1001 TIPŮ A TRIKŮ PRO

Ľuboslav Lacko

SQL

Sbírka nejlepších programátorských postupů a řešení

CD obsahuje

veškeré zdrojové kódy z knihy,
SQL Server 2008 EE
a další užitečné nástroje

- Normalizace, pohledy, indexy, kurzory a transakce
- Výběr, vkládání, aktualizace a seskupování údajů
- XML, procedury, funkce, komprese a zálohování
- Optimalizace dotazů a struktur, ukládání multimédií

CPRESS

Luboslav Lacko

1001 tipů a triků pro SQL

Computer Press, a. s.
Brno
2011

1001 tipů a triků pro SQL

Luboslav Lacko

Computer Press, a. s., 2011. Vydání první.

Překlad: Lukáš Krejčí
Jazyková korektura: Martina Mojzesová
Sazba: René Kašík
Rejstřík: Daniel Štreit
Obálka: Martin Sodomka
Komentář na zadní straně obálky: Martin Herodek

Technická spolupráce: Jiří Matoušek, Zuzana Šindlerová, Dagmar Hajdajová
Odpovědný redaktor: Martin Herodek
Technický redaktor: Jiří Matoušek
Produkce: Petr Baláš

Computer Press, a. s.,
Holandská 3, 639 00 Brno

Objednávky knih:
http://knihy.cpress.cz
distribuce@cpress.cz
tel.: 800 555 513

ISBN 978-80-251-3010-0
Prodejní kód: K1932
Vydalo nakladatelství Computer Press, a. s., jako svou 4133. publikaci.

Stručný obsah

Obsah

Úvod

Moderní databáze už dávno neslouží jen pro bezpečné a spolehlivé ukládání údajů, nové verze poskytují stále více služeb pro podnikové informační systémy, ať už se jedná o analýzu, reportování či práci s geografickými informacemi. Moderní webové aplikace a služby pracují hlavně s nerelačními údaji čili s dokumenty a multimédii. Stále větší oblibě se těší dokumenty XML. Proto je i tato publikace koncipována tak, aby na praktických ukázkách demonstrovala možnosti moderních databázových platforem.

Komu je kniha určena

Tipy jsou rozděleny do tří kategorií pro začátečníky, pokročilé a znalce. Rozdělení se ale netýká tolik náročnosti, jako spíše jejich potřebnosti.

Tip je určen začátečníkům, kteří se s SQL a databázemi seznamují. Nevyžaduje žádné znalosti.

Tip je určen těm, kteří již zvládají základy SQL. Najdete zde pokročilejší techniky.

Tip se zaměřuje na řešení nestandardních a komplikovaných problémů. Je určen pokročilým programátorům.

Konvence použité v knize

V knize se používá neproporcionální písmo pro ukázky dotazů a jejich výstupů. V ukázkách mají dotazy šedé pozadí, aby je čtenář snadněji odlišil od výstupů databázového systému. Dotazy použité v knize najdete i na přiloženém CD.

Doprovodné CD

Doprovodný disk obsahuje kromě zdrojových kódů také řadu odkazů na užitečné stránky a také několik užitečných nástrojů, jež vám práci s databázemi a SQL výrazně usnadní nebo alespoň zpříjemní.

CD stačí vložit do počítače a rozhraní se spustí automaticky. Pokud nemáte automatické spouštění disků povoleno, vyhledejte na CD kořenový adresář a otevřete soubor spustit_CD.html.

Jestliže rozhraní CD otevřete v prohlížeči Internet Explorer, Opera nebo Google Chrome, budete z CD moci instalovat doprovodný software okamžitě. V případě jiných prohlížečů se zobrazí výzva k uložení instalačního souboru na pevný disk. V tomto případě doporučujeme spustit instalaci přímo z CD. Obsah CD najdete ve složce obsah.

Zpětná vazba od čtenářů

Nakladatelství a vydavatelství Computer Press, které pro vás tuto knihu připravilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

redakce PC literatury
Computer Press
Spielberk Office Centre
Holandská 3
639 00 Brno

nebo

sefredaktor.pc@cpress.cz

Computer Press neposkytuje rady ani jakýkoli servis pro aplikace třetích stran. Pokud budete mít dotaz k programu, obraťte se prosím na jeho tvůrce.

Errata

Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nedá. Pokud v některé z našich knih najdete chybu, ať už chybu v textu nebo v kódu, budeme rádi, pokud nám ji nahlásíte. Ostatní uživatele tak můžete ušetřit frustrace a pomoci nám zlepšit následující vydání této knihy.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/k1932* po klepnutí na odkaz Soubory ke stažení.

Výběr vhodné databáze

1 Databáze jako základní pilíř informačního systému

Základem každého informačního systému je dobře navržená databáze, od níž očekáváme spolehlivost a stabilitu, rychlost, bezpečnost uložených dat a víceuživatelský přístup. Pro každou společnost je databáze životně důležitý orgán, který žije, dýchá a chrání cenná data.

V současnosti existuje více softwarových společností, které nabízejí relační databáze, jako jsou například Oracle, Microsoft, IBM a mnoho dalších. Otázkou je, zda si vybrat volně dostupnou nebo komerční distribuci databáze. Každá z databází má však osobitně specifikované hardwarové nároky co se týče procesorové kapacity, paměti a diskového prostoru.

2 Výběr databáze pro informační systém

Použijeme analogii z oblasti letecké dopravy. Zdálo by se, že kromě variability vnitřního vybavení kabiny a vnějšího nátěru nemají zákazníci neboli letecké společnosti mnoho možností pro přizpůsobení letadla, které přebírají od výrobce. Omyl. Zákazník si musí jednu z nejdůležitějších součástí, motor, vybrat sám. K dispozici jsou motory od různých dodavatelů. Například pro Airbus jsou k dispozici motory Rolls-Royce, General Electric, Pratt & Whitney... Letecké společnosti si dodavatele motorů vybírají podle dlouhodobých zkušeností, možností servisu, přičemž se samozřejmě snaží o co největší unifikaci. Podobná situace je výběru databáze pro informační systém.

Poznámka: Databázový server se svým IT okolím je zpravidla poměrně velký investiční celek, takže do hry vstupují různá výběrová kritéria.

Databázové servery jsou specializované uzly informačních systémů zaměřené na databázové zpracování, přičemž plní funkci distribuovaného systému řízení báze dat. Zabezpečují operace zadávání údajů, jejich modifikace a vyhledávání.

Moderní databázový systém vytváří základ stabilního a kvalitního přístupu k údajům. Disponuje moderními bezpečnými technologiemi a díky své struktuře klient-server-databáze umožňuje přistupovat k údajům systému přes Internet prakticky odkudkoli na světě.

3 Jak bude databáze v rámci informačního systému používána?

Bude databáze využívaná hlavně jako úložiště operačních dat pro transakce (OLTP), nebo budou údaje skladovány a analyzovány (OLAP)? OLTP (Online Transaction Processing) je technologie uložení dat v databázi, která umožňuje jejich co nejjednodušší a nejbezpečnější modifikaci ve víceuživatelském prostředí. Jedná se o přístup,

který se v současnosti používá v převážné většině databázových aplikací (z historického hlediska ještě poměrně nedávno dokonce ve všech databázových aplikacích). Jako protiklad k OLTP se především pro analytické účely nad rozsáhlými databázemi používá technologie OLAP (Online Analytical Processing). Základní rozdíl mezi databázemi OLAP a OLTP spočívá ve způsobu jejich používání. Zatímco v databázi OLTP se data často modifikují, v databázi OLAP jde o ukládání velkého množství dat, která se často nemění a nad nimiž se vykonávají složité databázové dotazy. Výběr správného typu databáze může výrazně ovlivnit chod a výkon aplikace.

4 Úloha databázového systému v podnikové informatice

Paralelně s vývojem databázových technologií začal nastupovat nejen přechod na architekturu klient-server, ale i trend decentralizace informačních technologií. Vynutila si ji globalizace ekonomiky a s ní související dynamické změny v řízení a struktuře firem. Databázový server se stal jedním ze základních pilířů třívrstvé architektury klient-server. Databázový a aplikační server lze realizovat na tomtéž hardwarovém serveru, jako middlewarová vrstva nebo na samostatných uzlech.

5 Vývojem prošly nejen technologie, ale i cenová politika

Není to tak dávno, co licence pro jeden procesor u některých špičkových databází stála několik desítek tisíc dolarů. Avšak možnosti odbytu na tyto trhy nejsou pyramidová hra a trh zákonitě začal saturovat. Postupně, s nasycením trhu informačních systémů pro velké firmy, si začali jejich dodavatelé stále více všímat i sektoru SMB. Začalo to databázovými servery. U nejnovějších databází velkých firem (Oracle 11g, IBM DB2, Microsoft SQL Server 2008) lze vypozorovat nejen zařazení verzí situovaných pro SMB, ale dokonce i volně šiřitelné verze „Express" pro nasazení mimo komerční sféru. Hlavní výhodou edicí SMB a „Express" je možnost rozběhu businessu s velmi nízkými náklady, přičemž v případě potřeby je možné bez jakýchkoli úprav přejít na kteroukoli komerční verzi na vhodné velikosti serveru.

6 Komerční produkt versus Open Source

Velmi často se vedou nekonečné polemiky, zda vybrat komerční produkt, nebo produkt s licencí Open Source. Zkuste si představit polemiku, zda použít nejznámější komerční databázovou platformu Oracle nebo nejpopulárnější databázi MySQL s licencí Open Source. Databázová platforma MySQL patří společnosti Sun Microsystems, která se po akvizici ocitla ve vlastnictví společnosti Oracle.

7 Kritéria pro výběr databáze

Mnohé subsystémy podnikové informatiky umožňují variabilní výběr databázové platformy. Typickým příkladem je ERP a systém SAP. Před výběrem vhodného databázového systému je nutné zvážit velké množství faktorů. Jde nejen o výkonová, kapacitní a bezpečnostní kritéria, ale také o účel, z něhož vyplývá předpokládané zatížení. Důležitým faktorem je existence informační infrastruktury ve formě a potenciál lidských zdrojů.

8 Jaká edice databáze je vhodná pro konkrétní informační systém?

Komerční databázové servery se dodávají ve více edicích, takže umožňují škálovatelnost a optimální možnost výběru vhodné verze v souvislosti s předpokládanými scénáři nasazení a objemu zátěže. Názvy edic se na jednotlivých platformách liší, nicméně při troše nadhledu se dají zevšeobecnit.

9 Co umí edice Enterprise

Pod tímto názvem se skrývá ucelená datová platforma splňující vysoké nároky podnikových aplikací pro zpracování transakcí online a pro datové sklady. Umožňuje konsolidovat servery a vykonávat online zpracování velkého objemu transakcí a generování sestav. Díky technologiím, jež chrání data před nákladnými lidskými chybami, zajišťuje obchodní kontinuitu a zkracuje čas potřebný pro obnovení po havárii.

Vytváří infrastrukturu s ověřenými schopnostmi zpracování velkých množství dat a vysokého podnikového zatížení. Splňuje požadavky na ochranu osobních dat a soulad s legislativními normami a nabízí integrované funkce pro ochranu dat před neoprávněným přístupem. Verze Enterprise nabízí správu infrastruktury s automatickou diagnostikou, optimalizací a konfigurací s cílem snížit provozní náklady a současně omezit nutnost údržby a správy velkých objemů dat. Umožňuje dotazování a analýzu velkých množství dat v datových skladech a datových tržištích, a usnadňuje tak získávání širšího pohledu na tato data.

10 Typické scénáře nasazení pro edici Enterprise

- Provozování nenahraditelných aplikací pro správu dat se škálovatelností, vysokou dostupností a zabezpečením na podnikové úrovni.
- Správa online zpracování transakcí (OLTP) ve velkém objemu.
- Pokročilá analýza velkých objemů dat v datových skladech.
- Generování sestav na základě analýzy velkých objemů dat.

11 Co umí edice Standard

Přívlastek Standard naznačuje platformu pro správu dat s jednoduchou spravovatelností určenou primárně pro provoz aplikací firemních oddělení. Tato edice má zpravidla vyváženou cenu tak, aby se dala použít i pro sektor SMB čili pro malé a střední firmy. Ve většině případů poskytuje funkce pro zabezpečenou vzdálenou synchronizaci a správu.

12 Typické scénáře nasazení pro edici Standard

- Aplikace firemních oddělení požadující dobrou spravovatelnost a jednoduché použití.
- Aplikace pro online zpracování transakcí (OLTP) v malém až středním objemu.
- Systémy pro podporu rozhodování požadující základní funkce pro generování sestav a analýzy.

13 Co umí edice Web

Nabízí nízké náklady, vysokou škálovatelnost pro vysoko dostupné webové aplikace či hostovaná řešení a vysokou dostupnost internetových prostředí webových služeb. Používá se pro:

- Sdílená a dedikovaná hostitelská řešení.
- Rozsáhlé webové prezentace s primárně webovým obchodním modelem a rostoucí potřebou škálování.

14 Co umí edice Compact a Mobile

Tyto odlehčené edice jsou určeny pro „přibalení" k aplikaci a volné šíření spolu s aplikací. Jsou k dispozici bezplatně a umožňují vytvářet samostatné a příležitostně připojené aplikace pro mobilní zařízení, klientské počítače a webové klienty.

15 Co umí edice Express

Edice Express je k dispozici bezplatně a je ideální pro studium a vytváření aplikací pro klientské počítače a malé servery a pro distribuci nezávislými výrobci softwaru.

Poznámka: Edice Express je v rozsahu základní funkčnosti plně kompatibilní se všemi ostatními edicemi příslušné platformy (SQL Server, Oracle...). Bude-li vaše aplikace a firma úspěšná a nároky na výkon a škálovatelnost budou růst, je možné aplikaci bez úprav přesunout na požadovanou komerční verzi.

16 Typické scénáře nasazení pro edici Express

- Základní a studijní databáze.
- Funkčně bohaté aplikace pro klientské počítače.
- Bezplatné práva na distribuci pro nezávislé výrobce softwaru (ISV).

IT směruje do cloudu

Vybrat pro informační systém databázi Oracle, IBM DB2, SQL Server, Informix…? Třeba více napoví analogie s otázkou: Vybrat do firemního vozového parku jako další vozidlo BMW, Mercedes, Volkswagen...? Možná se vám tato analogie nelíbí, především z toho důvodu, že většina moderních firem už nemá vlastní vozový park, ale tuto službu outsourcuje. A proč neoutsourcovat ukládání údajů jako službu poskytovanou datovými centry? Koncepce IT jako služby přináší úplnou abstrakci údajů od technologických či komerčních aspektů jejich ukládání. U pronajatého vozidla záleží jen na jeho kategorii, ale ne na značce, protože případná unifikace kvůli údržbě a servisu je problém, který řeší poskytovatel služby a zákazníka nezajímá. Rovněž nebude u spolehlivě fungující služby pro ukládání údajů nikoho zajímat, na jaké platformě jsou údaje uloženy.

17 XML jako alternativa malé databáze?

Odpověď na otázku, kam s údaji, je zpravidla jednoduchá – do databáze. Skutečně je to ale vždy nejoptimálnější řešení? Abychom podali uspokojivou odpověď na tuto otázku a neodchýlili se k polemice komerční vs. volně šiřitelné databáze, zavedeme hypotetický předpoklad, že databázový server máme k dispozici zdarma. Rozhodující je množství údajů a náklady na instalaci a administraci. Představte si aplikaci například pro evidenci v půjčovně DVD (tu můžete koneckonců vytvořit v Excelu) nebo nějaký specializovaný program třeba pro malou restauraci a podobně. Kromě instalace aplikace je nutné nainstalovat databázový server, vytvořit databázi, v ní vytvořit struktury a nakonec je zpravidla zapotřebí naplnit databázi nějakými z krátkodobého hlediska konstantními údaji: budeme potřebovat například ceník, kurzovní lístek a podobně. Představa, že si to bude instalovat, udržovat a v případě nehody obnovovat například provozní v restauraci, která je jinak počítačově docela gramotná, je celkem nereálná. Řešením může být embedded databáze, případně údaje v dokumentu XML.

18 Microsoft SQL Server 2008

Aktuální komerční verze databázového serveru Microsoft SQL Server 2008 R2 je svými vlastnostmi, robustností a spolehlivostí předurčena i pro aplikace typu „mission-critical", přičemž současně snižuje nároky na infrastrukturu a její správu. Komerční verze je na trhu od prosince roku 2007, v době psaní této publikace už byla k dispozici první CTP verze nového databázového produktu společnosti Microsoft s kódovým označením Denali. Cennou devizou SQL Serveru 2008 je možnost jeho navázání na ostatní produkty a řešení společnosti Microsoft, především vývojového prostředí Visual Studio.

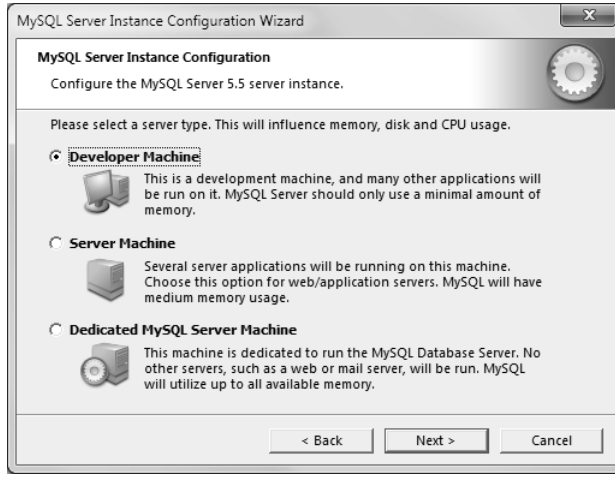
19 Je to ještě MySQL, nebo už Oracle?

Tento volně šiřitelný databázový server je oblíbený a nasazuje se hlavně u webových aplikací. Je portován pro Unix, Linux, Solaris, OS/2 i Windows. Je ideálním řešením pro malé firmy a amatérské využití, například pro publikační portál nebo webové aplikace pro sběratele, hobby, diskuzní fóra, seznamky a podobně. Instalační soubory databázového serveru MySQL je možné získat z webu, například ze stránky *www.mysql.org*.

20 MySQL: Instalace a konfigurace

Kompletní instalace je zabalená do jednoho souboru s velikostí přibližně 30 MB (pro Windows). Pro cvičné účely můžete použít databázi TEST, která se vytvoří při instalaci a je určena právě pro tyto účely. Pro zadávání příkazů můžete použít textovou konzolovou aplikaci MySQL Command Line Client. Do testovací databáze se přepnete příkazem:

```
USE TEST.
```



Obrázek 1: Databázový server a instanci databáze můžete nakonfigurovat pomocí nástroje MySQLInstanceConfig

Populární Open Source databázový server MySQL momentálně patří společnosti Oracle, která ho získala akvizicí společnosti Sun Microsystems. V průběhu akvizice ubezpečili Larry Ellison (CEO společnosti Oracle) spolu se Scottem McNealym (bývalý šéf společnosti Sun) příznivce této platformy, že bude i nadále podporována a rozvíjena, že databáze Oracle a MySQL si nikdy nebudou konkurovat. Nuže uvidíme.

Poznámka: Kromě „železa" neboli serverového portfolia, a databázové platformy získala společnost Oracle akvizicí Sunu i populární programovací jazyk Java, který je už dlouho jedním ze základních stavebních kamenů softwaru společnosti Oracle. Nejlépe tuto platformu charakterizuje jeden jediný údaj – implementace v 6,5 miliardách zařízení. Nejedná se jen o PC a mobily, ale i technologické a řídicí systémy a specializované mikročipy ve spotřební elektronice.

21 Oracle XE: konfigurace uživatelského přístupu

začátečník

Databáze Oracle Express Edition (XE) je postavena na kódové základně Oracle Database 11g (či 10g) a je plně kompatibilní se všemi ostatními edicemi databáze Oracle. Verze XE obsahuje plnou podporu jazyka SQL včetně jeho procedurální nadstavby PL/SQL. Databáze je k dispozici ke stažení na adrese *www.oracle.com/technology/xe*. Instalace databáze trvá přibližně tři minuty. Jediným údajem, který se v průběhu instalace zadává, je společné heslo pro účty SYS a SYSTEM.

Poznámka: Aby se tato volně šiřitelná verze nenasazovala do produkčních systémů, má určitá omezení. Databáze využívá maximálně jeden procesor či jedno jádro vícejádrového procesoru, udržuje využití paměti pod jedním gigabajtem, je omezena na jednu relaci v systému a umožňuje uchovávat maximálně čtyři gigabajty uživatelských dat.

22 Oracle XE: Využití cvičného schématu HR

začátečník

Pro svoje pokusy můžete vytvořit vlastní novou databázi nebo využít cvičnou databázi, která se nainstalovala spolu s databázovým serverem. Ve verzi databáze Oracle XE je implicitně nainstalováno **schéma testovacího klienta HR** (Human Resources – lidské zdroje). Schéma je po nainstalování databáze implicitně zamčeno, proto jako první administrátorský úkon proveďte její zpřístupnění a nastavení parametrů pro přihlášení klienta k této databázi.

Při prvním přístupu se musíte přihlásit jako administrátor, potřebujete-li odemknout testovací účet. Postup odemčení se pro jednotlivé verze liší. Uvedeme postup pro verzi Oracle XE 11g i pro starší verzi Oracle XE 10g.

23 Oracle 11g XE: Odemčení účtu

začátečník

V nabídce operačního systému spusťte konzolovou aplikaci Run SQL Command Line. Následně napište příkaz:

```
connect
```

a přihlaste se jako uživatel SYSTEM:

- Enter user-name: SYSTEM
- Enter password: <heslo, které jste zadali a potvrdili při instalaci databáze>

Odemkněte účet HR příkazem:

```
ALTER USER hr ACCOUNT UNLOCK;
```

Zadejte heslo, které budete pro tento cvičný účet používat:

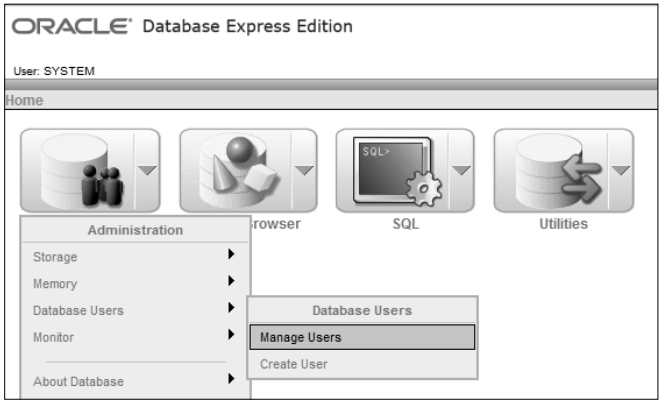
```
ALTER USER hr IDENTIFIED BY <heslo>;
```

24 Oracle 10g XE: Odemčení účtu

začátečník

Přihlaste se jako uživatel SYSTEM:

- Username: SYSTEM
- Password: <heslo, které jste zadali a potvrdili při instalaci databáze>



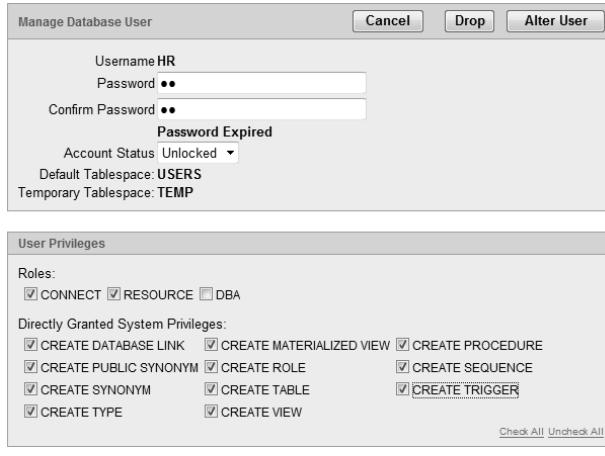
Obrázek 2: Odemčení schématu testovacího klienta se provede pomocí postupné aktivace ikon „Administrations“ a „Database Users“

Pro cvičné schéma HR zadejte heslo, nastavte parametr **Account Status** na hodnotu *Unlocked* a povolte přístup pro role CONNECT a RESOURCE. Vizuálně vykonané změny potvrdíte tlačítkem **Alter User**.

Poznámka: Vzhledem k tomu, že se jedná o cvičnou databázi, můžete povolit všechna oprávnění ve skupině Directly Granted System Privileges.

25 Oracle 10g XE: Nastavení oprávnění

Pro účely kurzu povolte všechna oprávnění ve skupině Directly Granted System Privileges.



Obrázek 3: Po spuštění webové konzole se nejdříve zobrazí dialogové okno pro přihlášení

26 Jak na praktické pokusy

Při praktických pokusech se neobejdete bez otestování možností jednotlivých triků a postupů, které vás zaujaly a plánujete je využívat. Po nainstalování libovolné databázové platformy se zpravidla nainstalují i výukové nebo testovací databáze s tabulkami, které se během instalace naplní cvičnými údaji. Máte tedy v podstatě tři možnosti. Buď můžete využít testovací tabulky nainstalované spolu s databázovým serverem, nebo využít tabulky, které se používají jako námět příkladů v této publikaci, anebo si můžete navrhnout vlastní tabulky z oblasti, v níž sami pracujete.

Poznámka: Doporučujeme vytvořit si dvě sady tabulek. Jedna sada bude testovací, to znamená, že ji vytvoříte a naplníte údaji jen jednou, a druhá sada bude cvičná. Do těchto tabulek můžete údaje zapisovat, případně je můžete editovat nebo vymazat.

27 Jak připravit cvičnou databázi

Naše cvičná a testovací databáze bude obsahovat tři jednoduché testovací tabulky `zakaznici`, `objednavky` a `zbozi` naplněné údaji. Mezi těmito tabulkami jsou náznaky

určitých vazeb – i laik odhalí souvislost, že objednávku provedl nějaký zákazník a že objednávka obsahuje zboží. Návrhové struktury jsou zřejmé z tabulkového přehledu:

Tabulka `zakaznici`

Název	Datový typ
id_zak	int
firma	varchar(20)
kontakt_jmeno	varchar(20)
adresa	varchar(20)
mesto	varchar(15)
obrat	money
dluh	money

Tabulka `objednavky`

Název	Datový typ
id_obj	int
id_zak	int
datum_obj	datetime

Tabulka `zbozi`

Název	Datový typ
id_zbo	int
id_obj	int
nazev	varchar(30)
jedn_cena	money
mnozstvi	int

Skripty pro vytvoření a naplnění tabulek pro jednotlivé platformy najdete v příloze a na webu.

28 Co ukládá tabulka pro testování výrazů a funkcí

Kromě klasických cvičných tabulek budete potřebovat ještě jednu jednoduchou tabulku na procvičování funkcí. Pro tento účel postačí jednoduchá testovací tabulka `TEST`, která má jen jeden sloupec `NN VARCHAR2(1)`. V tabulce je uložen jeden záznam, který ve sloupci `NN` obsahuje hodnotu „X“.

Základy databázové teorie

29 Co je to databáze, server, platforma

Pochopení pojmů „relační databáze“ a „databázová platforma“ není úplně triviální záležitost. Relační databáze je definována jako soubor nástrojů pro efektivní a spolehlivé ukládání údajů a pro manipulaci s těmito údaji. Naproti tomu pochopení pojmu databázová tabulka nečiní problém téměř nikomu. Ještě ošemetnější je situace u na první pohled jednoduchého pojmu databáze. Pod pojmem databáze si můžete představit prakticky cokoli. Od skříňové kartotéky u lékaře přes „kapesní databanky“, různé záznamy až po údaje organizované pod správou databázového serveru. Proto je na tomto místě potřebné objasnit pojmy, které budeme v publikaci využívat.

- **Databáze** – pojem zapouzdřuje údaje a nástroje pro jejich ukládání a manipulaci s těmito údaji.
- **Databázový server** – představuje soubor softwarových prostředků jednak pro práci s údaji, ale i pro organizování a realizaci přístupu klientů k těmto údajům.
- **Databázová platforma** (Oracle, SQL Server, IBM DB2, MySQL...) – pojem zapouzdřuje databázi, databázový server, soubor nástrojů pro správu a zabezpečení údajů v databázi.

30 Co je to databázový systém

Databázový systém je tvořen systémem řízení báze dat (SŘBD) a databází. Databázové systémy mohou být:

- hierarchické a síťové, kdy jsou aplikační programy závislé na databázi, z čehož vyplývá například problematická údržba a podobně;
- relační, pro něž je typická neprocedurální manipulace s daty, ukládání jednoduchých dat s pevnou strukturou, tedy v tabulkové formě;
- objektové databázové systémy používají složité datové struktury a složitá pravidla založená na obchodní logice (business rules).

31 Co znamená transakční zpracování údajů

Transakční zpracování znamená, že složitější manipulace s údaji, která se skládá z posloupnosti určitých kroků, se vykonává jako transakce, která převede databázi z jednoho konzistentního stavu do druhého. Zjednodušeně řečeno, buď všechny operace v transakci proběhnou úspěšně, nebo neproběhnou vůbec a databáze se uvede do konzistentního stavu, v jakém byla před transakcí.

32 Zotavení z chyb a nehod

Pokud při zpracování údajů, a to ve všeobecnosti, nejen při transakcích, dojde k chybě, systém se musí z této chyby zotavit. Dojde-li vlivem nějaké nepředvídané události, hává-

rie nebo živelní pohromy ke ztrátě údajů, měl by se systém po těchto událostech zotavit a pokračovat v práci například ze záložní databáze a podobně.

33 Jak na víceuživatelský přístup

Víceuživatelský přístup předpokládá efektivní řízení přístupu k údajům ze strany více uživatelů nebo klientských aplikací. SŘBD proto musí umožňovat definování přístupových práv jednotlivých uživatelů k databázovým objektům a specifikovat rozsah jejich oprávnění. Někteří uživatelé mohou údaje do databáze zapisovat, případně je mazat, zatímco jiní mají zpřístupněné jen čtení údajů.

34 Jak se definuje ochrana údajů

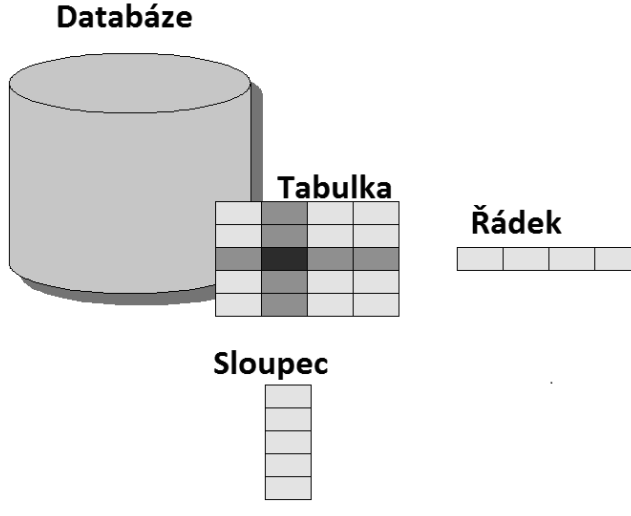
Pojem ochrana údajů lze definovat jako ochranu před ztrátou údajů, například při haváriích hardwaru a podobně. Můžete jej ale taktéž definovat jako ochranu údajů před jejich možnou krádeží či zneužitím, případně jako nutnost zabránění přístupu neoprávněných osob k údajům v databázích.

35 Co jsou to databázové tabulky

Údaje jsou v prostředí relačních databází uloženy v databázových tabulkách. Je to jednoduchá, dvojrozměrná struktura, formou téměř shodná s tabulkami ve formulářích nebo s tabulkami, s nimiž pracujeme v tabulkových kalkulátorech (Excel...). Strukturu takovéto tabulky zná každý – skládá se z řádků a sloupců.

Sloupec (atribut) je množina údajů jediného datového typu v tabulce. Někdy se sloupce označují i jako atributy.

Řádek (záznam) je kombinace sloupcových hodnot v tabulce. Každý řádek je zpravidla jednoznačně identifikován jedním sloupcem, který musí tím pádem obsahovat unikátní hodnoty a který se označuje jako primární klíč.



Obrázek 4: Vztah mezi databází, tabulkou, řádkem a sloupcem

Hodnoty (údaje) se nacházejí v průsečících řádků a sloupců.

Primární klíč je sloupec, případně kombinace několika sloupců, které slouží k identifikování každého řádku tabulky.

36 Relační vztahy mezi údaji uloženými ve více databázových tabulkách

Údaje jsou v databázích uloženy zpravidla ve více tabulkách, mezi nimiž jsou relační vazby. Definujme si některé pojmy z oblasti relačních databází:

- **Doména** je množina hodnot stejného významového typu.
- **Relace** je podmnožina kartézského součinu nad několika doménami, množina vztahů mezi prvky několika domén, dá se zachytit jako tabulka.
- **Atribut relace** představuje jméno použité hodnoty z domény v relaci.
- **Relační schéma** se dá vyjádřit jako jméno relace + jména atributů, v čase je stálé na rozdíl od těla relace.
- **Klíč** – atribut (či skupina atributů), jehož hodnota (kombinace hodnot) identifikuje n-tici relace.

37 Jaké existují typy domén

Doména je množina hodnot stejného typu pro daný atribut. Reprezentuje definiční obor příslušného atributu. Domény mohou být jednoduché a kompozitní.
Jednoduchá doména se váže na jednoduchý atribut a reprezentuje jen hodnoty tvořící přípustnou množinu hodnot tohoto atributu.

Kompozitní doména je definována jako kartézský součin hodnot jednoduchých domén atributů tvořících kompozitní doménu.

Poznámka: Typickým příkladem kompozitní domény je datum, které se skládá z atributů den, měsíc a rok.

38 Jaký je význam domén

Domény mají význam hlavně při relačních operacích, jako je porovnávání či spojování. Budete-li tedy provádět operace předpokládající příslušnost do stejných domén, je nutné definovat stejné domény pro atributy v různých relacích.

39 Co říkají podmínky relačnosti

Teorii relačních databází velmi pěkně a jednoduše vystihují tři podmínky minimální relačnosti, které byly zformulovány průkopníkem v této oblasti, Dr. E. F. Coddem:

- Všechny údaje v databázi jsou uloženy v tabulkách.
- Fyzická struktura údajů a jejich uložení jsou nezávislé a úplně od uživatele odstraněné, což znamená, že neexistují žádné pro uživatele viditelné přístupové cesty (včetně indexů).

- Pro práci s údaji v databázi předpokládáme existenci databázového jazyka, který umožňuje realizovat minimální operace selekce, restrikce, projekce a spojení.

40 Co jsou to integritní omezení

Pod pojmem integritní omezení si můžeme představit pravidla pro zajištění správnosti a konzistence uložených dat. Integritní omezení můžete zavést na třech úrovních:

1. **Entitní integrita** – zajištění jednoznačné identifikace každého řádku relace – jednoznačný primární klíč.
2. **Doménová integrita** – zajištění, aby každá hodnota atributu byla v souladu s množinou přípustných hodnot.
3. **Referenční integrita** – cizí klíče (tj. atributy či skupina atributů tvořící v jiné relaci primární klíč) nemohou nabývat hodnoty, které jsou v rozporu s hodnotami odkazovaného primárního klíče – různé způsoby zajištění.

41 Co je to primární klíč

Primární klíč je jednoznačný identifikátor každého záznamu. Může to být sloupec, případně kombinace více sloupců, které slouží k jednoznačné identifikaci každého řádku tabulky. Hodnota pole primárního klíče musí být v rámci tabulky jedinečná.

Poznámka: Občana bychom teoreticky mohli jednoznačně identifikovat jeho rodným číslem. Bohužel systém přidělování rodných čísel nebyl dokonalý, a tak všichni musíme ve většině důležitých formulářů vyplňovat kromě rodného čísla i jméno, příjmení, datum a někdy i místo narození.

Pole primárního klíče musí obsahovat konkrétní hodnotu, takže nikdy nesmí nabývat hodnotu `NULL`. Bez primárního klíče není možné definovat relace mezi tabulkami.

42 Primární klíč z pohledu relační integrity

Nechť R je relace a K(a1,a2,...,an) je množina atributů relace. Primární klíč je vybraná podmnožina z atributů, přičemž platí, že pro 2 n-tice neexistují stejné hodnoty atributů. Primární klíč slouží k odlišení entit. Atributy, které jsou součástí primárního klíče, se nazývají klíčové, ostatní atributy jsou neklíčové.

Poznámka: V jediné relaci může být definován jen jeden jediný primární klíč.

43 Primární klíč mohou vytvořit jen silné entity

Každé schéma relace by mělo mít definovaný primární klíč. Entity, které mají málo atributů a nejsou schopny vytvořit primární klíč, se nazývají slabé entity. Entity, které mají dostatek atributů na vytvoření primárního klíče, se nazývají silné entity.

44 Jednoduchý a kompozitní primární klíč

Primární klíč tvořený pouze jedním atributem se nazývá jednoduchý. Je-li primární klíč vybudován z více atributů, nazývá se kompozitní.

45 Co je to unikátní klíč

Unikátní klíč je definován pravidlem, podle něhož jeden nebo skupina několika atributů může nabývat jen unikátní hodnoty. Na rozdíl od primárního klíče může databázová tabulka obsahovat více unikátních klíčů.

46 Co je to cizí klíč

Cizí klíč je sloupec, případně kombinace několika sloupců, které jsou propojeny na primární klíč v jiné tabulce. V tabulce objednávek je cizím klíčem `id_zak`, který ukazuje do tabulky zákazníků a identifikuje danou objednávku vůči konkrétnímu zákazníkovi.

Tabulka zákazníků:

id_zak	firma	kontakt_jmeno	adresa	mesto	obrat	dluh
1	Procesory, s.r.o.	Vonasek Jiri	Volkrova 12	Praha	2567892.30	365000.00
2	Matice, a.s.	Kucera Jan	Zaoralova 16	Brno	753275.65	100000.00
3	Kosmetika	Chocholousek Tibor	Dolni 654	Uhersky Brod	212356.20	0.00

Tabulka objednávek

id_obj	id_zak	datum_objednavky
1	2	2002-10-01
2	1	2002-10-01
3	3	2002-10-02
4	5	2002-10-03

47 Cizí klíč z pohledu relační integrity

Nechť R2 je základní tabulka (relace). Potom cizím klíčem v R2 je podmnožina z množiny všech atributů R2, pro které platí:

- existuje relace R1 k relaci R2, v níž je vytvořen primární klíč PK;
- po celou dobu je každá hodnota cizího klíče z běžných hodnot relace R2 identická s hodnotami primárního klíče nějaké n-tice relace R1, nebo je hodnota atributu cizího klíče prázdná.

48 Jednoduchý a kompozitní cizí klíč

Cizí klíč může být kompozitní (může mít více atributů) jen tehdy, pokud je s ním spojený primární klíč také kompozitní.

49 Pravidla pro relační databázové systémy

Dr. E. F. Codd definoval kromě tří základních podmínek minimální relačnosti i dvanáct pravidel pro relační databázové systémy, které si postupně vysvětlíme ve zjednodušené podobě.

50 Co říká pravidlo informace

Informace v relační databázi musejí být reprezentovány explicitně na logické úrovni pomocí relačních tabulek.

51 Co říká pravidlo zaručeného přístupu

Údaje uložené v relační databázi musejí být přístupné kombinací názvu tabulky, názvu sloupce a hodnoty primárního klíče. V síťovém nebo distribuovaném databázovém prostředí se udává i název serveru a název databáze.

52 Jak na systematické ošetření prázdných hodnot

Musí existovat identifikátor chybějící hodnoty nebo hodnoty, kterou neznáme (rozdílný od čísla nula a prázdného řetězce). Tento identifikátor označujeme `NULL`.

53 Proč je popis struktury je založen na relačním modelu

Databáze musí umožňovat autorizovaným uživatelům přístup nejen k údajům, ale i k jejich popisům (metadatům) pomocí stejného databázového jazyka. Popis databáze se na logické úrovni reprezentuje stejně jako běžné údaje.

54 Co říká pravidlo komplexního datového jazyka

Databázový jazyk musí být jednoduchý a uživatelsky přívětivý, přičemž musí umožňovat interaktivní i programový režim. Kromě toho musí umožňovat definování údajů a entitních omezení, manipulaci s údaji a definování transakcí a přístupových práv.

55 Co je aktualizace pohledů

Relační databázový systém musí poskytovat nějaký způsob pro definování pohledů a musí umožnit pro tyto pohledy povolit či zakázat vkládání a rušení řádků nebo aktualizaci sloupců v základních tabulkách, nad nimiž je pohled vytvořen.

56 Co umí vysokoúrovňová manipulace s údaji

Relační databázový systém musí umožňovat množinové operace s celými tabulkami nejen při vyhledávání, ale i při vkládání, aktualizaci a rušení dat.

57 Co je fyzická datová nezávislost

Aplikační logika nesmí vyžadovat modifikaci v případě změny interního uložení nebo metody přístupu k údajům.

58 Co je logická datová nezávislost

Aplikační logika nesmí vyžadovat modifikaci v případě změny základních tabulek nevyvolávající ztrátu informace (zrušení nebo přidání sloupce do tabulky).

59 Co je nezávislost integrity

Aplikační logika nesmí vyžadovat modifikaci v případě změn integritních omezení definovaných pomocí databázového jazyka a uložených v katalogu dat.

60 Co je to distribuční nezávislost

Aplikační logika nesmí vyžadovat modifikaci v případě, jsou-li data distribuována na různých počítačích.

61 Co říká pravidlo nenarušení

Má-li databázový systém nízkoúrovňový (procedurální) programovací jazyk, nesmí být tomuto jazyku umožněno rušit nebo měnit omezení definovaná databázovým jazykem.

62 Jak na modelování

Podstatou činnosti každého informačního systému, který využívá databáze, je transformace informací z vnějšího světa do dat. Tento proces můžeme na různých úrovních modelovat. Model informačního systému by měl být:

- srozumitelný – měl by vyjadřovat fakta a pravidla v jednoduchém jazyce;
- vhodný – měl by podchytit co nejvíce pravidel vyplývajících z obchodní logiky navrhované aplikace;
- spolehlivý – měl by umožňovat ověření pravidel v přirozeném jazyce na jednoduchých příkladech;
- stálý – je potřebné minimalizovat obsah změn;
- vykonatelný – model musí být jednak realizovatelný na technické úrovni pomocí dostupných hardwarových a softwarových prostředků a taktéž musí byť vhodný pro provoz.

63 Modelování databázové aplikace

Modelování databázové aplikace se skládá ze dvou etap:

- modelování obchodních požadavků, které může být vnější (analýza vnějších vztahů) a konceptuální (analýza obchodní logiky);
- modelování databází, které může být na logické i fyzické úrovni.

64 Modelovaní obchodních požadavků

Vzhledem k tomu, že se jedná o publikaci věnovanou databázím, a ne ekonomii, nebudeme tuto etapu podrobněji rozvádět. Budeme předpokládat, že předtím, než přistoupíme k modelování databáze, už máme analýzu vnějších vztahů a obchodní požadavky ujasněné.

65 Modelování databází

Při modelování databází přecházíme od konceptuálního modelu, který je úplně hardwarově a softwarově nezávislý, k logickému a fyzickému modelu. Tento proces má nejen své hardwarové a softwarové specifikace, ale musíme přitom akceptovat i reálný čas. Nelze přeci ukládat údaje do databáze pomaleji, než vznikají, a podobně jako u konceptuálního modelu i požadavky obchodní logiky. Pokud například manažeři požadují

souhrny ve formě určitých typů grafů, bude rozumné navrhnout databázové tabulky tak, aby sloužily jako podklady pro požadované typy grafů.

66 Co je to databázové schéma

Základní informaci o tom, jakým způsobem jsou data v databázi uložena, obsahuje databázové schéma. Příkladem takového schématu pro uložení informací o zákaznících může být schéma:

```
zakaznici(id_zak, firma, kontakt_jmeno, adresa, mesto, psc, stat, telefon)
```

přičemž `zakaznici` je název schématu a `id_zak`, `firma`, `kontakt_jmeno`, `adresa`, `mesto`, `psc`, `stat`, `telefon` jsou názvy položek (atributů údajů, které do databáze chceme ukládat).

67 Co popisuje konceptuální model

Někdy označujeme proces systémové analýzy a systémového návrhu také jako konceptuální model. Takovýto model popisuje údaje v databázi zcela nezávisle na jejich fyzickém uložení a při jeho návrhu (tento proces se nazývá konceptuální modelování) se zaměřujeme na aplikační logiku, ale z pohledu člověka, ne z pohledu později použitých hardwarových a softwarových technologií. Při tvorbě konceptuálních modelů zpravidla vnímáme objekty reálného světa, vztahy mezi nimi a funkce, s jejichž pomocí se tyto vztahy realizují, takže konceptuální modelování je v podstatě objektově orientovaný proces. Kromě objektů zpravidla vstupuje do hry i jejich hierarchické uspořádání, například dědičnost, což znamená, že objekty mohou být vytvořeny na základě jiných objektů, přičemž zdědí část vlastností a podobně.

68 Co je to entita

Ještě předtím, než se budeme věnovat entitně-relačním modelům, je užitečné definovat pojem entita. Entita (angl. entity) je objekt reálného světa, který je schopen nezávislé existence a je jednoznačně odlišný od ostatních objektů. Příkladem entity může být například občan Josef Novák, narozený 22. 2. 1960 v Příbrami, bytem v Olomouci, rodné číslo 600222/1234.

69 Co je to entitně-relační model

Entitně-relační model (E-R model) je množina pojmů, s jejichž pomocí popisujeme příslušnou aplikaci za účelem následné specifikace struktury databáze. Proces návrhu systému spočívá jednak v identifikaci typů entit jako množin objektů stejného typu, v identifikaci typů vztahů, do nichž budou entity vstupovat, a v přiřazení atributů, které blíže popisují vlastnosti jednotlivých entit a vztahů. V této definici se vyskytlo několik pojmů, které je rovněž nutné definovat.

70 K čemu slouží vztah

Vztah (angl. relationship) je vazba mezi dvěma nebo více entitami.

Jako příklad vztahu můžeme uvést například „Josef Novák, narozený...“ je manželem „Ivety Novákové, narozené...“

71 Co je to atribut

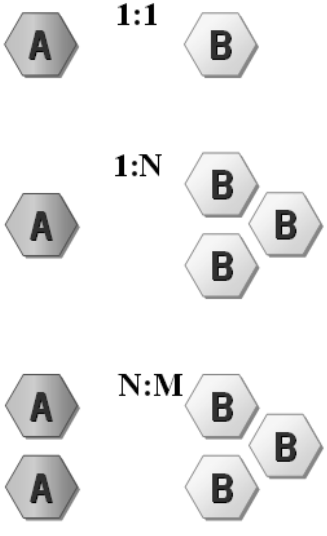
Atribut (angl. attribute) je funkce, která přiřazuje jednotlivým entitám či vztahům hodnotu, která určuje některou podstatnou vlastnost entity či vztahu. Například jméno občana, jeho datum narození, rodné číslo a podobně. Každá entita může mít více atributů.

Poznámka: Je nutné si uvědomit, že vymezení pojmů entita, vztah a atribut je poměrně volné, přičemž záleží hlavně na úhlu pohledu analytika. Určitým vodítkem může být analogie s přirozeným jazykem, kdy se pro popis entit používají podstatná jména a pro popis vztahů slovesa.

72 Co říká kardinalita vztahů mezi entitami

Relace mezi tabulkami vlastně popisují vztahy mezi objekty reálného světa, které tyto tabulky představují. Při návrhu databázových tabulek, na které navazuje aplikační logika, můžete definovat několik druhů vztahů. S ohledem na kardinalitu budeme entity nazývat první a druhá. Mezi nimi bude vztah (relace):

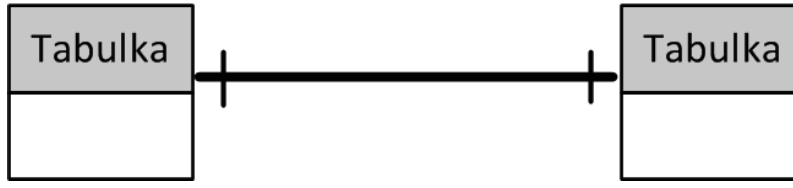
- 1:1 – první entitě, například záznamu v databázové tabulce, odpovídá maximálně jedna druhá entita, tedy záznam z jiné databázové tabulky.
- 1:N – první entitě odpovídá více druhých entit. Ale naopak, druhé entitě odpovídá maximálně jedna první entita.
- M:N – první entitě odpovídá více druhých entit. A taktéž i naopak, druhé entitě odpovídá více prvních entit.



Obrázek 5: Kardinalita vztahů mezi entitami

73 Co říká vztah jedna ku jedné (1:1, one-to-one)

V tomto vztahu první entitě neboli záznamu v databázové tabulce, odpovídá maximálně jedna druhá entita neboli záznam z jiné databázové tabulky. Každý řádek primární tabulky je tedy možné svázat právě s jedním řádkem sekundární tabulky. Například řidič řídí maximálně jeden automobil, nebo naopak, automobil je řízen maximálně jedním řidičem. Takovouto relaci zajistíte pomocí unikátních klíčů v obou tabulkách.

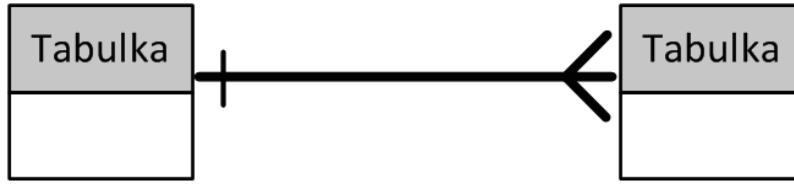


Obrázek 6: Vztah jeden ku jedné

Do této kategorie vztahů patří i tzv. částečné vztahy 1:0 a 0:1, v našem případě řidič z nějakého důvodu neřídí žádné vozidlo (má ho například v servisu, sedí v kině...).

74 Co říká vztah jeden ku více (1:N, many-to-one)

Každý řádek primární tabulky je možné svázat s jedním nebo více řádky sekundární tabulky.

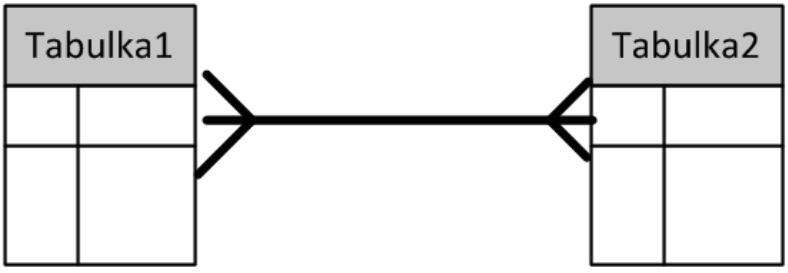


Obrázek 7: Vztah jeden k více

Například relace cestující-autobus. V autobuse může být výrazně víc cestujících, ale naopak, jeden cestující se nemůže vést naráz ve více autobusech. Při vztahu 1:N je významný směr.

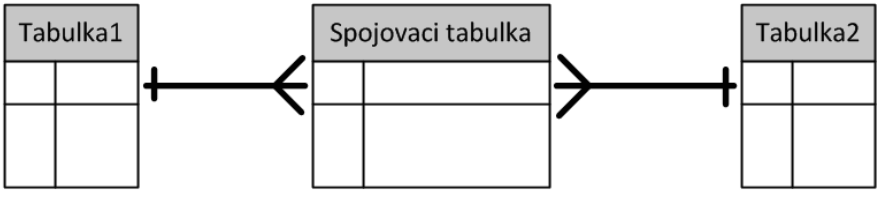
75 Co říká vztah více ku více (N:M, many-to-many)

Více řádků primární tabulky může být svázáno s více řádky sekundární tabulky. Vztah N:M lze v praxi pozorovat například mezi výrobky a vlastnostmi. Jeden výrobek může mít více vlastností, a naopak, některou vlastnost může mít více výrobků. Jiný příklad: relace M:N vznikne tak, že jeden autor může napsat více knih, ale na druhé straně, jednu knihu může napsat více autorů současně.



Obrázek 8: Vztah více ku více

Protože většina databázových systémů nedokáže přímo pracovat se vztahy typu N:M, používá se v praxi dekompozice, což znamená, že se tento vztah implementuje pomocí spojovací tabulky. Vztah N:M se tak rozloží na dva vztahy typu N:1.



Obrázek 9: Rozložení vztahu pomocí spojovací tabulky

76 Co je to parcialita vztahu

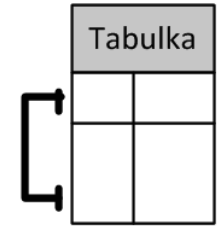
pokročilý

Kromě kardinality vztahu lze ještě rozlišit povinnost či volitelnost jeho existence. Typickým volným vztahem je „učitel-třídní učitel". Každý učitel nemusí být třídní.

77 Co je to unární relace

pokročilý

Až dosud jsme uváděli vztahy mezi dvěma tabulkami. V praxi se ale vyskytují i relace samotné tabulky se sebou. Nejvyšší hierarchická úroveň je v této vazbě tvořen jediným prvkem, který je svázán s prvky o jednu úroveň níže. Tyto prvky mohou mít vazbu na další prvky z nižší úrovně. Každý prvek kromě prvku na nejvyšší úrovni má tedy vazbu na jeden prvek z vyšší úrovně a žádnou, jednu nebo více vazeb na prvky nižší úrovně. Pomocí unární relace se často vyjadřuje hierarchický vztah nadřízený-podřízený. Sloupec tabulky může obsahovat vazbu na primární klíč jeho nejbližšího nadřízeného. Jedno políčko v takovémto sloupci v některém řádku tabulky je obvykle prázdné. Jedná se o nejvyššího nadřízeného.



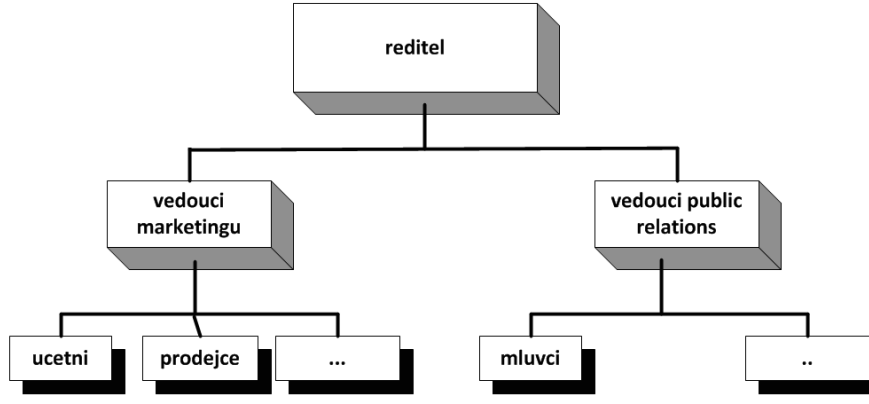
Obrázek 10: Unární relace

78 Příklad unární relace

Možná bude názornější vysvětlení na konkrétním příkladu. Mějme jednoduchou tabulku pracovníků. Tabulka má definovanou unární vazbu pomocí sloupce `nadrizeny`:

```
id_prac  jmeno            funkce              nadrizeny
-------- ---------------- ------------------- -----------
1        Novak Alfonz     reditel             NULL
2        Plha Jan         vedouci marketingu  1
3        Kecal Josef      vedouci pr          1
4        Rach Petr        ucetni              2
5        Kubikova Jana    prodejce            2
6        Pekna Kamila     mluvci              3
```

Sloupec `nadrizeny` obsahuje pro každého zaměstnance ID jeho nejbližšího nadřízeného. Jedno políčko v tabulce je prázdné (obsahuje hodnotu `NULL`). Jde tedy o nejvyššího nadřízeného. Jemu jsou podřízeni vedoucí oddělení marketingu a PR. Účetní a obchodnice jsou podřízeni vedoucímu marketingu a tisková mluvčí firmy je podřízená vedoucímu oddělení PR. Schematicky vypadá celá situace následovně:



Obrázek 11: Hierarchická struktura pracovníků firmy

79 Problémy spojené s implementací hierarchické struktury

Takováto tabulka je dobrá pro zápis údajů o hierarchické struktuře pro účely jejího výpisu, nehodí se však pro operace na hierarchických úrovních. V čem je problém? Pro názornost vytvoříme jednodušší tabulku, v níž bude hierarchická struktura implementována pomocí jmen, a naplníme ji třemi záznamy.

```
CREATE TABLE pracovnici
(
  id INT,
  jmeno CHAR(10),
  nadrizeny CHAR(10),
  mzda MONEY
);
```

```
INSERT INTO pracovnici VALUES (1, 'Suchy', 'Balaban', 10.0000);
INSERT INTO pracovnici VALUES (2, 'Jurista', 'none', 9.0000);
INSERT INTO pracovnici VALUES (3, 'Balaban', 'Jurista', 11.0000);
```

Implementovali jsme tuto hierarchii:

*	Jurista	9 000
**	Balaban	11 000
***	Suchy	10 000

Zkuste s takto vytvořenou tabulkou pracovat. Představte si, že uvidíte značnou nesourodost mezi postavením ve firemní hierarchii a platem. Proto zkusíte o 20 procent snížit mzdu každému pracovníkovi, který vydělává víc než jeho nadřízený. Intuitivně tedy zkusíte příkaz:

```
UPDATE pracovnici
  SET mzda = mzda * 0.8000
  WHERE mzda > (SELECT mzda
                  FROM pracovnici AS P2
                  WHERE P2.jmeno = pracovnici.nadrizeny);
```

Podívejme se, jak to dopadlo:

```
id          jmeno      nadrizeny  mzda
----------- ---------- ---------- --------
1           Suchy      Balaban    10.0000
2           Jurista    none       9.0000
3           Balaban    Jurista    8.8000
```

Mzda se snížila jen zaměstnanci Balabánovi, čímž se dostal pod úroveň svého šéfa Juristu. Ale vidíme, že tyto konstrukce fungují jen na jedné úrovni hierarchie, protože zaměstnanci Suchému se mzda nesnížila, neboť vydělával sice méně než jeho přímý nadřízený Balabán, ale nebralo se v potaz, že Suchý vydělává více než nejvyšší šéf Juristy.

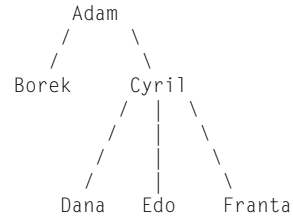
Poznámka: Po prostudování části o normálových formách pochopíte, v čem je problém: tato tabulka totiž není normalizovaná.

80 Návrh hierarchické struktury umožňující operace

Řešení samozřejmě existuje, jen musíte hierarchickou strukturu v rámci jedné tabulky implementovat jinak. Aby to bylo názorné i pro více záznamů, budeme implementovat takovouto hierarchickou strukturu zaměstnanců.

```
jmeno    nadrizeny
=================
Adam     NULL
Borek    Adam
Cyril    Adam
Dana     Cyril
Edo      Cyril
Franta   Cyril
```

Názornější to bude na hierarchickém grafu:



Definovat tuto hierarchii pomocí unární relace je hračka (viz tabulku), ale jak jste viděli, z hlediska aplikační logiky to za moc nestojí. Lidová moudrost „za málo peněz málo muziky“ se projevila i v tomto případě.

81 Implementace hierarchické struktury umožňující operace

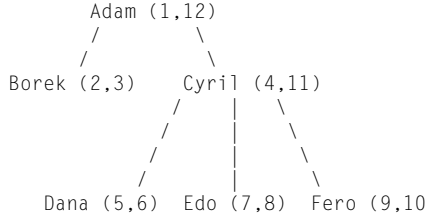
Řešením je implementace hierarchické stromové struktury pomocí dvou sloupců `LEVY` a `PRAVY`. S pomocí těchto sloupců můžete nejen přesně definovat polohu uzlu (listu) ve stromové struktuře, ale i správně vykonávat operace při zachování logiky hierarchie.

Poznámka: Předem upozorňujeme na to, že ačkoli se jedná o jednoduchý princip, je nutné ho nejdříve pochopit, a proto bude možná nutné pročíst si následující stať i dvakrát :-)

Nejlépe to celé pochopíte na hotovém příkladě. Naši strukturu byste pomocí sloupců `LEVY` a `PRAVY` implementovali takto:

```
Jmeno       LEVY  PRAVY
=======================
Adam           1     12
Borek          2      3
Cyril          4     11
Dana           5      6
Edo            7      8
Fero           9     10
```

V grafickém vyjádření to bude vypadat následovně:



82 Vytvoření a naplnění hierarchické tabulky

Tabulku vytvoříte a naplníte pomocí následujícího příkazového skriptu:

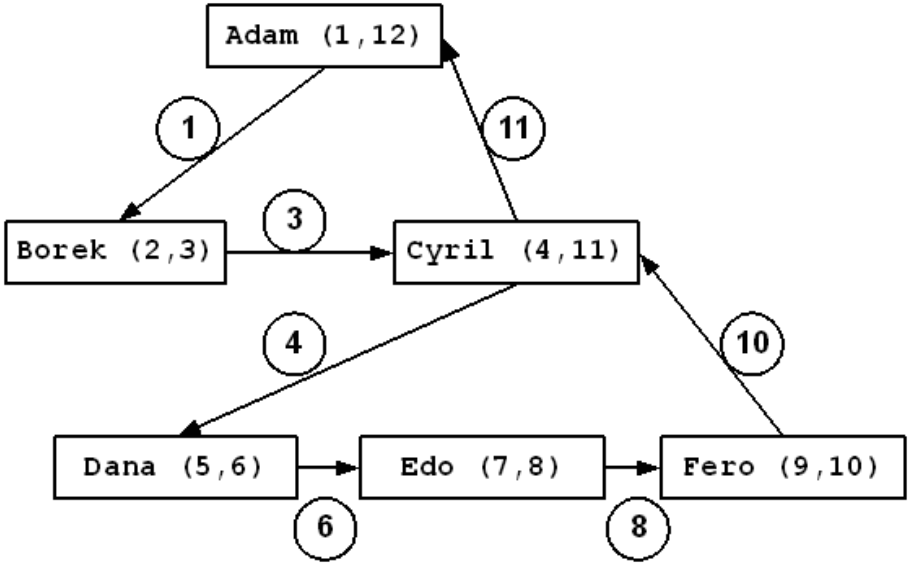
```
CREATE TABLE prac
(
  jmeno CHAR(10),
  LEVY  INT,
```

```
  PRAVY INT
);

INSERT INTO prac VALUES ('Adam', 1,12)
INSERT INTO prac VALUES ('Borek',  2,3)
INSERT INTO prac VALUES ('Cyril', 4,11)
INSERT INTO prac VALUES ('Dana', 5,6)
INSERT INTO prac VALUES ('Edo', 7,8)
INSERT INTO prac VALUES ('Fero',  9,10)
```

83 Jak to vlastně funguje?

Je to jednoduché až na jednu maličkost. Je nutné pochopit smysl a význam sloupců LEVY a PRAVY, umět jim přidělovat hodnoty a umět s nimi pracovat. Vysvětlíme si, jak se jednotlivé hodnoty přidělují. Podívejte se na obrázek:



Obrázek 12: Přidělování hodnot sloupcům LEVY a PRAVY

Budete postupně procházet stromovou strukturu shora dolů a zleva doprava a při tom při vstupu do uzlu inkrementovat pomyslný čítač. Má-li uzel podřízené uzly, přidělíte mu jen hodnotu sloupce LEVY, pokud podřízené uzly nemá, přiřadíte mu podle aktuálního stavu čítače obě hodnoty. To je případ uzlu Borek, kde do sloupce LEVY zapíšete číslo dva, do sloupce PRAVY číslo 3 a s tímto číslem uzel opustíte. V uzlu Cyril zapíšete jen hodnotu do sloupce LEVY a vydáte se na „pouť" jeho podřízenými uzly. Nakonec se vracíte nazpět do hlavního uzlu, přičemž přidělujete hodnoty do sloupce PRAVY.

Dá se na to jít i matematicky, ale je potřebné znát teorii grafů. Nejvýše v hierarchii je Adam (1,12), přičemž ve sloupci LEVY má hodnotu 1 a ve sloupci PRAVY hodnotu 12, což je celkový počet záznamů vynásobený dvěma. Řečeno výrazovými prostředky jazyka SQL:

```
PRAVY = 2 * (SELECT COUNT(*) FROM tabulka))
```

84 Výběr záznamu a jeho „nadřízených“

Důležité je taktéž s takto organizovanou strukturou umět pracovat. Chcete-li vybrat pracovníka a všechny jeho nadřízené, použijte příkaz ve tvaru:

```
SELECT P2.* FROM prac AS P1, prac AS P2
  WHERE P1.LEVY BETWEEN P2.LEVY AND P2.PRAVY AND P1.jmeno = 'Cyril';
```

```
jmeno      LEVY        PRAVY
---------- ----------- -----------
Adam       1           12
Cyril      4           11
```

Pro pracovníka jménem „Dana“ by byl výsledek následující:

```
jmeno      LEVY        PRAVY
---------- ----------- -----------
Adam       1           12
Cyril      4           11
Dana       5           6
```

Poznámka: Řečeno srozumitelněji, nadřízený konkrétního pracovníka je ten pracovník, u něhož leží hodnota sloupce `LEVY` mezi hodnotami sloupců `LEVY` a `PRAVY` podřízeného pracovníka.

85 Výběr záznamu a jeho „podřízených“

Samozřejmě to funguje i naopak. Chcete-li vypsat konkrétního pracovníka a všechny jeho podřízené, použijte „symetrický“ příkaz:

```
SELECT P2.* FROM prac AS P1, prac AS P2
  WHERE P2.PRAVY BETWEEN P1.LEVY AND P1.PRAVY AND P1.jmeno = 'Cyril';
```

```
jmeno      LEVY        PRAVY
---------- ----------- -----------
Cyril      4           11
Dana       5           6
Edo        7           8
Fero       9           10
```

Pro pracovníka jménem „Adam“ (nejvyšší nadřízený) by byl výsledek:

```
jmeno      LEVY        PRAVY
---------- ----------- -----------
Adam       1           12
Borek      2           3
Cyril      4           11
Dana       5           6
Edo        7           8
Fero       9           10
```

86 Operace v tabulce obsahující hierarchickou strukturu

Vraťme se k výše uvedenému problému snižování mezd. Pokud byste příslušnou tabulku implementovali takto:

```
CREATE TABLE pracovnici1
(
```

```
  id INT,
  jmeno CHAR(10),
  LEVY INT,
  pravý INT,
  mzda MONEY
);

INSERT INTO pracovnici1 VALUES (1, 'Suchy', 3,4, 10.0000);
INSERT INTO pracovnici1 VALUES (2, 'Jurista', 1,6, 9.0000);
INSERT INTO pracovnici1 VALUES (3, 'Balaban', 2,5, 11.0000);
```

příkaz pro snížení mzdy o 20 procent každému pracovníkovi, který vydělává víc než jeho nadřízený, byste mohli napsat takto:

```
UPDATE pracovnici1 SET mzda = mzda * 0.8000
  WHERE mzda > ANY (SELECT mzda FROM pracovnici1 AS PX
                    WHERE pracovnici1.LEVY BETWEEN PX.LEVY AND PX.PRAVY);
```

Tentokrát se příkaz vzhledem k hierarchické struktuře vykoná správně:

```
jmeno      LEVY        PRAVY
---------- ----------- -----------
Adam       1           12
Borek      2           3
Cyril      4           11
Dana       5           6
Edo        7           8
Fero       9           10
```

Základy jazyka SQL

87 K čemu slouží jazyk SQL

SQL je dotazovací jazyk, takže přes propojenou aplikaci se serveru odevzdá dotaz a databázový server na něj odpoví, obvykle tím, že vygeneruje nějakou množinu výstupních údajů. Tento princip komunikace s databázovým serverem je velmi jednoduchý a efektivní. Samozřejmě ale jen z hlediska uživatele. Jazyk SQL totiž připomíná klasický přirozený jazyk (samozřejmě anglický), má však přesně definovaná syntaktická a lexikální pravidla. Z pohledu serveru se SQL příkaz přenáší, dekóduje, zpracovává, optimalizuje a vykonává, takže podrobné schéma tohoto mechanismu by bylo velmi složité.

Poznámka: Jazyk SQL můžete použít jednak jako dotazovací jazyk pro práci s údaji v relační databázi, případně jako část hostitelského jazyka pro vývoj databázových aplikací.

88 Stručný pohled do historie jazyka SQL

Databázový jazyk SQL (Structured Query Language) vznikl na základě projektu firmy IBM s názvem SEQUEL (Structured English Query Language), jehož cílem bylo vytvořit jazyk blízký angličtině pro práci s údaji v databázi. Postupně se k tomuto standardu přidávaly další firmy (Oracle, SyBase, Informix) a tak vznikl „nepsaný standard" databázového jazyka s názvem SQL. Postupně byly přijaté vylepšené a upravené standardy jazyka SQL s názvem SQL-86 a později SQL-92. Pro verzi SQL-92 se vžil zkrácený název SQL-2. Ve vývoji jsou verze s pracovním názvem SQL-3 a dokonce už byly zahájeny přípravné práce na normě SQL-4.

89 SQL – slovní zásoba

Každý jazyk má určité náležitosti, především slovní zásobu a gramatiku. Když se podíváte na příkazy jazyka SQL, je to v podstatě gramatický předpis jednoduché anglické slovní zásoby. Slovní zásoba obsahuje několik desítek jednoduchých anglických slov.

- `CREATE` – vytvoř
- `SELECT` – vyber
- `INSERT` – vlož
- `INTO` – do
- `NULL` – prázdné, nic
- `NUMBER` – číslo
- `TABLE` – tabulka
- `VALUE` – hodnota
- ...

90 SQL – gramatika


pokročilý


Poznámka: Pokud vám příkazy z této publikace připomínají úsečnou řeč manželky manželovi, nemýlíte se. I ten, kdo není jazykovědec, si určitě všimne, že v každé větě (v každém příkaze jazyka SQL) je minimálně jedno sloveso v rozkazovacím způsobu. Vždyť i proto se tomu říká příkaz.

Z gramatických pravidel angličtiny stačí znát jedno jediné – jde o příkazy, a proto budou všechny „věty" jazyka SQL v rozkazovacím způsobu. Rozkazovací větu v angličtině tvoří sloveso v infinitivu, předmět a další větné členy. Například CREATE TABLE – vytvoř tabulku. Příkazy standardní množiny jazyka SQL lze rozdělit do několika podmnožin, viz následující části.

91 Data Definition Language (DDL)


pokročilý

Pomocí příkazů z podmnožiny DDL můžete definovat, vytvářet, měnit a rušit (odstraňovat) různé objekty a struktury v relačních databázích, jako jsou například tabulky, indexy, spouště, uložené procedury a podobně. Taktéž můžete přidělovat a odebírat uživatelská oprávnění jednotlivým uživatelům a skupinám uživatelů. Do této skupiny patří například tyto příkazy:

```
CREATE DATABASE
CREATE TABLE
ALTER TABLE
DROP TABLE
CREATE INDEX
DROP INDEX
CREATE VIEW
ALTER VIEW
DROP VIEW
DROP INDEX
CREATE SEQUENCE
ALTER SEQUENCE
DROP SEQUENCE
CREATE PROCEDURE
DROP PROCEDURE
CREATE TRIGGER
DROP TRIGGER
```

92 Data Manipulation Language (DML)


pokročilý

Do této skupiny, jak vyplývá už z jejího názvu, patří příkazy pro manipulaci s údaji neboli příkazy pro vkládání, aktualizaci a mazání údajů a samozřejmě velmi často používaný příkaz SELECT pro výběr údajů.

```
SELECT
INSERT
UPDATE
DELETE
```

93 Data Control Language (DCL)

Skupina zahrnuje speciální příkazy pro řízení provozu a údržby databáze.

```
GRANT
ALTER USER
DROP USER
REVOKE
```

94 Příkazy pro řízení transakcí (Transaction Control Commands)

Do této skupiny patří například příkazy:

```
CREATE TRANSACTION
COMMIT
```

95 Praktický příklad DCL

Kategorizaci příkazů si ukážeme na jednoduchém příkladu. Prezentovaný blok příkazů patří do podmnožiny **DCL (Data Control Language).** Obsahuje příkazy pro připojení se k databázovému serveru, odemknutí a změnu hesla uživatele schématu HR (Oracle).

```
CONNECT SYS AS SYSDBA
ALTER USER HR ACCOUNT UNLOCK;
ALTER USER HR IDENTIFIED BY HRPASSWORD;
CONNECT HR/HRPASSWORD;
```

Pokud byste tuto posloupnost příkazů připomínající anglické věty přeložili do češtiny (zvládla by to snadno i humanitně orientovaná překladatelka), získali byste posloupnost vět:

1. Připoj se jako SYSDBA.
2. Modifikuj uživatele jménem HR, odemkni jeho účet.
3. Modifikuj uživatele jménem HR, identifikuj ho podle hesla HRPASSWORD.
4. Připoj se jako HR.

96 Praktický příklad DDL

Příkazy z podmnožiny DDL (Data Definition Language) v tomto příkladě slouží pro vytvoření databázové tabulky zaměstnanců poboček.

```
CREATE TABLE emp
(
  empno    NUMBER(4) NOT NULL,
  ename    VARCHAR2(10),
  job      VARCHAR2(9),
  mgr      NUMBER(4),
  hiredate DATE,
  sal      NUMBER(7,2),
  comm     NUMBER(7,2),
  deptno   NUMBER(2)
);
```

```
CREATE TABLE dept
(
  deptno NUMBER(2) NOT NULL,
  dname VARCHAR2(14),
  loc VARCHAR2(13)
);
```

97 Praktický příklad DML

Naplnění tabulek údaji je typická úloha pro příkazy ze skupiny **DML (Data Manipulation Language)**.

```
INSERT INTO EMP VALUES
       (7369, 'SMITH',  'CLERK',     7902,
       TO_DATE('17-DEC-1980', 'DD-MON-YYYY'),  800, NULL, 20);
INSERT INTO EMP VALUES
       (7499, 'ALLEN',  'SALESMAN',  7698,
       TO_DATE('20-FEB-1981', 'DD-MON-YYYY'), 1600,  300, 30);
INSERT INTO EMP VALUES
       (7521, 'WARD',   'SALESMAN',  7698,
       TO_DATE('22-FEB-1981', 'DD-MON-YYYY'), 1250,  500, 30);
...
```

98 Středník za SQL příkazem

Hned na začátku používání syntaxe jazyka SQL narazíte na první náznak určité, v tomto případě ne až tak podstatné nekompatibility mezi databázovými platformami. Týká se psaní středníku (;) za SQL příkazem. Některé platformy středník nevyžadují, ale akceptují ho, například Microsoft SQL Server, jiné platformy ho striktně vyžadují, například Oracle. Tuto nekompatibilitu lehce překonáte tím, že za příkazy jazyka SQL budete vždy psát středník.

99 Jak používat komentáře

Každý programátor, který programuje víc než několikařádkový kód, určitě nepochybuje o významu komentářů. Při psaní kódu se vám zdá všechno jasné, ale bude to tak i za týden, za měsíc nebo za rok, kdy budete potřebovat provést v kódu drobné, případně i větší úpravy? Mírné zdržení při psaní komentářů se vám ve většině případů později bohatě vyplatí. Komentáre v jazyce SQL mohou být jednořádkové nebo víceřádkové. Jednořádkové komentáře začínají dvěma pomlčkami.

```
-- Toto je jednořádkový komentář
```

Víceřádkový komentář začíná a končí párovými znaky /* */, například:

```
/* Víceřádkový komentář
   se používá například při komentování
  rozsáhlejších funkčních bloků
*/
```

Poznámka: Platforma MySQL využívá pro řádkový komentář místo dvou pomlček znak #.

100 Víceslovné názvy objektů

Názvy objektů v jazyce SQL, například názvy tabulek, sloupců, pohledů a podobně, musejí začínat alfanumerickými znaky a–z či A–Z nebo podtržítkem (_). Pokud se názvy objektů skládají z více slov, musejí být ohraničeny uvozovkami nebo hranatými závorkami:

```
SELECT * FROM "Název tabulky"
```

nebo:

```
SELECT * FROM [Název tabulky]
```

101 Konvence pro názvy objektů

Pro názvy objektů v databázi platí všeobecné pravidlo, že název objektu musí být v systému jedinečný:

```
databáze.vlastník.objekt
```

Z toho například vyplývá skutečnost, že v databázi můžete mít dva různé objekty, například databázové tabulky se stejnými jmény, za předpokladu, že mají jiné vlastníky. Pro názvy identifikátorů podle normy SQL-92 platí, že musejí být tvořeny minimálně jedním (na to byste přišli i sami, ale normotvůrci musejí být exaktní) a maximálně 128 znaky. U některých platforem je tento maximální počet znaků nižší. První znak identifikátoru musí být písmeno nebo znaky _, @ a #. Dalšími znaky mohou být i číslice. Obsahuje-li identifikátor mezery, musíme ho povinně uzavřít do hranatých závorek nebo do uvozovek.

102 Jak zadat příkaz databázovému serveru

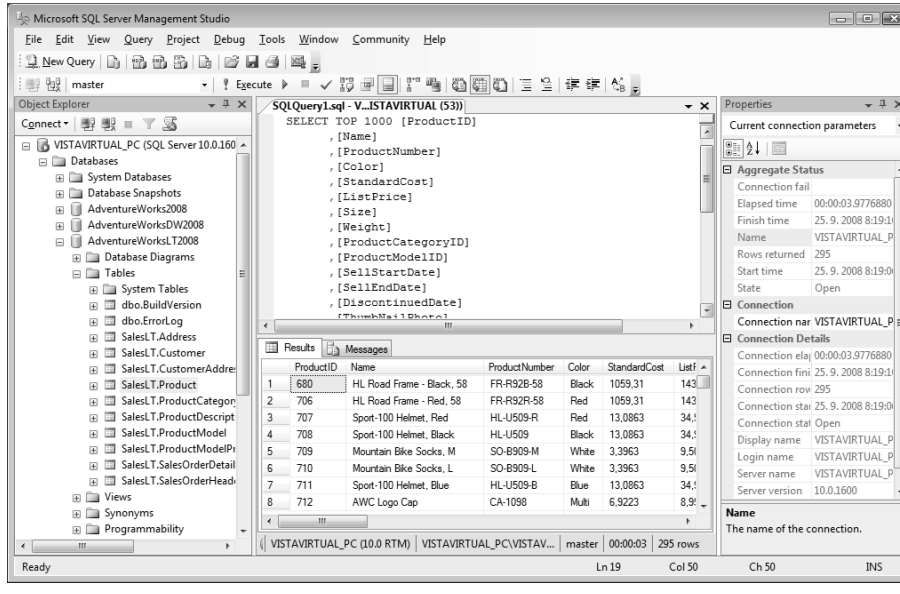
Pro úplného začátečníka v oblasti databází je potřebné ještě před začátkem seznamování se s jazykem SQL ujasnit si, jakým způsobem lze příkazy jazyka SQL předávat databázovému serveru a kde a v jaké formě získá výsledek. Potřebujete **klientskou konzolovou aplikaci**, s jejíž pomocí zadáváte databázovému serveru příkazy SQL a v jejímž okně taktéž vidíte výstupy, které databázový server vygeneruje jako odezvu na vaše příkazy. A vývojář kromě klientské konzolové aplikace občas potřebuje **nástroj pro správu databáze**. Prostřednictvím této aplikace můžete vytvářet nové databáze, vytvářet a spravovat účty jednotlivých uživatelů a přidělovat a rušit jejich oprávnění pro práci s jednotlivými objekty databáze. Pomocí nástroje pro správu databáze je možné nastavit i strategii údržby a zálohování údajů v databázi a podobně.

103 SQL Server: SQL Server Management Studio

Pro práci s databázovým serverem Microsoft SQL Server 2008 je k dispozici aplikace SQL Server Management Studio. Pracovní plocha aplikace je rozdělena na několik částí: Okno Object Explorer poskytuje grafický, přehledný a hierarchicky uspořádaný pohled na objekty databáze, například v případě databázových tabulek až na úroveň sloupců a index. Okno je možné přepnout i do režimu Registered Servers. V tomto režimu je zobrazen seznam zaregistrovaných serverů, k nimž se lze pomocí Management Studia

připojit. Pomocí tohoto okna se mohou administrátoři, kteří spravují více serverů, přepínat mezi jednotlivými instancemi.

Hlavní okno ve střední části pracovní plochy bývá v případě zadávání příkazů jazyka SQL rozděleno na dvě části. V horní části je okno alokováno pro zadávání příkazů a v dolní části okno pro zobrazování výsledků. Příkazy je možné v závislosti na tom, na jakou službu jsme připojeni, zadávat v jazycích a skriptovacích systémech Transact-SQL, XMLA, MDX, DMX nebo XQuery. Můžete taktéž vytvářet příkazy určené pro SQL Server Compact Edition. Výsledky je možné zobrazovat v textové nebo tabulkové formě. Pravé svislé okno Properties je určeno k zobrazení parametrů vybraného objektu.



Obrázek 13: SQL Server Management Studio

Pro správu databázového serveru a ladění databázové části aplikací lze využít i jednoduchou interaktivní textovou konzolovou aplikaci SQLCMD. Slouží pro zadávání příkazů jazyka SQL databázovému serveru a zobrazování výstupů vygenerovaných databázovým serverem, například výpisů údajů z databázových tabulek, potvrzení vykonání příkazů, chybové hlášení a podobně.

104 Oracle: Klientské a administrátorské aplikace

začátečník

Aby bylo možné jednoduše pracovat s databází na dálku pomocí příkazů jazyka SQL, je součástí dodávky databáze Oracle od verze 10g i webová konzole **iSQL*Plus**. Pro správu databáze Oracle 10g je určena webová aplikace **Oracle Enterprise Manager** (OEM). I při dálkovém přístupu poskytne tato aplikace komplexní možnosti pro správu a konfiguraci. Dále nabízí podrobné informace o stavu, v jakém se databázový server momentálně nachází, případně jsou signalizovány potenciální problémy.

Obrázek 14: Zadávání příkazů jazyka SQL přes webovou konzoli iSQL*Plus

V porovnání s dosud představenými konzolovými aplikacemi se mění jen způsob klientského přístupu přes prohlížeč webového obsahu. Filozofie a částečně i vzhled aplikace, princip přihlašování a práce s údaji pomocí SQL příkazů zůstávají beze změn.

105 MySQL: Konzolová aplikace

začátečník

Pro práci s databází pod správou databázového serveru MySQL je určena klasická klientská konzolová aplikace MySQL pro příkazový řádek. Pamětníkům operačního systému MS DOS bude připomínat příkazový řádek operačního systému nebo známý řádkový editor EDLIN.

Databázové tabulky

106 Vytvoření databáze

Pro pokusy s databázovým serverem je vhodné vytvořit novou databázi. Můžete ji vytvořit SQL příkazem `CREATE DATABASE`, například:

```
CREATE DATABASE pokusy
```

107 Vytvoření databázové tabulky příkazem CREATE TABLE

Příkazem `CREATE TABLE` vytvoříme v databázi novou tabulku. Úplná definice syntaxe tohoto příkazu je poměrně složitá, proto uvádíme jeho zjednodušenou podobu.

```
CREATE TABLE [schema.]název_tabulky
(
    Název_sloupce   datový_typ [DEFAULT vyraz] [,
    Název_sloupce2  datový_typ [DEFAULT vyraz],
    ...
    Název_sloupceN  datový_typ [DEFAULT vyraz]]
)
```

Volba `DEFAULT` specifikuje implicitní hodnotu, která se uplatní při vkládání nového řádku, pokud pro tento sloupec nebyla zadána konkrétní hodnota. Například:

```
CREATE TABLE zakaznici
(
       id_zak INT NOT NULL,
       firma VARCHAR(20) NOT NULL,
       kontakt_jmeno VARCHAR(20),
       adresa VARCHAR(20),
       mesto VARCHAR(15),
       obrat MONEY,
       dluh MONEY
);
```

108 Oracle – vytvoření dočasné tabulky

Ne vždy je potřebné vytvářet databázové tabulky jak se říká „na věčné časy“. Pro účely různých pokusů, přehledů a podobně je možné s výhodou využít dočasné tabulky. Jejich platnost je omezena jen do konce přihlášení klienta.

```
CREATE GLOBAL TEMPORARY TABLE pomocna
(
       id INT,
       jmeno VARCHAR(20)
);
```

Nebo při vytváření dočasné tabulky na základě jiné tabulky:

```
CREATE GLOBAL TEMPORARY TABLE dluznici
   ON COMMIT PRESERVE ROWS
   AS SELECT * FROM zakaznici WHERE dluh >50000;
```

109 SQL Server – vytvoření dočasné tabulky

Na platformě SQL Server se dočasné tabulky vytvářejí s prefixem #. Například:

```
CREATE TABLE #pomocna
(
       id INT,
       jmeno VARCHAR(20)
);
```

110 Oracle – vytvoření databázové tabulky podle už existující tabulky

Novou tabulku můžete vytvořit podle stávající tabulky. Nechcete-li do ní umístit údaje z „mateřské“ tabulky, použijte podmínku, která nevrátí žádný záznam. Na platformě Oracle a MySQL je postup zřejmý z následujícího příkladu:

```
CREATE TABLE employees_2
AS SELECT * FROM employees
WHERE 1 = 0;
```

111 SQL Server – vytvoření databázové tabulky podle už existující tabulky

Na platformě SQL Server se pro vytvoření databázové tabulky podle už existující tabulky využívá příkaz `SELECT INTO` ve tvaru:

```
SELECT INTO employees_2
FROM employees
WHERE 1 = 0;
```

112 Vytvoření databázové tabulky prostřednictvím vizuálního návrhu

Většina moderních databázových platforem má k dispozici i vizuální nástroje pro vytváření databázových objektů, například průvodce pro vytvoření tabulky. Je to alternativa k vytváření objektů přímo pomocí příkazu jazyka SQL.

Poznámka: Výsledkem vizuálního návrhu je v konečném důsledku taktéž příkaz jazyka SQL, který následně databázový server vykoná.

Oba postupy mají svoje výhody a nevýhody. Návrh tabulky pomocí Průvodce je trochu rozvláčný, ale na druhé straně tím, že vás zbaví břemene „kódování“, vám ponechá více prostoru pro přemýšlení nad aplikační a systémovou logikou. Návrh pomocí příkazu SQL je jednoduchý a stručný, chce to ovšem trošku praxe a zkušeností a v neposlední řadě i zvýšenou pozornost, abyste při návrhu tabulky nezapomněli na něco důležitého z hlediska aplikační logiky.

113 Co jsou to datové typy

Datové typy jsou měřítkem atributů a používáte je i v běžném životě, aniž byste si to uvědomovali. Používáte přece datum a čas, textové věty a slova, oběživo a podobně.

Poznámka: Datové typy v jazyce SQL jsou téměř analogické datovým typům, které se používají v jiných programovacích jazycích. Databázové datové typy znáte i z kancelářských aplikací, například z tabulkových kalkulátorů, kde se používají texty, celá či desetinná čísla, údaje o datu a čase a podobně.

114 Jaké číselné datové typy jsou k dispozici

Slouží k ukládání číselných údajů, ať už celých čísel, čísel s pevnou řádovou čárkou (využívá se například v účetnictví, kdy se účtuje na dvě desetinná místa) nebo desetinných čísel používaných při vědeckých a technických výpočtech. U číselných datových typů je vždy definován jejich rozsah, například 0 až 255, -32 768 až 32 767 a podobně. Obvykle se používají tyto typy:

- **decimal** – rozsah datového typu `decimal` je od $-10^{38} + 1$ do $10^{38} - 1$;
- **float** – datový typ s pohyblivou desetinnou čárkou v rozsahu od $-1{,}79^{308}$ až $-2{,}23^{308}$, 0, $2{,}23^{308}$ až $1{,}79^{308}$;
- **double** – datový typ s pohyblivou desetinnou čárkou s dvojnásobnou přesností;
- **real** – datový typ s pohyblivou desetinnou čárkou v rozsahu od $-3{,}4^{38}$ až $-1{,}18^{38}$, 0, $1{,}18^{38}$ až $3{,}4^{38}$.

115 Jaké existují datové typy pro vyjádření finančních částek

Do skupiny číselných datových typů patří i datové typy pro vyjádření finanční částky v peněžní měně. Finanční částky se vyjadřují a počítají na pevný počet desetinných míst, zpravidla na dvě nebo čtyři desetinná místa. Pokud databázová platforma tento typ nepodporuje, můžete jej nahradit jiným datovým typem, například na platformě Oracle datovým typem `number(14,2)`.

- **money** – rozsah tohoto datového typu je od -2^{63} (-922 337 203 685 477,5808) do $2^{63} - 1$ (922 337 203 685 477,5807);
- **smallmoney** – rozsah je od -214 748,3648 do 214 748,3647.

116 Jaké existují datové typy na uložení celočíselných hodnot

- **bit** – tento datový typ slouží k vyjádření jen dvou hodnot, 0 nebo 1, neboli řečeno jinak pravdivostních hodnot PRAVDA / NEPRAVDA (`TRUE` / `FALSE`);
- **int** (integer) – celé číslo v rozsahu od -2^{31} (-2 147 483 648) do 2^{31} -1 (2 147 483 647) – tento datový typ zabírá 4 bajty;
- **smallint** – celé číslo v rozsahu od 2^{15} (-32 768) do $2^{15} - 1$ (32 767) – tento datový typ zabírá 2 bajty;
- **tinyint** – celé číslo v rozsahu od 0 do 255 – datový typ `tinyint` zabírá jeden bajt.

117 Oracle – datový typ Number

Oracle jako základní a v podstatě jediný nativní číselný typ používá datový typ `number(n_číslic, m_číslic_za_desetinnou_čárkou)`, který slouží k ukládání číselných údajů, ať už celých čísel, čísel s pevnou řádovou čárkou (využívá se například v účetnictví, kdy se účtuje na dvě desetinná místa) nebo desetinných čísel s pohyblivou řádovou čárkou používaných při vědeckotechnických výpočtech. Rozsah datového typu `number` je $1{,}0 \times 10^{-130}$ až $9{,}9...9 \times 10^{125}$ s přesností na 38 platných číslic. Celé číslo se deklaruje jako `number(n_číslic)` nebo ekvivalentním zápisem `number(n_číslic,0)`. Můžete dokonce deklarovat počet desetinných míst jako záporné číslo, například `number(7,-2)`. Takováto deklarace vede ke zrušení platnosti dvou platných číslic před desetinnou čárkou, tedy k zaokrouhlení na stovky.

118 Jaké existují znakové datové typy

Znakové datové typy slouží pro uložení textových údajů. Tyto údaje se skládají z písmen, číslic a jiných znaků. Můžeme sem zařadit datové typy:

- **char**(délka) – tento datový typ slouží pro uložení textového řetězce pevné délky, která je dána parametrem v závorce;
- **nchar**(délka) – tento datový typ slouží pro uložení textového řetězce pevné délky, která je dána parametrem v závorce ve vybrané národní znakové sadě;
- **varchar**(délka) – tento datový typ slouží pro uložení textového řetězce proměnné délky. Maximální délka textového řetězce je dána parametrem v závorce;
- **nvarchar**(délka) – tento datový typ slouží pro uložení textového řetězce proměnlivé délky ve zvolené národní znakové sadě. Maximální délka textového řetězce je dána parametrem v závorce.

119 Jaké existují datové typy pro uložení data a času

Tyto typy není nutné podrobněji popisovat, neboť vše podstatné je už vyjádřeno v jejich názvu. Pomocí datových typů patřících do této skupiny můžeme jednak vyjádřit konkrétní datum a čas, jednak i datumový a časový interval.

Poznámka: Bohužel datové typy pro uložení datumových a časových hodnot se asi nejvíce podílejí na určité složitosti migrace kódu z jedné databázové platformy na jinou. Jejich variabilita vyplývá hlavně z národních formátů pro datum a čas.

Abychom hlavně začátečníkům ulehčili základní orientaci v těchto datových typech, ukážeme si rozdíly pro tento datový typ přehledným způsobem pomocí tabulek:

Definice datového typu:

Microsoft SQL Server	`DATETIME`
Oracle	`DATE`
MySQL	*`1. DATETIME, TIMESTAMP`*

Vložení aktuálního data a času:

Microsoft SQL Server	`GETDATE()`
Oracle	*2. SYSDATE*
MySQL	`CURRENT_TIMESTAMP()`

120 K čemu slouží hodnota NULL

Jednotlivé sloupce různých datových typů mohou kromě konkrétních hodnot obsahovat i hodnotu `NULL`. Tato hodnota vlastně vyjadřuje neexistenci hodnoty proměnné. Je ale třeba přísně rozlišovat mezi nulovou hodnotou proměnné nebo prázdným řetězcem a hodnotou `NULL`. Nula je celkem konkrétní číslo, ale hodnota `NULL` znamená, že tento údaj neznáme.

Porovnání s hodnotou `NULL` vrací vždy hodnotu `NULL` (tuto větu si řekněte raději několikrát, jak uvidíte dále při tvorbě podmínek, je velmi důležitá), například:

```
SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
```

121 SQL Server – jak na řídké sloupce

Poměrně často se v praxi využívají v relačních databázích tabulky, které mají velký počet atributů, ačkoli v mnoha záznamech se ne vždy použijí úplně všechny. Není ojedinělé, když se z několika desítek či stovek atributů využijí v některých záznamech jen dva nebo tři a ostatní mají hodnotu `NULL`. SQL Server od verze 2008 dokáže takto řídké sloupce (anglicky sparse columns) uložit ve zhuštěné podobě, čímž se v mnohých případech ušetří velké množství místa v úložišti údajů. Zhuštěná tabulka má kromě „řídkých atributů“ pro každý záznam i jeden takzvaný souhrnný XML atribut, který obsahuje hodnoty těch sloupců, které se pro daný záznam vyskytují. Tento atribut je možné použít pro jednoduché zadávání údajů, kdy klasický příkaz pro vložení záznamu obsahuje mnoho názvů sloupců. Vkládání údajů do tabulky je standardní:

```
CREATE TABLE Produkty
(
      Id int,
      Typ nvarchar(16),
      Megapixely int SPARSE,
      Zoom nvarchar(8) SPARSE,
      ObvodPasu int SPARSE,
      Delka int SPARSE,
      PixelyH int SPARSE,
      PixelyV int SPARSE,
      Charakteristika XML COLUMN_SET FOR ALL_SPARSE_COLUMNS
);

INSERT INTO Produkty(Id, Typ, Megapixely, Zoom)
       VALUES (7024, 'Kamera', '6', '3x');
INSERT INTO Produkty(Id, Typ, ObvodPasu, Delka)
       VALUES (20956, 'Kalhoty', 32, 32)
```

```
INSERT INTO Produkty(Id, Typ, PixelyH, PixelyV)
       VALUES (1629, 'TV', 1280, 1024)
```

122 SQL Server – jak na filtrované indexy nad řídkými sloupci

Indexy dokážou při správném použití významným způsobem zrychlit vyhledávání údajů v databázi, je s nimi ale spojena i určitá režie a v některých případech zabírají na disku poměrně hodně místa. Podobně jako řídké sloupce najdou i filtrované indexy uplatnění hlavně v tabulkách obsahujících nerovnoměrně rozmístěné údaje. Filtrovaný index se týká jen těch položek, pro které je definován pomocí podmínky v klauzuli `WHERE`. Například pro schematickou tabulku zčásti věnované řídkým sloupcům bychom filtrovaný index vytvořili pomocí příkazu:

```
CREATE INDEX fi1 ON Tabulka(p1) WHERE jmeno=A OR jmeno=D
```

id	jmeno	p1	p2	p3	p4	p5	p6	p7	p8	p9
1	A	1								9
2	B		2		4					
3	C						6	7		
4	D	1				5				
5	E				4				8	
...										

123 SQL Server – použití filtrovaných indexů nad řídkými sloupci

I námět pro ukázkový příklad si můžeme vypůjčit z předchozí části a vytvořit například filtrovaný index nad některým řídkým sloupcem. Záměrem pro vytvoření tohoto indexu z hlediska aplikační logiky je urychlení vyhledávání údajů o kamerách, a právě na tento segment chce obchodní oddělení zaměřit svoji pozornost a na základě dotazů a analýz se pokusit o zvýšení prodeje.

```
CREATE INDEX ix on Produkty(Zoom) WHERE typ='Kamera' ;
```

Tento index výrazně zrychlí dotazy typu:

```
SELECT  Zoom, * FROM Produkty WHERE typ = 'Kamera'
```

aniž by samotný index zabíral v databázi nepřiměřeně mnoho místa.

124 SQL Server – jak na filtrovanou statistiku v tabulkách s řídkými sloupci

V tabulkách s řídkými sloupci má velký význam i filtrovaná statistika. Bylo by například úplně zbytečné provádět statistiku pro atributy `Megapixely` a `Zoom` u textilních výrobků, a naopak, pro `obvod pasu` u digitálních fotoaparátů:

```
CREATE STATISTICS stx ON Produkty(ObvodPasu) WHERE Typ='Kalhoty' ;
```

Statistiku je možné zobrazit příkazem typu:

```
DBCC SHOW_STATISTICS ("Produkty", stx) ;
```

případně v podobě histogramu příkazem:

```
DBCC SHOW_STATISTICS ("Produkty", stx) WITH HISTOGRAM ;
RANGE_HI_KEY RANGE_ROWS    EQ_ROWS       DISTINCT_RANGE_ROWS  AVG_RANGE_ROWS
------------ ------------- ------------- -------------------- --------------
32           0             1             0                    1
```

125 Definování uživatelských datových typů


pokročilý

Kromě implementovaných datových typů je možné definovat i vlastní datové typy. Požadavek na používání vlastních datových typů vyplývá z aplikační logiky. Velmi často se používají některé hodnoty z reálného života, například poštovní směrovací čísla, rodná čísla a podobně, dokonce tak často, že je výhodné definovat pro ně vlastní datový typ. Pro ukládání poštovního směrovacího čísla byste mohli vytvořit vlastní datový typ PSC, který by byl ekvivalentem datového typu `CHAR(6)`. Vysvětlování syntaxe pro vytvoření vlastního datového typu by zabralo dost místa, proto si ukážeme jen ilustrační příklad pro definování datového typu PSC pro poštovní směrovací číslo.

126 SQL Server – definování datového typu


pokročilý

Na platformě MS SQL Server se pro definování vlastního datového typu používá systémová uložená procedura `SP_ADDTYPE`:

```
USE master
EXEC sp_addtype psc, 'CHAR(6)', 'NOT NULL'
```

127 Oracle – definování datového typu


pokročilý

Na platformě Oracle můžete vytvářet tři druhy datových typů: podtypy, záznamy a tabulky PL/SQL. U podtypů není povoleno definovat délku:

```
SUBTYPE psc IS VARCHAR2 ;
```

Záznamy (records) můžete definovat například takto:

```
CREATE TYPE lokalita AS OBJECT
(
  mesto VARCHAR(25),
  stat  VARCHAR(20)
) ;
```

128 SQL Server – k čemu slouží datový typ HierarchyID


znalec

Datový typ `HierarchyID` je určen pro definování pozice v hierarchické struktuře kořenu k listům. Pomocí metod lze například definovat kořen, zjistit pozici prvku v hierarchii, zjistit jeho rodiče, zjistit, zda má potomky, nebo jej případně předat jinému rodiči:

```
CREATE TABLE Pracovnici
(
   Pozice        hierarchyid,
   Uroven        AS Pozice.GetLevel(),
   Id_prac       int UNIQUE NOT NULL,
```

```
   Jmeno         varchar(25),
   Funkce        varchar(20)
) ;
```

129 SQL Server – vložení kořenového elementu do hierarchické struktury

Jako první se do tabulky vloží kořenový element neboli element stojící na nejvyšší úrovni hierarchické struktury, v tomto případě nejvyšší nadřízený:

```
INSERT Pracovnici (Pozice, Id_prac, Jmeno, Funkce)
  VALUES (hierarchyid::GetRoot(), 1, 'Novak Alfonz', 'reditel') ;
```

130 SQL Server – vložení potomka do hierarchické struktury

Vytvoření tabulky a dokonce i vložení kořenového záznamu nevybočuje z běžné praxe vkládání záznamů pomocí příkazu `INSERT`. Při vkládání ostatních elementů, které jsou včleněny v hierarchické úrovni, je nutné uvést přímého předka. V tomto příkladu bude předkem zaměstnanec na kořenové úrovni hierarchie. Vkládaný záznam se totiž bude týkat zaměstnance, který je přímým podřízeným ředitele:

```
DECLARE @root hierarchyid
SELECT @root = hierarchyid::GetRoot() FROM pracovnici;
INSERT Pracovnici (Pozice, Id_prac, Jmeno, Funkce)
   VALUES (@root.GetDescendant(NULL, NULL), 2,'Plha Jan', 'vedouci
   marketingu') ;
```

131 SQL Server – uložení procedury na vložení potomka do hierarchické struktury

Vkládání záznamů na zadanou hierarchickou pozici vám může zjednodušit uložená procedura, která obsahuje kód založený na stejném principu, který byl použit pro vkládání potomka ředitele, jen je zevšeobecněný pro libovolnou hierarchickou pozici. První dva parametry uložené procedury určují pozici vkládaného záznamu v hierarchické struktuře. První parametr obsahuje identifikátor přímého nadřízeného a druhý parametr identifikátor vkládaného záznamu.

```
CREATE PROC Zapis(@id_m int, @id_p int, @jmeno varchar(25), @funkce
varchar(20))
AS
BEGIN
   DECLARE @manager hierarchyid, @potomek hierarchyid
   SELECT @manager = Pozice FROM Pracovnici
      WHERE Id_prac = @id_m
   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
   BEGIN TRANSACTION
      SELECT @potomek = max(Pozice) FROM Pracovnici
         WHERE Pozice.GetAncestor(1) = @manager;
      INSERT Pracovnici (Pozice, Id_prac, Jmeno, Funkce)
         VALUES(@manager.GetDescendant(@potomek, NULL), @id_p, @jmeno,
               @funkce)
   COMMIT
END ;
```

Pomocí uložené procedury je možné vložit další zaměstnance na správnou úroveň podnikové hierarchie poměrně intuitivně, stačí jen znát jejich nadřízeného:

```
EXEC Zapis 1, 3, 'Kecal Josef', 'vedouci pr' ;
EXEC Zapis 2, 4, 'Rach Petr', 'ucetni' ;
EXEC Zapis 2, 5, 'Kubikova Jana', 'prodejce' ;
EXEC Zapis 3, 6, 'Pekna Kamila', 'mluvci' ;
```

132 SQL Server – výpis hierarchické struktury

Po vložení záznamů se můžete pomocí příkazu SELECT podívat, v jaké podobě jsou údaje uloženy. Hierarchická pozice je vypsána v textové formě a poměrně srozumitelně vyjadřuje pozici příslušného záznamu:

```
Text_Pozice   Pozice       Id_prac     Uroven Jmeno           Funkce
------------- ------------ ----------- ------ --------------- ----------
/             0x           1           0      Novak Alfonz    reditel
/1/           0x58         2           1      Plha Jan        vedouci
                                                              marketingu
/1/1/         0x5AC0       4           2      Rach Petr       ucetni
/1/2/         0x5B40       5           2      Kubikova Jana   prodejce
/2/           0x68         3           1      Kecal Josef     vedouci pr
/2/1/         0x6AC0       6           2      Pekna Kamila    mluvci
```

133 SQL Server – vyhledávání v hierarchické struktuře

V hierarchické struktuře je možné zjišťovat údaje ve vztahu k pozici záznamu v této struktuře. Kořenový prvek celé hierarchické struktury lze zjistit pomocí funkce `GetRoot`:

```
SELECT Pozice.ToString() AS Text_Pozice, * FROM Pracovnyici
WHERE Pozice = hierarchyid::GetRoot() ;
```

```
Text_Pozice   Pozice       Id_prac     Uroven Jmeno           Funkce
------------- ------------ ----------- ------ --------------- ----------
/             0x           1           0      Novak Alfonz    reditel
```

Pro výpis přímých podřízených konkrétního pracovníka je možné použít dotaz využívající funkci `GetAncestor()`:

```
DECLARE @manager hierarchyid
SELECT @manager = Pozice FROM Pracovnici WHERE ID_prac = 1 ;
SELECT Pozice.ToString() AS Text_Pozice, * FROM Pracovnici
   WHERE Pozice.GetAncestor(1) = @manager ;
```

```
Text_Pozice   Pozice       Id_prac     Uroven Jmeno           Funkce
------------- ------------ ----------- ------ --------------- ----------
/1/           0x58         2           1      Plha Jan        vedouci
                                                              marketingu
/2/           0x68         3           1      Kecal Josef     vedouci pr
```

134 SQL Server – změna pozice v hierarchické struktuře

Snad největší výhodou hierarchického datového typu je jednoduchá možnost změny pozice ve struktuře, například v situaci, kdy v naší fiktivní firmě bude z organizačních důvodů přesunuta pracovnice Kamila Pěkná z funkce mluvčí na funkci prodejce. To v konečném důsledku znamená, že její přímým nadřízeným už nebude vedoucí PR Josef Kecal, ale vedoucí marketingu Jan Plha.

```
DECLARE @zam hierarchyid , @OldManag hierarchyid, @NewManag hierarchyid
SELECT @zam = Pozice FROM Pracovnici WHERE ID_prac = 6;      --Pekna
SELECT @OldManag = Pozice FROM Pracovnici WHERE ID_prac = 3; --Kecal
SELECT @NewManag = Pozice FROM Pracovnici WHERE ID_prac = 2; --Plha

UPDATE Pracovnici
   SET Pozice = @Zam.GetReparentedValue(@OldManag, @NewManag)
   WHERE Pozice = @zam;
GO
```

135 SQL Server – k čemu slouží datový typ TABLE

`TABLE` je uživatelem definovaný datový typ umožňující ukládat tabulková data. Nejčastěji se používá jako vstupní parametr pro funkce a uložené procedury. Musí však být `READONLY`. Pro tento datový typ lze definovat indexy a omezení. Podobně jako u jiných uživatelsky definovaných datových typů je i velikost typu `TABLE` omezena na 2 GB. V souvislosti s datovým typem TABLE přibyl i nový systémový pohled `sys.table_types`.

136 SQL Server – příklad bez použití datového typu TABLE

Typickým scénářem použití je předávání údajů tabulkového typu funkcím a uloženým procedurám. Na praktickém příkladě můžete porovnat standardní způsob vkládání nových záznamů přes parametrickou uloženou proceduru a přes parametr typu `TABLE`. V příkladu použijeme jednoduchou tabulku s evidencí zaměstnanců.

```
CREATE TABLE zamestnanci
(
   ID int NOT NULL,
   jmeno nvarchar(30),
   email nvarchar(30)
)
```

Klasicky by se údaje mohly vkládat přes parametrickou uloženou proceduru pro přidávání záznamů.

```
CREATE PROCEDURE MultiInsert(@id int, @meno nvarchar(30), @email nvarchar(30))
AS
BEGIN
   INSERT INTO dbo.zamestnanci
   VALUES(@id, @meno, @email)
END
```

Můžete vyzkoušet přidat pomocí této uložené procedury několik záznamů:

```
EXECUTE MultiInsert 1, 'Mickey Mouse', 'mickey@disney.com'
EXECUTE MultiInsert 2, 'Donald Duck', 'donald@disney.com'
EXECUTE MultiInsert 3, 'Snehurka', 'snehurka@disney.com'
```

137 SQL Server – příklad použití datového typu TABLE

Stejnou funkčnost lze realizovat pomocí nového datového typu `TABLE` s vhodnou strukturou pro ukládání údajů o zaměstnancích:

```
CREATE TYPE ZamestnanciTabTyp AS TABLE
( ID int NOT NULL,  meno nvarchar(30),  email nvarchar(30))
```

Poznámka: O vytvoření datového typu TABLE se můžete přesvědčit pomocí nástroje SQL Server Management Studio. Informace o něm získáte přes kontextovou nabídku v záložce Programability –> Types –> User-Defined Table Types.

Tento datový typ bude vstupním parametrem uložené procedury pro přidávání záznamů s využitím datového typu `TABLE`:

```
CREATE PROCEDURE MultiInsert2(@zamestnanec ZamestnanciTabTyp READONLY)
AS
BEGIN
   INSERT INTO dbo.zamestnanci
      SELECT * FROM @zamestnanec
END
```

138 SQL Server – přidávání údajů s využitím datového typu TABLE

Údaje se budou ukládat do dočasné proměnné uživatelem definovaného tabulkového datového typu `TABLE`. Následně se pomocí uložené procedury uloží z proměnné do databázové tabulky:

```
DECLARE @zamestnanci ZamestnanciTabTyp
INSERT INTO @zamestnanci
   VALUES (1, 'Mickey Mouse', 'mickey@disney.com')
INSERT INTO @zamestnanci
   VALUES (2, 'Donald Duck', 'donald@disney.com')
INSERT INTO @zamestnanci
   VALUES (3, 'Snehurka', 'snehurka@disney.com')
EXECUTE MultiInsert2 @zamestnanci
```

Výhoda postupu s využitím parametru datového typu `TABLE` spočívá v tom, že se v tomto případě uložená procedura nevolala třikrát, ale jen jednou. Taktéž v případě změny struktury tabulkových údajů není nutné uloženou proceduru rekompilovat.

139 SQL Server – informace o datových typech TABLE

Informace o datových typech `TABLE` můžete získat ze systémových pohledů:

```
SELECT * FROM sys.types
SELECT * FROM sys.table_types
```

Datový typ `TABLE` je součástí databázového schématu, takže při deklarování proměnných můžete určovat i vlastníky datových typů:

```
DECLARE @t AS dbo.ZamestnanciTabTyp
```

Nový, uživatelem definovaný datový typ `TABLE` je výhodný i pro dávkové aktualizace, migraci údajů a při velkém objemu údajů posílaných z klienta na server.

140 Omezení pro atributy databázových tabulek

Aby se zabránilo zadávání nesprávných hodnot do databázových tabulek, je v některých případech potřebné zavést pro některé atributy databázových tabulek určitá omezení. Bylo by nesmyslné stanovit některý sloupec jako primární klíč, podle něhož se identifikují záznamy, pokud by polovina záznamů měla v tomto sloupci hodnotu `NULL` a podob-

ně. Omezení se mohou vztahovat ke konkrétnímu atributu (sloupci) nebo k celé tabulce. Syntaktický zápis je následovný:

```
sloupec [CONSTRAINT název_omezení] typ_omezení
```

141 Vyloučení hodnoty NULL – NOT NULL

Nejčastěji se používá omezení `NOT NULL`, kdy atribut, na nějž se toto omezení vztahuje, nesmí obsahovat hodnotu `NULL`.

```
CREATE TABLE pokus_not_null
(
  sloupec INT NOT NULL
);
```

Pokud byste chtěli použít úplnou syntaxi a omezení i pojmenovat, příkaz by byl ve tvaru:

```
CREATE TABLE pokus_not_null
(
  sloupec INT CONSTRAINT identifikator NOT NULL
);
```

142 Co je implicitní hodnota v sloupci – DEFAULT

Při zadávání údajů, které se ukládají do databázových tabulek, se často vyskytuje situace, kdy se při vytvoření záznamu zadají jen některé hodnoty atributů a ostatní se vyplní později, přičemž v případě, že se nevyplní, zůstane v nich implicitní hodnota, kterou je možné pro příslušný atribut definovat při vytváření databázové tabulky. Pro tento účel se využívá klauzule `DEFAULT`, například:

```
CREATE TABLE pokus_default
(
  jmeno VARCHAR(20,
  stav VARCHAR(10) DEFAULT 'nedefinovaný'
);
```

143 Jak na kontrolu zadávaných hodnot – CHECK

Pomocí omezení `CHECK` můžete definovat podmínky a omezení, které musejí být splněny pro každý záznam. Můžete například definovat omezení pro sloupec databázové tabulky tak, aby hodnoty v tomto sloupci byly z intervalu 1 až 40.

```
CREATE TABLE pokus_check
(
  sloupec INT,
  CHECK (sloupec BETWEEN 1 AND 40)
);
```

144 Co je omezení na unikátní hodnotu – UNIQUE

Toto omezení stanovuje unikátnost hodnoty. To znamená, že sloupec (nebo kombinace sloupců) musí být jednoznačný pro všechny záznamy v tabulce. Na rozdíl od primárního klíče můžete toto omezení aplikovat i na více sloupců tabulky.

```
CREATE TABLE pokus_unique_key
(
```