



28  
hodinových  
lekci

Ryan Stephens, Ron Plew, Arie D. Jones

# Naučte se SQL

## za 28 dní

Stačí hodina  
denně

Od základů až  
po komplexní dotazy

Krátké nenáročné lekce  
umožňující rychlý pokrok

Otázky a odpovědi, kvíz a cvičení  
na konci každé kapitoly



Ke stažení příkazy pro  
vytvoření databázových  
tabulek z knihy

 C P R E S S

**SAMS**

**Ryan Stephens, Ron Plew, Arie D. Jones**

# **Naučte se SQL za 28 dní**

**Computer Press  
Brno  
2012**

# Naučte se SQL za 28 dní

**Ryan Stephens, Ron Plew, Arie D. Jones**

**Překlad:** Lukáš Krejčí

**Obálka:** Martin Sodomka

**Odpovědný redaktor:** Martin Herodek

**Technický redaktor:** Jiří Matoušek

Authorized translation from the English language edition, entitled SAMS TEACH YOURSELF SQL IN ONE HOUR A DAY, 5th Edition, 0672330253 by STEPHENS, RYAN; PLEW, RON; JONES, ARIE D., published by Pearson Education, Inc, publishing as Sams Publishing, Copyright © 2009 by Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc. CZECH language edition published by Albatros Media a.s., Copyright © 2010.

Autorizovaný překlad z originálního anglického vydání SAMS TEACH YOURSELF SQL IN ONE HOUR A DAY. Originální copyright: published by Pearson Education, Inc, publishing as Sams Publishing, Copyright © 2009.

Překlad: © Albatros Media a.s., 2010.

Objednávky knih:

<http://knihy.cpress.cz>

[www.albatrosmedia.cz](http://www.albatrosmedia.cz)

[eshop@albatrosmedia.cz](mailto:eshop@albatrosmedia.cz)

bezplatná linka 800 555 513

ISBN 978-80-251-2700-1

Vydalo nakladatelství Computer Press v Brně roku 2012 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 16 001.

© Albatros Media a. s. Všechna práva vyhrazena. Žádná část této publikace nesmí být kopírována a rozmnožována za účelem rozšiřování v jakékoli formě či jakýmkoli způsobem bez písemného souhlasu vydavatele.

Dotisk 1. vydání.

  
**ALBATROS** MEDIA a.s.

# Stručný obsah

Úvod	25
ČÁST I	
Úvod do SQL	29
1. Seznámení s jazykem SQL	31
2. Začínáme s dotazy	45
3. Výrazy, podmínky a operátory	61
4. Klauzule v dotazech jazyka SQL	103
5. Spojování tabulek	135
6. Vkládání poddotazů do dotazů	161
7. Formování dat pomocí vestavěných funkcí	185
ČÁST II	
Návrh databáze	229
8. Normalizace databáze	231
9. Tvorba a údržba tabulek	241
10. Řízení integrity dat	263
ČÁST III	
Manipulace s daty	279
11. Manipulace s daty	281
12. Datum a čas v jazyku SQL	303
13. Tvorba pohledů	321
14. Řízení transakcí	341
ČÁST IV	
Administrace databáze	355
15. Tvorba indexů na tabulkách pro zlepšení výkonu	357
16. Racionalizace příkazů jazyka SQL pro zlepšení výkonu	373
17. Databázová bezpečnost	393
18. Datový slovník (systémový katalog)	413

<b>ČÁST V</b>	
<b>Další SQL objekty</b>	<b>439</b>
19. Dočasné tabulky, uložené procedury, spouštěče a kurzory	441
20. Nové objekty v současném standardu	459
<b>ČÁST VI</b>	
<b>Pokročilé techniky SQL</b>	<b>473</b>
21. Generování příkazů jazyka SQL pomocí jazyka SQL	475
22. Tvorba komplexních dotazů jazyka SQL	497
23. Ladění příkazů jazyka SQL	515
24. Vkládání kódu jazyka SQL při programování aplikací	535
<b>ČÁST VII</b>	
<b>SQL v různých databázových implementacích</b>	<b>545</b>
25. Použití nástroje SQL*Plus databázového systému Oracle pro generování zpráv	547
26. Úvod do jazyka PL/SQL databázového systému Oracle	585
27. Seznámení s jazykem Transact-SQL	613
28. Databázový systém MySQL na unixovém systému	635
<b>ČÁST VIII</b>	
<b>Přílohy</b>	<b>647</b>
A. Odpovědi	649
B. Ukázky kódu pro vytvoření tabulek	677
C. Ukázky kódu pro naplnění tabulek	689
D. Instalace databázového systému MySQL pro cvičení	703
E. Přehled nejčastěji používaných příkazů jazyka SQL	705
F. Přehled nejčastěji používaných funkcí jazyka SQL	711

# Obsah

O autorech	23
Věnování	24
Poděkování	24
Poznámka redakce českého vydání	24
<b>Úvod</b>	<b>25</b>
Komu je kniha určena	25
Uspořádání knihy	25
Použité konvence	26
Praktická cvičení v databázovém systému MySQL	27
Zdrojový kód	27

## ČÁST I

### Úvod do SQL

---

LEKCE 1	
<b>Seznámení s jazykem SQL</b>	<b>31</b>
Stručná historie jazyka SQL	31
Stručná historie databází	32
Současná podoba databází	36
Jazyk pro více produktů	37
Prvotní implementace	37
Jazyk SQL a vývoj aplikací typu klient-server	38
Přehled jazyka SQL	38
Populární implementace jazyka SQL	39
MySQL	39
Oracle	39
Microsoft SQL Server a Sybase	40
IBM DB2	40
ODBC	40
Pozice kódu jazyka SQL ve vytvářené aplikaci	41
Shrnutí	43
Otázky a odpovědi	43
Úkoly pro vás	44
Kvíz	44
Cvičení	44

## LEKCE 2

<b>Začínáme s dotazy</b>	<b>45</b>
Pozadí jazyka SQL	45
Osvojení základní syntaxe dotazů	45
Stavební bloky pro získávání dat: SELECT a FROM	47
Dotazy v praxi	48
Píšeme první dotaz	49
Ukončení příkazu jazyka SQL	50
Vybírání jednotlivých sloupců	51
Změna pořadí sloupců	51
Vybírání jiných tabulek	53
Vybírání odlišných hodnot	54
Shrnutí	56
Otázky a odpovědi	56
Úkoly pro vás	56
Kvíz	58
Cvičení	59

## LEKCE 3

<b>Výrazy, podmínky a operátory</b>	<b>61</b>
Pracujeme s dotazovými výrazy	61
Podmínky v dotazech	62
Jak používat operátory	63
Aritmetické operátory	64
Porovnávací operátory	75
Znakové operátory	83
Logické operátory	89
Množinové operátory	93
Ostatní operátory: IN a BETWEEN	97
Shrnutí	99
Otázky a odpovědi	99
Úkoly pro vás	100
Kvíz	101
Cvičení	101

## LEKCE 4

<b>Klauzule v dotazech jazyka SQL</b>	<b>103</b>
Specifikace kritérií pomocí klauzule WHERE	104
Klauzule ORDER BY	106
Klauzule GROUP BY	115
Klauzule HAVING	121

<b>Kombinování klauzulí</b>	<b>127</b>
Příklad 4.1	127
Příklad 4.2	128
Příklad 4.3	128
Příklad 4.4	130
<b>Shrnutí</b>	<b>132</b>
<b>Otázky a odpovědi</b>	<b>132</b>
<b>Úkoly pro vás</b>	<b>132</b>
Kvíz	133
Cvičení	133
<b>LEKCE 5</b>	
<b>Spojování tabulek</b>	<b>135</b>
<b>Spojování více tabulek v jediném příkazu SELECT</b>	<b>135</b>
Křížové spojování tabulek	136
Hledání správného sloupce	141
<b>Spojování tabulek na základě rovnosti</b>	<b>142</b>
<b>Spojování tabulek na základě nerovnosti</b>	<b>149</b>
<b>Vnější a vnitřní spojení</b>	<b>151</b>
<b>Spojení tabulky se sebou</b>	<b>155</b>
<b>Shrnutí</b>	<b>157</b>
<b>Otázky a odpovědi</b>	<b>157</b>
<b>Úkoly pro vás</b>	<b>158</b>
Kvíz	159
Cvičení	160
<b>LEKCE 6</b>	
<b>Vkládání poddotazů do dotazů</b>	<b>161</b>
<b>Sestavujeme poddotazy</b>	<b>163</b>
<b>Agregační funkce v poddotazech</b>	<b>168</b>
<b>Vnořování poddotazů</b>	<b>170</b>
<b>Vnější reference s korelovanými poddotazy</b>	<b>173</b>
<b>Klíčová slova EXISTS, ANY a ALL</b>	<b>176</b>
<b>Shrnutí</b>	<b>181</b>
<b>Otázky a odpovědi</b>	<b>181</b>
<b>Úkoly pro vás</b>	<b>182</b>
Kvíz	182
Cvičení	183



## LEKCE 7

<b>Formování dat pomocí vestavěných funkcí</b>	<b>185</b>
<b>Agregační funkce pro sumarizaci dat</b>	<b>185</b>
Funkce COUNT	186
Funkce SUM	186
Funkce AVG	188
Funkce MAX	189
Funkce MIN	189
Funkce VARIANCE	190
Funkce STDDEV	191
<b>Funkce pro formátování data a času</b>	<b>192</b>
Funkce ADD_MONTHS/DATE_ADD	192
Funkce LAST_DAY	194
Funkce MONTHS_BETWEEN	195
Funkce NEXT_DAY	196
Funkce SYSDATE	197
<b>Funkce pro aritmetické operace</b>	<b>198</b>
Funkce ABS	198
Funkce CEIL a FLOOR	199
Funkce EXP	200
Funkce LN a LOG	200
Funkce MOD	201
Funkce POWER	202
Funkce SIGN	202
Funkce SQRT	203
<b>Funkce pro změnu vzhledu znakových hodnot</b>	<b>204</b>
Funkce CHR	204
Funkce CONCAT	204
Funkce INITCAP	206
Funkce LOWER a UPPER	206
Funkce LPAD a RPAD	207
Funkce LTRIM a RTRIM	208
Funkce REPLACE	209
Funkce SUBSTR	211
Funkce TRANSLATE	215
Funkce INSTR	215
Funkce LENGTH	216
<b>Převodní funkce</b>	<b>216</b>
Funkce TO_CHAR	217
Funkce TO_NUMBER	218
<b>Ostatní funkce</b>	<b>218</b>

Funkce GREATEST a LEAST	218
Funkce USER	219
<b>Doplňující příklady znakových funkcí databázového systému MySQL</b>	<b>219</b>
Funkce LENGTH	220
Funkce LOCATE	220
Funkce INSTR	220
Funkce LPAD	220
Funkce RPAD	221
Funkce LEFT	221
Funkce RIGHT	221
Funkce SUBSTRING	221
Funkce LTRIM	222
Funkce RTRIM	222
Funkce TRIM	222
<b>Doplňující příklady funkcí databázového systému MySQL pro práci s datem</b>	<b>222</b>
Funkce DATE_FORMAT	223
Funkce TIME_FORMAT	224
Funkce CURDATE	224
Funkce CURTIME	224
<b>Shrnutí</b>	<b>224</b>
<b>Otázky a odpovědi</b>	<b>225</b>
<b>Úkoly pro vás</b>	<b>225</b>
Kvíz	226
Cvičení	227

## ČÁST II

### Návrh databáze

#### LEKCE 8

<b>Normalizace databáze</b>	<b>231</b>
Normalizace databáze	231
Holá databáze	231
Logický návrh databáze	231
Potřeby koncového uživatele	232
Redundance dat	232
<b>Normální formy</b>	<b>233</b>
První normální forma	233
Druhá normální forma	234
Třetí normální forma	235
<b>Normalizace v praxi</b>	<b>236</b>
Referenční integrita	236
<b>Výhody normalizace</b>	<b>237</b>

Nevýhody normalizace	237
Denormalizace databáze	238
Shrnutí	238
Otázky a odpovědi	239
Úkoly pro vás	239
Kvíz	239
Cvičení	239

## LEKCE 9

<b>Tvorba a údržba tabulek</b>	<b>241</b>
Začínáme příkazem CREATE DATABASE	241
Možnosti příkazu CREATE DATABASE	242
Návrh databáze	243
Tvorba datového slovníku (systémového katalogu)	244
Tvorba klíčových polí	245
Rozbití dat	245
Definování tabulek pomocí příkazu CREATE TABLE	246
Název tabulky	247
Název pole	247
Datové typy pole	247
Umístění a velikost tabulky	252
Vytvoření tabulky ze stávající tabulky	253
Změna struktury tabulky pomocí příkazu ALTER TABLE	255
Příkaz DROP TABLE	258
Příkaz DROP DATABASE	259
Práce s příkazy DROP TABLE a DROP DATABASE	259
Shrnutí	259
Otázky a odpovědi	259
Úkoly pro vás	260
Kvíz	260
Cvičení	261

## LEKCE 10

<b>Řízení integrity dat</b>	<b>263</b>
Seznámení s omezeními	263
Integrita dat	263
Proč používat omezení	264
Typy omezení	264
Omezení NOT NULL	265
Omezení ve formě primárního klíče	266
Omezení ve formě jedinečnosti	268

Omezení ve formě cizího klíče	269
Omezení ve formě kontroly	270
<b>Správa omezení</b>	<b>272</b>
Správné pořadí omezení	272
Různé přístupy ke tvorbě omezení	273
Ukázková hlášení referenční integrity databázového systému Oracle	273
<b>Shrnutí</b>	<b>276</b>
<b>Otázky a odpovědi</b>	<b>277</b>
<b>Úkoly pro vás</b>	<b>277</b>
Kvíz	278
Cvičení	278

## ČÁST III

### Manipulace s daty

#### LEKCE 11

<b>Manipulace s daty</b>	<b>281</b>
Seznámení s příkazy pro manipulaci s daty	281
Zadávání dat pomocí příkazu INSERT	282
Zadávání jednoho záznamu pomocí příkazu INSERT...VALUES	282
Vkládání hodnot NULL	284
Vkládání jedinečných hodnot	285
Zadávání většího počtu záznamů pomocí příkazu INSERT...SELECT	286
Modifikace stávajících dat pomocí příkazu UPDATE	289
Odstraňování informací pomocí příkazu DELETE	292
Importování a exportování dat z cizích zdrojů	296
Microsoft Access	296
Microsoft SQL Server	297
Oracle	298
MySQL	298
Shrnutí	299
Otázky a odpovědi	299
Úkoly pro vás	300
Kvíz	300
Cvičení	301

#### LEKCE 12

<b>Datum a čas v jazyku SQL</b>	<b>303</b>
Způsob uložení data a času	303
Datové typy standardu ANSI pro datum a čas	303
Prvky datového typu DATETIME	304
Implementace specifických datových typů	304

<b>Aplikace funkcí pro práci s časem v dotazech</b>	<b>305</b>
Aktuální datum	305
Časová pásma	307
Přičítání času ke kalendářním datům	307
Odečítání kalendářních dat	309
Porovnávání datových a časových období	311
Další funkce pro práci s datem	311
<b>Převod mezi formáty kalendářních dat</b>	<b>312</b>
Datové obrazy	313
Převod kalendářních dat na znakové řetězce	315
Převod znakových řetězců na kalendářní data	316
<b>Shrnutí</b>	<b>317</b>
<b>Otázky a odpovědi</b>	<b>317</b>
<b>Úkoly pro vás</b>	<b>317</b>
Kvíz	318
Cvičení	318
<b>LEKCE 13</b>	
<b>Tvorba pohledů</b>	<b>321</b>
Seznámení s pohledy	321
Používáme pohledy	322
Jednoduchý pohled	324
Přejmenování sloupců	326
Zpracování pohledů	327
Omezení klauzule SELECT	331
Modifikace dat v pohledu	331
Nejčastější využití pohledů	334
Odstranění pohledu příkazem DROP VIEW	337
Shrnutí	338
Otázky a odpovědi	338
Úkoly pro vás	339
Kvíz	339
Cvičení	339
<b>LEKCE 14</b>	
<b>Řízení transakcí</b>	<b>341</b>
Správa transakcí	341
Bankovní aplikace	342
Zahájení transakce	343
Dokončení transakce	345
Zrušení transakce	347

Záchytné body transakce	350
Shrnutí	352
Otázky a odpovědi	353
Úkoly pro vás	353
Kvíz	353
Cvičení	353

## ČÁST IV

### Administrace databáze

#### LEKCE 15

<b>Tvorba indexů na tabulkách pro zlepšení výkonu</b>	<b>357</b>
Seznámení s indexy	357
Rady pro práci s indexy	365
Vytváření indexů na více než jednom poli	365
Klíčové slovo UNIQUE v příkazu CREATE INDEX	368
Indexy a spojování tabulek	369
Klastrované indexy	370
Shrnutí	371
Otázky a odpovědi	371
Úkoly pro vás	371
Kvíz	371
Cvičení	372

#### LEKCE 16

<b>Racionalizace příkazů jazyka SQL pro zlepšení výkonu</b>	<b>373</b>
Pište příkazy jazyka SQL čitelně	374
Nepoužívejte skenování celé tabulky	375
Přidání nového indexu	375
Uspořádání prvků v dotazu	376
Procedury	378
Nepoužívejte operátor OR	378
OLAP a OLTP	379
Dolaďování systému OLTP	380
Dolaďování systému OLAP	380
Dávkové zátěže a transakční zpracování	380
Optimalizace načítání dat zahazením indexů	382
Příkaz COMMIT	382
Přestavování tabulek a indexů v dynamickém prostředí	384
Dolaďování databáze	385
Identifikování výkonnostních překážek	388

Použití vestavěných doladovacích nástrojů	389
Shrnutí	389
Otázky a odpovědi	390
Úkoly pro vás	390
Kvíz	390
Cvičení	391
<b>LEKCE 17</b>	
<b>Databázová bezpečnost</b>	<b>393</b>
Role bezpečnosti při správě databáze	393
Oblíbené databázové produkty a bezpečnost	394
Bezpečnost v databázových systémech Oracle Express a MySQL	395
Tvorba uživatelů	395
Tvorba rolí	397
Uživatelská oprávnění	399
Použití pohledů pro účely zabezpečení	406
Synonyma místo pohledů	407
Řešení bezpečnostních problémů pomocí pohledů	408
Klauzule WITH GRANT OPTION	409
Shrnutí	410
Otázky a odpovědi	410
Úkoly pro vás	411
Kvíz	411
Cvičení	411
<b>LEKCE 18</b>	
<b>Datový slovník (systémový katalog)</b>	<b>413</b>
Seznámení s datovým slovníkem	413
Identifikování uživatelů datového slovníku	414
Obsah datového slovníku	414
Datový slovník databázového systému Oracle	415
Datový slovník databázového systému MySQL	415
Pohled do datového slovníku databázového systému Oracle	415
Pohledy pro uživatele	416
Pohledy pro správce databáze	423
Pohledy dynamického výkonu	431
Pohled do datového slovníku databázového systému MySQL	432
Příkazy pro zobrazení tabulek v databázovém systému MySQL	433
Databáze INFORMATION_SCHEMA	433
Shrnutí	435
Otázky a odpovědi	436

Úkoly pro vás	436
Kvíz	436
Cvičení	437

## ČÁST V

### Další SQL objekty

#### LEKCE 19

<b>Dočasné tabulky, uložené procedury, spouštěče a kurzory</b>	<b>441</b>
Vytváříme dočasné tabulky	441
Používáme kurzory	445
Vytvoření kurzoru	446
Otevření kurzoru	446
Posouvání kurzoru	446
Testování stavu kurzoru	447
Uzavření kurzoru	448
Rozsah platnosti kurzorů	448
Vytváříme a používáme uložené procedury	449
Odstranění uložené procedury	450
Navrhujeme a používáme spouštěče	451
Spouštěče a transakce	452
Omezení při používání spouštěčů	453
Vnořené spouštěče	453
Používáme vložený kód jazyka SQL	453
Statický a dynamický kód jazyka SQL	454
Shrnutí	455
Otázky a odpovědi	456
Úkoly pro vás	456
Kvíz	456
Cvičení	457

#### LEKCE 20

<b>Nové objekty v současném standardu</b>	<b>459</b>
Příkaz CREATE ROLE	459
Tvorba spouštěčů	461
Příkaz CREATE TYPE	463
Regulární výrazy	467
Datový typ BLOB	468
Krátký příklad kódu jazyka XML	469
Shrnutí	470
Otázky a odpovědi	470



<b>Úkoly pro vás</b>	<b>470</b>
Kvíz	471
Cvičení	471

## ČÁST VI

### Pokročilé techniky SQL

#### LEKCE 21

<b>Generování příkazů jazyka SQL pomocí jazyka SQL</b>	<b>475</b>
Generování příkazů jazyka SQL	475
Nové povely nástroje SQL*Plus	476
Povel SET ECHO	477
Povel SET FEEDBACK	477
Povel SET HEADING	477
Povel SPOOL	477
Povel START	478
Povel EDIT	478
Počítání řádků v tabulkách	478
Udělení systémových práv více uživatelům	482
Udělení práv na vlastní tabulky jinému uživateli	484
Deaktivace omezení tabulky kvůli načtení dat	486
Tvorba více synonym jednou ranou	487
Tvorba pohledů na svých tabulkách	490
Vyprázdnění všech tabulek v daném schématu	491
Generování systémových skriptů pomocí jazyka SQL	492
Praktická aplikace generování kódu jazyka SQL a dalších principů	493
Shrnutí	494
Otázky a odpovědi	494
Úkoly pro vás	495
Kvíz	495
Cvičení	496

#### LEKCE 22

<b>Tvorba komplexních dotazů jazyka SQL</b>	<b>497</b>
Příkazy CREATE TABLE	497
Příklady složitých dotazů	500
Výpočet věku z data narození	500
Rozdělení části dne na hodiny, minuty a vteřiny	501
Převod bajtů na kilobajty a megabajty	503
Zpráva o fragmentaci databáze	504
Poddotazy v jazyku DML	504

Formátování kalendářních dat	505
Poddotaz zahrnující maximální hodnotu	506
Více poddotazů	507
Formátování číselných hodnot pomocí lomítek a mezer	507
Zvyšování číselných hodnot o zadaný podíl	508
Zjištění další nejvyšší hodnoty ve sloupci	508
Práce s hodnotami NULL	510
<b>Tipy pro sestavování komplexních dotazů</b>	<b>512</b>
<b>Shrnutí</b>	<b>513</b>
<b>Otázky a odpovědi</b>	<b>513</b>
<b>Úkoly pro vás</b>	<b>514</b>
Kvíz	514
Cvičení	514
<b>LEKCE 23</b>	
<b>Ladění příkazů jazyka SQL</b>	<b>515</b>
<b>Běžné chyby v příkazech jazyka SQL</b>	<b>515</b>
Neexistující tabulka či pohled	515
Neplatné uživatelské jméno nebo heslo	516
Chybí klíčové slovo FROM	516
Nesprávně použitá seskupující funkce	517
Neplatný název sloupce	518
Chybějící klíčové slovo	519
Chybějící levá závorka	519
Chybějící pravá závorka	520
Chybějící čárka	520
Nejednoznačně definovaný sloupec	521
Nesprávně ukončený příkaz jazyka SQL	521
Chybějící výraz	522
Nedostatek argumentů pro funkci	522
Nedostatek hodnot	523
Porušení integritního omezení – rodičovský klíč nenalezen	523
Databáze Oracle není k dispozici	524
Vkládaná hodnota je pro sloupec příliš velká	524
TNS: Posluchač nemohl vyhodnotit identifikátor SID uvedený v deskriptoru připojení	525
Nedostatečné právo pro udělování práv	525
Přepínací znak v příkazu – neplatný znak	525
Nelze vytvořit soubor operačního systému	526

<b>Běžné logické chyby</b>	<b>526</b>
Rezervovaná slova v příkazech jazyka SQL	526
Příkaz DISTINCT při výběru více sloupců	527
Zahození nekvalifikované tabulky	527
Veřejná synonyma v databázi s více schématy	528
Obávaný kartézský součin	528
Neschopnost prosadit vstupní standardy	529
Neschopnost prosadit konvence v oblasti struktury systému souborů	529
Rozsáhlé tabulky a výchozí parametry úložiště	529
Umísťování objektů do systémového prostoru tabulek	530
Neschopnost zkomprimovat rozsáhlé soubory zálohy	531
Neschopnost rozplánovat systémové prostředky	531
<b>Jak se vyhnout problémům s daty</b>	<b>531</b>
<b>Shrnutí</b>	<b>531</b>
<b>Otázky a odpovědi</b>	<b>532</b>
<b>Úkoly pro vás</b>	<b>532</b>
Kvíz	532
Cvičení	533
<b>LEKCE 24</b>	
<b>Vkládání kódu jazyka SQL při programování aplikací</b>	<b>535</b>
Letmý pohled na několik nástrojů pro vývoj aplikací	535
ODBC	535
Oracle Express	536
SQL v jazyku Java přes rozhraní JDBC	536
SQL v prostředí .NET přes rozhraní OleDb	536
Přípravy pro databázový systém Oracle	536
<b>Tvorba databáze</b>	<b>537</b>
<b>Jazyk SQL v prostředí Javy</b>	<b>540</b>
<b>Jazyk SQL v prostředí .NET</b>	<b>542</b>
<b>Shrnutí</b>	<b>543</b>
<b>Otázky a odpovědi</b>	<b>543</b>
<b>Úkoly pro vás</b>	<b>544</b>
Kvíz	544
Cvičení	544

## ČÁST VII

**SQL v různých databázových implementacích**

## LEKCE 25

**Použití nástroje SQL\*Plus databázového systému****Oracle pro generování zpráv****547**

Seznámení s nástrojem SQL\*Plus

547

Paměť nástroje SQL\*Plus

547

Zobrazení struktury tabulky pomocí příkazu DESCRIBE

552

Zobrazení nastavení pomocí příkazu SHOW

553

Souborové příkazy pro manipulaci se soubory

554

Příkazy SAVE, GET a EDIT

554

Zahájení souboru

555

Nasměrování výstupu dotazu

556

Přizpůsobení pracovního prostředí pomocí příkazů SET

558

Vynulování nastavení příkazem CLEAR

561

Formátování výstupu

561

TTITLE a BTITLE

561

Formátování sloupců (COLUMN, HEADING, FORMAT)

562

Tvorba zprávy a skupinových souhrnů

564

Příkaz BREAK ON

564

Příkaz COMPUTE

565

Proměnné v nástroji SQL\*Plus

567

Substituční proměnné (&amp;)

568

Příkaz DEFINE

568

Příkaz ACCEPT

569

Povel NEW\_VALUE

571

Tabulka DUAL

572

Funkce DECODE

573

Převody kalendářních dat

575

Spuštění série souborů s kódem jazyka SQL

578

Komentáře ve skriptech jazyka SQL

579

Tvorba pokročilých zpráv

580

Shrnutí

581

Otázky a odpovědi

582

Úkoly pro vás

582

Kvíz

582

Cvičení

582

## LEKCE 26

<b>Úvod do jazyka PL/SQL databázového systému Oracle</b>	<b>585</b>
Seznámení s jazykem PL/SQL	585
Struktura bloku jazyka PL/SQL	586
Oddíl DECLARE	587
Oddíl PROCEDURE	590
Oddíl EXCEPTION	595
Řízení transakcí v jazyku PL/SQL	598
Praktické příklady	598
Ukázkové tabulky a data	599
Jednoduchý blok jazyka PL/SQL	599
Rozvinutější příklad bloku jazyka PL/SQL	602
Používáme uložené procedury, balíčky a spouštěče	606
Ukázková procedura	606
Ukázkový balíček	607
Ukázkový spouštěč	608
Shrnutí	610
Otázky a odpovědi	610
Úkoly pro vás	611
Kvíz	611
Cvičení	611

## LEKCE 27

<b>Seznámení s jazykem Transact-SQL</b>	<b>613</b>
Přehled jazyka Transact-SQL	613
Rozšíření standardu ANSI SQL	614
Kdo může používat jazyk Transact-SQL	614
Základní prvky jazyka Transact-SQL	614
Datové typy	614
Znakové řetězce	615
Číselné datové typy	615
Datové typy pro práci s kalendářním datem	615
Datové typy pro práci s finančními částkami	615
Binární řetězce	616
Logický datový typ bit	616
Přístup k databázi pomocí jazyka Transact-SQL	616
Databáze BASEBALL	617
Tabulka BATTERS	617
Tabulka PITCHERS	618
Tabulka TEAMS	618
Deklarace lokálních proměnných	619

Deklarace globálních proměnných	619
Praktické použití proměnných	621
Příkaz PRINT	622
<b>Řízení toku programu</b>	<b>623</b>
Příkazy BEGIN a END	623
Příkazy IF...ELSE	623
Podmínka EXISTS	625
Testování výsledku dotazu	626
Cyklus WHILE	626
Příkaz BREAK	627
Příkaz CONTINUE	627
Průchod tabulkou pomocí cyklu WHILE	628
<b>Zástupné symboly v jazyku Transact-SQL</b>	<b>629</b>
<b>Převody kalendářních dat</b>	<b>630</b>
<b>Příkazy SET jakožto diagnostické nástroje</b>	<b>631</b>
<b>Shrnutí</b>	<b>631</b>
<b>Otázky a odpovědi</b>	<b>631</b>
<b>Úkoly pro vás</b>	<b>632</b>
Kvíz	632
Cvičení	632
<b>LEKCE 28</b>	
<b>Databázový systém MySQL na unixovém systému</b>	<b>635</b>
<b>Správa databázového systému MySQL</b>	<b>635</b>
Instalace databázového systému MySQL	636
Spuštění a zastavení databázového systému MySQL	637
Počáteční práva v databázového systému MySQL	637
<b>Terminálový monitor databázového systému MySQL</b>	<b>638</b>
Připojení k databázi	638
Volby příkazového řádku	639
Zadávaní příkazů monitoru databázového systému MySQL	641
Historie příkazového řádku	643
Dávkový režim	643
Příkaz SHOW	644
<b>Pomocné nástroje databázového systému MySQL</b>	<b>645</b>
<b>Shrnutí</b>	<b>645</b>
<b>Otázky a odpovědi</b>	<b>646</b>
<b>Úkoly pro vás</b>	<b>646</b>
Kvíz	646
Cvičení	646

## ČÁST VIII

## Přílohy

PŘÍLOHA A	
<b>Odpovědi</b>	<b>649</b>
PŘÍLOHA B	
<b>Ukázky kódu pro vytvoření tabulek</b>	<b>677</b>
PŘÍLOHA C	
<b>Ukázky kódu pro naplnění tabulek</b>	<b>689</b>
PŘÍLOHA D	
<b>Instalace databázového systému MySQL pro cvičení</b>	<b>703</b>
Pokyny pro instalaci v systému Windows	703
Pokyny pro instalaci v systému Linux	704
PŘÍLOHA E	
<b>Přehled nejčastěji používaných příkazů jazyka SQL</b>	<b>705</b>
PŘÍLOHA F	
<b>Přehled nejčastěji používaných funkcí jazyka SQL</b>	<b>711</b>
Řetězcové funkce	711
Číselné funkce	713
Agregační funkce	713
Funkce pro práci s datem a časem	714
<b>Rejstřík</b>	<b>715</b>

## O autorech

Již více než 10 let se autoři věnují studiu, aplikaci a dokumentaci standardu jazyka SQL a jeho praktického použití na kritické databázové systémy v této knize. **Ryan Stephens** a **Ron Plew** jsou provozovateli, mluvčími a spoluzakladateli rychle se rozvíjející firmy Perpetual Technologies, Inc. (PTI), která se orientuje na management a poradenství v oblasti informačních technologií. Společnost PTI se specializuje na databázové technologie, především pak na databázové systémy Oracle a SQL Server provozované na platformách UNIX, Linux a Microsoft. Oba autoři začínali jako analytici dat a správci databáze a nyní vedou tým skvělých odborníků, kteří se starají o databáze klientů po celém světě. Vytvořili kurzy databází pro univerzitu Purdue v Indianapolis a pět let je vyučovali a napsali více než desítku knih o databázovém systému Oracle, jazyku SQL, návrhu databází a o zajištění vysoké dostupnosti kritických systémů.

**Arie D. Jones** je hlavním konzultantem společnosti Microsoft pro firmu PTI. Vede tým společnosti PTI složený z expertů na plánování, návrh, vývoj, nasazení a správu databázových prostředí a aplikací s cílem dosáhnout pro každého z klientů co nejlepší kombinace nástrojů a služeb. Pravidelně přednáší na setkání odborníků a napsal několik knih a článků, v nichž se věnuje tématům souvisejícím s databázemi. Jeho nejnovější kniha vydaná nakladatelstvím Wrox Publishing nese název „SQL Functions Programmer's Reference“ (Funkce jazyka SQL – příručka programátora).



## Věnování

*Tato kniha je věnována mým rodičům, Thomasu a Karlyn Stephensovým, kteří mě vždy vedli k tomu, že pokud budu chtít, tak dosáhnu čehokoliv. Tato kniha je věnována také mému úžasnému synu Danielovi a mým nádherným dcerám Autumn a Alivii – nikdy se nespokojte s ničím menším než se svými sny.*

—Ryan

*Tato kniha je věnována mé rodině: mé ženě Lindě, mé matce Betty, mým dětem Leslie, Nancy, Angele a Wendy, mým vnukům Andymu, Ryanovi, Holly, Morgan, Schyler, Heather, Gavinovi, Regan, Caleigh a Cameron a mým zeťům Jasonovi a Dallasovi. Děkuji vám, že jste se mnou během tohoto rušného období měli trpělivost. Všechny vás mám rád.*

—Poppy

*Tuto knihu bych rád věnoval mé ženě Jackie za to, že mi během těch dlouhých hodin, které jsem věnoval práci na této knize, projevovala pochopení a podporu.*

—Arie

## Poděkování

Děkujeme všem lidem v našich životech, kteří byli během všech vydání této knihy nesmírně trpěliví – především našim ženám Tině a Lindě. Děkujeme Ariemu Jonesovi za jeho nedocenitelnou pomoc při práci na tomto vydání. Děkujeme také všem v redakci vydavatelství Sams za jejich tvrdou práci, aby toto vydání bylo ještě lepší než to předchozí. Bylo pro nás potěšením s každým z vás pracovat.

## Poznámka redakce českého vydání

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

Computer Press  
redakce počítačové literatury

Holandská 8  
639 00 Brno

nebo

*knihy@cpress.cz.*

Další informace a případné opravy českého vydání knihy najdete na internetové adrese <http://knihy.cpress.cz/K1733>. Prostřednictvím uvedené adresy můžete též naší redakci zaslat komentář nebo dotaz týkající se knihy. Na vaše reakce se srdečně těšíme.

# Úvod

V průběhu poslední dekády se prostor informačních technologií výrazným způsobem posunul ke světu zaměřenému na data. Společnosti začaly více než kdy předtím hledat způsoby pro využití své vlastní datové sítě k provádění rozumných obchodních rozhodnutí. To zahrnuje schopnost efektivně shromažďovat, uchovávat a vybírat údaje na potenciálně rozsáhlé množině dat v mnoha formátech. Proto nabyla role správců a vývojářů databáze v náležitě implementaci a správa těchto systémů přímo strategický význam.

Základním kamenem jakéhokoli databázového projektu je jazyk, který se bude používat pro interakci s databázovým systémem. Naštěstí jisté sdružení ustanovilo standardní dotazovací jazyk pro databázová prostředí známý jako standard ANSI SQL. Dodržováním tohoto známého standardu se všechny databázové dotazovací jazyky setkávají ve společných rysech, což umožňuje vývojářům, aby se tento standard naučili a poté pracovali v libovolném počtu databázových systémů jen s drobnými změnami.

V této knize se zaměříme především na to, aby čtenáři získali základní znalosti o jazyku SQL, díky čemuž budou mít pevný základ pro budoucí studium. V současném podnikovém prostředí je na osvojení nových věcí mnohdy velmi málo času, neboť většinu času zhltnou každodenní pracovní činnosti. Kniha se soustředí na lekce menšího rozsahu a na logické členění částí ve stylu odrazového můstku, což čtenářům umožní učit se jazyk SQL jejich vlastním tempem a v rámci jejich vlastních časových možností.

## Komu je kniha určena

Kniha je určena všem, kteří se chtějí rychle naučit základy jazyka SQL (Structured Query Language – strukturovací dotazovací jazyk). Prostřednictvím bezpočtu příkladů jsou představeny všechny hlavní složky jazyka SQL společně s možnostmi, které jsou k dispozici v nejrůznějších databázových implementacích. Takto získané znalosti byste pak měli být schopni využít v relačních databázích tradičního podnikového prostředí.

## Uspořádání knihy

Kniha je rozdělena na sedm částí, které logicky rozčleňují strukturu jazyka ANSI SQL na snadno osvojitelné celky:

- Část I, tvořená prvními sedmi lekcemi, se věnuje základním konceptům v pozadí jazyka SQL a zaměřuje se především na dotazy jazyka SQL.
- Část II je věnována tématu umění návrhu databáze, jako je správné vytváření databází a databázových objektů, což je často základem pro vývoj aplikace v prostředí relačního databázového systému.
- Část III se soustřeďuje na manipulaci s daty a na používání jazyka SQL pro aktualizaci (UPDATE), vkládání (INSERT) a mazání (DELETE) dat v databázi. Jedná se o základní příkazy, které budete používat při každodenní práci s databází.

- Část IV je věnována správě databáze, což zahrnuje témata, jako je bezpečnost, řízení a výkon, která vám umožňují udržovat integritu a výkon své databáze.
- Část V se zaměřuje na pokročilejší objekty jazyka SQL, kam patří spouštěče a uložené procedury. Díky těmto objektům můžete sáhnout po důmyslnějších technikách pro manipulaci s daty, jejichž realizace by ve standardní syntaxi jazyka SQL byla velice obtížná.
- Část VI se zabývá pokročilejším programováním v jazyku SQL. Pomocí pokročilejšího programování v jazyku SQL můžete provádět složitější dotazy a manipulaci s daty v databázi.
- Část VII vám představí jazyk SQL v nejrůznějších databázových implementacích. Rozšíření jazyka SQL (např. PL/SQL) vám umožňují využít jedinečných rysů konkrétního databázového prostředí (např. databázový systém Oracle).
- V knize se nachází také šest příloh, v nichž kromě správných řešení cvičení každé lekce najdete také ukázky kódu pro vytvoření a naplnění tabulek používaných v celé knize.

Po prostudování této knihy se budete skvěle orientovat v jazyku SQL a tyto znalosti budete schopni aplikovat v praxi.

#### POZNÁMKA

Pokud již základy a historii jazyka SQL znáte, pak první lekci jen tak přelette očima a začněte naostro až od lekce 2.

Po vysvětlení syntaxe jazyka SQL si ji procvičíme prostřednictvím příkladů pro databázový systém MySQL, jehož implementace se nejvíce přibližuje standardu ANSI SQL, a také pro databázový systém Oracle, na němž si ukážeme některá rozšíření jazyka ANSI SQL.

## Použité konvence

Knihy používá pro snazší čitelnost a přehlednost textu následující typografické zásady:

- Názvy nabídek jsou od položek odděleny zvláštním znakem >. Například Soubor > Otevřít znamená zvolit položku Otevřít v nabídce Soubor.
- Nové pojmy jsou *zvýrazněny*.
- V některých výpisech je jak vstup, tak i výstup (**Vstup/výstup ▼**). V těchto případech je veškerý kód, který píšete (vstup), zvýrazněn tučným písmem, zatímco výstup zůstává ve standardním písmu se stejnou roztečí.
- Nadpisy **Vstup ▼** a **Výstup ▼** označují povahu uvedeného kódu.
- Řada termínů souvisejících s kódem jazyka SQL je v textu vysázena také písmem se stejnou roztečí.
- Zástupné symboly v kódu jsou uváděny *skloněným písmem se stejnou roztečí*.
- Odstavce nadepsané jako **Analýza ▼** vysvětlují předcházející ukázkou kódu.
- Nadpis **Syntaxe ▼** uvádí syntaxi příkazu.
- Text knihy je dále doplněn speciálními prvky:

**POZNÁMKA**

Poznámky vysvětlují zajímavé nebo důležité body, které mohou pomoci při porozumění technikám a koncepcím v pozadí jazyka SQL.

**TIP**

Tipy jsou malé útržky informací, které vám pomohou v praktických situacích. Tipy často nabízejí zkratky, díky nimž lze danou činnost provést snadněji nebo rychleji.

**UPOZORNĚNÍ**

Upozornění poskytují informace o problémech s negativním dopadem na výkon nebo o nebezpečných chybách. Varováním proto věnujte zvýšenou pozornost.

## Praktická cvičení v databázovém systému MySQL

V této edici jsme pro praktická cvičení zvolili databázový systém MySQL. V předchozích edicích jsme nechali na čtenáři, aby si zajistil přístup k libovolné implementaci jazyka SQL. Rozhodli jsme se, že by bylo lepší nabídnout databázi SQL s otevřeným zdrojovým kódem, která by všem čtenářům umožnila začít na stejné úrovni se stejným softwarem. Zvolili jsme databázový systém MySQL, protože jde v současnosti o nejoblíbenější databázi s otevřeným zdrojovým kódem, kterou lze snadno stáhnout a používat.

Databázový systém MySQL má však i svá omezení. Existuje několik prvků standardního jazyka SQL, které vůbec nepodporuje. Proto jsme se snažili rozlišovat mezi cvičeními, která databázový systém MySQL podporují, a cvičeními, která jej nepodporují. Ve cvičeních, která MySQL nepodporují, se zaměříme především na edici Express databázového systému Oracle. Krása jazyka SQL spočívá v tom, že se jedná o standardní jazyk, i když každá implementace má své odlišnosti. Pokud si budete základy jazyka SQL procvičovat v databázovém systému MySQL, budete schopni osvojené znalosti snadno využít v libovolné implementaci jazyka SQL.

## Zdrojový kód

V přílohách najdete zdrojový kód pro vytvoření všech objektů používaných v této knize. To zahrnuje všechny používané tabulky a data. Kromě toho je zdrojový kód možné stáhnout z webové stránky knihy (<http://knihy.cpress.cz/K1733>). Záznamy si tak můžete jednoduše zkopírovat do svého rozhraní, takže nemusíte trávit většinu svého času psaním, a můžete se tak soustředit na probíranou látku.



# ČÁST I

## Úvod do SQL

- Lekce 1: Seznámení s jazykem SQL
- Lekce 2: Začínáme s dotazy
- Lekce 3: Výrazy, podmínky a operátory
- Lekce 4: Klauzule v dotazech jazyka SQL
- Lekce 5: Spojování tabulek
- Lekce 6: Vkládání poddotazů do dotazů
- Lekce 7: Formování dat pomocí vestavěných funkcí



# LEKCE 1

## Seznámení s jazykem SQL

Vítejte na první lekci kurzu jazyka SQL. Tuto lekci zahájíme stručnou historií jazyka SQL a databází a získáte základy, na nichž budete stavět ve zbývajících částech knihy. Konkrétně se naučíte následující:

- Seznámíte se s historií jazyka SQL a databází.
- Naučíte se 12 pravidel Dr. Codda pro relační model databáze.
- Dozvíte se, jak navrhovat strukturu databáze.
- Seznámíte se s populárními implementacemi jazyka SQL.
- Dozvíte se, proč je otevřená propojitelnost databází (open database connectivity – ODBC) důležitá.

### Stručná historie jazyka SQL

Historie jazyka SQL začala v laboratoři společnosti IBM v San Jose v Kalifornii. Zde byl na konci sedmdesátých let dvacátého století jazyk SQL vyvinut. Zkratka SQL znamená *Structured Query Language* (strukturovaný dotazovací jazyk) a samotný jazyk je často označován jako „sequel“. Původně byl vyvinut pro produkt společnosti IBM s názvem DB2 (což je relační databázový systém neboli RDBMS, který lze i nyní zakoupit pro nejrůznější platformy a prostředí). Ve skutečnosti byla existence relačního databázového systému možná právě díky jazyku SQL. Na rozdíl od do té doby vytvořených procedurálních jazyků nebo jazyků třetí generace (3GL), jako je COBOL nebo C, se jedná o jazyk neprocedurální.

#### POZNÁMKA

*Neprocedurální* nepopisuje, jak se má něco provést, ale spíše tím co se má provést. Kupříkladu jazyk SQL nepopisuje, jak se má s daty pracovat, ale na jakých datech se má pracovat.

Charakteristická odlišnost databázového systému od relačního databázového systému spočívá v tom, že relační databázový systém používá množinově orientovaný databázový jazyk. *Množinová orientace* označuje způsob, jakým jazyk SQL zpracovává data – jako *množiny* nebo *skupiny*.

Dvě standardizační organizace propagují jazyk SQL jako průmyslový standard: ANSI (American National Standards Organization – americká národní standardizační organizace) a ISO (International Standards Organization – mezinárodní standardizační organizace). ANSI SQL je standard pro jazyk SQL, který budeme používat v rámci celé knihy. Ačkoliv tyto standardy vytvářející orgány připravují standardy, které mají návrháři databázových systémů dodržovat, všechny databázové produkty se od standardu



ANSI do určité míry odchylují. Popravdě řečeno, i když se standard ANSI docela rozrostl, tak množství prvků, které musí daný relační databázový systém implementovat, aby tento standard splňoval, je docela malé. Většina systémů nabízí určitá proprietární rozšíření jazyka SQL, která z něj činí skutečný procedurální jazyk.

V této knize budeme probírat nejrůznější relační databázové systémy, přičemž v části 7 se podrobněji podíváme na rozličné varianty jazyka SQL pro určité implementace.

## Stručná historie databází

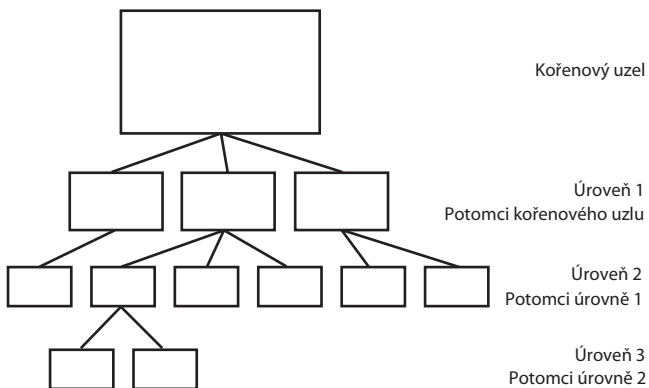
Trocha informací o vývoji databází a teorie databází vám pomůže pochopit fungování jazyka SQL. Databázové systémy se starají o uchovávání dat snad v každém myslitelném podnikovém prostředí. Databázové systémy uchovávají a distribuují data, na která se spoléháte – počínaje velkými sledovacími databázemi, jako jsou rezervační systémy letů, a konče kolekcí kartiček s fotbalisty. Před několika málo desetiletími bylo možné provozovat databáze pouze na velkých sálových počítačích. Tyto stroje byly tradičně velice nákladné na návrh, nákup i údržbu. Nicméně současná generace výkonných, laciných pracovních stanic umožňuje programátorům navrhovat software, který dokáže spravovat a distribuovat data rychle a levně.

Nejpopulárnějším modelem datového úložiště je relační databáze, která se zrodila z klíčové studie s názvem „A Relational Model of Data for Large Shared Data Banks“ (relační model dat pro rozsáhlé sdílené banky dat), kterou napsal Dr. E. F. Codd v roce 1970. Jazyk SQL se vyvinul tak, aby sloužil principům relačního modelu databáze. Dr. Codd definoval pro relační model 13 pravidel, kterým se kupodivu říká 12 pravidel Dr. Codda:

0. Relační databázový systém musí být schopen spravovat databáze jen s využitím svých relačních schopností.
1. **Informace:** Všechny informace v relační databázi (včetně názvů tabulek a sloupců) jsou reprezentovány explicitně jako hodnoty v tabulkovém formátu.
2. **Zaručený přístup:** U každé hodnoty v relační databázi je zaručeno, že bude přístupná prostřednictvím kombinace názvu tabulky, hodnoty primárního klíče a názvu sloupce.
3. **Systematická podpora nulitních hodnoty:** Databázový systém poskytuje systematickou podporu pro práci s *nulitnými hodnotami* (neznámá či nepoužitelná data), které jsou odlišné od výchozích hodnot a nezávislé na jakékoli doméně.
4. **Aktivní relační katalog dostupný online:** Popis databáze a jejího obsahu je reprezentován na logické úrovni v tabulkové formě, a proto lze nad ním spouštět dotazy pomocí databázového jazyka.
5. **Ucelený datový podjazyk:** Alespoň jeden podporovaný jazyk musí mít dobře definovanou syntaxi a musí být ucelený. Musí podporovat definici dat, manipulaci s daty, integritní pravidla, autorizaci a transakce.
6. **Aktualizování pohledů:** Veškeré pohledy, které lze teoreticky aktualizovat, mohou být aktualizovány prostřednictvím systému.
7. **Vkládání, aktualizace a mazání na úrovni množin:** Databázový systém podporuje nejen získávání dat na úrovni množin, ale také vkládání, aktualizace a mazání na úrovni množin.

8. **Fyzická datová nezávislost:** Změna fyzických přístupových metod nebo úložných struktur nemá z logického hlediska na aplikační a ad hoc programy žádný vliv.
9. **Logická datová nezávislost:** Změna tabulkových struktur nemá z logického hlediska na aplikační a ad hoc programy v maximální možné míře vliv.
10. **Integritní nezávislost:** Databázový jazyk musí být schopen definovat integritní pravidla. Tato pravidla musejí být uložena v katalogu dostupném online a nesmí existovat možnost pro jejich obejití.
11. **Nezávislost distribuce:** První distribuce nebo redistribuce dat nemá na aplikační programy a ad hoc požadavky z logického hlediska žádný vliv.
12. **Nenarušitelnost:** Nesmí existovat možnost umožňující obejití integritních pravidel definovaných v databázovém jazyku pomocí jazyků nižší úrovně.

Většina databází obsahuje vztah „rodič/potomek“, což znamená, že rodičovský uzel obsahuje ukazatele na své potomky (viz obrázek 1.1).



**Obrázek 1.1:** Coddův relační databázový systém

Tato metoda má několik výhod a řadu nevýhod. V její prospěch hovoří fakt, že fyzická struktura dat na disku se stane nepodstatnou. Programátor jednoduše uloží ukazatele na další umístění, takže k datům lze pak přistupovat tímto způsobem. Data lze kromě toho snadno přidávat i mazat. Nicméně různé skupiny informací nelze snadno spojit k vytvoření nové informace. Formát dat na disku nelze po vytvoření databáze libovolně měnit. Taková změna by totiž vyžadovala vytvoření nové databázové struktury.

Coddův přístup k relačnímu databázovému systému využívá matematickou koncepci relační algebry pro rozbití dat na množiny a s nimi související společné podmnožiny.

Vzhledem k tomu, že informace lze přirozeně seskupovat do různých množin, uspořádal Dr. Codd svůj databázový systém právě kolem této koncepce. V relačním modelu jsou data oddělena do množin, které připomínají tabulkovou strukturu. Tuto tabulkovou strukturu tvoří jednotlivé datové elementy nazývané *sloupce* nebo též *pole*. Jedna sada skupiny polí se označuje jako *záznam* nebo též *řádek*. Například k vytvoření relační databáze sestávající z dat o zaměstnancích můžete začít s tabulkou nazvanou `EMPLOYEE` (zaměstnanec), jež obsahuje následující

informace: EMP\_ID (identifikátor zaměstnance), LNAME (příjmení), FNAME (jméno) a DOB (datum narození). Tyto čtyři části dat tvoří pole v tabulce EMPLOYEE (viz tabulka 1.1).

**Tabulka 1.1:** Tabulka EMPLOYEE

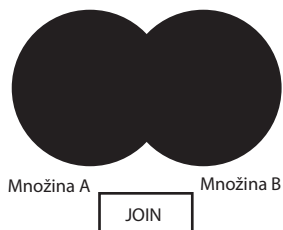
EMP_ID	LNAME	FNAME	DOB
1	NOVÁK	JAN	25-JUL-75
2	KOVAŘÍK	MICHAL	16-MAR-80
3	BUBÍKOVÁ	MARTINA	27-MAY-67
4	MAJEROVÁ	KRISTÝNA	01-MAR-69
5	VIDRA	MAREK	14-FEB-85
6	PLAŠIL	JIŘÍ	17-JUN-71
7	HODINKA	RADEK	22-OCT-52
8	STRÁNSKÝ	BARTOLOMĚJ	25-DEC-68

Osm výše uvedených řádků představuje záznamy v tabulce EMPLOYEE. Pokud bychom chtěli z této tabulky získat záznam například pro Michala Kovaříka, pak bychom vydali databázovému systému pokyn pro načtení záznamů, v nichž se pole LNAME rovná hodnotě Kovařík. Pokud bychom vydali pokyn pro načtení všech polí v záznamu, obdrželi bychom pole EMP\_ID, LNAME, FNAME a DOB. Pro předávání takovýchto pokynů databázi se používá jazyk SQL. Příkaz jazyka SQL může vypadat například takto:

```
SELECT *
FROM EMPLOYEE;
```

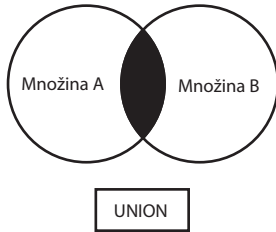
Uvědomte si, že přesná syntaxe není v tomto okamžiku podstatná. Podrobně se jí budeme věnovat v následující lekci.

Různé datové prvky lze seskupovat podle zřejmých vztahů (např. vztah příjmení zaměstnance k datu jeho narození), a proto relační model databáze nabízí návrhářům databází velkou míru flexibility pro popis vztahů mezi datovými elementy. Prostřednictvím matematických koncepcí JOIN (spojení) a UNION (sjednocení) mohou relační databáze rychle získávat části dat z různých množin (tabulek) a vracet je uživateli či programu jako jedinou „spojenou“ kolekci dat (viz obrázek 1.2). Funkce spojení umožňuje návrhářům ukládat množiny informací do samostatných tabulek a snížit tak výskyt opakovaně uložených dat.



**Obrázek 1.2:** Funkce spojení

Obrázek 1.3 ukazuje sjednocení. Funkce sjednocení vrátí pouze data společná oběma zdrojům.



**Obrázek 1.3:** Funkce sjednocení

Zde je jednoduchý příklad, který ukazuje, jak lze data logicky rozdělit do dvou tabulek. Tabulka 1.2 s názvem `DEPENDENTS` obsahuje pět sloupců: `EMP_ID` (identifikátor zaměstnance), `LNAME` (příjmení), `FNAME` (jméno), `SEX` (pohlaví) a `RELATIONSHIP` (vztah).

**Tabulka 1.2:** Tabulka `DEPENDENTS`

EMP_ID	LNAME	FNAME	SEX	RELATIONSHIP
1	NOVÁKOVÁ	MARIE	Ž	MANŽELKA
1	NOVÁK	TOMÁŠ	M	SYN
1	NOVÁKOVÁ	LUCIE	Ž	DCERA
2	KOVAŘÍKOVÁ	TAMARA	Ž	MANŽELKA
2	KOVAŘÍK	VILÉM	M	SYN
3	BUBÍK	MIREK	M	MANŽEL
3	BUBÍK	VILÉM	M	SYN
3	BUBÍKOVÁ	VERONIKA	Ž	DCERA
3	BUBÍK	ROBERT	M	SYN
3	BUBÍKOVÁ	JANA	Ž	DCERA
4	MAJER	PETR	M	MANŽEL
7	HODINKOVÁ	KRISTÝNA	Ž	MANŽELKA
8	STRÁNSKÁ	ELIŠKA	Ž	MANŽELKA

Bylo by nesprávné duplikovat pro každý záznam pole zaměstnance `EMP_ID`, `LNAME`, `FNAME` a `DOB`. Později by zbytečně duplikovaná data zabrala velkou část prostoru na pevném disku a prodloužila přístupovou dobu pro relační databázový systém. Pokud však pole `EMP_ID` a data náležící členům rodiny uložíme do samostatné tabulky s názvem `DEPENDENTS`, mohou uživatelé tabulky `DEPENDENTS` a `EMPLOYEE` spojit na poli `EMP_ID`. Při vydání pokynu relačnímu databázovému systému pro získání všech polí z tabulek `DEPENDENTS` a `EMPLOYEE`, v nichž se pole `EMP_ID` rovná hodnotě 3, obdržíme data uvedená v tabulce 1.3.

**Tabulka 1.3:** Hodnoty vrácené pro záznamy s polem EMP\_ID rovnajícím se hodnotě 3

LNAME	FNAME	FNAME	RELATIONSHIP
3	BUBÍK	MARTINA	MANŽEL
3	BUBÍK	MARTINA	SYN
3	BUBÍKOVÁ	MARTINA	DCERA
3	BUBÍK	MARTINA	SYN
3	BUBÍKOVÁ	MARTINA	DCERA

S podrobnějšími příklady spojování začneme pracovat v lekcii 5.

## Současná podoba databází

Výpočetní technologie způsobuje neustálou změnu ve způsobu fungování podniků na celém světě. Informace, které byly kdysi uloženy ve skladišti plném kartoték, jsou nyní přístupné pouhým stiskem tlačítka myši. Objednávky prováděné zákazníky v cizích zemích lze nyní okamžitě zpracovat na půdě výrobní haly.

I když se před 20 lety většina těchto informací přenášela do databází v podnikových sálových počítačích, fungovaly kanceláře i přesto v prostředí s dávkovým zpracováním. Pokud bylo nutné provést nějaký dotaz, musel někdo upozornit oddělení správy informačních systému (SIS) a poté se požadovaná data doručila v nejbližším možném termínu (často to však nebylo nijak brzy).

Kromě vývoje relačního modelu databáze přispěly k prudkému nárůstu toho, co se dnes označuje jako *databázové systémy typu klient-server*, další dvě technologie. První významnou technologií byl osobní počítač. Levné, snadno použitelné aplikace, jako je Lotus 1-2-3 nebo WordPerfect, umožňovaly zaměstnancům (a uživatelům domácích počítačů) rychleji a přesněji vytvářet dokumenty a spravovat data. Uživatelé si zvykli neustále modernizovat své systémy, a to jednak kvůli rychlosti probíhajících změn, jednak kvůli nepřetržitému pádu cen pokročilejších systémů.

Druhou významnou technologií byla *místní počítačová síť* (LAN – local area network) a její integrace do kanceláří na celém světě. Uživatelé sice byli zvyklí na terminálová připojení k podnikovému sálovému počítači, soubory pro zpracování textu však již bylo možné ukládat lokálně v rámci kanceláře a přistupovat k nim z libovolného počítače připojeného k síti. Když počítač Apple Macintosh představil přívětivé grafické uživatelské rozhraní (GUI – graphical user interface), staly se počítače nejen levnými a výkonnými, ale též snadno použitelnými. Ba co víc, šlo k nim přistupovat ze vzdálených sítí a uložení velkých objemů dat bylo možné přenést na oblastní datové servery.

Během této doby prudkých změn a pokroku se objevil nový typ systému, který se nazývá *vývoj typu klient-server*, protože zpracování je rozděleno mezi klientské počítače a databázový server. Tento nový druh aplikace představoval radikální změnu oproti programování aplikací založených na sálových počítačích. Mezi mnoho výhod tohoto typu architektury patří následující:

- nižší náklady na údržbu,
- nižší zátěž sítě (zpracování probíhá na databázovém serveru nebo klientském počítači),

- možná spolupráce více operačních systémů sdílejících společný síťový protokol,
- lepší integrita dat díky centralizovanému umístění dat.

Bernard H. Boar definuje v knize „Implementing Client/Server Computing“ výpočetní model typu klient-server takto (přeloženo z anglického originálu knihy):

Výpočetní model typu klient-server představuje model zpracování dat, v němž je jediná aplikace rozdělena na více zpracovatelů (typu front-end neboli klient, a back-end neboli server), kteří spolupracují (transparentně vzhledem ke koncovému uživateli) na dokončení zpracování jako jediná unifikovaná úloha. Produkt na bázi modelu klient-server svazuje tyto zpracovatele dohromady, čímž poskytuje dojem (iluzi) jediného systému. Společně používané prostředky hrají roli žádajících klientů, kteří přistupují k autorizovaným službám. Tato architektura je nekonečně rekurzivní, poněvadž servery se mohou stávat klienty a požadovat služby od jiných serverů na síti.

Tento typ vývoje aplikací vyžaduje zcela novou sadu programátorských dovedností. Programování uživatelského rozhraní je nyní zaměřeno na grafická uživatelská rozhraní, ať už jde o systém MS Windows, IBM OS/2, Apple Macintosh nebo UNIX X Window. Pomocí jazyka SQL a síťového připojení může aplikace pracovat s databází sídlící na vzdáleném serveru. Díky vzrůstajícímu výkonu hardwaru osobních počítačů lze kritické databázové informace uchovávat na relativně levném samostatném serveru. Ten lze navíc nahradit jen s nepatrnou nebo vůbec žádnou změnou na straně klientských aplikací.

## Jazyk pro více produktů

Základní principy představené v této knize můžete použít v mnoha prostředích, například v databázi Microsoft Access běžící v jednouživatelské aplikaci Windows nebo v databázi SQL Server běžící se stovkou připojených uživatelů. Jedna z největších výhod jazyka SQL spočívá v tom, že se jedná o jazyk skutečně použitelný na více platformách a ve více produktech. Vzhledem k tomu, že se jedná o jazyk, který programátoři označují jako jazyk vysoké úrovně nebo jazyk čtvrté generace (4GL), lze pomocí menšího počtu řádků kódu odvést větší kus práce.

## Prvotní implementace

Společnost Oracle Corporation vydala první komerční relační databázový systém, který používal jazyk SQL. Ačkoliv původní verze byly vyvinuty pro systémy VAX/VMS, byla Oracle jedním z prvních dodavatelů verze svého relačního databázového systému pro systém DOS. (Databáze Oracle je nyní dostupná pro více než 70 platform.) V polovině osmdesátých let vypustila společnost Sybase svůj relační databázový systém s názvem SQL Server. S klientskými knihovnami pro přístup k databázi, podporou pro uložené procedury a schopností komunikovat skrze nejrůznější sítě se databáze SQL Server stala poměrně úspěšným produktem, především pak v prostředích typu klient-server.

Jednou z nejsilnějších stránek obou zmíněných výkonných databázových systémů je jejich škálovatelnost napříč platformami. Kód v jazyku C (zkombinovaný s kódem jazyka SQL) napsaný pro databázový systém Oracle na osobním počítači je virtuálně identický se svým protějškem napsaným pro databázový systém Oracle běžící na systému VAX.

## Jazyk SQL a vývoj aplikací typu klient-server

Společným jmenovatelem vývoje aplikací typu klient-server je použití jazyka SQL a relačních databází. Kromě toho použití této databázové technologie v jednouuživatelské podnikové aplikaci dává této aplikaci potenciál pro budoucí růst.

## Přehled jazyka SQL

Jazyk SQL je de facto standardním jazykem používaným pro manipulaci a získávání dat z relačních databází. Pomocí jazyka SQL může programátor nebo správce databáze provádět následující:

- Upravovat strukturu databáze.
- Měnit nastavení zabezpečení systému.
- Přidávat uživatelská oprávnění k databázím či tabulkám.
- Dotázat se databáze na nějakou informaci.
- Aktualizovat obsah databáze.

### POZNÁMKA

Termín SQL může být poněkud matoucí. Je docela jasné, že písmenko „S“ znamená strukturovaný a „L“ znamená jazyk (language), ale písmenko „Q“ je trošku zavádějící. Znamená samozřejmě dotaz (query), který by vás při doslovné interpretaci omezil jen na kladení dotazů databázi. Jenže jazyk SQL nabízí mnohem více než kladení dotazů. Pomocí něho můžete také vytvářet tabulky, přidávat data, mazat data, spojovat data dohromady, spouštět akce na základě změn v databázi a ukládat dotazy do programu či databáze.

Pro poslední písmenko však neexistuje žádná vhodná náhrada. Je jasné, že zkratka SAMDJSTQL (Structured Add Modify Delete Join Store Trigger and Query Language – strukturovaný přidávací modifikační mazací spojovací ukládací spouštěcí a dotazovací jazyk) je poněkud těžkopádná. Ve jménu harmonie tedy zůstaneme u označení SQL. Nyní již ale víte, že funkce tohoto jazyka je mnohem větší než jeho jméno.

Nejčastěji používaným příkazem v jazyku SQL je příkaz `SELECT` (viz lekce 2), který získává data z databáze a vrací je uživateli. Tabulka `EMPLOYEE` představuje typický příklad pro použití příkazu `SELECT`. Kromě příkazu `SELECT` nabízí jazyk SQL příkazy pro vytváření nových databází, tabulek, polí a indexů a dále příkazy pro vkládání a mazání záznamů. Standard ANSI SQL kromě toho doporučuje základní skupinu funkcí pro manipulaci s daty. Jak se sami přesvědčíte, řada databázových systémů poskytuje rovněž nástroje k zajištění integrity dat a prosazování bezpečnosti (viz lekce 14), které programátorům umožňují zastavit provádění skupiny příkazů, pokud dojde k určité podmínce.

# Populární implementace jazyka SQL

V této části si představíme některé z populárnějších implementací jazyka SQL. Implementace se navzájem liší, přičemž každá má své silné i slabé stránky. Zatímco některé implementace jazyka SQL byly vytvořeny pouze pro osobní počítače a jejich cílem byla snadná obsluha ze strany uživatelů, jiné byly vyvinuty tak, aby vyhovovaly velmi rozsáhlým databázím. V této části se seznámíme s vybranými klíčovými funkcemi některých implementací.

## POZNÁMKA

Tato kniha neslouží jen jako referenční příručka jazyka SQL, ale obsahuje též řadu praktických příkladů z oblasti vývoje softwaru. Jazyk SQL je užitečný jen tehdy, když řeší problémy z reálného světa, k nimž dochází ve vašem kódu.

## MySQL

V této knize se setkáte s ukázkami kódu pro databázový systém MySQL při demonstraci syntaxe jazyka SQL na příkazovém řádku. MySQL (viz <http://www.mysql.com/>) se stahuje a instaluje relativně snadno a jakožto databázový systém se těší stále větší popularitě. Detailní postup pro získání a instalaci databázového systému MySQL najdete v příloze D.

## Oracle

My budeme používat databázový systém Oracle, který představuje obsáhlejší podnikový databázový svět, k demonstraci příkazů jazyka SQL na příkazovém řádku a technik pro správu databáze. (Tyto techniky jsou důležité, protože dny samostatných strojů se chýlí ke svému konci stejně jako dny, kdy stačilo ovládat jednu databázi nebo jeden operační systém.) Na příkazovém řádku SQL se zadávají jednoduché, samostatné příkazy jazyka SQL do nástroje SQL\*Plus pracujícího nad databázovým systémem Oracle. Získaná data se pak vypíší na obrazovku, kde si je uživatel může prohlédnout nebo provést příslušnou akci nad databází.

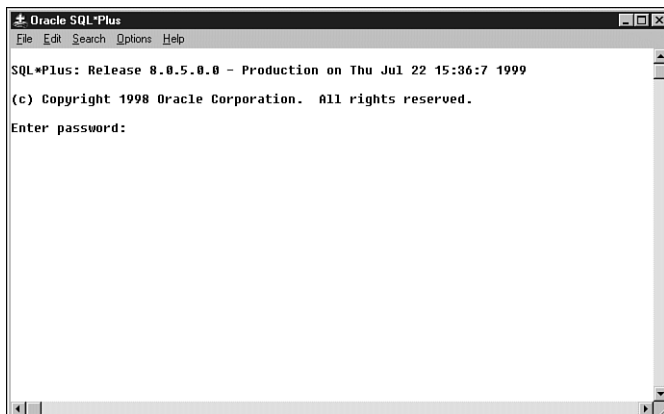
Většina příkladů je směřována k začínajícím programátorům nebo nováčkům v oblasti používání jazyka SQL. Začneme těmi nejjednoduššími příkazy jazyka SQL a budeme postupovat až k řízení transakcí a programování uložených procedur. Relační databázový systém Oracle se dodává s grafickými nástroji pro správu databází, uživatelů a objektů a také s pomůckou SQL\*Loader, která se používá pro import a export dat do a z databázového systému Oracle.

Relační databázový systém Oracle jsme zvolili hned z několika důvodů:

- Obsahuje téměř všechny nástroje nezbytné k demonstraci témat probíraných v této knize.
- Je virtuálně dostupný na každé v současnosti používané platformě a představuje jeden z nejpůlárnějších relačních databázových systémů na světě.
- Z webového serveru společnosti Oracle Corporation (<http://www.oracle.com>) lze stáhnout bezplatnou edici Express.

Obrázek 1.4 zachycuje nástroj SQL\*Plus z této sady nástrojů.





**Obrázek 1.4:** Nástroj SQL\*Plus pro databázový systém Oracle

#### TIP

Mějte na paměti, že téměř veškerý kód jazyka SQL v této knize lze přenést i do jiných databázových systémů. V případech, kdy se syntaxe mezi produkty různých výrobců značně liší, budou uvedeny příklady pro ilustraci těchto odlišností.

## Microsoft SQL Server a Sybase

Společnost Sybase má na svědomí původní implementaci databáze SQL Server, která byla původně navržena pro operační systém OS/2. Později uzavřela dohodu o vývoji kódu se společností Microsoft, která přenesla aplikaci pro OS/2 na svou platformu Windows. V roce 1993 se tyto dvě společnosti rozhodly jít každá vlastní cestou. Společnost Sybase nyní přejmenovala svůj produkt na Sybase Adaptive Server Enterprise a společnost Microsoft vydala systém SQL Server ve verzi 2008.

## IBM DB2

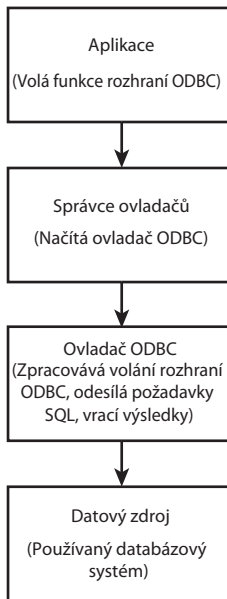
Společnost IBM původně vyvíjela databázi SQL na sklonku sedmdesátých let pro svou platformu DB2. Jakožto přední světový dodavatel hardwaru se společnost IBM rozhodla svou platformu DB2 transformovat do produktové řady známé jako Universal Database.

## ODBC

ODBC (Open Database Connectivity – otevřená databázová konektivita) je knihovna navržena tak, aby poskytovala společné aplikační rozhraní k základním databázovým systémům. S databází komunikuje prostřednictvím knihovničního ovladače, tedy stejně, jako když systém Windows komunikuje s tiskárnou skrze ovladač tiskárny. V závislosti na používané databázi může být pro připojení k vzdálené databázi vyžadován síťový ovladač. Architekturu knihovny ODBC znázorňuje obrázek 1.5.

Jedinečná vlastnost knihovny ODBC (ve srovnání s knihovnami společnosti Oracle či Sybase) spočívá v tom, že žádná z jejích funkcí není specifická pro některého z výrobců databáze. Můžete tak například použít stejný kód jen s malou nebo vůbec žádnou úpravou k provedení dotazů nad tabulkou Microsoft Access nebo nad databází Informix. Opět je nutné poznamenat, že většina výrobců přidává ke standardu SQL určitá proprietární rozšíření, jako je například Transact-SQL společností Microsoft a Sybase nebo PL/SQL společnosti Oracle.

Před zahájením práce s novým datovým zdrojem byste měli vždy nahlédnout do dokumentace. Knihovna ODBC se vyvinula do standardu přijatého do řady produktů, mezi něž patří Visual Basic, Visual C++, FoxPro, Borland Delphi a PowerBuilder. A jako vždy, vývojáři aplikací musí zvážit výhodu použití rozvíjejícího se standardu ODBC, který umožňuje navrhovat kód bez ohledu na určitou databázi, oproti rychlosti získané při použití knihovny specificky navržené pro určitou databázi. Jinými slovy, při použití knihovny ODBC je kód sice přenositelnější, ale zato ve srovnání s kódem postaveným nad knihovnami společnosti Oracle či Sybase pomalejší.



**Obrázek 1.5:** Struktura knihovny ODBC

## Pozice kódu jazyka SQL ve vytvářené aplikaci

Jazyk SQL byl v roce 1986 ustaven jako standard ANSI. Standard ANSI z roku 1989 (často označovaný jako SQL-89) definuje tři typy propojení kódu jazyka SQL s aplikačním programem:

- **Jazyk převedený na modul:** Používá procedury uvnitř programů. Tyto procedury lze volat aplikačním programem, přičemž prostřednictvím parametrů mohou vracet hodnoty.

- **Vložený kód jazyka SQL:** Používá příkazy jazyka SQL vložené do kódu vlastního programu. Tato metoda často vyžaduje použití předkompilace pro zpracování příkazů jazyka SQL. Standard definuje příkazy pro Pascal, FORTRAN, COBOL a PL/1.
- **Přímé vyvolání:** Ponecháno na implementaci.

Před tím, než se vyvinula koncepce dynamického jazyka SQL, byl nejpobulárnějším způsobem použití jazyka SQL v rámci programovacího prostředí vložený kód jazyka SQL, který se stále používá. Vložený kód jazyka SQL využívá *statický* jazyk SQL, což znamená, že příkaz jazyka SQL se zkompileje do aplikace a nelze jej za běhu měnit.

Princip je v podstatě stejný jako při porovnávání kompilátoru s interpretací kódu. Výkon pro tento typ SQL je sice dobrý, nicméně není flexibilní, a nemůže tedy vždy naplnit potřeby současných měnících se prostředí. K dynamickému SQL se dostaneme vzápětí.

Standard ANSI z roku 1992 (SQL-92) jazyk dále rozšířil a stal se celosvětovým standardem. Definuje tři úrovně shody: základní (entry), střední (intermediate) a úplnou (full). Mezi nové prvky zavedené standardem SQL-92 patří následující:

- připojení k databázím,
- posuvné kurzory,
- dynamické SQL,
- vnější spojení.

Největší revize standardu ANSI (SQL3) má pět vzájemně propojených dokumentů, přičemž v blízké budoucnosti mohou být přidány další dokumenty. Pět částí vypadá takto:

- **Část 1 (SQL/Framework – rámec):** Specifikuje obecné požadavky na shodu a definuje základní principy jazyka SQL.
- **Část 2 (SQL/Foundation – základ):** Definuje syntaxi a operace jazyka SQL.
- **Část 3 (SQL/Call-Level Interface – rozhraní na úrovni volání):** Definuje rozhraní k SQL pro aplikační programování.
- **Část 4 (SQL/Persistent Stored Modules – trvale uložené moduly):** Definuje řídicí struktury, které pak definují rutiny SQL. Část 4 dále definuje moduly, jež obsahují rutiny SQL.
- **Část 5 (SQL/Host Language Bindings – vazby na hostitelský jazyk):** Definuje, jak vkládat příkazy jazyka SQL do aplikačních programů, které jsou napsané ve standardním programovacím jazyku.

Standard SQL má dvě úrovně minimální shody, k nimž se mohou databázové systémy hlásit: základní podporu jazyka SQL (Core SQL Support) a rozšířenou podporu jazyka SQL (Enhanced SQL Support).

V této knize se budeme věnovat nejen všem těmto rozšířením, ale také několika proprietárním rozšířením používaným výrobcí relačních databázových systémů. Dynamické SQL dovoluje přípravu příkazu jazyka SQL za běhu programu. I když výkon tohoto typu SQL není tak dobrý jako v případě vkládaného kódu jazyka SQL, nabízí vývojářům (a uživatelům) aplikací obrovskou míru flexibility. Rozhraní na úrovni volání, jako je knihovna ODBC nebo DB-Library společnosti Sybase, je příkladem dynamického SQL.

Rozhraní na úrovni volání by pro programátora aplikací neměl být neznámý pojem. Kupříkladu při používání knihovny ODBC jednoduše naplníte proměnnou příkazem jazyka SQL a zavoláte funkci pro jeho odeslání do databáze. Chyby nebo výsledky lze v programu získat prostřednictvím volání dalších, k tomuto účelu navržených funkcí. Výsledky se získávají skrze proces známý jako *svazování* proměnných.

## Shrnutí

V této lekci jsme se věnovali stručné historii a struktuře v pozadí jazyka SQL. Vzhledem k tomu, že jazyk SQL a relační databáze spolu velmi úzce souvisejí, podívali jsme se také (třebaže jen stručně) na historii a funkci relačních databází. Jak již nyní víte, databáze se v té či oné formě používají ve většině organizací pro správu důležitých podnikových dat. Bez databází by organizace byly nuceny i nadále uchovávat data v tištěné formě. Bez standardního databázového jazyka, jako je jazyk SQL, by uživatelé postrádali robustní a snadno použitelné rozhraní, které umožňuje komunikaci s databázovým prostředím. Zapamatujte si také pravidla Dr. Codda pro relační model databáze, neboť tvoří základ pro všechny relační databázové systémy. Lekce 2 je věnována té nejdůležitější součásti jazyka SQL: dotazům.

## Otázky a odpovědi

**Otázka:** Proč bych se měl zajímat o jazyk SQL?

**Odpověď:** Pokud jste až dosud nepracovali na rozsáhlém databázovém systému, pak máte pravděpodobně jen povrchní znalost jazyka SQL. S nástupem vývojových nástrojů typu klient-server (např. Visual Basic, Visual C++, ODBC, Delphi společnosti Borland nebo PowerBuilder společnosti Sybase) a přesunem několika rozsáhlých databází (Oracle a Sybase) na platformu PC je pro většinu v současnosti vyvíjených aplikací praktická znalost jazyka SQL naprosto nezbytná.

**Otázka:** Proč je potřeba k práci s jazykem SQL znát také něco z teorie relačních databází?

**Odpověď:** Jazyk SQL byl vyvinut pro relační databáze. Bez minimální znalosti teorie relačních databází jej nebudete schopni kromě těch nejtriviálnějších případů efektivně používat.

**Otázka:** Díky všem těm novým nástrojům s grafickým uživatelským rozhraním mi stačí pro vytvoření kódu jazyka SQL jen klepnout na nějaké tlačítko. Proč bych měl věnovat čas tomu, abych se naučil psát kód jazyka SQL ručně?

**Odpověď:** Nástroje s grafickým uživatelským rozhraním mají své místo na slunci stejně jako ruční zápis kódu jazyka SQL. Ručně zapsaný kód jazyka SQL je obecně efektivnější než kód generovaný grafickým uživatelským rozhraním. Ten navíc není tak snadno čitelný jako ručně psané příkazy jazyka SQL, přičemž složitější dotazy mohou být těžkopádnější. Koneckonců povědomí o tom, co se při používání nástrojů GUI děje v pozadí, vám pomůže z nich vytěžit maximum.

**Otázka:** Je-li tedy jazyk SQL standardizovaný, měl bych být schopen psát kód jazyka SQL pro libovolnou databázi?

Odpověď: Ne, budete schopni psát kód jazyka SQL pouze pro relační databázové systémy, jež podporují jazyk SQL, jako je například Microsoft Access, Oracle, Microsoft SQL Server, Sybase a Informix. I když implementace každého z dodavatelů se od ostatních malinko liší, měli byste být schopni použít jazyk SQL jen s velmi drobnými úpravami.

## Úkoly pro vás

Tato část nabízí kvízové otázky, které vám pomohou s upevněním získaných znalostí, a dále cvičení, jež vám poskytnou praktické zkušenosti s používáním osvojené látky. Pokuste se před nahlédnutím na odpovědi v příloze A odpovědět na otázky v kvízu a ve cvičení.

### Kvíz

1. Co činí jazyk SQL neprocedurálním jazykem?
2. Podle čeho lze říci, zda je daná databáze skutečně relační?
3. K čemu lze jazyk SQL použít?
4. Uveďte název modelu, který odděluje data do odlišných, jedinečných množin.

### Cvičení

Nalistujte přílohu D. Stáhněte si databázový systém MySQL a nainstalujte jej na svůj počítač, abyste byli připraveni na cvičení v následujících lekcích. Databázový systém MySQL budeme používat pro co největší počet cvičení, protože se jedná o databázi z valné části vyhovující standardu ANSI, která je bezplatná a snadno se stahuje a používá. V některém cvičení s databázovým systémem MySQL můžeme používat syntaxi, která je malinko odlišná od té, kterou používáme v příkladech s databázovým systémem Oracle. V takových případech se budeme snažit upozornit na odlišnosti nebo neshody se standardem ANSI, s nimiž se můžete v databázi MySQL setkat.

## LEKCE 2

# Začínáme s dotazy

Vítejte v lekcí 2! Na jejím konci budete ovládat následující:

- Co je dotaz a jak se používá.
- Syntaxe a způsob použití příkazu `SELECT` a klauzule `FROM`.
- Výběr a výpis všech řádků a sloupců z tabulky.
- Výběr a výpis některých sloupců z tabulky.
- Výběr a výpis sloupců z různých tabulek.

## Pozadí jazyka SQL

K plnému využití síly relační databáze, kterou jsme si stručně popsali v lekcí 1, je nutné umět s ní komunikovat. Nejlepší by bylo, kdybyste se jen otočili ke svému počítači a řekli jasným a zřetelným hlasem: „Ukaž mi všechny hnědooké leváky, kteří pracovali pro tuto společnost nejméně 10 let.“ Někteří z vás už takto možná s počítačem komunikují. Všichni ostatní ale potřebují poněkud konvenčnější způsob získávání informací z databáze. A právě k tomu lze využít dotazovací schopnosti jazyka SQL vyjádřené prostředním písmenem „Q“ (query – dotaz).

Jak jsme si řekli již v lekcí 1, termín dotaz je v tomto kontextu poněkud zavádějící. Dotaz jazyka SQL neznamená nutně otázku položenou databázi. Může jít o příkaz k provedení některé z následujících činností:

- sestavení či smazání tabulky,
- vložení, modifikace či smazání řádků nebo polí,
- vyhledání specifických informací v několika tabulkách a vrácení výsledku v určitém pořadí,
- úprava informací o zabezpečení.

Dotazem může být též jednoduchá otázka položená databázi. K využití tohoto mocného nástroje se musíte naučit, jak psát dotazy jazyka SQL.

## Osvojení základní syntaxe dotazů

Jak za okamžik sami uvidíte, syntaxe jazyka SQL je docela flexibilní, ačkoliv jako v každém programovacím jazyku existují i zde určitá pravidla, která je nutné dodržovat. Jednoduchý dotaz ilustruje základní syntaxi příkazu `SELECT` jazyka SQL. Věnujte velkou pozornost velikosti písmen, mezerám a logickému oddělení částí každého dotazu jazyka SQL klíčovými slovy.

## Syntaxe ▼

```
SELECT NAME, STARTTERM, ENDTERM
FROM PRESIDENTS
WHERE NAME = 'MASARYK';
```

## Analýza ▼

V tomto příkladu je vše zapsáno velkými písmeny, což ale není vůbec nutné. Výše uvedený dotaz by pracoval stejně dobře i při následujícím zápisu:

## Syntaxe ▼

```
select name, startterm, endterm
from presidents
where name = 'MASARYK';
```

## Analýza ▼

Všimněte si, že slovo `MASARYK` je v obou příkladech zapsáno velkými písmeny. I když se u samotných příkazů jazyka SQL nerozlišuje velikost písmen, u odkazů na data v databázi na velikosti písmen záleží. Řada společností například ukládá svá data s velkými písmeny. V předchozím příkladu také předpokládáme, že sloupec `name` uchovává svůj obsah s velkými písmeny. Proto by dotaz hledající ve sloupci `name` text „Masaryk“ nenašel žádná data. V otázce požadavků na velikost písmen konzultujte svou implementaci anebo zásady společnosti.

### POZNÁMKA

U příkazů jazyka SQL se velikost písmen nerozlišuje.

Podívejte se znovu na ukázkový dotaz. Je snad něco magického na rozestupech jeho částí? A opět si musíme říci, že nikoliv. Následující kód by fungoval stejně dobře:

## Syntaxe ▼

```
select name, startterm, endterm from presidents where name = 'MASARYK';
```

Pokud však budete věnovat trošku pozornosti též rozmístění a velikost písmen, budou vaše příkazy mnohem čitelnější. Stanou-li se navíc součástí vašeho projektu, budou se též snadněji udržovat.

Dalším důležitým prvkem ukázkového dotazu je středník na konci výrazu. Toto interpunkční znaménko říká programu SQL na příkazové řádce, že váš dotaz je hotový.

Není-li nějaká magie ve velikosti písmen ani ve formátu, pak se můžete zeptat, které prvky jsou vlastně důležité. Odpověď je: klíčová slova nebo slova v jazyku SQL, která jsou rezervovaná jako součást jeho syntaxe. (V závislosti na daném příkazu jazyka SQL může být klíčové slovo buď povinným, nebo volitelným prvkem příkazu.) Výše uvedený příklad obsahuje následující klíčová slova:

- SELECT
- FROM
- WHERE

Nahlédněte do obsahu, kde najdete některá klíčová slova jazyka SQL společně s informací, ve které lekci se je naučíte používat. V této lekci si osvojíte klíčová slova `SELECT` a `FROM`.

## Stavební bloky pro získávání dat: `SELECT` a `FROM`

S přibývajícím zkušenostmi s jazykem SQL zjistíte, že slova `SELECT` a `FROM` píšete častěji než kterákoli jiná slova ze slovníku jazyka SQL. Nejsou sice tak kouzelná jako `CREATE` nebo nemilosrdná jako `DROP`, jsou zato naprosto nepostradatelná v jakékoli konverzaci s počítačem, v níž se snažíte získat nějaká data.

Nejdříve se zaměříme na klíčové slovo `SELECT`, protože většina vašich příkazů bude začínat právě slovem `SELECT`:

### Syntaxe ▼

```
SELECT názvy_sloupců
```

#### POZNÁMKA

V této knize budou uváděny příklady kódu a výsledky po jeho provedení. Příklady začínající

SQL>

byly vytvořeny pomocí databázového systému Oracle. Příklady začínající

mysql>

byly vytvořeny pomocí databázového systému MySQL. Těmto značkám se říká výzva (PROMPT). Příklady kódu bez výzvy jsou ukázkou syntaxe.

Základní příkaz `SELECT` již nemohl být jednodušší. Nicméně `SELECT` nepracuje o samotě. Pokud do svého systému napíšete slovo `SELECT`, pak obdržíte následující odpověď:

### Vstup/výstup ▼

```
SQL> SELECT;
```

```
SELECT
```

```
*
```

```
ERROR at line 1:
```

```
ORA-00936: missing expression
```

Pokud pracujete v databázovém systému MySQL, pak chyba, kterou obdržíte, může vypadat takto:

### Vstup/výstup ▼

```
mysql> select;
```

```
ERROR 1064: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server for the right syntax to use near '' at line 1
```



## Analýza ▼

---

Hvězdička pod chybučícím řádkem označuje místo, kde podle databázového systému Oracle došlo k chybě. Chybová zpráva říká, že zde něco chybí. To chybějící něco je klauzule FROM:

## Syntaxe ▼

---

```
FROM tabulka
```

Příkaz SELECT a klauzule FROM spolu vytvářejí platný příkaz jazyka SQL. Pro upřesnění si můžeme říci, že tento příkaz je na nejvyšší úrovni syntaxe jazyka SQL. Platný příkaz jazyka SQL se formuje na základní úrovni pomocí klíčových slov, jež tvoří klauzule. Takže následující příklad

## Syntaxe ▼

---

```
SELECT NAME FROM PRESIDENTS;
```

můžeme rozdělit na následující logické části:

## Syntaxe ▼

---

```
Příkaz: SELECT NAME FROM PRESIDENTS;
Klauzule: SELECT NAME (klauzule SELECT)
          FROM PRESIDENTS (klauzule FROM)
Klíčová slova: SELECT
               FROM
```

Základní syntaxi jednoduchého příkazu jazyka SQL již tedy znáte, a proto se podíváme na jeho praktické příklady.

# Dotazy v praxi

Ještě před tím, než budeme pokračovat dále, se podíváme na ukázkovou databázi, která tvoří základ pro následující příklady. Diagram databázových tabulek a ukázkový kód pro jejich vytvoření najdete v příloze B. Tato databáze ilustruje základní funkce klíčových slov SELECT a FROM. Ve skutečném světě byste k sestavení této databáze použili techniky popsané v lekcí 11, pro účely popisu, jak používat SELECT a FROM, však předpokládejme, že již existuje. V tomto příkladu používáme tabulku CHECKS pro získávání informací o šecích napsaných danou osobou.

Tabulka CHECKS vypadá v databázovém systému Oracle takto:

## Výstup ▼

---

CHECK#	PAYEE	AMOUNT	REMARKS
1	Nákupní centrum	1500	Příště vzít syny
2	ČD	245	Vlak do Prahy
3	Nákupní centrum	2000	Mobilní telefon
4	Místní benzínka	980	Benzín
5	Diskont	1500	Nákup

6 Hospoda	2500 Divoká noc
7 Benzinka ve městě	250 Benzín

Tabulka CHECKS v databázovém systému MySQL vypadá takto:

### Výstup ▼

check	payee	amount	remarks
1	Nákupní centrum	1500.00	Příště vzít syny
2	ČD	245.00	Vlak do Prahy
3	Nákupní centrum	2000.00	Mobilní telefon
4	Místní benzinka	980.00	Benzín
5	Diskont	1500.00	Nákup
7	Benzinka ve městě	250.00	Benzín
6	Hospoda	2500.00	Divoká noc

### POZNÁMKA

Mezi výpisem obou implementací je patrný určitý rozdíl. Databázový systém MySQL prezentuje výstup v jakémsi rámu, kdežto databázový systém Oracle prezentuje výstup v jednodušší formě.

## Píšeme první dotaz

Následující příkaz jazyka SQL vybere všechny sloupce z tabulky CHECKS. Hvězdička (\*) znamená „vše“.

### Vstup ▼

```
SQL> select * from checks;
mysql> select * from checks;
```

Podstatné je, že příkaz `SELECT` je téměř vždy doprovázen klauzulí `FROM`. V průběhu této lekce začnete rozeznávat používané vzory, což vám v budoucnu usnadní psaní vlastních dotazů pomocí správné syntaxe.

Výstup uvedeného příkazu jazyka SQL je uveden níže, nejprve v databázovém systému Oracle a poté v MySQL:

### Výstup ▼

CHECK#	PAYEE	AMOUNT	REMARKS
1	Nákupní centrum	1500	Příště vzít syny
2	ČD	245	Vlak do Prahy
3	Nákupní centrum	2000	Mobilní telefon
4	Místní benzinka	980	Benzín
5	Diskont	1500	Nákup
6	Hospoda	2500	Divoká noc
7	Benzinka ve městě	250	Benzín

7 rows selected.

check	payee	amount	remarks
1	Nákupní centrum	1500.00	Příště vzít syny
2	ČD	245.00	Vlak do Prahy
3	Nákupní centrum	2000.00	Mobilní telefon
4	Místní benzinka	980.00	Benzín
5	Diskont	1500.00	Nákup
7	Benzinka ve městě	250.00	Benzín
6	Hospoda	2500.00	Divoká noc

7 rows in set (0.20 sec)

## Analýza ▼

Výstup vypadá úplně stejně jako kód v obou příkladech. Všimněte si, že v prvním příkladu jsou sloupce 1 a 3 zarovnány doprava a sloupce 2 a 4 doleva. Tento formát se řídí konvencí pro zarovnání, v níž jsou číselné datové typy zarovnávané doprava a znakové datové typy doleva. V příkladu s databázovým systémem MySQL je sloupec 1 zarovnán doprava, kdežto sloupec 3 je zarovnán na střed, i když se jedná o stejný datový typ (datové typy budeme probírat v lekci 9). Vždy prostudujte dokumentaci k databázové platformě, kterou používáte, abyste měli jistotu, že rozumíte konvencím, kterými se řídí a které se mohou lišit od toho, co je uvedeno v této knize.

Hvězdička (\*) v příkazu `SELECT *` říká databázi, aby vrátila všechny sloupce spojené se zadanou tabulkou popsanou v klauzuli `FROM`. Databáze určí pořadí, ve kterém se sloupce vrátí. Databáze také určuje pořadí vrácených řádků, pokud jí ovšem uživatel explicitně „neřekne“, jak je má vrátit. Nelze předpokládat, že data budou vrácena v tom či onom pořadí. Pokud například zadáte sadu zaměstnanců do tabulky `Employee` v pořadí podle jejich data narození a poté nad stejnou tabulkou napíšete příkaz `SELECT *`, tak svá data s největší pravděpodobností obdržíte ve zcela jiném pořadí. Později se naučíte, jak používat klauzuli `ORDER BY` ke stanovení správného uspořádání vrácené datové sady.

## Ukončení příkazu jazyka SQL

V některých implementacích jazyka SQL říká středník na konci příkazu interpretu, že jste svůj dotaz již dopsali. Například nástroj SQL\*Plus pro databázový systém Oracle zadávaný dotaz nespustí, dokud v něm neobjeví středník (nebo lomítko). Databázový systém MySQL provede zadávaný příkaz jen tehdy, pokud interpret narazí na středník. Na druhou stranu některé implementace jazyka SQL nepoužívají středník pro ukončování příkazů. Například databáze SQL Server společnosti Microsoft provede příkaz bez ohledu na to, zda obsahuje středník či nikoliv. Následující sada dotazů by se tedy v databázi SQL Server provedla úplně stejně:

## Vstup ▼

```
> select * from checks;
> select * from checks
```

## Vybírání jednotlivých sloupců

Předpokládejme, že nechcete sledovat všechny sloupce v databázi. Příkaz `SELECT *` jste použili pro zjištění, které informace jsou dostupné, a nyní se chcete soustředit jen na čísla šeků a přípsanou částku. Napíšete tedy příkaz

### Vstup ▼

---

```
SQL> SELECT CHECK#, amount from checks;
```

který vrátí

### Výstup ▼

---

```
CHECK# AMOUNT
-----
1      1500
2       245
3      2000
4       980
5      1500
6      2500
7       250
```

7 rows selected.

Nyní máme jen sloupce, které chceme vidět. Všimněte si použití velkých a malých písmen v dotazu. Na výsledek to nemá vůbec žádný vliv.

Co ale v případě, kdy potřebujeme uspořádat sloupce odlišným způsobem?

## Změna pořadí sloupců

V některých z předchozích příkladů používáme `*` pro výběr všech sloupců v tabulce, přičemž jejich pořadí zleva doprava ve výstupu určuje databáze. Pro stanovení pořadí sloupců lze napsat následující příkaz:

### Vstup ▼

---

```
SQL> SELECT payee, remarks, amount, check# FROM checks;
mysql> select payee, `check`, remarks, amount from checks;
```

### Analýza ▼

---

Všimněte si, že v klauzuli `SELECT` je uveden název každého sloupce. Pořadí, ve kterém jsou tyto sloupce uvedeny, odpovídá pořadí, ve kterém se objeví ve výstupu. Všimněte si dále, že názvy sloupců jsou odděleny čárkou a že za posledním názvem sloupce a následující klauzulí (v tomto případě se jedná o klauzuli `FROM`) je mezera. Výstup výše uvedeného příkazu vypadá takto:

Databázový systém Oracle:

### Výstup ▼

PAYEE	REMARKS	AMOUNT	CHECK#
Nákupní centrum	Příště vzít syny	1500	1
ČD	Vlak do Prahy	245	2
Nákupní centrum	Mobilní telefon	2000	3
Místní benzinka	Benzín	980	4
Diskont	Nákup	1500	5
Hospoda	Divoká noc	2500	6
Benzinka ve městě	Benzín	250	7

7 rows selected.

Databázový systém MySQL:

### Výstup ▼

payee	check	remarks	amount
Nákupní centrum	1	Příště vzít syny	1500.00
ČD	2	Vlak do Prahy	245.00
Nákupní centrum	3	Mobilní telefon	2000.00
Místní benzinka	4	Benzín	980.00
Diskont	5	Nákup	1500.00
Benzinka ve městě	7	Benzín	250.00
Hospoda	6	Divoká noc	2500.00

Níže je uveden další způsob, jak zapsat stejný příkaz:

Databázový systém Oracle:

### Vstup ▼

```
SELECT payee, remarks, amount, check#
FROM checks;
```

Databázový systém MySQL:

### Vstup ▼

```
mysql> select payee, `check`, remarks, amount,
-> from checks;
```

### Analýza ▼

Všimněte si, že klauzuli FROM jsme přenesli na druhý řádek. Tato konvence je sice záležitostí osobního vkusu při psaní kódu jazyka SQL, jejím cílem je však čitelnější kód. Výstup nyní vypadá takto:

Databázový systém Oracle:

### Výstup ▼

PAYEE	REMARKS	AMOUNT	CHECK#
Nákupní centrum	Příště vzít syny	1500	1
ČD	Vlak do Prahy	245	2
Nákupní centrum	Mobilní telefon	2000	3
Místní benzinka	Benzín	980	4
Diskont	Nákup	1500	5
Hospoda	Divoká noc	2500	6
Benzinka ve městě	Benzín	250	7

7 rows selected.

Databázový systém MySQL:

### Výstup ▼

payee	check	remarks	amount
Nákupní centrum	1	Příště vzít syny	1500.00
ČD	2	Vlak do Prahy	245.00
Nákupní centrum	3	Mobilní telefon	2000.00
Místní benzinka	4	Benzín	980.00
Diskont	5	Nákup	1500.00
Benzinka ve městě	7	Benzín	250.00
Hospoda	6	Divoká noc	2500.00

### Analýza ▼

Výstup je identický, protože jedině, co jsme změnili, je formát příkazu. Nyní již dovedete stanovit pořadí sloupců, budete tedy schopni také určovat, které sloupce chcete vidět.

### Vybírání jiných tabulek

Předpokládejme, že máme tabulku s názvem DEPOSITS s následující strukturou:

### Výstup ▼

DEPOSIT#	WHOPAID	AMOUNT	REMARKS
1	Bohatý strýček	20000	Vyzvednout seznam dáreků
2	Zaměstnavatel	25000	Výplata za červenec
3	Banka	5000	Půjčka

Nyní stačí jen změnit klauzuli FROM na požadovanou tabulku a zapsat následující příkaz:

### Vstup ▼

```
SQL> select * FROM deposits
```

Výsledek vypadá takto:

DEPOSIT#	WHOPAID	AMOUNT	REMARKS
1	Bohatý strýček	20000	Vyzvednout seznam dáreků
2	Zaměstnavatel	25000	Výplata za červenec
3	Banka	5000	Půjčka

333 rows selected.

Pomocí jediné malé změny jsme získali nový datový zdroj.

## Vybírání odlišných hodnot

V původní tabulce CHECKS je patrné, že se některá data opakují. Podíváte-li se kupříkladu na sloupec AMOUNT pomocí příkazu

### Vstup ▼

```
SQL> select amount from checks;
```

uvidíte následující:

### Výstup ▼

```
AMOUNT
-----
1500
245
2000
980
1500
2500
250
```

7 rows selected.

### Analýza ▼

Všimněte si, že částka 1500 se opakuje. Co kdybychom chtěli zjistit, kolik různých částek se nachází v tomto sloupci? Zkuste následující příkaz:

### Vstup ▼

```
SQL> select DISTINCT amount from checks;
```

Výsledek by měl vypadat takto:

### Výstup ▼

```
AMOUNT
-----
2500
```

```

250
980
1500
2000
245

```

6 rows selected.

## Analýza ▼

Všimněte si, že se vybralo pouze šest řádků. V příkazu jsme uvedli `DISTINCT`, a proto se vypíše pouze jedna instance duplikovaných dat, což znamená, že obdržíme méně řádků. Co se ale stane, pokud do výsledné sady přidáme další sloupec?

## Vstup ▼

```
SQL> select DISTINCT ,payee, amount from checks;
```

Výsledek vypadá takto:

## Výstup ▼

PAYEE	AMOUNT
Nákupní centrum	1500
ČD	245
Nákupní centrum	2000
Místní benzinka	980
Diskont	1500
Hospoda	2500
Benzinka ve městě	250

7 rows selected.

## Analýza ▼

V tomto příkladu vrátí klíčové slovo `DISTINCT` odlišnou sadu kombinací sloupců. Mějte to na paměti, až začneme psát složitější příkazy.

Klíčové slovo `ALL` se skrytě používá v základním příkazu `SELECT`. Téměř nikdy jej však nevidíte, protože příkazy `SELECT <sloupec> FROM <tabulka>` a `SELECT ALL <sloupec> FROM <tabulka>` dávají tytéž výsledky.

Vyzkoušejte následující příklad – poprvé (a naposledy!) ve své kariéře programátora v jazyku SQL:

## Vstup/výstup ▼

```
SQL> SELECT ALL AMOUNT
2> FROM CHECKS;
AMOUNT
-----
1500
245

```



```
2000
 980
1500
2500
 250
```

```
7 rows selected.
```

Jedná se o tentýž příkaz jako `SELECT AMOUNT`. Kdo by se zdržoval psaním kódu navíc?

## Shrnutí

Klíčová slova `SELECT` a `FROM` umožňují dotazu získávat data. Pomocí příkazu `SELECT *` můžete vytvořit obsáhlejší příkaz se všemi sloupci nebo můžete požadované sloupce přeuspořádat nebo si vyžádat jen některé sloupce. Klíčové slovo `DISTINCT` omezuje výstup tak, aby neobsahoval duplicitní hodnoty, a to buď ve sloupci, nebo v sadě sloupců. A nakonec: klíčové slovo `ALL` představuje výchozí nastavení a znamená, že si přejete zobrazit veškeré výsledky. V následující lekci se naučíte, jak psát ještě selektivnější dotazy.

## Otázky a odpovědi

**Otázka:** Odkud se tato data vzala a jak se k nim připojím?

**Odpověď:** Data byla vytvořena pomocí metod popisovaných v lekci 11. Připojení k databázi závisí na tom, jakým způsobem se používá jazyk SQL. My používáme tradiční metodu příkazového řádku, která se používá v komerčních databázích. Tyto databáze byly tradičně doménou sálových počítačů nebo pracovních stanic, v současnosti se však přesouvají do osobních počítačů.

**Otázka:** Dobrá, pokud ale žádnou z těchto databází nepoužívám, jak budu používat jazyk SQL?

**Odpověď:** Jazyk SQL můžete používat také v rámci jiného programovacího jazyka. Vkládaný kód jazyka SQL funguje běžně jako rozšíření jiného jazyka, s nímž se lze nejčastěji setkat v jazyku COBOL, kde se kód jazyka SQL píše uvnitř programu a kompiluje s programem. Společnost Microsoft vytvořila aplikační rozhraní, jež umožňuje programátorům používat jazyk SQL z prostředí jazyků Visual Basic, C nebo C++. Knihovny dostupné pro databázové systémy SQL Server a Oracle vám též umožňují používat ve svých programech kód jazyka SQL. Společnost Borland zapouzdřila v prostředí Delphi jazyk SQL do databázových objektů. Principy probírané v této knize se vztahují na všechny tyto jazyky a také na mnoho dalších jazyků.

## Úkoly pro vás

Tato část nabízí kvízové otázky, které vám pomohou s upevněním získaných znalostí, a dále cvičení, jež vám poskytnou praktické zkušenosti s používáním osvojené látky. Pokuste se před nahlédnutím na odpovědi v příloze A odpovědět na otázky v kvízu a ve cvičení.

Příloha B obsahuje příkazy `CREATE TABLE`, které jsou nezbytné pro práci s příklady v této knize. Příloha C obsahuje příkazy `INSERT`, které vaše tabulky naplní daty. I když principy související s těmito příkazy budeme teprve probírat, je vhodné se na ně podívat už nyní. Pokud tedy používáte databázový systém MySQL, můžete napsat následující příkaz:

```
mysql> show databases;
+-----+
| Database           |
+-----+
| information_schema |
| michael            |
| mysql              |
| payments           |
| test               |
+-----+
5 rows in set (0.06 sec)
```

Databáze `mysql`, `information_schema` a `test` jsou standardní součástí databázového systému MySQL a databáze `michael` a `payments` jsme vytvořili již dříve. Pokud jste si v databázovém systému MySQL ještě nevytvořili vlastní databázi, můžete tak učinit právě nyní. Názvem databáze může být vaše křestní jméno, jméno vaší kočky nebo prostře cokoliv, co vám přijde na mysl. Následující příkaz vytvoří databázi s názvem `kuba`.

```
mysql> create database kuba;
```

Zde je výsledek příkazu `CREATE DATABASE`:

```
Query OK, 1 row affected (0.00 sec)
```

Pro ověření toho, co jsme právě provedli, můžeme znovu vydat příkaz `SHOW DATABASES`.

```
mysql> show databases;
+-----+
| Database           |
+-----+
| information_schema |
| kuba               |
| michael            |
| mysql              |
| payments           |
| test               |
+-----+
6 rows in set (0.06 sec)
```

K práci uvnitř nějaké databáze je nutné oznámit databázovému systému MySQL, že chceme používat určitou databázi.

```
mysql> use kuba
Database changed
```

Pro zobrazení tabulek v databázi napište:

```
mysql> show tables;
Empty set (0.00 sec)
```

Nemáme žádné tabulky, takže nemůžeme pokračovat, dokud je nevytvoříme. Tabulky nám však příliš nepomohou, pokud nebudou obsahovat nějaká data, takže je musíme ještě naplnit daty.

Máte-li přístup k elektronické verzi příkazů `CREATE TABLE` a `INSERT` (např. z webu nakladatelství), můžete je zkopírovat do svého příkazového řádku MySQL. V opačném případě použijte následující příkaz k vytvoření tabulky `CHECKS`:

```
mysql> create table checks
-> (`check` numeric(6) not null,
-> payee varchar(20) not null,
-> amount decimal(6,2) not null,
-> remarks varchar(20) not null);
```

Měli byste obdržet následující výsledek:

```
Query OK, 0 rows affected (0.05 sec)
```

Před vkládáním (a též vypisováním) dat z konzoly je nutné nastavit znakovou sadu tak, aby odpovídala kódování používané v konzole. V případě české (či slovenské) verze Windows tedy nastavíme znakovou sadu `latin2` (v případě anglické verze znakovou sadu `latin1`):

```
mysql> set character set latin2;
Query OK, 0 rows affected (0.00 sec)
```

Zde jsou příkazy `INSERT`:

```
insert into checks values
('1', 'Nákupní centrum', '1500', 'Příště vzít syny');
insert into checks values
('2', 'ČD', '245', 'Vlak do Prahy');
insert into checks values
('3', 'Nákupní centrum', '2000', 'Mobilní telefon');
insert into checks values
('4', 'Místní benzinka', '980', 'Benzín');
insert into checks values
('5', 'Diskont', '1500', 'Nákup');
insert into checks values
('6', 'Hospoda', '2500', 'Divoká noc');
insert into checks values
('7', 'Benzinka ve městě', '250', 'Benzín');
```

Gratulujeme! Nyní máte úspěšně vytvořenou a naplněnou svou první databázi.

## Kvíz

1. Vracejí následující příkazy stejný výstup?

```
SELECT * FROM CHECKS;
select * from checks;
```

2. Žádný z následujících dotazů nefunguje. Proč?

a.

```
Select *
```

b.

```
Select * from checks
```

c.

```
Select amount name payee FROM checks;
```

3. Který z následujících příkazů jazyka SQL bude fungovat?

a.

```
select *  
from checks;
```

b.

```
select * from checks;
```

c.

```
select * from checks
```

4. Napište příkaz jazyka SQL, který vrátí z tabulky `CHECKS` pouze číslo šeku a částku.
5. Napište příkaz jazyka SQL, který vrátí z tabulky `CHECKS` pouze odlišné (`DISTINCT`) názvy příjemců plateb.
6. Napište příkaz jazyka SQL v databázovém systému MySQL, který vrátí z tabulky `CHECKS` odlišnou sadu názvů příjemců plateb a částek. Záleží v tomto případě na pořadí sloupců?
7. Zaručuje následující příkaz nějaké konkrétní uspořádání dat nebo sloupců zleva doprava?

```
select * from checks;
```

## Cvičení

1. Pomocí tabulky `CHECKS` z dřívější části této lekce napište dotaz, který vrátí jen čísla šeků a poznámky.
2. Přepište dotaz z cvičení 1 tak, aby se poznámky objevily ve výsledku dotazu jako první sloupec.
3. Pomocí tabulky `CHECKS` napište dotaz, který vrátí všechny odlišné poznámky.
4. Napište dotaz, který z tabulky `CHECKS` vrátí pouze sloupce `CHECK` a `AMOUNT`.
5. V databázovém systému MySQL zobrazte všechny databáze.
6. V databázovém systému MySQL použijte jinou databázi.
7. V databázovém systému MySQL zobrazte všechny tabulky v aktuální databázi.
8. V databázovém systému MySQL se vraťte (použijte) ke své původní databázi (tj. k databázi obsahující tabulku `CHECKS`).



## LEKCE 3

# Výrazy, podmínky a operátory

V lekci 2 jste pomocí klauzulí `SELECT` a `FROM` zajímavými (a užitečnými) způsoby manipulovali s daty. V této lekci se o těchto klauzulích dozvíte ještě mnohem více. Základní dotaz obohatíte o nové termíny, novou klauzuli a skupinu užitečných prvků, kterým se říká *operátory*. Na konci této lekce budete ovládat následující:

- Co je výraz a jak se používá.
- Co je podmínka a jak se používá.
- Základní použití klauzule `WHERE`.
- Používání aritmetických, porovnávacích, znakových, logických a množinových operátorů.
- Praktická znalost některých smíšených operátorů.

### POZNÁMKA

Pro generování příkladů v této lekci používáme databázové systémy Oracle a MySQL. Jiné implementace jazyka SQL se mohou malinko lišit ve způsobu zadávání příkazů nebo zobrazování výstupu, výsledky jsou ale ve všech implementacích splňujících standard ANSI v podstatě stejné.

### POZNÁMKA

Tato lekce je jedna z nejdelších v této knize a také jedna z nejdůležitějších, neboť obsahuje základy, na nichž staví většinu ostatních lekcí. V této lekci budete vstřebávat mnoho příkladů. Nepokoušejte se všechny si je zapamatovat, ale spíše se naučte principy, na kterých fungují. V následujících lekcích budete mít spoustu příležitostí aplikovat to, co se zde naučíte.

## Pracujeme s dotazovými výrazy

Definice výrazu je jednoduchá: *výraz* vrací nějakou hodnotu. Existuje celá řada typů výrazů pokrývajících nejrůznější datové typy, jako jsou řetězce nebo číselné a logické hodnoty. Ve skutečnosti je výrazem téměř cokoliv, co následuje za nějakou klauzulí (jako je např. `SELECT` nebo `FROM`). V následujícím příkladu je `AMOUNT` výraz, který vrací hodnotu obsaženou ve sloupci `AMOUNT`:

## Syntaxe ▼

---

```
SELECT AMOUNT FROM CHECKS;
```

Níže uvedené je samozřejmě také považováno za číselný výraz. Pamatujte si, že podstatnou vlastností výrazu je, že vrací nějakou hodnotu.

## Syntaxe ▼

---

```
SELECT AMOUNT*10 FROM CHECKS;
```

V následujícím příkazu jsou výrazy `NAME`, `ADDRESS`, `PHONE` a `ADDRESSBOOK`:

## Syntaxe ▼

---

```
SELECT NAME, ADDRESS, PHONE  
FROM ADDRESSBOOK;
```

Podívejme se nyní na následující klauzuli `WHERE`:

## Syntaxe ▼

---

```
WHERE NAME = 'SOVA'
```

Obsahuje podmínku `NAME = 'SOVA'`, což je příklad logického výrazu. Výraz `NAME = 'SOVA'` bude mít hodnotu `TRUE` nebo `FALSE` v závislosti na podmínce =.

# Podmínky v dotazech

Pokud chcete ve své databázi najít konkrétní prvek či skupinu prvků, pak musíte sáhnout po jednom či více podmínkách. Podmínky se umísťují do klauzule `WHERE`. V předchozím příkladu vypadá podmínka takto:

## Syntaxe ▼

---

```
NAME = 'SOVA'
```

K vyhledání všech zaměstnanců, kteří za poslední měsíc odpracovali více než 100 hodin, bychom mohli použít následující podmínku:

## Syntaxe ▼

---

```
NUMBEROFHOURS > 100
```

Díky podmínkám můžete provádět selektivní dotazy. Ve své nejběžnější formě se podmínky skládají z proměnné, konstanty a porovnávacího operátoru. V prvním příkladu máme proměnnou `NAME`, konstantu `'SOVA'` a porovnávací operátor `=`. Ve druhém příkladu je proměnná `NUMBEROFHOURS`, konstanta `100` a porovnávací operátor `>`. Před tím, než budete moci psát podmíněné dotazy, se musíte seznámit ještě se dvěma dalšími prvky: s klauzulí `WHERE` a s operátory.

Syntaxe klauzule `WHERE` vypadá takto:

## Syntaxe ▼

```
WHERE vyhledávací_podmínka
```

Mezi nejčastěji používané klauzule jazyka SQL patří `SELECT`, `FROM` a `WHERE`. Klauzule `WHERE` prostě způsobí, že vaše dotazy budou selektivnější. Bez klauzule `WHERE` nemůžete dělat nic užitečného, než zobrazit všechny záznamy ve zvolené tabulce či tabulkách:

## Vstup ▼

```
SQL> SELECT * FROM BIKES;
```

Tento příkaz vypíše data na všech řádcích tabulky `BIKES`.

## Výstup ▼

NAME	FRAMESIZE	ZMPOSITION	KMRIDDEN	TYPE
Trek 2300	22.5	Uhlíkové vlákno	3500	Závodní
Burley	22	Oceľ	2000	Dvojkoľo
Obr	19	Oceľ	1500	Městské
Fuji	20	Oceľ	500	Trekíngové
Spec	16	Oceľ	100	Horské
Dělo	22.5	Hliník	3000	Závodní

6 rows selected.

Pokud chcete zobrazit určité kolo, pak můžete napsat:

## Vstup ▼

```
SQL> SELECT *
  2 FROM BIKES
  3 WHERE NAME = 'Burley';
```

Nyní obdržíte pouze jediný záznam:

## Výstup ▼

NAME	FRAMESIZE	ZMPOSITION	KMRIDDEN	TYPE
Burley	22	Oceľ	2000	Dvojkoľo

1 rows selected.

Z těchto jednoduchých příkladů je patrné, jak lze klást podmínky na data, která si přejete získat.

# Jak používat operátory

Operátory jsou prvky, které se používají uvnitř výrazů pro stanovení, jak má daná podmínka získat data. Operátory spadají do šesti skupin: aritmetické, porovnávací, znakové, logické,



množinové a smíšené. Jazyk SQL nabízí tři typy operátorů: aritmetické, porovnávací a logické.

## Aritmetické operátory

Mezi aritmetické operátory patří plus (+), minus (-), dělení (/), násobení (\*) a modulo (%). První čtyři operátory nepotřebují žádný další komentář. Operátor modulo vrací celočíselný zbytek po dělení. Zde jsou dva příklady:

```
5% 2 = 1
6% 2 = 0
```

Operátor modulo nepracuje s datovými typy, které mají desetinná místa, jako jsou typy `Real` nebo `Number`.

Pokud umístíte několik aritmetických operátorů do výrazu bez závorek, pak se vyhodnotí v tomto pořadí: násobení, dělení, modulo, sčítání a odčítání. Například výraz:

```
2 * 6 + 9 / 3
```

se vyhodnotí takto:

```
12 + 3 = 15
```

Nicméně výraz:

```
2 * (6 + 9) / 3
```

bude vyhodnocen jako:

```
2 * 15 / 3 = 10
```

Dávejte proto pozor, kam umísťujete závorky! Někdy se stane, že výraz provede přesně to, co jste mu zadali, ale ne to, co jste od něj očekávali. Totéž platí pro jazyk SQL jako celek.

V následující části se podrobněji podíváme na aritmetické operátory a napíšeme si několik dotazů.

### Plus (+)

Znaménko plus lze použít několika způsoby. Napište následující příkaz pro zobrazení tabulky `PRICE` (ceny):

### Vstup/výstup ▼

```
SQL> SELECT * FROM PRICE;
ITEM          WHOLESALE
-----
Rajčata             34
Brambory            51
Banány              67
Tuřiny              45
Sýr                 89
Jablka              23
6 rows selected.
```

Nyní napište příkaz:

### Vstup ▼

```
SQL> SELECT ITEM, WHOLESALE, WHOLESALE + 15
      2 FROM PRICE;
```

Zde jsme pomocí operátoru + přidali ke každé položce 15 korun:

### Výstup ▼

ITEM	WHOLESALE	WHOLESALE+15
Rajčata	34	49
Brambory	51	66
Banány	67	82
Tuřiny	45	60
Sýr	89	104
Jablka	23	38

6 rows selected.

### Analýza ▼

Co ale znamená poslední sloupec s nevzhledným záhlavím WHOLESALE+15? V původní tabulce totiž vůbec není. Jazyk SQL vám umožňuje kombinací nebo modifikací stávajících sloupců vytvářet virtuální či odvozené sloupce.

Znovu napište původní příkaz:

### Vstup ▼

```
SQL> SELECT * FROM PRICE;
```

Vypíše se následující tabulka:

### Výstup ▼

ITEM	WHOLESALE
Rajčata	34
Brambory	51
Banány	67
Tuřiny	45
Sýr	89
Jablka	23

6 rows selected.

### Analýza ▼

Z tohoto výstupu je jasné, že původní data se nijak nezměnila a že záhlavní sloupce WHOLESALE+15 není jejich trvalou součástí. Ve skutečnosti je toto záhlavní sloupce tak nevzhledné, že bychom s tím měli něco udělat:

Napište následující příkaz:

### Vstup ▼

```
SQL> SELECT ITEM, WHOLESale, (WHOLESale + 15) RETAIL
      2 FROM PRICE;
```

Zde je výsledek:

### Výstup ▼

ITEM	WHOLESale	RETAIL
Rajčata	34	49
Brambory	51	66
Banány	67	82
Tuřiny	45	60
Sýr	89	104
Jablka	23	38

6 rows selected.

### Analýza ▼

To je skvělé! Nejenže můžete vytvářet nové výstupní sloupce, ale můžete je také rovnou přejmenovat. Pomocí syntaxe *název\_sloupce alias* můžete přejmenovat libovolné sloupce. (Všimněte si mezery mezi názvem sloupce a jeho aliasem.)

Kupříkladu dotaz:

### Vstup ▼

```
SQL> SELECT ITEM PRODUCE, WHOLESale, WHOLESale + 25 RETAIL
      2 FROM PRICE;
```

prejmenuje sloupce takto:

### Výstup ▼

PRODUCE	WHOLESale	RETAIL
Rajčata	34	59
Brambory	51	76
Banány	67	92
Tuřiny	45	70
Sýr	89	114
Jablka	23	48

6 rows in set (0.00 sec)

**POZNÁMKA**

Některé implementace jazyka SQL používají syntaxi <název sloupce> = <alias>. Předchozí příklad bychom tedy zapsali takto:

```
SQL> SELECT ITEM = PRODUCE,
      2 WHOLESAL,
      3 WHOLESAL + 25 = RETAIL,
      4 FROM PRICE;
```

Standard jazyka SQL dále umožňuje používat klíčové slovo AS, které je implementováno v mnoha databázových systémech a které se používá následovně:

```
SQL> SELECT ITEM AS PRODUCE,
      2 WHOLESAL,
      3 WHOLESAL + 25 = RETAIL,
      4 FROM PRICE;
```

Přesnou syntaxi si ověřte ve vlastní implementaci.

**POZNÁMKA**

V databázovém systému MySQL můžete specifikovat aliasy sloupců smíšeným způsobem.

Možná jste zvědaví, k čemuž jsou aliasy sloupců, pokud nepoužíváme jazyk SQL na příkazovém řádku. Nu dobrá. Zajímalo vás někdy, jak fungují generátory hlášení? Až vás jednoho dne někdo požádá, abyste napsali generátor hlášení, pak si na tuto možnost jistě vzpomenete, a nebudete tak muset strávit týdny znovuobjevováním toho, co již jednou napsal Dr. Codd a společnost IBM.

V některých implementacích jazyka SQL lze znaménko plus používat také jako znakový operátor. K této stránce znaménka plus se vrátíme v pozdější části této lekce.

**Minus (-)**

Minus má také dvojí použití. Za prvé dokáže změnit znaménko čísla. K demonstraci této funkce můžeme využít tabulku HILOW (teplotní extrémy).

**Vstup/výstup ▼**

```
SQL> SELECT * FROM HILOW;
COUNTRY LOWS   HIGHS
-----
CZ         -20      34
SK         -23      33
PL         -28      31
GB         -15      28
FR         -20      35
```

Níže uvedený příklad ukazuje způsob, jak lze manipulovat s daty:

### Vstup/výstup ▼

```
SQL> SELECT COUNTRY, - LOWS, - HIGHS
      2 FROM HILOW;
```

COUNTRY	LOWS	HIGHS
CZ	20	-34
SK	23	-33
PL	28	-31
GB	15	-28
FR	20	-35

#### POZNÁMKA

Všimněte si, že znaménko minus obrátilo teploty.

Druhé (a zcela zřejmé) použití znaménka minus spočívá v odečtení jednoho sloupce od druhého:

### Vstup/výstup ▼

```
SQL> SELECT COUNTRY,
      2 LOWS,
      3 HIGHS,
      4 (HIGHS - LOWS) DIFFERENCE
      5 FROM HILOW;
```

COUNTRY	LOWS	HIGHS	DIFFERENCE
CZ	-20	34	54
SK	-23	33	56
PL	-28	31	59
GB	-15	28	43
FR	-20	35	55

Pokud použijete znaménko minus omylem na znakové pole, pak obdržíte výsledek podobný následujícímu:

### Vstup/výstup ▼

```
SQL> SELECT -COUNTRY FROM HILOW;
```

```
ERROR:
ORA-01722: invalid number
no rows selected
```

Přesná chybová zpráva se může v jednotlivých implementacích lišit. Zde je stejný příklad v databázi MySQL:

### Vstup/výstup ▼

```
mysql> select -country
-> from hilow;
```

```
+-----+
| -country |
+-----+
|         0 |
|         0 |
|         0 |
|         0 |
+-----+
4 rows in set (0.00 sec)
```

Databázový systém MySQL sice příkaz `SELECT` vyhodnotil, ale jak můžete vidět, výsledky jsou v podstatě nesmyslné.

### Dělení (/)

Operátor dělení má jen jediný zřejmý smysl. Nyní se vrátíme k tabulce `PRICE`. Napište následující příkaz:

### Vstup/výstup ▼

```
SQL> SELECT * FROM PRICE;
ITEM      WHOLESALE
-----
Rajčata   34
Brambory  51
Banány    67
Tuřiny    45
Sýr       89
Jablka    23
6 rows selected.
```

```
mysql> select * from price;
+-----+-----+
| item      | wholesale |
+-----+-----+
| Rajčata   | 34.00     |
| Brambory  | 51.00     |
| Banány    | 67.00     |
| Tuřiny    | 45.00     |
| Sýr       | 89.00     |
| Jablka    | 23.00     |
+-----+-----+
6 rows in set (0.26 sec)
```

Zapsáním následujícího příkazu se zobrazí prodejní ceny při slevě dvě položky za cenu jedné:

### Vstup/výstup ▼

```
SQL> SELECT ITEM, WHOLESALE, (WHOLESALE/2) SALEPRICE
2 FROM PRICE;
ITEM      WHOLESALE  SALEPRICE
-----
```

```
Rajčata      34      17
Brambory    51      25.5
Banány     67      33.35
Tuřiny     45      22.5
Sýr       89      44.5
Jablka     23      11.5
6 rows selected.
```

Stejný příklad vypadá v databázovém systému MySQL takto:

### Vstup/výstup ▼

```
mysql> select ITEM, WHOLESALE, (WHOLESALE/2) Saleprice
-> from price;
```

```
+-----+-----+-----+
| ITEM      | WHOLESALE | Saleprice |
+-----+-----+-----+
| Rajčata   | 34.00    | 17.00    |
| Brambory  | 51.00    | 22.50    |
| Banány    | 67.00    | 33.35    |
| Tuřiny    | 45.00    | 22.50    |
| Sýr       | 89.00    | 44.50    |
| Jablka    | 23.00    | 11.50    |
+-----+-----+-----+
```

```
6 rows in set (0.26 sec)
```

Použití operátoru dělení v předchozím příkazu SELECT je vskutku jednoduché (tedy až na to, padesátníky a menší mince už dávno neplatí).

### Násobení (\*)

Použití operátoru násobení je také docela přímočaré. Opět využijeme tabulku PRICE:

### Vstup/výstup ▼

```
SQL> SELECT * FROM PRICE;
```

```
ITEM      WHOLESALE
-----
Rajčata   34
Brambory  51
Banány    67
Tuřiny    45
Sýr       89
Jablka    23
6 rows selected.
```

Výstup následujícího dotazu představuje paušální slevu ve výši 10 %. Samotná data v tabulce se nijak nezmění.

### Vstup/výstup ▼

```
SQL> SELECT ITEM, WHOLESALE, WHOLESALE * 0.9 NEWPRICE
2 FROM PRICE;
```

```

ITEM      WHOLESale  NEWPRICE
-----
Rajčata   34          30.6
Brambory  51          45.9
Banány    67          60.3
Tuřiny    45          40.5
Sýr       89          80.1
Jablka    23          20.7
6 rows selected.

```

Stejný příklad vypadá v databázovém systému MySQL takto:

### Vstup/výstup ▼

```

mysql> select Item,
-> Wholesale, Wholesale * 0.9 "New Price"
-> from price;

```

```

+-----+-----+-----+
| ITEM      | WHOLESale | New Price |
+-----+-----+-----+
| Rajčata   | 34.00    | 30.60    |
| Brambory  | 51.00    | 45.90    |
| Banány    | 67.00    | 60.30    |
| Tuřiny    | 45.00    | 40.50    |
| Sýr       | 89.00    | 80.10    |
| Jablka    | 23.00    | 20.70    |
+-----+-----+-----+
6 rows in set (0.00 sec)

```

### POZNÁMKA

Ještě jedna poznámka ohledně aliasů. Svému sloupci můžete pomocí uvozovek přiřadit dvouslovné záhlaví. Někdy může jít o jednoduché a někdy o dvojité uvozovky. Povolený zápis najdete v dokumentaci ke své implementaci jazyka SQL.

Pomocí uvedených operátorů můžete v příkazu SELECT provádět zajímavé výpočty.

### Modulo (%)

Operátor modulo vrací celočíselný zbytek operace dělení. Nyní použijeme tabulku REAMINS (zbytky):

### Vstup/výstup ▼

```

SQL> SELECT * FROM REAMINS;
NUMERATOR  DENOMINATOR
-----
10          5
8           3
23          9

```



```

      40          17
     1024         18
      85          34
6 rows selected. 3

```

Stejný příklad vypadá v databázi MySQL takto:

### Vstup/výstup ▼

```

mysql> select * from remains;
+-----+-----+
| numerator | denominator |
+-----+-----+
|         10 |          5 |
|          8 |          3 |
|         23 |          9 |
|         40 |         17 |
|        1024 |         16 |
|         85 |         34 |
+-----+-----+
6 rows in set (0.43 sec)

```

Nyní můžeme vytvořit nový výstupní sloupec s názvem REMAINDER (zbytek), který bude ucho-  
vávat hodnoty NUMERATOR % DENOMINATOR:

### Vstup/výstup ▼

```

SQL> SELECT NUMERATOR,
      2 DENOMINATOR,
      3 NUMERATOR % DENOMINATOR REMAINDER
      4 FROM REMAINS;

```

```

NUMERATOR  DENOMINATOR  REMAINDER
-----
      10          5          0
       8          3          2
      23          9          5
      40         17          6
     1024         16          0
      85         34         17
6 rows selected.

```

Stejný příklad vypadá v databázovém systému MySQL takto:

### Vstup/výstup ▼

```

mysql> select numerator, denominator, numerator % denominator remainder
      -> from remains;
+-----+-----+-----+
| numerator | denominator | remainder |
+-----+-----+-----+
|         10 |          5 |          0 |
|          8 |          3 |          2 |

```

```

|      23 |      9 |      5 |
|      40 |     17 |      6 |
|     1024 |     16 |      0 |
|      85 |     34 |     17 |
+-----+
6 rows in set (0.01 sec)

```

## Analýza ▼

Některé implementace jazyka SQL implementují modulo jako funkci s názvem MOD (viz lekce 7). Následující příkaz vytvoří výsledky, které jsou identické s výsledky předchozího příkazu:

## Vstup/výstup ▼

```

SQL> SELECT NUMERATOR,
       2 DENOMINATOR,
       3 MOD(NUMERATOR,DENOMINATOR) REMAINDER
       4 FROM REMAINS;

```

```

NUMERATOR  DENOMINATOR  REMAINDER
-----
          10             5             0
           8             3             2
          23             9             5
          40            17             6
         1024            16             0
           85            34            17
6 rows selected.

```

Stejný příklad vypadá v databázovém systému MySQL takto:

## Vstup/výstup ▼

```

mysql> select numerator, denominator,
-> mod(numerator,denominator) remainder
-> from remains;
+-----+
| numerator | denominator | remainder |
+-----+
|      10 |         5 |         0 |
|       8 |         3 |         2 |
|      23 |         9 |         5 |
|      40 |        17 |         6 |
|     1024 |        16 |         0 |
|       85 |        34 |        17 |
+-----+
6 rows in set (0.00 sec)

```

## Precedence operátorů

*Precedence* je pořadí, v jakém daná implementace vyhodnotí různé operátory stejného výrazu. V této části se podíváme na precedenci v příkazu `SELECT`. K tomuto účelu využijeme tabulku `PRECEDENCE`:

### Vstup/výstup ▼

```
SQL> SELECT * FROM PRECEDENCE;
```

```
  N1  N2  N3  N4
-----
    1   2   3   4
   13  24  35  46
    9   3  23   5
   63   2  45   3
    7   2   1   4
```

5 rows selected.

```
mysql> select * from precedence;
```

```
+-----+-----+-----+-----+
| n1 | n2 | n3 | n4 |
+-----+-----+-----+
|  1 |  2 |  3 |  4 |
| 13 | 24 | 35 | 46 |
|  9 |  3 | 23 |  5 |
| 63 |  2 | 45 |  3 |
|  7 |  2 |  1 |  4 |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

Použijte následující úryvek kódu k otestování precedences:

### Vstup/výstup ▼

```
SQL> SELECT
```

```
  2 N1+N2*N3/N4,
  3 (N1+N2)*N3/N4,
  4 N1+(N2*N3)/N4
  5 FROM PRECEDENCE;
```

```
N1+N2*N3/N4 (N1+N2)*N3/N4 N1+(N2*N3)/N4
```

```
-----
    2.5          2.25          2.5
 31.26087      28.152174      31.26087
    22.8          55.2          22.8
    93            975           93
    7.5           2.25          7.5
```

5 rows selected.

```
mysql> select n1+n2*n3/n4,
-> (n1+n2)*n3/n4,
-> n1+(n2*n3)/n4
-> from precedence;
```

```
+-----+-----+-----+-----+

```

```

| n1+n2*n3/n4 | (n1+n2)*n3/n4 | n1+(n2*n3)/n4 |
+-----+-----+-----+
|          2.50 |           9     |          2.50 |
|          31.26 |          1295   |          31.26 |
|          22.80 |           276   |          22.80 |
|          93.00 |          2925   |          93.00 |
|           7.50 |           9     |           7.50 |
+-----+-----+-----+
5 rows in set (0.00 sec)

```

Všimněte si, že první a poslední sloupce obsahují stejné hodnoty. Pokud byste přidali čtvrtý sloupec  $N1+N2*(N3/N4)$ , pak by jeho hodnoty byly stejné jako hodnoty prvního a posledního sloupce. Pravidla pro precedenci se řídí obvyklou algebrou, takže hodnoty jsou normálně prováděny v následujícím pořadí zleva doprava:

1. závorky,
2. násobení nebo dělení,
3. součet nebo rozdíl.

## Analýza ▼

Nejdříve jsou prostě vyhodnoceny hodnoty v závorkách, poté se provedou operace násobení a dělení a nakonec operace součtu a rozdílu. Tato pravidla je důležité si pamatovat, neboť se vám budou hodit, až začnete psát složitější výpočtu pro analýzu dat.

## Porovnávací operátory

Věrní svému jménu provádějí porovnávací operátory porovnávání výrazů a přitom vracejí jednu ze tří hodnot: TRUE, FALSE nebo UNKNOWN. No počkat! TRUE a FALSE jsou jasné, ale co znamená UNKNOWN?

K tomu, abyste pochopili, jak můžete obdržet hodnotu UNKNOWN, musíte vědět něco o koncepci NULL. V databázích představuje NULL absenci dat v nějakém poli. To ovšem neznamená, že daný sloupec v něm má nulu nebo prázdný řetězec. Nula i prázdný řetězec jsou určité hodnoty. NULL znamená, že v daném poli není vůbec nic.

Pokud provedete porovnání, jako například `Field = 9`, přičemž jedinou přijatelnou hodnotou pro `Field` je NULL, pak obdržíte hodnotu UNKNOWN. Vzhledem k tomu, že UNKNOWN je prapodivná podmínka, provádí většina variant jazyka SQL její převod na hodnotu FALSE a poskytuje speciální operátor `IS NULL`, který testuje přítomnost podmínky NULL.

Zde je příklad vyhodnocení na NULL. Předpokládejme, že nějaký záznam v tabulce PRICE neobsahuje pro pole WHOLESALE žádnou hodnotu. Výsledky dotazu mohou vypadat takto:

## Vstup/výstup ▼

```
SQL> SELECT * FROM PRICE;
```

```

ITEM      WHOLESALA
-----
Rajčata   34
Brambory  51

```

Banány	67
Tuřiny	45
Sýr	89
Jablka	23
Pomeranče	

7 rows selected.

## Analýza ▼

Všimněte si, že v případě záznamu Pomeranče není v poli WHOLESale uvedena žádná hodnota. Hodnota pole WHOLESale pro položku Pomeranče je tedy NULL. Hodnota NULL je v tomto případě jasně patrná, protože leží v číselném sloupci. Pokud by se však objevila ve sloupci ITEM, bylo by nemožné zjistit rozdíl mezi hodnotou NULL a prázdným řetězcem.

Vyzkoušejte následující příkaz, který vyhledá řádek s hodnotou NULL:

## Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM PRICE
      3 WHERE WHOLESale IS NULL;
ITEM      WHOLESale
-----
Pomeranče
```

1 rows selected.

Jak je patrné z výstupu, jediným prvkem, jehož pole WHOLESale má hodnotu NULL nebo neobsahuje žádnou hodnotu, je prvek Pomeranče. Co se ale stane, když místo operátoru IS NULL použijeme znaménko rovnosti (=)?

## Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM PRICE
      3 WHERE WHOLESale = NULL;
```

no rows selected

## Analýza ▼

Nic bychom nenašli, protože porovnání WHOLESale = NULL vrátí hodnotu FALSE, což je dáno tím, že výsledek byl neznámý (UNKNOWN). Místo operátoru = by bylo vhodnější použít operátor IS NULL, takže příkaz WHERE by obsahoval WHERE WHOLESale IS NULL. V takovém případě bychom obdrželi všechny řádky, v nichž by se vyskytovala hodnota NULL.

Tento příklad dále ilustruje použití nejčastějšího porovnávacího operátoru (=) a také oblast působnosti všech porovnávacích operátorů, kterou je klauzule WHERE. Klauzuli WHERE již znáte, nyní se tedy ve stručnosti podíváme na znaménko rovnosti.

## Znaménko rovnosti (=)

V dřívější části této lekce jsme si ukázali, jak některé implementace jazyka SQL přiřazují alias pomocí znaménka rovnosti. V klauzuli `WHERE` patří znaménko rovnosti k nejčastěji používaným porovnávacím operátorům. Samo o sobě představuje velmi pohodlný způsob výběru jedné hodnoty z mnoha. Vyzkoušejte následující příkaz:

### Vstup/výstup ▼

```
SQL> SELECT * FROM FRIENDS;
LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Lužný     Bob        602 111222  OL  79821
Matyáš    Bob        773 123456  BR
Strouhal  Fido       700 343434  ZL  76701
Přikryl   Tom        581 010101  OT  72000
Černý     Franta     777 000999  OL  79801
5 rows selected.
```

Pokusme se nyní najít Tomův řádek. (S krátkým seznamem vypadá tento úkol jako velmi snadný, ale může se stát, že budete mít mnohem více přátel, než máme my, nebo můžete mít nějaký seznam s tisíci záznamy.)

### Vstup/výstup ▼

```
SQL> SELECT *
  2 FROM FRIENDS
  3 WHERE FIRSTNAME = 'Tom';

LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Přikryl   Tom        581 010101  OT  72000
1 rows selected.
```

```
mysql> select * from friends
-> where firstname = 'Tom';
+-----+-----+-----+-----+-----+-----+
| lastname | firstname | callprefix | phone  | re  | zip  |
+-----+-----+-----+-----+-----+-----+
| Přikryl  | Tom      |          581 | 010101 | OT  | 72000 |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.37 sec)
```

Obdrželi jsme výsledek dle očekávání. Zkuste nyní tento příkaz:

### Vstup/výstup ▼

```
SQL> SELECT *
  2 FROM FRIENDS
  3 WHERE FIRSTNAME = 'Bob';
```

```

LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Lužný     Bob         602 111222  OL  79821
Matyáš    Bob         773 123456  BR
2 rows selected.

```

**POZNÁMKA**

Zde můžete vidět, že operátor = dokáže vytáhnout i více záznamů. Všimněte si, že druhý záznam má pole ZIP (PSC) prázdné. Jedná se o znakové pole (o vytváření a naplňování tabulek se dozvíte v lekcí 9), přičemž z tohoto záznamu je patrné, že hodnotu NULL ve znakovém poli je nemožné odlišit od pole s prázdným řetězcem.

Zde je další velice důležitá lekce týkající se rozlišování velikosti písmen:

**Vstup/výstup ▼**

```
SQL> SELECT FIRSTNAME FROM FRIENDS
      2 WHERE FIRSTNAME = 'Fido';
```

```

FIRSTNAME
-----
Fido
1 row selected.

```

```

mysql> select firstname from friends where firstname = 'Fido';
+-----+
| firstname |
+-----+
| Fido      |
+-----+
1 row in set (0.00 sec)

```

Nyní vyzkoušejte tento příkaz:

**Vstup/výstup ▼**

```
SQL> select FIRSTNAME from friends
      2 where firstname = 'FIDO';
```

```
no rows selected.
```

```

mysql> select firstname
      -> from friends
      -> where firstname = 'fido';
+-----+
| firstname |
+-----+
| Fido      |
+-----+
1 row in set (0.01 sec)

```

## Analýza ▼

Ikdyž v syntaxi jazyka SQL se velikost písmen nerozlišuje, u dat tomu tak není, tedy přinejmenším v některých implementacích. Jak je patrné z výše uvedených příkladů, u dat uložených v databázovém systému Oracle (SQL\*Plus) se velikost písmen rozlišuje, zatímco v databázovém systému MySQL se nerozlišuje.

Většina společností ukládá data kvůli jejich konzistenci ve tvaru s velkými písmeny. Někdy může být vhodnější ukládat data buď jen s velkými, nebo jen s malými písmeny, bez ohledu na tu databázi, s níž pracujete. Můžete se totiž stát, že by míchání velikosti písmen mohlo vést k obtížím při získávání přesných dat prostřednictvím porovnávání v klauzuli WHERE.

## Větší než (>) a větší nebo rovno (>=)

Operátor větší než funguje takto:

### Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE CALLPREFIX > 700;
LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Matyáš    Bob        773 123456 BR
Černý     Franta     777 000999 OL 79801
```

2 rows selected.

V tomto příkladu vyhledáváme všechny záznamy s telefonní předvolbou menší než 700 (ale ne rovnající se hodnotě 700). Pro začlenění předvolby 700 napište následující příklad:

### Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE CALLPREFIX >= 700;
LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Matyáš    Bob        773 123456 BR
Strouhal  Fido       700 343434 ZL 76701
Černý     Franta     777 000999 OL 79801
```

3 rows selected.

```
mysql> select * from friends
-> where callprefix >= 700;
+-----+-----+-----+-----+-----+-----+
| lastname | firstname | callprefix | phone | re | zip |
+-----+-----+-----+-----+-----+-----+
| Matyáš | Bob | 773 | 123456 | BR | NULL |
| Strouhal | Fido | 700 | 343434 | ZL | 76701 |
| Černý | Franta | 777 | 000999 | OL | 79801 |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.34 sec)
```



Po této změně obdržíme všechny záznamy s telefonní předvolbou 700 a vyšší. Stejného výsledku byste dosáhli i podmínkou `CALLPREFIX > 699`.

**POZNÁMKA**

Všimněte si, že hodnota 700 není v žádném ze dvou předchozích příkladů obklopena uvozovkami. Pole číselného typu uvozovky nevyžadují.

**Menší než (<) a menší nebo rovno (<=)**

Jak již asi tušíte, tyto porovnávací operátory fungují úplně stejně jako operátory `>` a `>=`, ale obráceně:

**Vstup/výstup ▼**

```
SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE REGION < 'Z';
LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Lužný     Bob        602         111222  OL  79821
Matyáš    Bob        773         123456  BR
Příkryl   Tom        581         010101  OT  72000
Černý     Franta     777         000999  OL  79801
4 rows selected.
```

```
mysql> select * from friends where region < 'T';
+-----+-----+-----+-----+-----+-----+
| lastname | firstname | callprefix | phone | re | zip |
+-----+-----+-----+-----+-----+-----+
| Lužný | Bob | 602 | 111222 | OL | 79821 |
| Matyáš | Bob | 773 | 123456 | BR | NULL |
| Příkryl | Tom | 581 | 010101 | OT | 72000 |
| Černý | Franta | 777 | 000999 | OL | 79801 |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

**POZNÁMKA**

Pokud má sloupec v databázovém systému Oracle pouze dva znaky, pak se jeho název ve vrácených řádcích zkrátí na dva znaky. Pokud se tedy nějaký sloupec jmenuje `COWS`, pak se zkrátí na `C0`. Šířka sloupců `CALLPREFIX` a `PHONE` je větší než jejich název, a proto se nezkrátí.

**Analýza ▼**

Ale moment. Jak to, že jsme použili operátor `<` na znakové pole? Kterýkoli z těchto operátorů lze použít na libovolný datový typ. Výsledek pak závisí na použitém datovém typu. Například v následujícím hledání podle kraje použijeme malé písmeno:

**Vstup/výstup ▼**

```
SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE REGION < 'z';
LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Lužný     Bob        602  111222  OL  79821
Matyáš    Bob        773  123456  BR
Strouhal  Fido       700  343434  ZL  76701
Přikryl   Tom        581  010101  OT  72000
Černý     Franta     777  000999  OL  79801
5 rows selected.
```

```
mysql> select * from friends where region < 'z';
```

```
+-----+-----+-----+-----+-----+-----+
| lastname | firstname | callprefix | phone | re | zip |
+-----+-----+-----+-----+-----+-----+
| Lužný    | Bob      |          602 | 111222 | OL | 79821 |
| Matyáš   | Bob      |          773 | 123456 | BR | NULL |
| Přikryl  | Tom      |          581 | 010101 | OT | 72000 |
| Černý    | Franta   |          777 | 000999 | OL | 79801 |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Velká písmena jsou obvykle seřazena před malými písmeny. Proto jsou vrácené kódy velkých písmen menší než písmeno z. Pro jistotu ještě nahlédněte do dokumentace pro svou implementaci.

**TIP**

Abyste měli jistotu, jak se tyto operátory budou chovat, zkontrolujte své jazykové tabulky. Většina implementací pro osobní počítače používá tabulky ASCII.

Pro začlenění Zlínského kraje do původního vyhledávání napište níže uvedený příkaz:

**Vstup/výstup ▼**

```
SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE REGION <= 'Z';
LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Lužný     Bob        602  111222  OL  79821
Matyáš    Bob        773  123456  BR
Strouhal  Fido       700  343434  ZL  76701
Přikryl   Tom        581  010101  OT  72000
Černý     Franta     777  000999  OL  79801
5 rows selected.
```

```
mysql> select * from friends where region <= 'Z';
```

```

+-----+-----+-----+-----+-----+-----+
| lastname | firstname | callprefix | phone | re | zip |
+-----+-----+-----+-----+-----+-----+
| Lužný | Bob | 602 | 111222 | OL | 79821 |
| Matyáš | Bob | 773 | 123456 | BR | NULL |
| Strouhal | Fido | 700 | 343434 | ZL | 76701 |
| Přikryl | Tom | 581 | 010101 | OT | 72000 |
| Černý | Franta | 777 | 000999 | OL | 79801 |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

```

### Nerovnost (<> nebo !=)

V okamžiku, kdy potřebujete vyhledat vše kromě určitých dat, použijte symbol nerovnosti, který může mít v závislosti na implementaci jazyka SQL tvar <> nebo !=. Kupříkladu k vyhledání všech, kteří se nejmenují Bob, použijte níže uvedený příkaz:

### Vstup/výstup ▼

```

SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE FIRSTNAME <> 'Bob';
LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Strouhal  Fido       700         343434  ZL  76701
Přikryl   Tom        581         010101  OT  72000
Černý     Franta     777         000999  OL  79801
3 rows selected.

```

```

mysql> select * from friends where firstname <> 'Bob';
+-----+-----+-----+-----+-----+-----+
| lastname | firstname | callprefix | phone | re | zip |
+-----+-----+-----+-----+-----+-----+
| Strouhal | Fido | 700 | 343434 | ZL | 76701 |
| Přikryl | Tom | 581 | 010101 | OT | 72000 |
| Černý | Franta | 777 | 000999 | OL | 79801 |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

K vyhledání všech, kteří nežijí v Pardubickém kraji, můžete použít následující příkaz:

### Vstup/výstup ▼

```

SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE REGION != 'PA';
LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Lužný     Bob        602         111222  OL  79821
Matyáš    Bob        773         123456  BR
Strouhal  Fido       700         343434  ZL  76701
Přikryl   Tom        581         010101  OT  72000
Černý     Franta     777         000999  OL  79801
5 rows selected.

```

```
mysql> select * from friends where region != 'PA';
+-----+-----+-----+-----+-----+-----+
| lastname | firstname | callprefix | phone | re | zip |
+-----+-----+-----+-----+-----+-----+
| Lužný | Bob | 602 | 111222 | OL | 79821 |
| Matyáš | Bob | 773 | 123456 | BR | NULL |
| Strouhal | Fido | 700 | 343434 | ZL | 76701 |
| Přikryl | Tom | 581 | 010101 | OT | 72000 |
| Černý | Franta | 777 | 000999 | OL | 79801 |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

**POZNÁMKA**

Je třeba poznamenat, že v obou uváděných implementacích lze „nerovnost“ vyjádřit pomocí kteréhokoli ze symbolů <> a !=.

3

## Znakové operátory

Znakové operátory lze použít pro manipulaci s reprezentací znakových řetězců, a to jak ve výstupu dat, tak i v procesu kladení podmínek na získávaná data. V této části si popíšeme dva znakové operátory: operátor LIKE a operátor ||, který představuje zřetězení.

### Operátor LIKE

Představte si, že potřebujete vybrat část databáze, která odpovídá nějakému vzoru, ale přitom nemusí jít o naprosto přesnou shodu. V takovém případě byste mohli použít znaménko rovnosti a vypsat všechny možné varianty. To by však bylo pracné a časově náročné. Mnohem lepší je sáhnout po operátoru LIKE. Podívejte se na následující příklad:

### Vstup/výstup ▼

```
SQL> SELECT * FROM PARTS;
NAME          LOCATION      PARTNUMBER
-----
Slepé střevo  Střed Břicho  1
Ohryzek       Hrdlo         2
Srdce         Hruď         3
Páteř         Záda         4
Kovadlinka    Ucho         5
Ledvina       Střed Záda    6
6 rows selected.
```

Jak vyhledáte všechny části umístěné na zádech? Rychlý vizuální průzkum této jednoduché tabulky odhalí, že se jedná o dvě části, jenže každá má trošku jiný název. Zkuste následující příkaz:

### Vstup/výstup ▼

```
SQL> SELECT *
2 FROM PARTS
3 WHERE LOCATION LIKE '%Záda%';
```

NAME	LOCATION	PARTNUMBER
Páteř	Záda	4
Ledvina	Střed Záda	6

2 rows selected.

## Analýza ▼

Všimněte si znaku procento (%) za operátorem LIKE. Uvnitř operátoru LIKE představuje zástupný symbol pro libovolný počet libovolných znaků. V tomto případě se ptáme na libovolný výskyt znaků Záda ve sloupci LOCATION. Dotaz upravíme takto:

## Vstup ▼

```
SQL> SELECT *
      2 FROM PARTS
      3 WHERE LOCATION LIKE 'Záda%';
```

Nyní obdržíme všechny záznamy, jejichž pole LOCATION začíná znaky Záda:

## Vstup/výstup ▼

NAME	LOCATION	PARTNUMBER
Páteř	Záda	4

1 rows selected.

```
mysql> select * from parts where location like 'Záda%';
+-----+-----+-----+
| name | location | partnumber |
+-----+-----+-----+
| Páteř | Záda | 4 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Pokud použijeme následující příkaz:

## Vstup ▼

```
SQL> SELECT *
      2 FROM PARTS
      3 WHERE NAME LIKE 'S%';
```

pak obdržíme všechny záznamy, jejichž název začíná písmenem S:

## Výstup ▼

NAME	LOCATION	PARTNUMBER
Slepé střevo	Střed Břicho	1
Srdce	Hrud'	3

2 rows selected.

Rozlišuje operátor `LIKE` velikost písmen v obou databázových systémech Oracle a MySQL? To hned zjistíme pomocí následujícího dotazu:

### Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM PARTS
      3 WHERE NAME LIKE 's%';
```

no rows selected

```
mysql> select * from parts where name like 's%';
```

```
+-----+-----+-----+
| name          | location      | partnumber |
+-----+-----+-----+
| Slepé střevo  | Střed Břicho |           1 |
| Srdce         | Hrud'        |           3 |
+-----+-----+-----+
```

2 rows in set (0.00 sec)

Odpověď je ano pro databázový systém Oracle a ne pro databázový systém MySQL. Odkazy na data jsou závislé na implementaci, s níž pracujete.

Představte si, že potřebujete vyhledat data, která v určitém vzoru odpovídají všem znakům kromě jednoho? V takovém případě byste mohli použít jiný typ zástupného symbolu: podtržítka.

### Podtržítka ( \_ )

Podtržítka je zástupný symbol představující jeden libovolný znak. Nyní použijeme upravenou verzi tabulky `FRIENDS`:

### Vstup/výstup ▼

```
SQL> SELECT * FROM FRIENDS;
```

```
LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Lužný     Bob        602 111222  OL  79821
Matyáš    Bob        773 123456  BR
Strouhal  Fido       700 343434  ZL  76701
Přikryl   Tom        581 010101  OT  72000
Černý     Franta     777 000999  OL  79801
Zavádil   Olda       911 111311  OL  79604
Berka     Standa     604 111234  ZL  72801
```

7 rows selected.

K vyhledání všech záznamů, jejichž pole `CALLPREFIX` začíná znaky 60, napište následující příkaz:

### Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE CALLPREFIX LIKE '60_';
```

```

LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Lužný     Bob        602  111222  OL  79821
Berka     Standa    604  111234  OT  72801
2 rows selected.

```

```
mysql> select * from friends where callprefix like '60_';
```

```

+-----+-----+-----+-----+-----+-----+
| lastname | firstname | callprefix | phone  | re  | zip  |
+-----+-----+-----+-----+-----+-----+
| Lužný    | Bob      | 602       | 111222 | OL  | 79821 |
| Berka    | Standa   | 604       | 111234 | OT  | 72801 |
+-----+-----+-----+-----+-----+-----+
2 row in set (0.00 sec)

```

V jednom příkazu můžete použít i více podtržíték:

### Vstup/výstup ▼

```

SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE PHONE LIKE '111___';
LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Lužný     Bob        602  111222  OL  79821
Zavadił   Ołda      911  111311  OL  79604
Berka     Standa    604  111234  OT  72801
3 rows selected.

```

Předchozí příkaz bychom mohli zapsat také takto:

### Vstup/výstup ▼

```

SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE PHONE LIKE '111%';
LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Lužný     Bob        602  111222  OL  79821
Zavadił   Ołda      911  111311  OL  79604
Berka     Standa    604  111234  OT  72801
3 rows selected.

```

Všimněte si, že výsledek je naprosto stejný. Tyto dva zástupné symboly lze libovolně kombinovat. V následujícím příkladu hledáme všechny přátele, jejichž příjmení má jako druhý znak písmeno a:

### Vstup/výstup ▼

```

SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE LASTNAME LIKE '_a%';
LASTNAME  FIRSTNAME  CALLPREFIX  PHONE  RE  ZIP
-----
Matyáš    Bob        773  123456  BR

```

```
Zavadil Olda                911 111311 0L 79604
2 rows selected.
```

## Zřetězení (||)

Operátor || (dvě svislé čáry) provádí zřetězení dvou řetězců. Vyzkoušejte tento příkaz:

### Vstup/výstup ▼

```
SQL> SELECT FIRSTNAME || LASTNAME ENTIRENAME
       2 FROM FRIENDS;
ENTIRENAME
-----
Bob Lužný
Bob Matyáš
Fido Strouhal
Tom Přikryl
Franta Černý
Olda Zavadil
Standa Berka
7 rows selected.
```

### Analýza ▼

Všimněte si, že místo operátoru + se používá operátor ||. Pokud byste pro zřetězení řetězců použili operátor +, pak by interpret jazyka SQL používaný pro tento příklad (Oracle) vrátil následující chybu:

### Vstup/výstup ▼

```
SQL> SELECT FIRSTNAME + LASTNAME ENTIRENAME
       2 FROM FRIENDS;

ERROR:
ORA-01722: invalid number
```

Uvedený příkaz hledá dvě čísla, která by sečetl. Žádná nenajde, a proto vyvolá chybu způsobenou neplatným číslem.

#### POZNÁMKA

Některé implementace jazyka SQL (např. SQL Server společnosti Microsoft) používají k spojování řetězců znaménko plus. Zkontrolujte tedy dokumentaci ke své implementaci.

#### POZNÁMKA

Databázový systém MySQL lze nastavit tak, aby umožňoval řetězení pomocí operátoru ||. Nejedná se však o jeho výchozí nastavení. Ve výchozím stavu se k tomuto účelu používá funkce `concat()`, která přijímá libovolný počet proměnných. Tato funkce se používá docela snadno. Pokud byste se rozhodli změnit parametry databázového systému MySQL tak, aby umožňoval řetězení pomocí operátoru ||, pak si nejdříve toto téma prostudujte v doprovodné dokumentaci.



## Vstup/výstup ▼

```
mysql> select concat(firstname," ",lastname) Entirename from friends;
+-----+
| Entirename |
+-----+
| Bob Lužný   |
| Fido Strouhal |
| Tom Přikryl |
| Franta Černý |
| Bob Matyáš  |
| Olda Zavadil |
| Standa Berka |
+-----+
7 rows in set (0.00 sec)
```

Zde je praktičtější příklad řetězení:

## Vstup/výstup ▼

```
SQL> SELECT LASTNAME || ',' || FIRSTNAME NAME
      2 FROM FRIENDS;
NAME
-----
Lužný      , Bob
Matyáš     , Bob
Strouhal   , Fido
Přikryl    , Tom
Černý     , Franta
Zavadil    , Olda
Berka      , Standa
7 rows selected.
```

```
mysql> select concat(lastname,""," ",firstname) Name from friends;
+-----+
| Name |
+-----+
| Lužný, Bob |
| Matyáš, Bob |
| Strouhal, Fido |
| Přikryl, Tom |
| Černý, Franta |
| Zavadil, Olda |
| Berka, Standa |
+-----+
7 rows in set (0.00 sec)
```

Příkaz pro databázový systém Oracle vkládá čárku mezi příjmení a křestní jméno. To je dáno tím, že při zřetězení s jiným řetězcem pracuje databázový systém Oracle (stejně jako i jiné implementace jazyka SQL) s úplnou délkou, jakou může daný sloupec mít. Tím se mezi hodnotami sloupců či řetězců vytváří přirozené rozestupy. V příkazu pro databázový systém MySQL vkládáme mezi dva sloupce čárku a mezeru. Databázový systém MySQL automaticky

převádí hodnoty sloupců či řetězců na jedničku, takže se jakékoliv „přirozené“ rozestupy mezi hodnotami ztratí.

### POZNÁMKA

Více k problémům s mezerami: Všimněte si mezer navíc mezi křestním jménem a příjmením v příkladech pro databázový systém Oracle. Tyto mezery jsou ve skutečnosti součástí dat. U určitých typů dat jsou na pravou stranu hodnot menších než celková délka přidělená danému poli doplňovány mezery (podívejte se do své implementace). Datové typy budeme probírat v lekcí 9. Kromě toho, pokud se pokusíte zřetěžit hodnotu NULL s řetězcem, bude mít výsledek celého výrazu hodnotu NULL. V takových případech je pravděpodobně vhodnější použít vestavěnou funkci k odstranění hodnot NULL. Tomuto tématu se budeme věnovat v lekcí 7.

Dosud jste v jednom příkazu prováděli pouze jedno porovnávání. V některých případech to stačí, ale představte si, že potřebujete vyhledat všechny osoby v práci s příjmením začínajícím písmenem P, kterým zbývají méně než tři dny dovolené. V takovém případě nám pomohou logické operátory.

## Logické operátory

Logické operátory oddělují dvě či více podmínek v klauzuli `WHERE` uvnitř příkazu jazyka SQL. Čas dovolených je na celém světě vždy žhavým tématem. Předpokládejme, že máme k dispozici následující tabulku `VACATION` (dovolené) pro účetní oddělení:

### Vstup/výstup ▼

```
SQL> SELECT * FROM VACATION;
```

LASTNAME	EMPLOYEEENUM	YEARS	LEAVETAKEN
Huňka	101	2	4
Bednář	104	5	23
Hruška	107	8	45
Zilvar	233	4	80
Hrbek	210	15	100
Ovčáček	211	10	78

6 rows selected.

Předpokládejme, že naše společnost dává každému zaměstnanci každý rok 12 dní volna. Z toho, co jsme se naučili, a pomocí logického operátoru nyní vyhledáme všechny zaměstnance se jménem začínajícím na písmeno H, kterým zůstává více jak 50 dnů volna.

### Vstup/výstup ▼

```
SQL> SELECT LASTNAME,
2 YEARS * 12 - LEAVETAKEN REMAINING
3 FROM VACATION
4 WHERE LASTNAME LIKE 'H%'
5 AND
6 YEARS * 12 - LEAVETAKEN > 50;
```

```

LASTNAME REMAINING
-----
Hruška          51
Hrbek           80
2 rows selected.

```

```

mysql> select lastname,
      -> years*12 - leavetaken remaining
      -> from vacation
      -> where lastname like 'H%'
      -> and years*12 - leavetaken > 50;

```

```

+-----+-----+
| lastname | remaining |
+-----+-----+
| Hruška   |         51 |
| Hrbek    |         80 |
+-----+-----+
2 rows in set (0.00 sec)

```

## Analýza ▼

Tento dotaz je tím nekomplikovanějším, který jste v této knize dosud provedli. V klauzuli SELECT (řádky 1 a 2) používáme aritmetické operátory pro stanovení počtu zbývajících dní dovolené každého zaměstnance. Uvážíme-li běžnou precedenci, pak můžeme klidně psát `YEARS * 12 - LEAVETAKEN` (srozumitelnější by asi bylo `(YEARS * 12) - LEAVETAKEN`).

Na řádku používáme operátor LIKE se zástupným symbolem % k vyhledání všech jmen začínajícím písmenem H. Na řádku hledáme pomocí operátoru > všechny výskyty větší než 50.

Na řádku 5 se vyskytuje nový prvek. Pomocí logického operátoru AND jsme zajistili, aby se vyhledaly pouze ty záznamy, které splňují obě kritéria uvedená na řádcích 4 a 5.

## Operátor AND (logický součin)

Operátor AND vyžaduje, aby se výrazy na obou stranách vyhodnotily na TRUE. Pokud se jeden z nich vyhodnotí na FALSE, pak vrátí FALSE. Například pro zjištění, kteří zaměstnanci jsou v společnosti 5 a méně let a vybrali si již více než 20 dnů volna, můžeme použít následující příkaz:

## Vstup/výstup ▼

```

SQL> SELECT LASTNAME
      2 FROM VACATION
      3 WHERE YEARS <= 5
      4 AND
      5 LEAVETAKEN > 20 ;
LASTNAME
-----
Bednář
Zilvar
2 rows selected.

```

```

mysql> select lastname from vacation

```

```

-> where years <= 5
-> and leavetaken > 20;
+-----+
| lastname |
+-----+
| Bednář   |
| Zilvar   |
+-----+
2 rows in set (0.00 sec)

```

Pomocí níže uvedeného příkazu zjistíme, kteří zaměstnanci jsou ve společnosti již 5 a více let a vybrali si méně než 50 procent svého volna:

### Vstup/výstup ▼

```

SQL> SELECT LASTNAME WORKAHOLICS
      2 FROM VACATION
      3 WHERE YEARS >= 5
      4 AND
      5 ((YEARS *12)-LEAVETAKEN)/(YEARS * 12) < 0.50;
WORKAHOLICS
-----
Hrbek
Ovčáček

2 rows selected.

```

```

mysql> select lastname Workaholics
      -> from vacation
      -> where years >= 5
      -> and ((years * 12) - leavetaken) / (years * 12) < 0.50;
+-----+
| Workaholics |
+-----+
| Hrbek       |
| Ovčáček     |
+-----+
2 rows in set (0.00 sec)

```

U těchto lidí byste si měli dát pozor na syndrom vyhoření

Velmi pozorně si projděte, jakým způsobem jsme pomocí operátoru AND zkombinovali tyto dvě podmínky dohromady.

### Operátor OR (logický součet)

Pro shrnutí více podmínek můžete též použít operátor OR. Má-li kterákoli z takovýchto podmínek hodnotu TRUE, pak operátor OR vrátí hodnotu TRUE. Pro ilustraci rozdílů nahradíme v posledním dotazu operátor AND operátorem OR:

### Vstup/výstup ▼

```

SQL> SELECT LASTNAME WORKAHOLICS
      2 FROM VACATION

```

```

3 WHERE YEARS >= 5
4 OR
5 ((YEARS *12)-LEAVETAKEN)/(YEARS * 12) < 0.50;

```

WORKAHOLICS

-----

Bednář  
Hruška  
Hrbek  
Ovčáček

4 rows selected.

```

mysql> select lastname
-> from vacation
-> where years >= 5
-> OR ((years*12)-leavetaken)/(years*12) < 0.50;

```

+-----+

lastname
Bednář
Hruška
Hrbek
Ovčáček

+-----+

4 rows in set (0.00 sec)

Původní jména jsou i nadále v seznamu, kromě nich tu však máme další tři záznamy (kdo by nám ale zablýžl, že jej nazýváme workoholikem). Tato tři nová jména se dostala do seznamu proto, že splňují jednu z podmínek. Operátor OR totiž pro vrácení dat vyžaduje, aby byla splněna pouze jedna z podmínek.

## Operátor NOT (logická negace)

Operátor NOT znamená logickou negaci. Pokud se podmínka, na níž se vztahuje, vyhodnotí na TRUE, pak z ní operátor NOT udělá FALSE. Pokud má podmínka za operátorem NOT hodnotu FALSE, pak se z ní stane TRUE. Kupříkladu následující příkaz SELECT vrací jména, která v tabulce nezačínají písmenem H:

## Vstup/výstup ▼

```

SQL> SELECT *
2 FROM VACATION
3 WHERE LASTNAME NOT LIKE 'H%';

```

LASTNAME	EMPLOYEEENUM	YEARS	LEAVETAKEN
Bednář	104	5	23
Zilvar	233	4	80
Ovčáček	211	10	78

-----

Bednář 104 5 23

Zilvar 233 4 80

Ovčáček 211 10 78

3 rows selected.

```
mysql> select * from vacation
-> where lastname not like 'H%';
+-----+-----+-----+-----+
| lastname | employeenum | years | leavetaken |
+-----+-----+-----+-----+
| Bednář   |          104 |     4 |          23 |
| Zilvar   |          233 |     5 |          80 |
| Ovčáček  |          211 |    10 |          78 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Operátor NOT lze při aplikaci na hodnotu NULL použít také s operátorem IS. Vzpomeňte si na tabulku PRICE, kde jsme u položky Pomeranče umístili hodnotu NULL do sloupce WHOLESale.

### Vstup/výstup ▼

```
SQL> SELECT * FROM PRICE;
ITEM      WHOLESale
-----
Rajčata   34
Brambory  51
Banány    67
Tuřiny    45
Sýr       89
Jablka    23
Pomeranče
7 rows selected.
```

K vyhledání všech prvků, jejichž pole WHOLESale neobsahuje hodnotu NULL, stačí napsat následující dotaz:

### Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM PRICE
      3 WHERE WHOLESale IS NOT NULL;

ITEM      WHOLESale
-----
Rajčata   34
Brambory  51
Banány    67
Tuřiny    45
Sýr       89
Jablka    23
6 rows selected.
```

## Množinové operátory

V lekci 1 jste se dozvěděli, že jazyk SQL je založen na teorii množin. V následující části se proto zaměříme na množinové operátory. Množinové operátory se používají ke kombinování odlišných množin dat vrácených různými dotazy do jediného dotazu, a tedy do jediné datové

sady. Jazyk SQL nabízí rozličné množinové operátory, které umožňují kombinovat nejrůznější datové sady podle aktuálních potřeb na zpracování dat.

### Operátory UNION (sjednocení) a UNION ALL

Operátor UNION vrací výsledky dvou dotazů bez duplicitních řádků. Následující dvě tabulky reprezentují soupis členů dvou týmů:

#### Vstup/výstup ▼

---

```
SQL> SELECT * FROM FOOTBALL;
```

```
NAME
-----
Huňka
Bureš
Cásek
Dokoupil
Edler
Frgál
Grumlich
7 rows selected.
```

```
SQL> SELECT * FROM SOFTBALL;
```

```
NAME
-----
Huňka
Bednář
Cásek
Daněk
Edler
Fojt
Grumlich
7 rows selected.
```

Kolik různých osob hraje v obou týmech?

#### Vstup/výstup ▼

---

```
SQL> SELECT NAME FROM SOFTBALL
  2 UNION
  3 SELECT NAME FROM FOOTBALL;
```

```
NAME
-----
Huňka
Bednář
Bureš
Cásek
Daněk
Dokoupil
Edler
```

```
Fojt
Frgál
Grumlich
10 rows selected.
```

Operátor `UNION` vrátil ze dvou seznamů 10 odlišných jmen. Kolik jmen je na obou seznamech (včetně duplicit)?

### Vstup/výstup ▼

```
SQL> SELECT NAME FROM SOFTBALL
      2 UNION ALL
      3 SELECT NAME FROM FOOTBALL;
```

```
NAME
-----
Huňka
Bednář
Cásek
Daněk
Edler
Fojt
Grumlich
Huňka
Bureš
Cásek
Dokoupil
Edler
Frgál
Grumlich
14 rows selected.
```

### Analýza ▼

Spojený seznam (vznikl díky operátoru `UNION ALL`) obsahuje 14 jmen. Operátor `UNION ALL` funguje stejně jako operátor `UNION` až na to, že neodstraňuje duplicity. Pamatujte si, že operátory `UNION` a `UNION ALL` budou fungovat jen tehdy, mají-li všechny použité příkazy `SELECT` stejné sloupce. V opačném případě vrátí chybovou zprávu. Nyní chceme zobrazit seznam hráčů, kteří jsou současně v obou týmech. Zde nám již operátor `UNION` nepomůže, a musíme proto sáhnout po operátoru `INTERSECT`.

### Operátor `INTERSECT` (průnik)

Operátor `INTERSECT` vrátí pouze ty řádky, které se nacházejí v obou dotazech. Následující příkaz `SELECT` zobrazí seznam hráčů, kteří hrají v obou týmech:

### Vstup/výstup ▼

```
SQL> SELECT * FROM FOOTBALL
      2 INTERSECT
      3 SELECT * FROM SOFTBALL;
```



```
NAME
-----
Huňka
Cásek
Edler
Grumlich
```

```
4 rows selected.
```

V tomto příkladu vyhledá operátor `INTERSECT` seznam těch hráčů, kteří hrají za oba týmy, což provede zkombinováním výsledků dvou příkazů `SELECT`. Na operátor `INTERSECT` jsou kladena stejná omezení jako na operátory `UNION` a `UNION ALL`, což znamená, že příslušné příkazy `SELECT` musejí obsahovat stejné sloupce.

### Operátor MINUS (množinový rozdíl)

Operátor `MINUS` vrací řádky z prvního dotazu, které se nevyskytují ve druhém:

#### Vstup/výstup ▼

---

```
SQL> SELECT * FROM FOOTBALL
      2 MINUS
      3 SELECT * FROM SOFTBALL;
```

```
NAME
-----
Bureš
Dokoupil
Frgál
```

```
3 rows selected.
```

Tento dotaz zobrazil tři fotbalové hráče, kteří nejsou členy softbalového týmu. Pokud převrátíme pořadí příkazů `SELECT`, pak obdržíme tři softbalové hráče, kteří nepatří do fotbalového týmu:

#### Vstup/výstup ▼

---

```
SQL> SELECT * FROM SOFTBALL
      2 MINUS
      3 SELECT * FROM FOOTBALL;
```

```
NAME
-----
Bednář
Daněk
Fojt
```

```
3 rows selected.
```

## Ostatní operátory: IN a BETWEEN

Operátory IN a BETWEEN poskytují zkratky pro funkce, se kterými již umíte pracovat. Chcete-li najít přátele z Brněnského, Olomouckého a Ostravského regionu, pak můžete použít následující dotaz:

### Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE REGION = 'OL'
      4 OR
      5 REGION = 'BR'
      6 OR
      7 REGION = 'OT';
```

LASTNAME	FIRSTNAME	CALLPREFIX	PHONE	RE	ZIP
Lužný	Bob	602	111222	OL	79821
Matyáš	Bob	773	123456	BR	
Příkryl	Tom	581	010101	OT	72000
Černý	Franta	777	000999	OL	79801
Zavadił	Olda	911	111311	OL	79604

5 rows selected.

Další možnost demonstruje níže uvedený dotaz:

### Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE REGION IN('OL','BR','OT');
```

LASTNAME	FIRSTNAME	CALLPREFIX	PHONE	RE	ZIP
Lužný	Bob	602	111222	OL	79821
Matyáš	Bob	773	123456	BR	
Příkryl	Tom	581	010101	OT	72000
Černý	Franta	777	000999	OL	79801
Zavadił	Olda	911	111311	OL	79604

5 rows selected.

```
mysql> select * from friends
      -> where region in ('OL','BR','OT');
```

lastname	firstname	callprefix	phone	re	zip
Lužný	Bob	602	111222	OL	79821
Matyáš	Bob	773	123456	BR	NULL
Příkryl	Tom	581	010101	OT	72000
Černý	Franta	777	000999	OL	79801
Zavadił	Olda	911	111311	OL	79604

5 rows in set (0.20 sec)

Druhý příklad je ve srovnání s prvním kratší a čitelnější. Nikdy nevíte, kdy bude potřeba vrátit se zpět a pracovat na něčem, co jste napsali před několika měsíci. Operátor `IN` pracuje také s čísly. Podívejte se na následující příklad, v němž je sloupec `CALLPREFIX` číselného typu:

### Vstup/výstup ▼

---

```
SQL> SELECT *
      2 FROM FRIENDS
      3 WHERE CALLPREFIX IN(602,581,911);
```

LASTNAME	FIRSTNAME	CALLPREFIX	PHO	NE	RE	ZIP
Lužný	Bob	602	111222	OL		79821
Přikryl	Tom	581	010101	OT		72000
Zavadil	Olga	911	111311	OL		79604

3 rows selected.

Potřebujete-li nějaký rozsah dat z tabulky `PRICE`, pak můžete napsat následující dotaz:

### Vstup/výstup ▼

---

```
SQL> SELECT *
      2 FROM PRICE
      3 WHERE WHOLESALE > 25
      4 AND
      5 WHOLESALE < 75;
```

ITEM	WHOLESALE
Rajčata	34
Brambory	51
Banány	67
Tuřiny	45

4 rows selected.

Další možností je použít operátor `BETWEEN`:

### Vstup/výstup ▼

---

```
SQL> SELECT *
      2 FROM PRICE
      3 WHERE WHOLESALE BETWEEN 25 AND 75;
```

ITEM	WHOLESALE
Rajčata	34
Brambory	51
Banány	67
Tuřiny	45

4 rows selected.

```
mysql> select * from price
      -> where wholesale between 25 and 75;
```

```

+-----+-----+
| item   | wholesale |
+-----+-----+
| Rajčata | 34.00    |
| Brambory | 51.00    |
| Banány  | 67.00    |
| Tuřiny  | 45.00    |
+-----+-----+
4 rows in set (0.08 sec)

```

A opět: druhý příklad nabízí ve srovnání s prvním čistší a čitelnější řešení.

#### POZNÁMKA

Pokud by v poli WHOLESALÉ v tabulce PRICE existovala hodnota 25, pak bychom obdrželi také tento záznam. Parametry operátoru BETWEEN se do výsledku též zahrnují.

3

## Shrnutí

Na začátku této lekce jste uměli používat klauzule `SELECT` a `FROM`. Nyní víte, jak používat hromadu operátorů, díky nimž si můžete své požadavky na databázi pěkně doladit. Naučili jste si, jak používat aritmetické, porovnávací, znakové, logické a množinové operátory. Tato výkonná sada nástrojů je základním kamenem vašich vědomostí v oblasti jazyka SQL. V lekci 4 se dozvíte, jak při dolování dat zvýšit sílu dotazů jazyka SQL integrováním dalších klauzulí, jako je klauzule `WHERE`, které budou ve vašich dotazech provádět operace související se seskupováním a uspořádáváním.

## Otázky a odpovědi

**Otázka:** Jak se mnou všechny tyto informace souvisejí v případě, že nepoužívám SQL na příkazovém řádku jako v příkladech?

**Odpověď:** Ať už používáte SQL v jazyku COBOL ve formě vloženého kódu nebo v knihovně ODBC společnosti Microsoft, stále pracujete se stejnými základními konstrukcemi. Své znalosti z těchto prvních lekcí budete při práci s jazykem SQL využívat neustále.

**Otázka:** Proč mám pořád kontrolovat svou implementaci? Myslel jsem si, že existuje nějaký standard!

**Odpověď:** Existuje sice standard ANSI (poslední verze byla vydána ke konci roku 2008), nicméně většina dodavatelů si jej modifikuje tak, aby vyhovoval jejich databázím. Základy jsou podobné, ne-li stejné, přičemž každá instance nabízí rozšíření, které ostatní dodavatelé kopírují a vylepšují. Jakožto výchozí bod jsme zvolili standard ANSI a na případné rozdíly budeme upozorňovat v průběhu výkladu.

## Úkoly pro vás

Tato část nabízí kvízové otázky, které vám pomohou s upevněním získaných znalostí, a dále cvičení, jež vám poskytnou praktické zkušenosti s používáním osvojené látky. Pokuste se před nahlédnutím na odpovědi v příloze A odpovědět na otázky v kvízu a ve cvičení.

Zde jsou příkazy `CREATE TABLE` a `INSERT` pro tabulky `FRIENDS` a `PRICE`. Opište následující kód do svého databázového systému MySQL, pokud jste tak již neučinili.

```
create table friends
(lastname  varchar(15) not null,
 firstname varchar(15) not null,
 callprefix number(9)  null,
 phone    varchar(10) null,
 region   char(2)     not null,
 zip      varchar(5)  null);

insert into friends values
('Lužný', 'Bob', '602', '111222', '0L', '79821');

insert into friends values
('Matyáš', 'Bob', '773', '123456', 'BR', NULL);

insert into friends values
('Strouhal', 'Fido', '300', '343434', 'ZL', '76701');

insert into friends values
('Příkryl', 'Tom', '581', '010101', 'PL', '72000');

insert into friends values
('Černý', 'Franta', '777', '000999', '0L', '79801');

insert into friends values
('Zavdil', 'Olda', '911', '111311', '0L', '79604');

insert into friends values
('Berka', 'Standa', '604', '111234', 'ZL', '72801');

create table price
(item      varchar(15) not null,
 wholesale decimal(4,2) not null);

insert into price values
('Rajčata', '34');

insert into price values
('Brambory', '51');

insert into price values
('Banány', '67');

insert into price values
('Tuříný', '45');
```

```
insert into price values
('Syr', '89');
```

```
insert into price values
('Jablka', '23');
```

## Kvíz

K vyřešení následujících otázek použijte tabulku FRIENDS.

LASTNAME	FIRSTNAME	CALLPREFIX	PHONE	RE	ZIP
Lužný	Bob	602	111222	OL	79821
Matyáš	Bob	773	123456	BR	
Strouhal	Fido	700	343434	ZL	76701
Přikryl	Tom	581	010101	OT	72000
Černý	Franta	777	000999	OL	79801
Zavadil	Olda	911	111311	OL	79604
Berka	Standa	604	111234	ZL	72801

- Napište dotaz, který vrátí každého v databázi, jehož příjmení končí písmenem a.
- Napište dotaz, který vrátí každého, kdo žije v Olomouckém kraji a jmenuje se Bob.
- Máme dvě tabulky (PART1 a PART2) obsahující sloupce s názvem PARTNO (číslo součástky). Jak zjistíte, která čísla součástek jsou v obou tabulkách? Napište dotaz.
- Jakou zkratku použijete místo výrazu WHERE a >= 10 AND a <= 30?
- Jaký výsledek vrátí následující dotaz?

```
SELECT FIRSTNAME
FROM FRIENDS
WHERE FIRSTNAME = 'Bob'
AND LASTNAME = 'Černý';
```
- Jaký je hlavní rozdíl ve výsledné sadě při použití operátoru UNION versus UNION ALL?
- Jaký je hlavní rozdíl v použití operátů INTERSECT a MINUS?

## Cvičení

- S použitím tabulky FRIENDS napište dotaz, který vrátí následující výsledek:

```
NAME REGION
-----
Bob Kraj OL
```

- S použitím tabulky FRIENDS napište dotaz, který vrátí následující výsledek:

```
NAME PHONE
-----
Matyáš, Bob 773 123456
Strouhal, Fido 700 343434
Černý, Franta 777 000999
```

- Vyberte všechny řádky tabulky PRICE, jejichž sloupec WHOLESAL je větší než 50.

4. Jaký výsledek obdržíte z následujícího dotazu?

```
mysql> select *  
-> from price  
-> where item like 'B%y';
```

5. Podporuje databázový systém MySQL množinové operátory UNION, UNION ALL, INTERSECT a MINUS?
6. Co je v následujícím dotazu špatně?

```
SELECT FIRSTNAME, LASTNAME FROM FRIENDS_1  
UNION  
SELECT FIRSTNAME FROM FRIENDS_2;
```

# LEKCE 4

## Klauzule v dotazech jazyka SQL

Tématem této lekce jsou klauzule. Klauzule jsou části příkazu jazyka SQL, díky nimž máte k dispozici jemnou kontrolu nad výsledkem dotazu. Na konci této lekce budete schopni používat následující klauzule:

- WHERE,
- ORDER BY,
- GROUP BY,
- HAVING.

Pro získání přehledu, kde se tyto funkce používají, poslouží obecná syntaxe příkazu SELECT:

### Syntaxe ▼

---

```
SELECT [DISTINCT | ALL] { *
      | { [schema.]{table | view | snapshot}.*
      | expr } [ [AS] c_alias ]
      [, { [schema.]{table | view | snapshot}.*
      | expr } [ [AS] c_alias ] ] ... }
FROM [schema.]{table | view | snapshot}[@dblink] [t_alias]
[, [schema.]{table | view | snapshot}[@dblink] [t_alias] ] ...
  [WHERE condition ]
  [GROUP BY expr [, expr] ... [HAVING condition] ]
  [{UNION | UNION ALL | INTERSECT | MINUS} SELECT command ]
  [ORDER BY {expr|position} [ASC | DESC]
  [, {expr|position} [ASC | DESC]] ...]
```

Totéž v databázovém systému MySQL:

### Syntaxe ▼

---

```
SELECT [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_
RESULT]
  [HIGH_PRIORITY]
  [DISTINCT | DISTINCTROW | ALL]
select_expression, ...
[INTO [OUTFILE | DUMPFILE] 'file_name' export_options]
[FROM table_references
  [WHERE where_definition]
  [GROUP BY {unsigned_integer | col_name | formula} [ASC | DESC],
...]
```



```
[HAVING where_definition]
[ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC] ,... ]
[LIMIT [offset,] rows]
[PROCEDURE procedure_name]
[FOR UPDATE | LOCK IN SHARE MODE]]
```

**POZNÁMKA**

Ze své praxe s jazykem SQL mohu říci, že standard ANSI je spíše jakýmsi „doporučením“ ANSI. Výše uvedená syntaxe ANSI SQL bude zpravidla fungovat s libovolným databázovým strojem SQL, můžete se však setkat s drobnými odchylkami. To je dáno tím, že ve srovnání se standardem ANSI jako celkem je jeho část, kterou musí daná implementace databáze podporovat, aby tento standard splňovala, velice malá.

Zatím jsme s komplikovaným syntaktickým diagramem nepracovali. Pro mnoho lidí není při osvojování něčeho nového příliš poučný, ale spíše tajemný, a proto se v této knize zaměříme raději na jednoduché příklady ilustrující konkrétní použití v praxi. Nyní se ovšem nacházíme v bodě, kde může syntaktický diagram pomoci spojit známé pojmy s tématem dnešní lekce.

S přesnou syntaxí si nedějte starosti, neboť v každé implementaci se stejně malinko liší. Raději se soustředte na vztahy. V horní části příkazu je klauzule `SELECT`, kterou jste používali mnohokrát v předchozích lekcích. Za ní následuje klauzule `FROM`, která by měla být v každém příkazu `SELECT` (o novém použití klauzule `FROM` se dozvíte v lekcí 6). Poté následují klauzule `WHERE`, `GROUP BY`, `HAVING` a `ORDER BY` (vnější klauzule diagramu – `UNION`, `UNION ALL`, `INTERSECT` a `MINUS` – jsme pobírali v lekcí 3). Každá klauzule hraje důležitou roli při výběru a manipulaci s daty.

## Specifikace kritérií pomocí klauzule `WHERE`

**POZNÁMKA**

Vzpomeňte si na lekcí 3, kde klauzule `WHERE` jednoduše činila vaše dotazy selektivnějším, že omezovala množství řádků vrácených ve výsledku.

S pouhými klauzulemi `SELECT` a `FROM` jste odkázáni jen na všechny řádky v tabulce. Pokud například použijete tato dvě klíčová slova na tabulku `CHECKS`, pak obdržíte všech sedm řádků:

### Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM CHECKS;
```

CHECK#	PAYEE	AMOUNT	REMARKS
1	Nákupní centrum	1500	Příště vzít syny
2	ČD	245	Vlak do Prahy
3	Nákupní centrum	2000	Mobilní telefon
4	Místní benzínka	980	Benzín

5 Diskont	1500 Nákup
6 Hospoda	2500 Divoká noc
7 Benzínka ve městě	250 Benzín

7 rows selected.

S klauzulí WHERE mohou být vaše dotazy selektivnější. Pro vyhledání všech šeků s hodnotou vyšší než 1000 korun můžete použít následující dotaz:

## Vstup ▼

```
SQL> SELECT *
      2 FROM CHECKS
      3 WHERE AMOUNT > 1000;
```

Klauzule WHERE vrátí čtyři instance v tabulce, které splňují zadanou podmínku:

## Výstup ▼

CHECK#	PAYEE	AMOUNT	REMARKS
1	Nákupní centrum	1500	Příště vzít syny
3	Nákupní centrum	2000	Mobilní telefon
5	Diskont	1500	Nákup
6	Hospoda	2500	Divoká noc

Klauzule WHERE dokáže též vyřešit oblíbené hádanky. S následující tabulkou jmen a lokací můžeme položit oblíbenou otázku: „Kde je Waldo?“

## Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM PUZZLE;
```

NAME	LOCATION
Dlaždič	Dvůr
Major	Kuchyně
Speedy	Obývací
Waldo	Garáž
Chlapec	Šatna
Arnold	Ložnice

6 rows selected.

```
SQL> SELECT LOCATION AS "Kde je Waldo?"
      2 FROM PUZZLE
      3 WHERE NAME = 'Waldo';
```

Kde je Waldo?

-----  
Garáž

Omlouváme se, ale tomuhle jsme prostě nemohli odolat. Slibujeme, že žádné sentimentální dotazy už nebudou. Nicméně na tomto dotazu je vidět, že sloupec použitý v podmínce klauzule WHERE nemusí být uveden v klauzuli SELECT. V tomto příkladu jsme vybrali sloupec LOCATION, ale v klauzuli WHERE jsme použili sloupec NAME, což je naprosto v pořádku. AS je volitelný operátor přiřazení, který přiřadí sloupci LOCATION alias „Kde je Waldo?“. S tímto operátorem se již možná nesetkáte, protože kromě psaní navíc nepřináší nic dalšího. Ve většině implementací můžete napsat následující:

### Vstup ▼

```
SQL> SELECT LOCATION "Kde je Waldo?"
      2 FROM PUZZLE
      3 WHERE NAME ='Waldo';
```

V tomto případě obdržíme stejný výsledek jako v předchozím dotazu, aniž bychom použili klíčové slovo AS:

### Výstup ▼

```
Kde je Waldo?
-----
Garáž
```

Po klauzulích SELECT a FROM je WHERE třetím nejpoužívanějším termínem jazyka SQL.

## Klauzule ORDER BY

Čas od času je nutné výsledky dotazu nějak seřadit. Jak již ale víte, z příkazu SELECT FROM obdržíte výpis v takovém pořadí, v jakém jste zadali řádky do tabulky (pokud jste ovšem nedefinovali primární klíč – viz lekce 15). Podívejte se na upravenou tabulku CHECKS:

### Vstup/výstup ▼

```
SQL> SELECT * FROM CHECKS;
```

CHECK#	PAYEE	AMOUNT	REMARKS
1	Nákupní centrum	1500	Příště vzít syny
2	ČD	245	Vlak do Prahy
3	Nákupní centrum	2000	Mobilní telefon
4	Místní benzínka	980	Benzín
5	Diskont	1500	Nákup
16	Hotovost	2500	Divoká noc
17	Benzínka ve městě	250	Benzín
9	Drogerie ABC	243	Superškrob
20	Drogerie ABC	105	Megačistič
8	Hotovost	600	Cesta do Prahy
21	Hotovost	340	Cesta do Ostravy

11 rows selected.

Totéž v databázovém systému MySQL:

## Vstup/výstup ▼

```
mysql> select * from checks;
```

check	payee	amount	remarks
1	Nákupní centrum	1500.00	Příště vzít syny
2	ČD	245.00	Vlak do Prahy
3	Nákupní centrum	2000.00	Mobilní telefon
4	Místní benzinka	980.00	Benzín
5	Diskont	1500.00	Nákup
16	Hotovost	2500.00	Divoká noc
17	Benzinka ve městě	250.00	Benzín
9	Drogerie ABC	243.00	Superškrob
20	Drogerie ABC	105.00	Megačistič
8	Hotovost	600.00	Cesta do Prahy
21	Hotovost	340.00	Cesta do Ostravy

11 rows in set (0.00 sec)

## Analýza ▼

Ve většině implementací je pořadí řádku ve výpisu stejné jako pořadí jejich zadání do tabulky. Nahlédněte do dokumentace k vaší implementaci, kde se dozvíte, zda nepoužívá nějaké jiné výchozí uspořádání. Až si přečtete lekci 11 a budete vědět, jak pomocí příkazu `INSERT` plnit tabulky daty, můžete si sami otestovat, jak se data řadí ve výchozím stavu.

Klauzule `ORDER BY` nabízí možnost seřadit výsledky. Kupříkladu k seřazení předchozího výpisu podle čísla šeku můžete použít následující klauzuli `ORDER BY`:

## Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM CHECKS
      3 ORDER BY CHECK#;
```

CHECK#	PAYEE	AMOUNT	REMARKS
1	Nákupní centrum	1500	Příště vzít syny
2	ČD	245	Vlak do Prahy
3	Nákupní centrum	2000	Mobilní telefon
4	Místní benzinka	980	Benzín
5	Diskont	1500	Nákup
8	Hotovost	600	Cesta do Prahy
9	Drogerie ABC	243	Superškrob
16	Hotovost	2500	Divoká noc
17	Benzinka ve městě	250	Benzín
20	Drogerie ABC	105	Megačistič
21	Hotovost	340	Cesta do Ostravy

11 rows selected.

Totéž v databázovém systému MySQL:

### Vstup/výstup ▼

```
mysql> select * from checks order by `check`;
```

check	payee	amount	remarks
1	Nákupní centrum	1500.00	Příště vzít syny
2	ČD	245.00	Vlak do Prahy
3	Nákupní centrum	2000.00	Mobilní telefon
4	Místní benzínka	980.00	Benzín
5	Diskont	1500.00	Nákup
8	Hotovost	600.00	Cesta do Prahy
9	Drogerie ABC	243.00	Superškrob
16	Hotovost	2500.00	Divoká noc
17	Benzínka ve městě	250.00	Benzín
20	Drogerie ABC	105.00	Megačistič
21	Hotovost	340.00	Cesta do Ostravy

11 rows in set (0.00 sec)

Nyní jsou data seřazena tak, jak chcete, a ne tak, jak jste je zadali do tabulky. Jak je patrné z následujícího příkladu, za klíčovým slovem `ORDER` musí být vždy klíčové slovo `BY`, které tedy není volitelné:

### Vstup/výstup ▼

```
SQL> SELECT * FROM CHECKS ORDER CHECK#;
```

```
SELECT * FROM CHECKS ORDER CHECK#
                                *
```

```
ERROR at line 1:
ORA-00924: missing BY keyword
```

Představte si, že potřebujete seznam dat v opačném pořadí s nejvyšším číslem či písmenem na prvním místě. Následující dotaz vygeneruje seznam seřazený od konce abecedy podle příjemců plateb:

### Vstup/výstup ▼

```
SQL> SELECT *
      2 FROM CHECKS
      3 ORDER BY PAYEE DESC;
```

CHECK#	PAYEE	AMOUNT	REMARKS
2	ČD	245	Vlak do Prahy
3	Nákupní centrum	2000	Mobilní telefon
1	Nákupní centrum	1500	Příště vzít syny
4	Místní benzínka	980	Benzín
16	Hotovost	2500	Divoká noc

8	Hotovost	600	Cesta do Prahy
21	Hotovost	340	Cesta do Ostravy
9	Drogerie ABC	243	Superškrob
20	Drogerie ABC	105	Megačistič
5	Diskont	1500	Nákup
17	Benzinka ve městě	250	Benzín

11 rows selected.

Totéž v databázovém systému MySQL:

### Vstup/výstup ▼

```
mysql> select * from checks order by payee desc;
+-----+-----+-----+-----+
| check | payee           | amount | remarks           |
+-----+-----+-----+-----+
| 2     | ČD              | 245.00 | Vlak do Prahy    |
| 1     | Nákupní centrum | 1500.00 | Příklad vzít syny |
| 3     | Nákupní centrum | 2000.00 | Mobilní telefon  |
| 4     | Místní benzinka | 980.00  | Benzín           |
| 16    | Hotovost        | 2500.00 | Divoká noc       |
| 8     | Hotovost        | 600.00  | Cesta do Prahy   |
| 21    | Hotovost        | 340.00  | Cesta do Ostravy |
| 9     | Drogerie ABC    | 243.00  | Superškrob       |
| 20    | Drogerie ABC    | 105.00  | Megačistič       |
| 5     | Diskont         | 1500.00 | Nákup            |
| 17    | Benzinka ve městě | 250.00  | Benzín           |
+-----+-----+-----+-----+
11 rows in set (0.00 sec)
```

Klíčové slovo `DESC` na konci klauzule `ORDER BY` způsobí, že se seznam míst výchozího (vzestupného) pořadí seřadí sestupně. V následujícím příkazu se objevuje volitelné, zřídka používané klíčové slovo `ASC`:

### Vstup/výstup ▼

```
SQL> SELECT PAYEE, AMOUNT
      2 FROM CHECKS
      3 ORDER BY CHECK# ASC;
```

PAYEE	AMOUNT
Nákupní centrum	1500
ČD	245
Nákupní centrum	2000
Místní benzinka	980
Diskont	1500
Hotovost	600
Drogerie ABC	243
Hotovost	2500
Benzinka ve městě	250