychlení. Tento přístup naráží na dva zásadní problémy:

1. Každý index zabírá v paměti vyhrazené pro databázi nezanedbatelné množství místa (vzhledem k paměti vyhrazené pro tabulku). Při existenci mnoha indexů se může stát, že paměť zabraná pro jejich chod je skoro stejně velká, jako paměť zabraná jejími daty - zvláště u rozsáhlých tabulek (typu faktových tabulek v datovém skladu) může něco takového být nepřijatelné.
2. Každý index zpomaluje operace, které mění obsah indexovaných sloupců (například SQL příkazy UPDATE, INSERT). To je dáno tím, že databáze se v případě takové operace nad indexovaným sloupcem musí postarat nejen o změny v datech tabulky, ale i o změny v datech indexu.

Pro správné zvolení indexů by ten, kdo databázi navrhuje, měl vědět, jak často se vybírané záznamy z dané tabulky třídí podle zamýšleného sloupce (a kandidáta na index) a nakolik je důležité, aby vyhledávání a třídění podle něj bylo rychlé. U některých tabulek, které jsou např. číselníky o stálém počtu položek, řekněme max. několika desítkách, nemusí být třeba index definovat vůbec (Toto platí pouze v případě, že načtení celé tabulky (full table

scan) se provede za max. dvě operace čtení).

Příklad definice klíčů v jazyku SQL

```
CREATE TABLE diskuzni_forum(  
  id int(8) NOT NULL auto_increment,  
  jmeno varchar (30) NOT NULL,  
  email varchar (30) NOT NULL,  
  prispevek mediumtext NOT NULL,  
  cas_pridani datetime NOT NULL,  
  PRIMARY KEY (id),  
  INDEX (cas_pridani)  
)
```

Červený (sedmý) řádek udělá nad sloupcem `id` primární klíč; zelený (osmý) řádek povyšuje sloupec `cas_pridani` na (standardní) index - hledání a třídění podle něj bude od této chvíle optimalizováno databázovým strojem.

Druhy indexů

Databázové indexy (či indexsekvencní 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 naleznete 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.

Speciální druhy indexů

Parciální index

Parciální index je index, který spadá do jednoho z výše uvedených základních druhů (v drtivé většině případů **INDEX**), ale týká se jen určité podmnožiny řádků tabulky. Databázový stroj s ním pracuje jen za určité podmínky, tato podmínka se týká některých z ostatních polí v dané tabulce, tato podmínka je součástí definice parciálního klíče. Parciální klíče mají jen některé databáze (např. PostgreSQL). Situace, kdy záznamy tabulky příliš nerovnoměrně spadají do jedné velké podskupiny, je příkladem a důvodem pro vytvoření parciálního indexu, který v této podskupině (a pouze v ní) záznamy zindexuje.

Kandidátní index

Související informace naleznete také v článku Kandidátní klíč.

Cizí index

Související informace naleznete také v článku Cizí klíč.

Funkční index

Funkční nebo též výrazový index pracuje (tj. porovnává, vyhledává, vytváří si interní pořadové tabulky) s prostou hodnotou sloupce, nad kterým je definován, ale nad funkční hodnotou na něj aplikovanou nebo i výrazem z nich složených. Pokud sledovaný výraz umožňuje použití i více sloupců z tabulky, lze pomocí něj nastavit i složitější chování přímo v databázovém stroji, například:

- porovnávat nezávisle na velikosti písmen

```
CREATE INDEX test1_lower_coll_idx ON test1 (lower(coll));
```

- třídít podle rozkódované hodnoty sloupce

```
CREATE INDEX test1_decode_coll_idx ON test1 (decode(coll));
```

- nastavit index pouze podle prvních např. 2 písmen

```
CREATE INDEX test1_left2_coll_idx ON test1 (substr(coll1,0,2));
```

- nastavit unikátnost záznamu na základě shody dvou nebo více sloupců (například jména a příjmení)

```
CREATE INDEX full_name ON people ((first_name || ' ' || last_name));
```

V jiných databázích by se tato funkčnost provedla přiřazením sloupců, které mají dohromady být unikátní (v tomto případě `first_name` a `last_name`) do jednoho unikátního indexu. Podporuje-li databázový stroj funkční indexy s libovolným výrazem ze sloupců z dané tabulky, schopnost přiřadit více sloupců do jednoho indexu tím více než dostatečně supluje a objektivně se dá říci, že se jedná o flexibilnější řešení s většími možnostmi.

Funkčními indexy disponují například databáze PostgreSQL.

Odkazy

Související články

- Primární klíč

Externí odkazy

- Využití databázových indexů ^[1] na root.cz ^[2]
- Typy databázových indexů ^[3] a Ukázka použití indexů ^[4] - články Jakuba Vrány

Reference

[1] <http://www.root.cz/clanky/vyuziti-databazovych-indexu/>

[2] <http://www.root.cz/>

[3] <http://php.vrana.cz/typy-databazovych-indexu.php>

[4] <http://php.vrana.cz/ukazka-pouziti-indexu.php>

Normalizace

Normalizace databáze

Pojem **normalizace** je spjat s relačním modelem. V relačním modelu jsou data uložena v tabulkách, na které má jisté požadavky. Při splnění požadavků je tabulka označována jako *normalizovaná*. Pokud nejsou tyto požadavky splněny, jsou označovány jako *nenormalizované* a proces jejich převodu na tabulky se označuje jako normalizace. Při tomto procesu dochází k odstraňování nedostatků tabulek jako je redundance nebo možnost vzniku aktualizací anomálie, tj. nechtěného vedlejšího efektu operace nad databází, při kterém dojde ke ztrátě nebo nekonzistenci dat. Postup normalizace je rozdělen do několika kroků a po dokončení každého z nich se tabulka nachází v určité normální formě. V praxi se většinou normalizuje do Třetí normální formy, vyšší normální formy je vcelku obtížné porušit a vyžadují relativně velké znalosti, stejně jako návrh databází takové velikosti, kde je možné je porušit.

Normální formy

- 1NF: Každý atribut obsahuje pouze atomické hodnoty
- 2NF: Každý neklíčový atribut je plně závislý na primárním klíči
- 3NF: Všechny neklíčové atributy musí být vzájemně nezávislé
- BCNF: Atributy, které jsou součástí primárního klíče, musí být vzájemně nezávislé
- 4NF: Relace popisuje pouze příčinnou souvislost mezi klíčem a atributy
- 5NF: Relaci již není možno bezeztrátově rozložit

Pro splnění určité normální formy je nutné splnit i ty předchozí. V praxi se obvykle používají pouze první tři.

Ukázky záznamu v databázi

adresa v tabulce nesplňující 1NF

Adresa
Jan Novák, Nádražní 42, Zlín 12300

Jméno	Příjmení	Ulice	Číslo popisné	Město	Směrovací číslo
Jan	Novák	Nádražní	42	Zlín	12300

⊕ adresa v tabulce splňující 1NF

Třetí normální forma

Třetí normální forma (3NF) je soubor doporučení (metodika) pro návrh datové struktury databáze, jehož dodržení vede k optimálnímu využití vlastností systému OLTP při tvorbě databázových aplikací. 3NF obsahuje jako svou podmnožinu druhou normální formu (2NF) a první normální formu (1NF):

1. Eliminuj duplicitní sloupce v jednotlivých tabulkách.
2. Pro každou skupinu dat s jasně vymezeným významem vytvoř zvláštní tabulku, každý řádek opatří unikátním primárním klíčem.
3. Obsahem jednotlivých sloupců tabulky by měla být jednoduchá, dále nedělitelná informace.
4. Podmnožinu dat se shodnou hodnotou pro určitý sloupec tabulky převed' do samostatné tabulky a spoj s původní tabulkou cizím klíčem.
5. Odstraň z tabulky sloupce, které jsou přímo závislé na jiné skupině sloupců tabulky než pouze na primárním klíči.

Tak, jak jsou zde zapsány, působí jednotlivá doporučení velice obecně a (alespoň pro toho, kdo již někdy nějakou databázovou strukturu navrhoval) v podstatě samozřejmě. Pokusme se demonstrovat jejich použití na jednoduchém příkladu:

Příklad

Zadání: Navrhněte datovou strukturu (tabulku nebo skupinu tabulek) pro uložení seznamu oddělení firmy, který bude obsahovat číslo oddělení, adresu budovy, kde oddělení sídlí, jméno, příjmení a plat šéfa a jméno, příjmení a plat všech zaměstnanců oddělení.

První pokus o řešení

Vytvořím si jednu tabulku **Oddělení**, která bude obsahovat následující sloupce:

- číslo oddělení
- adresa budovy
- jméno, příjmení a plat šéfa
- jméno, příjmení a plat prvního zaměstnance
- jméno, příjmení a plat druhého zaměstnance
- jméno, příjmení a plat třetího zaměstnance
- ... a tak dále podle toho, kolik má zaměstnanců největší oddělení ve firmě

Moc hezké, ale zkuste v této struktuře napsat příkaz SELECT, který vybere deset nejlépe placených zaměstnanců. Nebo třeba SELECT, který vybere oddělení s více než deseti zaměstnanci.

Tato struktura rozhodně nesplňuje bod 3 z výše uvedených doporučení - adresa by měla být rozdělena na ulici, město, PSČ a stát a pro každého zaměstnance by mělo být uvedeno zvlášť jméno, zvlášť příjmení a zvlášť plat.

Tato struktura také rozhodně nesplňuje bod 2 - jsou zde dohromady v jedné tabulce smíchány údaje o lidech a o odděleních.

Druhý pokus o řešení

Vytvořím dvě tabulky:

Tabulka **Oddělení**:

- číslo oddělení (primární klíč)
- adresa budovy - ulice
- adresa budovy - číslo
- adresa budovy - město
- adresa budovy - PSČ
- adresa budovy - stát
- ID šéfa (cizí klíč do tabulky Lidé)

Tabulka **Lidé**:

- ID člověka (primární klíč - uměle vytvořené zaměstnanecké číslo, žádný z existujících sloupců ani jejich skupinu nemohu použít jako primární klíč, neboť ve firmě mohou existovat dva Josefové Novákové pracující ve stejném oddělení za stejný plat)
- jméno
- příjmení
- plat
- číslo oddělení (cizí klíč do tabulky Oddělení)

Už je to o něco lepší - rozhodně už dokážu pomocí SQL příkazu najít nejlépe placené zaměstnance nebo zjistit počet zaměstnanců jednotlivých oddělení. Problém je ještě v doporučení číslo 4 v případě tabulky Oddělení - firma bude nejspíš mít hodně oddělení, ale málo budov (možná dokonce jenom jednu), rozhodně bude hodně oddělení sedět vždy ve stejné budově. V tabulce Oddělení se mi tedy bude pořád dokola opakovat několik málo adres budov. Správně by tedy měly být adresové sloupce vyvedeny do zvláštní tabulky.

Třetí pokus o řešení

Vytvořím tři tabulky:

Tabulka **Oddělení**:

- číslo oddělení (primární klíč)
- ID šéfa (cizí klíč do tabulky Lidé)
- ID budovy (cizí klíč do tabulky Budovy)

Tabulka **Lidé**:

- ID člověka (primární klíč)
- jméno
- příjmení
- plat
- číslo oddělení (cizí klíč do tabulky Oddělení)

Tabulka **Budovy**:

- ID budovy (primární klíč - uměle vytvořené pořadové číslo budovy)
- adresa budovy - ulice
- adresa budovy - číslo
- adresa budovy - město
- adresa budovy - PSČ
- adresa budovy - stát

To už vypadá docela dobře. Dalo by se diskutovat o tom, jestli *číslo oddělení* je opravdu vhodný primární klíč, co se stane, když dojde k přečíslování oddělení, nebo třeba o tom, že podle PSČ a státu dokážu určit město, takže podle doporučení 5 je položka *adresa budovy - město* vlastně přebytečná, ale v zásadě jsme se díky pravidlům **3NF** dostali ke struktuře, se kterou by nemělo být složité pracovat pomocí příkazů jazyka SQL.

Externí odkazy

- Databáze a jazyk SQL ^[1] - popis prvních čtyř normálních forem (česky)

Reference

[1] <http://interval.cz/clanky/databaze-a-jazyk-sql/>

SQL

SQL

SQL (někdy vyslovováno anglicky *es-kjú-el* [ɛs kjuː ɛl] IPA, někdy též *síkvl* [si:kwəl] IPA) je standardizovaný dotazovací jazyk používaný pro práci s daty v relačních databázích. SQL je zkratka anglických slov **Structured Query Language** (strukturovaný dotazovací jazyk).

Historie SQL

V 70. letech 20. století probíhal ve firmě IBM výzkum relačních databází. Bylo nutné vytvořit sadu příkazů pro ovládání těchto databází. Vznikl tak jazyk **SEQUEL** (Structured English Query Language). Cílem bylo vytvořit jazyk, ve kterém by se příkazy tvořily syntakticky co nejlépe přirozenému jazyku (angličtině).

K vývoji jazyka se přidaly další firmy. V r. 1979 uvedla na trh firma Relational Software, Inc. (dnešní Oracle Corporation) svoji relační databázovou platformu *Oracle Database*. IBM uvedla v roce 1981 nový systém *SQL/DS* a v roce 1983 systém *DB2*. Dalšími systémy byly např. Progres, Informix^[1] a SyBase. Ve všech těchto systémech se používala varianta jazyka SEQUEL, který byl přejmenován na **SQL**.

Relační databáze byly stále významnější, a bylo nutné jejich jazyk standardizovat. Americký institut ANSI původně chtěl vydat jako standard zcela nový jazyk RDL. SQL se však prosadil jako *de facto* standard a ANSI založil nový standard na tomto jazyku. Tento standard bývá označován jako *SQL-86* podle roku, kdy byl přijat.

V dalších letech se ukázalo, že *SQL-86* obsahuje některé nedostatky a naopak v něm nejsou obsaženy některé důležité prvky týkající se hlavně integrity databáze. V roce 1992 byl proto přijat nový standard *SQL-92* (někdy se uvádí jen *SQL2*). Zatím nejnovějším standardem je *SQL3* (*SQL-99*), který reaguje na potřeby nejmodernějších databází s objektovými prvky.

Standards podporuje prakticky každá relační databáze, ale obvykle nejsou implementovány vždy všechny požadavky normy. A naopak, každá z nich obsahuje prvky a konstrukce, které nejsou ve standardech obsaženy. Přenositelnost SQL dotazů mezi jednotlivými databázemi je proto omezená.

Popis jazyka

SQL příkazy se dělí na čtyři základní skupiny:

- Příkazy pro manipulaci s daty (SELECT, INSERT, UPDATE, DELETE, ...)
 - Příkazy pro definici dat (CREATE, ALTER, DROP, ...)
 - Příkazy pro řízení přístupových práv (GRANT, REVOKE)
 - Příkazy pro řízení transakcí (START TRANSACTION, COMMIT, ROLLBACK)
 - Ostatní nebo speciální příkazy
-

Odkazy

Související články

- Databáze
- Primární klíč
- Tabulka
- Databázová transakce

Externí odkazy

- Tutoriál SQL ^[2]

Reference

[1] http://www.intax.cz/INFORMIX_Dynamic_Server.html

[2] <http://krokodata.vse.cz/SQL>

Příkazy jazyka SQL

Příkazy jazyka SQL obecně umožňují úplnou kontrolu nad systémem řízení báze dat. Podle svého účelu se dělí do následujících skupin:

Příkazy pro manipulaci s daty

Jsou to příkazy pro získání dat z databáze a pro jejich úpravy. Označují se zkráceně **DML** – *Data Manipulation Language* („jazyk pro manipulaci s daty“).

- **SELECT** – vybírá data z databáze, umožňuje výběr podmnožiny a řazení dat.
- **INSERT** – vkládá do databáze nová data.
- **UPDATE** – mění data v databázi (editace).
- **MERGE** – kombinace **INSERT** a **UPDATE** – data buď vloží (pokud neexistuje odpovídající klíč), pokud existuje, pak je upraví ve stylu **UPDATE**.
- **DELETE** – odstraňuje data (záznamy) z databáze.
- **EXPLAIN** – speciální příkaz, který zobrazuje postup zpracování SQL příkazu. Pomáhá uživateli optimalizovat příkazy tak, aby byly rychlejší.
- **SHOW** - méně častý příkaz, umožňující zobrazit databáze, tabulky nebo jejich definice

Příkazy pro definici dat

Těmito příkazy se vytvářejí struktury databáze – tabulky, indexy, pohledy a další objekty. Vytvořené struktury lze také upravovat, doplňovat a mazat. Tato skupina příkazů se nazývá zkráceně **DDL** – *Data Definition Language* („jazyk pro definici dat“).

- **CREATE** – vytváření nových objektů.
 - **ALTER** – změny existujících objektů.
 - **DROP** – odstraňování objektů.
-

Příkazy pro řízení dat

Do této skupiny patří příkazy pro nastavování přístupových práv a řízení transakcí. Označují se jako **DCL** – *Data Control Language* („jazyk pro ovládání dat“), někdy také **TCC** – *Transaction Control Commands* („jazyk pro ovládání transakcí“).

- GRANT – příkaz pro přidělení oprávnění uživateli k určitým objektům.
- REVOKE – příkaz pro odnětí práv uživateli.
- START TRANSACTION – zahájení transakce.
- COMMIT – potvrzení transakce.
- ROLLBACK – zrušení transakce, návrat do původního stavu.

Ostatní příkazy

Do této skupiny patří příkazy pro správu databáze. Pomocí nich lze přidávat uživatele, nastavovat systémové parametry (kódování znaků, způsob řazení, formáty data a času apod.). Tato skupina není standardizována a konkrétní syntaxe příkazů je závislá na databázovém systému. V některých dialektech jazyka SQL jsou přidány i příkazy pro kontrolu běhu, takže lze tyto dialekty zařadit i mezi programovací jazyky.

Komentáře

Do kódu SQL lze ve všech hlavních databázích zapisovat i komentáře. Prakticky všechny z nich podporují jednořádkové komentáře, některé i víceřádkové. Jednořádkové bývají uvozeny znaky jako `--`, `##` (či jenom `#`), s tím, že parser jazyka ignoruje vše až do konce řádku. Víceřádkové komentáře se mohou psát mezi `/*` a `*/`, kdy případný kód za koncovým `*/` se zpracovává.

```
-- jednořádkový komentář. SELECT "tento příkaz se nevykoná";

/* více-
-řádkový
komentář */ SELECT "tento příkaz se vykoná.";
```

Např. MySQL využívá tento typ komentářů k vkládání kódu specifického pro tento typ databáze, s tím, že specifický kód je umístěn mezi `/*!` a `*/`.

```
/*!40101 SET NAMES utf8 */;
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE table1.col1=5;
```

Je-li takovýto kód spuštěn v jiném databázovém prostředí, je vnitřek komentáře ignorován (můžeme hovořit o kompatibilitě s ostatními implementacemi SQL).

MySQL

MySQL

```

Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.0.51b-community-nt MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>

```

Screenshot příkazové řádky MySQL

Vývojář	Oracle Corporation
Aktuální verze	5.5.27 (2. srpen 2012)
Připravovaná verze	5.6.5 (10. duben 2011)
Operační systém	Linux, Solaris, FreeBSD, Mac OS, Windows
Typ softwaru	RDBMS
Licence	GPL nebo komerční licence
Web	www.mysql.org ^[1]

MySQL je databázový systém, vytvořený švédskou firmou MySQL AB, nyní vlastněný společností Sun Microsystems, dceřinou společností Oracle Corporation. Jeho hlavními autory jsou Michael „Monty“ Widenius a David Axmark. Je považován za úspěšného průkopníka dvojího licencování – je k dispozici jak pod bezplatnou licencí GPL, tak pod komerční placenou licencí.

MySQL je multiplatformní databáze. Komunikace s ní probíhá – jak už název napovídá – pomocí jazyka SQL. Podobně jako u ostatních SQL databází se jedná o dialekt tohoto jazyka s některými rozšířeními.

Pro svou snadnou implementovatelnost (lze jej instalovat na Linux, MS Windows, ale i další operační systémy), výkon a především díky tomu, že se jedná o volně šiřitelný software, má vysoký podíl na v současné době používaných databázích. Velmi oblíbená a často nasazovaná je kombinace Linux, MySQL, PHP a Apache jako základní software webového serveru („technologie LAMP“).

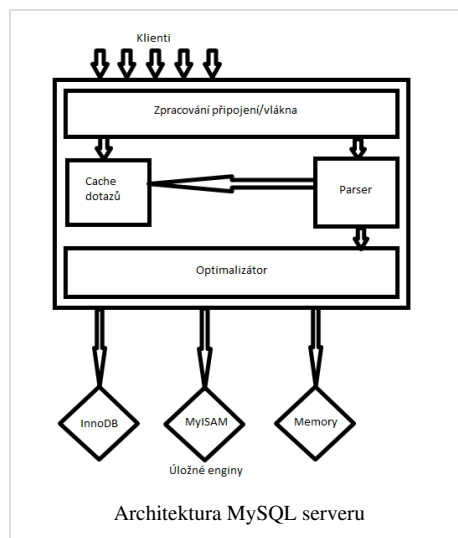
MySQL bylo od počátku optimalizováno především na rychlost, a to i za cenu některých zjednodušení: má jen jednoduché způsoby zálohování, a až donedávna nepodporovalo pohledy, trigger, a uložené procedury. Tyto vlastnosti jsou doplňovány teprve v posledních letech, kdy začaly nejčastějším uživatelům produktu – programátorům webových stránek – již poněkud scházet.

Architektura MySQL serveru

Architektura MySQL se velmi odlišuje od architektur jiných databázových serverů, má široký záběr a je užitečná pro řešení mnoha různorodých úloh. Vrstva, která je úplně nahoře, obsahuje služby, jež nejsou jedinečné pro MySQL. Obsluhují většinu potřebných nástrojů klient/server, které jsou založeny na síti.

Ve druhé vrstvě se nachází valná část mozku MySQL, včetně kódu pro rozbor (parsing), analýzu, optimalizaci a pro všechny zabudované funkce. Na této úrovni se nachází veškerá funkcionalita, která se poskytuje prostřednictvím úložných enginů.

Třetí vrstva obsahuje úložné enginy. Ty mají na starosti ukládání a získávání všech dat uložených v MySQL. Server komunikuje s úložnými enginy prostřednictvím API úložných enginů. Toto rozhraní skrývá rozdíly mezi jednotlivými úložnými enginy a činí je na vrstvě dotazů velmi transparentními. API obsahuje několik desítek nízkourovňových funkcí, které provádějí operace jako "zahájit transakci" nebo "získat řádek, který má tento primární klíč". Úložné enginy nedělají rozbor SQL a nekomunikují mezi sebou - jednoduše pouze odpovídají na požadavky serveru.



Správa připojení a bezpečnost

Každé klientské připojení dostane uvnitř serverového procesu vlastní vlákno (thread). Dotazy tohoto připojení se vykonávají uvnitř tohoto jediného vlákna, které zase sídlí na jednom jádru nebo CPU. Protože server udržuje vlákna v cache, nemusejí se vytvářet a likvidovat pro každé nové připojení. Autentizace je založena na uživatelském jménu, hostiteli, odkud pocházejí a heslu. Dají se také používat certifikáty X509 přes připojení SSL. Jakmile se klient připojí, server pro každý dotaz vydaný klientem ověřuje, zdali má patřičná oprávnění pro akci, kterou chce vykonat.

Optimalizace a vykonávání

MySQL provádí rozbor dotazů proto, aby vytvořil interní stromovou strukturu (parse tree), pak aplikuje všelijaké optimalizace. Může dotaz přepsat, určit pořadí, v němž bude číst tabulky, zvolit, které indexy použije atd. Prostřednictvím speciálních klíčových slov může programátor optimalizátoru předat tzv. pokyny (hints), jimiž se dá ovlivnit rozhodovací proces. Optimalizátor se ve skutečnosti nestará o to, který úložný engine používá konkrétní tabulka. Úložný engine ovšem ovlivňuje, jak server optimalizuje dotaz. Optimalizátor od úložného enginu zjišťuje, zdali má jistou výbavu, ptá se na náklady jistých operací a dotazuje se na statistiky o datech tabulky.

Ovšem ještě dříve než server začne s rozbořem dotazu, obrátí se na cache dotazů (query cache), kam může ukládat pouze příkazy SELECT společně s jejich výslednými sadami. Jestliže někdo vydá dotaz, který je identický s nějakým dotazem, který je už k dispozici v cache, server nemusí dělat vůbec žádný rozbor, nemusí nic optimalizovat a dokonce nemusí dotaz ani vykonat - jednoduše pouze předá zpět uloženou výslednou sadu.

Úložné enginy (úložiště dat)

MySQL ukládá každou databázi (také se jim říká schéma) do podadresáře svého datového adresáře na odkladovém souborovém systému (dá se změnit klauzulí `DATA DIRECTORY` a `INDEX DIRECTORY`). Když vytvoříte nějakou tabulku, MySQL ukládá definici tabulky, data tabulky a indexy tabulky do speciálních souborů s následujícími příponami:

- `.frm` – definice tabulky
- `.MYD` – data tabulky
- `.MYI` – index/y tabulky

Název souboru bez přípony se shoduje s názvem tabulky. Jelikož MySQL používá při ukládání definic souborový systém, otázka rozlišování velikosti písmen je závislá na platformě. Na instalaci MySQL na Windows se velikost písmen v názvech tabulek a databází nerozlišuje, na unixových systémech se velikost písmen rozlišuje. Každý úložný engine ukládá tabulky a indexy jinak, definici tabulky ovšem zpracovává samotný server.

MySQL nabízí několik typů úložných enginů (storage engine), které se liší svými možnostmi, použitím a způsobem ukládání dat do souborů:

- `ARCHIVE` – engine uzpůsobený pro ukládání velkého množství neindexovaných dat.
- `BLACKHOLE` – engine, který data přijímá, ale neukládá je (zahazuje je).
- `CSV` – ukládá data v textovém formátu CSV.
- `EXAMPLE` – nefunkční engine, který slouží jako ilustrační pro potřeby zdrojových kódů databáze MySQL a využijí ho tedy jen její vývojáři.
- `FEDERATED`
- `InnoDB`
- `MEMORY` nebo `HEAP` – vysoce výkonný engine, který data uchovává pouze v operační paměti, při restartu serveru jsou data ztracena.
- `MERGE` - sloučení dat z několika `MyISAM` tabulek o stejné struktuře, starší alternativa k `partition`.
- `MyISAM`

Před koupí MySQL Oraclem byl vyvíjen ještě engine `Falcon`, ale Oracle jeho vývoj ukončil.

Úložné enginy fungují jako moduly, které lze k distribuci tohoto databázového systému doinstalovat; jejich aktuální seznam lze zjistit příkazem `SHOW ENGINES`.

Přehled podporovaných vlastností

verze 3.23

- cizí klíče (podporovány v tabulkách typu `InnoDB`)
- transakce (podporovány v tabulkách typu `InnoDB`)
- Příkazová replikace

verze 4.0

- sjednocování dotazů pomocí `UNION`

verze 4.1

- podpora různých znakových sad a porovnávání na úrovni databáze, tabulky i sloupce
- podpora časových pásem v datech
- poddotazy
- R-stromy (v tabulkách typu `MyISAM`)
- podpora „audio“ funkce `SOUNDS_LIKE`

verze 5.0

- uložené procedury

- triggery
- pohledy
- práce s metadaty
- distribuované XA transakce (v tabulkách typu InnoDB)
- kurzory
- INFORMATION.SCHEMA

verze 5.1

- partitioning
- časování událostí (Event Scheduler)
- úložiště IBMDB2I (tabulky typu IBM DB2, podporující transakce)
- rozšiřitelné API
- replikace na úrovni řádků
- logování na straně serveru

plánováno ve verzi 6

- cizí klíče (i pro jiné tabulky než InnoDB)
- použití cizích i fulltextových klíčů současně

Kódování a znakové sady

Od verze 4.1 MySQL řeší ukládání řetězců s podporou Unicode pomocí nastavení znakové sady (CHARACTER SET) COLLATION. To představuje souhrn způsobů, jak k takto uloženému textu přistupovat – porovnávání (s ohledem na případné národnostní zvyklosti), řazení, citlivost velkých malých písmen, ligatur, transkripce speciálních znaků apod. Znaková sada a collation mohou být nastaveny individuálně pro daný (textový) sloupec, mimo je možnost nastavit defaultní sadu a collation pro tabulku (tu zdědí vytvářené sloupce, u kterých nebyla explicitně vybrána), i celá databáze (tu zase kaskádově zdědí v ní vytvářené tabulky, pokud pro ně není výslovně nastavena). I jednotlivé collations jsou modulární (existují v podobě textových souborů). Jejich aktuální výčet lze zjistit příkazem SHOW COLLATION.

Související články

- LAMP

nástroje pro správu objektů v MySQL

- phpMyAdmin
- Adminer

Externí odkazy

- **(anglicky)** www.mysql.com ^[2] – Oficiální stránky
- MySQL na Wikiverzitě beta ^[3]
- Seriály na ABC Linuxu:
 - Tvorba databází v MySQL, 6 dílů ^[4]
 - Správa databází v MySQL, 3 díly ^[5]
- Český seriál o MySQL na serveru Linuxsoft ^[6]
- Český MySQL manuál ^[7]
- Český webhosting s podporou MySQL ^[8]
- **(anglicky)** MySQL tuning ^[9] - jak vyladit nastavení MySQL serveru

Reference

- [1] <http://www.mysql.org/>
- [2] <http://www.mysql.com/>
- [3] <http://cs.wikiversity.org/wiki/MySQL>
- [4] <http://www.abclinuxu.cz/clanky/navody/tvorba-databazi-v-mysql-i>
- [5] <http://www.abclinuxu.cz/clanky/navody/sprava-databazi-v-mysql-i>
- [6] http://www.linuxsoft.cz/article_list.php?id_kategorie=232
- [7] <http://www.junext.net/mysql>
- [8] <http://www.savana.cz/>
- [9] <http://www.mysqlperformanceblog.com/2006/09/29/what-to-tune-in-mysql-server-after-installation/>

SELECT

SQL příkaz **SELECT** vrací množinu záznamů z jedné a nebo více tabulek.

Syntaxe

```
SELECT

[ALL | DISTINCT]

{[tabulka. | alias. | pohled.]* | sloupec | sloupec AS alias}

| AVG([tabulka. | alias. | pohled.]<sloupec>) [AS <alias>]

| MIN([tabulka. | alias. | pohled.]<sloupec>) [AS <alias>]

| MAX([tabulka. | alias. | pohled.]<sloupec>) [AS <alias>]

| COUNT([tabulka. | alias. | pohled.]<* | sloupec>) [AS <alias>]

}[, ...n]

[INTO jméno_nové_tabulky]

FROM <tabulka> [AS <alias>][, ... n]

[[INNER | FULL] JOIN <tabulka> ON <spojovací podmínka>

| <LEFT | RIGHT> OUTER JOIN <tabulka> ON <spojovací podmínka>

| CROSS JOIN <sloupec>

[AS <alias>]

[, ... n]]

[WHERE <podmínky>

| <sloupec> <operator> <sloupec | hodnota>

| <sloupec> <operator> <sloupec | hodnota> <AND | OR | NOT> <sloupec> <operator> <sloupec | hodnota>

| <sloupec> BETWEEN <hodnota> AND <hodnota>

| <sloupec> LIKE <regularní výraz>

| <sloupec> IN <vycet hodnot>

| <sloupec | výraz> <operator> ANY | SOME (poddotaz)

| EXISTS (poddotaz)]

[GROUP BY <nazev sloupce>[, ... n]]

[HAVING <omezující podmínka postavena na výsledcích klauzule GROUP BY>]

[ORDER BY <sloupec>[, ... n] [ASC | DESC]]
```

```
[UNION <SELECT dotaz>]
```

Míra implementace SQL dotazů se liší u každého SŘBD, proto je třeba mít při psaní konkrétních dotazů na zřeteli konkrétní SŘBD, na kterém bude dotaz prováděn. Bližší informace naleznete v referenčních manuálech.

Příklad

```
SELECT id, zakaznik, cena FROM smlouvy WHERE cena>10000 AND se_slevou=1  
ORDER BY cena DESC
```

Další vlastnosti

DISTINCT

Klíčové slovo `DISTINCT` (někdy používáno `DISTINCTROW`) z výpisu odstraní záznamy, které se v dané hodnotě pole opakují. Výsledkem pro daný sloupec bude seznam všech hodnot (vyhovující případné podmínce výpisu); každé zastoupené jen jednou. `SELECT s DISTINCT` vypisuje z logických důvodů většinou jen jedno pole.

Omezení počtu zobrazených řádků

Databázové stroje většinou umožňují pomocí nějakého klíčového slova v SQL omezit počet vybraných řádků na určitou hodnotu.

TOP

Databáze Microsoft Access, MSSQL mají klauzuli `TOP`, která se vkládá hned za `SELECT...`

```
SELECT TOP 10 jmeno_skladby FROM zebricek_skladeb ORDER BY  
poslouchanost DESC;
```

LIMIT a OFFSET

Databáze MySQL, PostgreSQL mají klauzuli `LIMIT`, která kromě maximálního počtu zobrazených řádků umožňuje určit i od jakého místa (ofsetu) z výsledných řádků dotazu (pomyslného celkového výběru) má vrácení výsledku začít. Například dotaz

```
SELECT jmeno_skladby FROM zebricek_skladeb ORDER BY poslouchanost LIMIT  
5,10;
```

by zobrazil záznamy na 5. až 15. místě. Je možné použít

- `LIMIT` s jedním parametrem – maximálním počtem vypsání řádků
- `LIMIT` se dvěma parametry – první je maximální počet vypsání řádků a druhý pozice, od které má výpis začínat (offset)
- `LIMIT` v kombinaci s klíčovým slovem `OFFSET` – místo varianty dvou čísel oddělených čárkou

```
SELECT jmeno_skladby FROM zebricek_skladeb ORDER BY poslouchanost LIMIT  
10 OFFSET 5;
```

SQL_CALC_FOUND_ROWS

Konkrétně MySQL navíc podporuje klíčové slovo `SQL_CALC_FOUND_ROWS` (není součástí žádného SQL standardu), které se umísťuje za `SELECT` a způsobí, že databázový stroj si i přes omezení dané klíčovým slovem `LIMIT` ve výběrovém dotazu uloží celkový počet záznamů splňujících podmínku v klauzuli `WHERE` (pokud je zadaná) a ten pak může poslat jako výsledek dotazu:

```
SELECT FOUND_ROWS ()
```

Výhodou je, že pro zjištění celkového počtu řádků nemusí být spouštěn další dotaz.

Externí odkazy

- implementace `SELECT` v MySQL ^[1]
- implementace `SELECT` v PostgreSQL ^[2]
- implementace `SELECT` v Microsoft SQL Serveru ^[3]

Reference

[1] <http://dev.mysql.com/doc/refman/5.0/en/select.html>

[2] <http://www.postgresql.org/docs/8.1/interactive/queries.html>

[3] <http://msdn2.microsoft.com/en-us/library/ms189499.aspx>

Agregační funkce

Agregační funkce jsou v SQL statistické funkce, pomocí kterých systém řízení báze dat umožňuje seskupit vybrané řádky dotazu (získané příkazem `SELECT`) a spočítat nad nimi výsledek určité aritmetické nebo statistické funkce. Agregací funkce se v SQL používají s konstrukcí `GROUP BY`.

Agregace

Při výběru řádků z tabulky (nebo tabulek) většina databázových strojů relačních databází dovoluje výsledné řádky seskupit (agregovat) podle zadaného sloupce nebo výrazu z nich složeného pomocí syntaktické konstrukce `GROUP BY`. Při použití této konstrukce lze získat i výsledek některých statistických funkcí (nejčastěji je to aritmetický průměr, minimum, maximum, složitější, tzv. populační funkce, směrodatné odchylky a variance; nebo naopak obyčejný součet či počet řádků, které byly pod každou jednotlivou hodnotou seskupeny).

Vícenásobné seskupení

Databázové stroje většinou podporují i vícenásobné seskupení / seskupení podle více sloupců (nebo výrazů z nich složených). V takovém případě se názvy sloupců za klíčovým slovem `GROUP BY` oddělují čárkou. Sloupec (nebo výraz) uvedený jako první má nejvyšší prioritu, poslední sloupec nebo výraz má prioritu nejnižší.

Nejčastější agregační funkce

Následující tabulka popisuje zkratky pro nejběžnější agregační funkce. Podpora těch ostatních může být závislá na konkrétním databázovém systému; taktéž zkratky se mohou lišit.

Název	Popis
AVG ()	Aritmetický průměr
SUM ()	Součet
COUNT ()	Počet
MIN ()	Minimum
MAX ()	Maximum

Uvnitř závorek se předpokládá použití některého z polí z tabulky, popř. hvězdička (*), reprezentující celý řádek. Povolené jsou také výrazy z názvů polí i literálů.

COUNT

Funkce COUNT () se od ostatních agregačních funkcí v několika věcech odlišuje:

- Zatímco ostatní svůj výsledek vrací nad specifikovanými sloupci nebo výrazy z nich, COUNT () vrací počet záznamů, které vyhovují zadané podmínce resp. seskupení, a proto je jedno, který ze sloupců má jako argument (a často se používá hvězdičková konvence).
- Výjimkou výše uvedeného bodu je případ, kdy je potřeba vrátit počet unikátních hodnot určitého sloupce použitého v tabulce. Pak se ke COUNT () přidává klíčové slovo DISTINCT:

Příklad:

```
SELECT COUNT(DISTINCT zeme) FROM navstevnici;
-- Z kolika zemí přicházejí návštěvníci?
```

- V případech, že nebyl nalezen ani jeden odpovídající řádek, COUNT () vrátí nulu (ostatní agregační funkce vrací hodnotu NULL).
- COUNT () může být v SQL voláno bez konstrukce GROUP BY. Touto výjimkou je SQL dotaz nad celou tabulkou (např. dotaz na maximální hodnotu určitého sloupce z celé tabulky), který vrací toliko jeden řádek:

```
SELECT COUNT(*) FROM zamestnanci;
-- Vrať počet řádků v tabulce `zamestnanci`.
```

Některé databázové systémy mají počet položek svých tabulek uloženy zvlášť, takže pokud není dotaz tohoto typu nijak dále omezen (jako ve výše uvedeném příkladu), „sáhnou“ si pro výsledek do informací o tabulce a k jejímu procházení vůbec nedojde (což se pozitivně projeví na rychlosti a zátěži na paměťové médium).

GROUP_CONCAT

GROUP_CONCAT () je speciální agregační funkce, kterou nabízejí některé databázové systémy (mezi nimi například MySQL). Jejím výsledkem je nikoli počet ale výčet nalezených hodnot, oddělených čárkou nebo jiným oddělovačem. Pro různé číselníky apod. tak může být GROUP_CONCAT () velice užitečná – bez ní by bylo potřeba hodnoty vybrat jiným SQL dotazem, výsledek projít záznam po záznamu a hodnoty zapsat jednu za druhou do pomocné řetězcové proměnné. U příliš obsáhlých tabulek může ovšem výsledek přesahovat maximum toho, co databázový systém může vrátit, a je třeba na to dávat pozor. Jako u COUNT () i u této funkce lze GROUP_CONCAT () kombinovat s klíčovým slovem DISTINCT pro eliminaci vícekrát se vyskytujících hodnot. Navíc lze výčet seřadit (vložením klauzule ORDER BY) a též si přizpůsobit formát výpisu specifikováním jiného oddělovače než defaultní čárky (uvedeným v klauzuli SEPARATOR).

Příklad:

```
SELECT student_name,
GROUP_CONCAT(DISTINCT test_score ORDER BY test_score DESC SEPARATOR ' ')
```

```
' )  
FROM student  
GROUP BY student_name;  
-- Z tabulky studentů vypíše jméno a unikátní skóre, seřazené sestupně,  
oddělené mezerou.
```

HAVING

Syntaktická konstrukce `HAVING`, za kterou následuje omezující podmínka, umožňuje omezit řádky, které se budou ve výsledku agregovat a u kterých se budou počítat výsledky agregačních funkcí. Na rozdíl od podmínek v klauzuli `WHERE`, kde to není povoleno, dovoluje podmínka v klauzuli `HAVING` používat agregační funkce. Sloupce, které se objeví v agregačních funkcích za `HAVING`, musejí být uvedeny v sekci za `GROUP BY`. Návrh SQL používá dvě různé konstrukce pro omezující podmínky výběru podle výskytu či absence agregačních funkcí proto, že agregovaný výběr se od toho běžného výrazně liší (co do výkonu, zapojení složitějších algoritmů, atd.).

Externí odkazy

- SQL dotazy s agregací ^[1], interval.cz
- SQL - Co jsou to ty agregace? ^[2], Živě.cz
- Agregační funkce ^[3] pro MySQL (anglicky)

Reference

[1] <http://interval.cz/clanky/sql-dotazy-s-agregaci/>

[2] <http://www.zive.cz/default.aspx?article=4641>

[3] <http://dev.mysql.com/doc/refman/5.0/en/group-by-functions.html>

Zdroje článků a přispěvatelé

Systém řízení báze dat *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=10751337> *Přispěvatelé:* Beren, Bilboq, Jendakol, Jvs, Kloban, Ludek, Mikiqex, Mormegil, Pastorius, Podvečerníček, Serval, ToOb, 6 anonymní úpravy

Databáze *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=11095413> *Přispěvatelé:* Beren, BilboqCyborg, Black.john, Bruce Shorty, CYBERDEMON, Hashar, HePal, Hidalgo944, Jana Lánová, Jvs, Kozuch, Lenka64, Li-sung, Matěj Suchánek, Mercy, Milda, MiroslavJosef, Mojza, Mormegil, NIGKDO, Palovska, Pasky, Pastorius, Petr Kopač, Porthos, Postrach, Quentar, Rashack, Serval, Spock lone wolf, Strepon, Sumivec, Tlusfa, Wiki-vr, Xzajic, ZbR, Zipacna1, 48 anonymní úpravy

Relační databáze *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=11082501> *Přispěvatelé:* A0, Bilboq, DaBler, Dflanderka, Hidalgo944, Honza Záruba, Jana Lánová, Jj14, Kejv2, Lesan, Lms, Mage1987, Milan Keršláger, Mirek256, MiroslavJosef, Mojza, Nolanus, Palovska, Pistekjakub, Postrach, Radulinek, Rionka, Slady, Tchoř, Tlusfa, Tomas Linka, VitaJindra, Vrba, 40 anonymní úpravy

Relační model *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=9837886> *Přispěvatelé:* Aktron, HePal, Jvs, Knytttr, Kyjo, Milan Keršláger, Mojza, P.matel, Palovska, Pastorius, Vebloud, 9 anonymní úpravy

Trigger (databáze) *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=10325408> *Přispěvatelé:* AL3X, Beren, Che, Dj.yogi, Elm, Hidalgo944, Michalsjx, MiroslavJosef, PaD, Palovska, Pastorius, Postrach, Ty-Dyt, 7 anonymní úpravy

Referenční integrita *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=9846856> *Přispěvatelé:* Beren, Che, Milda, Mojza, Palovska, Pastorius, Podvečerníček, Vebloud, ŠJů, 5 anonymní úpravy

Databázová integrita *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=10771715> *Přispěvatelé:* Elm, Jvs, MiroslavJosef, Palovska, Podvečerníček, ToOb, 1 anonymní úpravy

Kandidátní klíč *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=10282197> *Přispěvatelé:* Jana Lánová, Kaluzman, Nick519, P.matel, Palovska, 2 anonymní úpravy

Primární klíč *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=9846163> *Přispěvatelé:* Che, G3ron1mo, Hidalgo944, Hugo, Jvs, Mercy, Milan Keršláger, Misa.vacha, Mojza, P.matel, Pastorius, Podvečerníček, 14 anonymní úpravy

Cizí klíč *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=10795565> *Přispěvatelé:* Bab dz, Che, Hidalgo944, Lms, Mirek256, Mormegil, Palovska, Solomiyka, ZbR, 10 anonymní úpravy

Index (databáze) *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=9969881> *Přispěvatelé:* Ben Skála, Beren, Broucek, Chrupoš, DaBler, Dflanderka, G3ron1mo, Hidalgo944, Jvs, Kejv2, Kočičác Bonifák, LiMr, MiroslavJosef, Palovska, 6 anonymní úpravy

Normalizace databáze *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=10779762> *Přispěvatelé:* Achse, Chmee2, DaBler, Iwbrowse, Japo, Mage1987, MartinAFC, Matěj Suchánek, Pan BMP, Pastorius, Quentar, RomanM82, Snoogans, ToOb, Vebloud, 24 anonymní úpravy

Třetí normální forma *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=10429607> *Přispěvatelé:* Adam Zivner, Anomen, Chrupoš, GermanX, Iwbrowse, Vebloud, 5 anonymní úpravy

SQL *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=9837297> *Přispěvatelé:* Assax, Banter, Chrupoš, DaBler, Danny B., Elwikipedista, Franci, Gosht, Hidalgo944, Honys, Hugo, Jana Lánová, Karakal, Kronn, Ludek, Lzur, Mad, Mercy, Mojza, Mormegil, Palovska, Podvečerníček, Rashack, Skim, Zagothal, ZbR, 40 anonymní úpravy

Příkazy jazyka SQL *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=9182092> *Přispěvatelé:* Hidalgo944, Jj14, Miraceti, Zagothal

MySQL *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=10838335> *Přispěvatelé:* Adam Zivner, Aleš Tošovský, Che, Clary.Aldringen, CommonsDelinker, DaBler, Danny B., Elwikipedista, Franci, G3ron1mo, HePal, Hidalgo944, Honza Záruba, Hugo, Ioannes Pragensis, JAn Dudík, Jan.kachlik, Jj14, Jvs, Jx, Kibitzer, Kodlodot, Kychot, Lms, Lubos, Macronyx, McFly, Mercy, Miloslav Ponkrác, Mojza, Mormegil, Palu, Panther7, Patrias, Pavel Cvrček, Podvečerníček, Porod, Postrach, Ps, Raysonho, Studioworks, Sumil, Tabovl, Tchoř, Xzajic, Zirland, 38 anonymní úpravy

SELECT *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=10262192> *Přispěvatelé:* AL3X, Danny B., Hidalgo944, Jednorázovka, Mormegil, PaD, Pastorius, Podvečerníček, ZbR, 9 anonymní úpravy

Agregační funkce *Zdroj:* <https://cs.wikipedia.org/w/index.php?oldid=10074259> *Přispěvatelé:* Hidalgo944, Jenikkozak, Martin Kozák, 5 anonymní úpravy

Zdroje obrázků, licence a přispěvatelé

Soubor:Wikiquote-logo.svg *Zdroj:* <https://cs.wikipedia.org/w/index.php?title=Soubor:Wikiquote-logo.svg> *Licence:* Public Domain *Přispěvatelé:* -xfi-, Dbc334, Doodletoo, Elian, Guillom, Jeffq, Krinkle, Maderibeyza, Majorly, Nishkid64, RedCoat, Rei-artur, Rocket000, 11 anonymní úpravy

Soubor:Broom icon.svg *Zdroj:* https://cs.wikipedia.org/w/index.php?title=Soubor:Broom_icon.svg *Licence:* GNU General Public License *Přispěvatelé:* Bayo, Booyabazooka, Davepape, Dcoetzee, Herbythyme, Ilmari Karonen, Javierme, Perhelion, Rocket000, TMg, The Evil IP address, 11 anonymní úpravy

Soubor:Mysql-screenshot.PNG *Zdroj:* <https://cs.wikipedia.org/w/index.php?title=Soubor:Mysql-screenshot.PNG> *Licence:* Public Domain *Přispěvatelé:* Dereckson, Stephantom

Soubor:Mysql_architecture_schema.png *Zdroj:* https://cs.wikipedia.org/w/index.php?title=Soubor:Mysql_architecture_schema.png *Licence:* Public Domain *Přispěvatelé:* Clary.Aldringen

Licence

Creative Commons Attribution-Share Alike 3.0
[//creativecommons.org/licenses/by-sa/3.0/](https://creativecommons.org/licenses/by-sa/3.0/)
