KAPITOLA 7

POUŽITÍ SQL K VÝBĚRU INFORMACÍ Z TABULEK

- Vytvoření dotazu
- Použití SELECT příkazu
- Kvalifikovaný výběr – WHERE klauzule

V této kapitole si ukážeme, jak vybrat informace z tabulek. Naučíme vás, jak vynechat nebo přeuspořádat sloupce a jak automaticky vylučovat nadbytečná data z výstupu. Detailněji probereme příkaz SELECT, který jsme stručně představili v kapitole 2 a kterému se budeme věnovat ještě v několika dalších kapitolách.

Vytvoření dotazu

SQL značí Structured Query Language (strukturovaný dotazovací jazyk). V současnosti je zkratka mnohem běžněji používána než vlastní název. Název vyplývá ze skutečnosti, že dotaz (query) je nejčastěji používaný aspekt SQL. Co je dotaz? Dotaz je příkaz, ve kterém chcete, aby DBMS zjistil určité informace. Dotazy se v SQL vytváří jedním příkazem. Struktura tohoto příkazu je až ošidně jednoduchá, protože jej můžete rozšířit a umožnit vysoce promyšlené vyhodnocení a zpracování dat. Příslušný příkaz se nazývá SELECT.

Použití příkazu SELECT

V jeho nejjednodušší formě příkaz SELECT dává databázi pokyn k tomu, aby zobrazila obsah tabulek. Např. byste mohli zobrazit tabulku „Salespeople" zadáním následujícího příkazu:

```
SELECT snum, sname, city, comm
   FROM Salespeople;
```

Výstup dotazu ukazuje obr. 7.1.

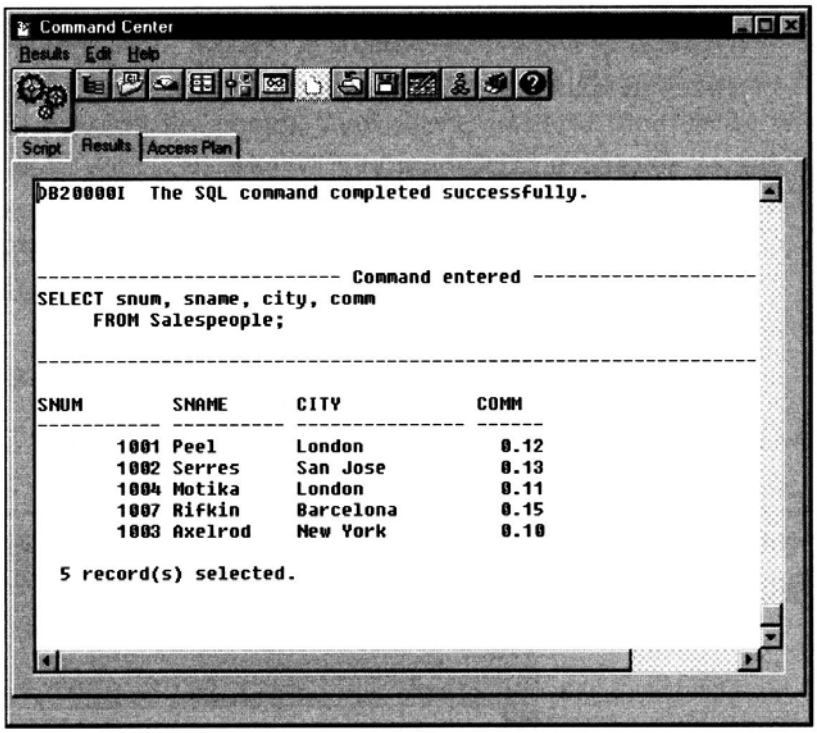
Jinými slovy tento příkaz jednoduše zobrazuje všechna data v tabulce. Zde je vysvětlení každé části příkazu:

SELECT — klíčové slovo, které říká databázi, že tento příkaz je dotaz. Všechny dotazy začínají tímto slovem po němž následuje bílý znak (pozn. překl.: mezera, nový řádek nebo tabulátor).

snum, sname ... — úplný seznam sloupců tabulky, které se vybírají dotazem. Sloupce neuvedené v seznamu se nezahrnou do výstupu příkazu.

FROM Salespeople — klíčové slovo jako SELECT, které musí být přítomné v každém dotazu. Následuje mezera a jméno tabulky použité jako zdroj informace. V tomto případě se jedná o tabulku „Salespeople".

OBRÁZEK 7.1:
Příkaz Select



Za zmínku stojí skutečnost, že tento dotaz neuspořádá svůj výstup nějakým konkrétním způsobem. Stejný příkaz provedený na stejných datech v různých časech může dokonce data seřadit různě. Výstup ze SQL příkazů můžete uspořádat použitím speciálních klauzulí. Později vysvětlíme, jak to udělat. Prozatím jednoduše řekneme, že v nepřítomnosti explicitního řazení nebude váš výstup v definovaném pořadí.

Jednoduchý výběr všech sloupců

Pokud chcete zobrazit všechny sloupce tabulky, existuje speciální zástupný znak. Celý seznam sloupců můžete nahradit hvězdičkou „*" jako v následujícím příkladu:

```
SELECT *
   FROM Salespeople;
```

Vytvoříme tím stejný výsledek jako v našem předcházejícím příkazu.

Shrneme-li dosavadní poznatky, pak víme, že příkaz SELECT začíná klíčovým slovem SELECT. Následuje seznam jmen sloupců, které si přejete vidět, oddělený čárkami. Pokud chcete vidět všechny sloupce tabulky, můžete seznam nahradit hvězdičkou. Další částí příkazu je klíčové slovo FROM následované mezerou a jménem tabulky, která je dotazována. Konec příkazu vyjadřuje středník.

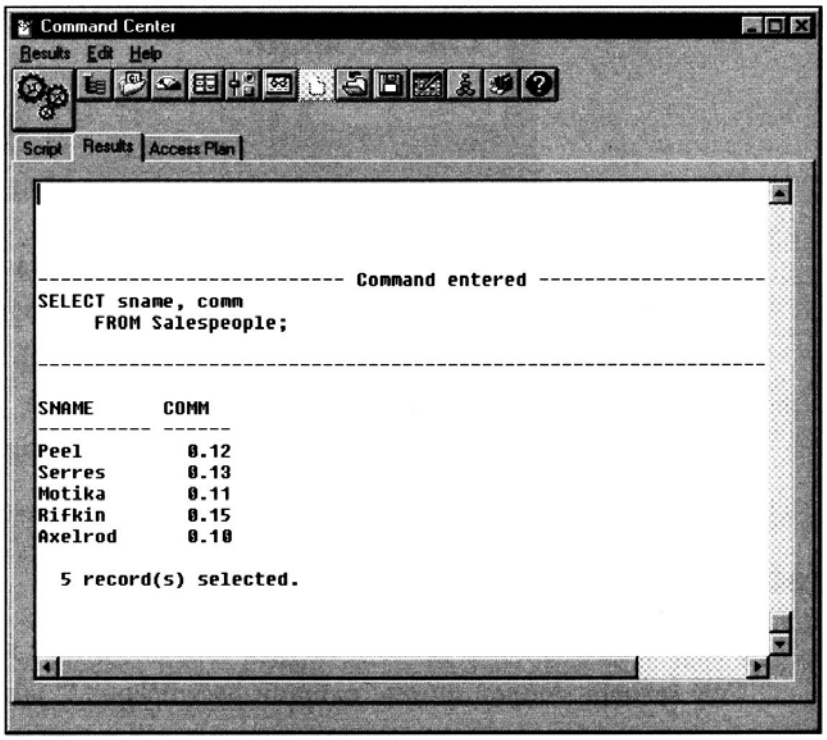
Výběr určitých sloupců

Abyste se podívali pouze na určité sloupce tabulky, stačí jednoduše z klauzule SELECT vynechat ty sloupce, které si nepřejete vidět. Příklad takového dotazu:

```
SELECT sname, comm
   FROM Salespeople;
```

vytvoří výstup na obr. 7.2.

OBRÁZEK 7.2:
Výběr určitých sloupců



Často se setkáte s tabulkami, které mají velký počet sloupců s daty, jež k vašemu účelu nepotřebujete. Tudíž shledáte možnost vybírat a volit sloupce za zcela užitečnou.

Mimochodem, ačkoliv sloupce tabulky mají specifické pořadí (na rozdíl od řádků), neznamená to, že je v něm musíte vybírat. Hvězdička zobrazí všechny sloupce v jejich původním pořadí, ale když vyjmenujete sloupce samostatně, můžete je seřadit v jakémkoliv pořadí, které chcete. Přestože jsme uvedli velmi jednoduché příklady, poukázali jsme při tom na důležitou vlastnost dotazů, kterou je schopnost vybírat a přeuspořádat vaše data. V příštích několika kapitolách, ozřejmíme, že jednoduchý příkaz SELECT může shromáždit, dát do vztahu a odvodit informace v rámci celé databáze.

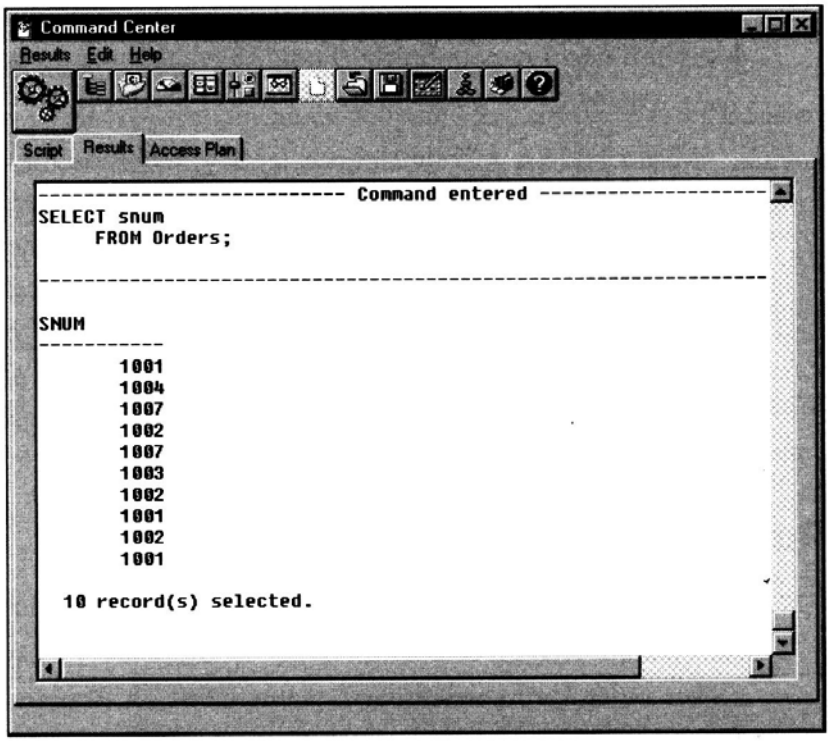
Rušení nadbytečných dat

Modifikátor DISTINCT poskytuje způsob, jak zrušit duplicitní hodnoty z vašeho výstupu. DISTINCT se umísťuje do SELECT klauzule. Předpokládejme, že chcete vědět, kteří prodejci mají momentálně objednávky v tabulce „Orders". Nezajímá vás kolik objednávek každý z nich má, potřebujete jen seznam čísel prodejců (snum). Mohli byste zadat

```
SELECT snum
   FROM Orders;
```

abyste dostali výstup z obr. 7.3.

OBRÁZEK 7.3:
SELECT s duplikáty

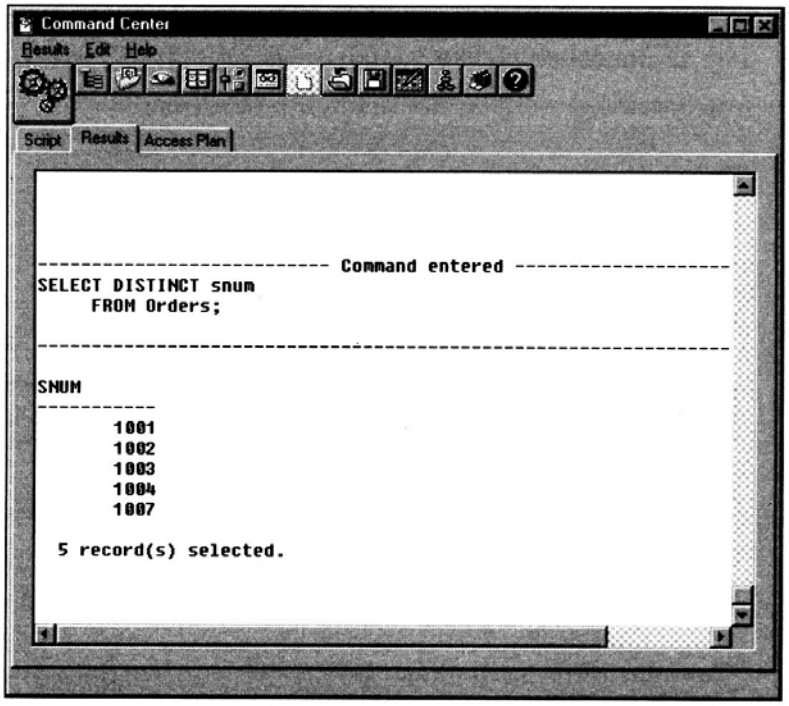


Snáze čitelný seznam bez duplikátů, byste mohli vytvořit následujícím příkazem:

```
SELECT DISTINCT snum
   FROM Orders;
```

Výstup tohoto dotazu ukazuje obr. 7.4.

OBRÁZEK 7.4:
SELECT bez duplikací



Jinými slovy DISTINCT sleduje, které hodnoty se již vyskytly, a duplikáty nezařadí na výstup. To je užitečný způsob, jak se vyhnout nadbytečným datům, ale je důležité si uvědomit, co děláte. *Jestliže nepracujete s redundantními daty, potom byste neměli používat DISTINCT, protože jeho použití může skrývat problém.* Např. byste mohli předpokládat, že všechna jména zákazníků jsou různá. Když ovšem někdo vloží druhého Clemense do tabulky „Customers" a vy použijete SELECT DISTINCT cname, neuvidíte důkaz duplikace. Oba se jeví jako jediný zákazník. Jelikož v tomto případě neočekáváte nadbytečnost, neměli byste DISTINCT raději použít. Také byste DISTINCT neměli používat, pokud jej skutečně nepotřebujete. Může totiž zpomalovat vaše dotazy, neboť systém dělá zbytečnou práci navíc.

Parametry DISTINCT

DISTINCT se používá ke všem sloupcům v klauzuli SELECT. Když kombinace všech výstupních sloupců duplikuje nějakou předchozí vybranou kombinaci, tento duplikát se vynechá. Tudíž může být toto klíčové slovo uvedeno maximálně jednou v jednom příkazu SELECT (výjimku tvoří použití s agregačními funkcemi, jak vysvětluje kapitola 8). Pokud by klauzule z obrázku 7.4 vybrala několik sloupců, DISTINCT by rušil řádky tam, kde by všechny hodnoty sloup-

ců byly stejné. Řádky, v kterých by některé hodnoty byly stejné a některé odlišné, by se ponechaly. Jelikož se klauzule DISTINCT používá na celý řádek nikoliv na specifický sloupec, nemá smysl ji opakovat.

DISTINCT versus ALL

Jako alternativu k DISTINCT můžete uvést ALL. Ta má opačný účinek: duplikované výstupní řádky se ponechávají. Jelikož vede ke stejnému výsledku, jako když neuvedete nic, ALL je tedy spíše vysvětlivka než funkční argument.

POZNÁMKA V praxi se na většině systémů vyskytuje klíčové slovo UNIQUE – synonymum k DISTINCT. Záměnou těchto klíčových slov vznikne identický výstup.

Kvalifikovaný výběr – klauzule WHERE

Tabulky postupně bobtnají — s přibývajícím časem se přidává více a více řádků. Jelikož vás v daném čase obvykle zajímají jen určité řádky, SQL vás nechá definovat kritéria určující, které řádky se vyberou na výstup. Klauzule WHERE příkazu SELECT vám umožňuje definovat predikát, který může pro daný řádek tabulky nabývat hodnoty TRUE, FALSE nebo UNKNOWN. Příkaz vybírá z tabulky jen ty řádky, pro které je predikát roven TRUE. S predikáty jsme již setkali v kapitolách 4 a 6, ale příkaz SELECT je používá asi sofistikovaněji než ostatní příkazy, a proto v této kapitole probereme predikáty podrobněji.

Mimochodem řádky tabulky, které se vyhodnocují predikátem, se nazývají *kandidátské řádky (candidate rows).* Řádky, v nichž predikát nabývá hodnoty TRUE, se nazývají *vybrané řádky (selected rows).*

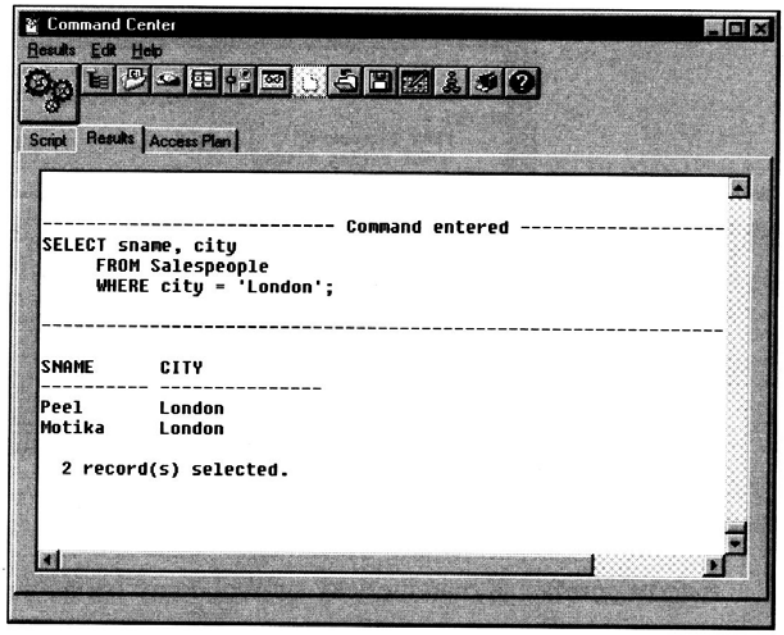
Předpokládejme, že chcete vidět jména a provize všech prodejců v Londýně. Mohli byste zadat tento příkaz:

```
SELECT sname, city
   FROM Salespeople
   WHERE city = 'London';
```

Když je přítomna klauzule WHERE, databázový program prochází celou tabulkou řádek po řádku a zkoumá každý řádek, aby určil, zda je predikát TRUE. Tudíž např. pro Peelův záznam se program podívá na hodnotu sloupce city, ur-

čí, že se rovná hodnotě 'London' a zahrne tento řádek do výstupu. Serresův záznam nezahrne atd. Výstup výše uvedeného dotazu ukazuje obrázek 7.5.

OBRÁZEK 7.5:
SELECT s WHERE klauzulí



Všimněte si, že sloupec city není součástí výstupu, přestože se jeho hodnota použila k určení, zda se řádky vyberou. To nám skvěle vyhovuje. Není nutno, aby sloupce použité WHERE klauzulí byly přítomny mezi těmi, které jsou vybrány na výstup.

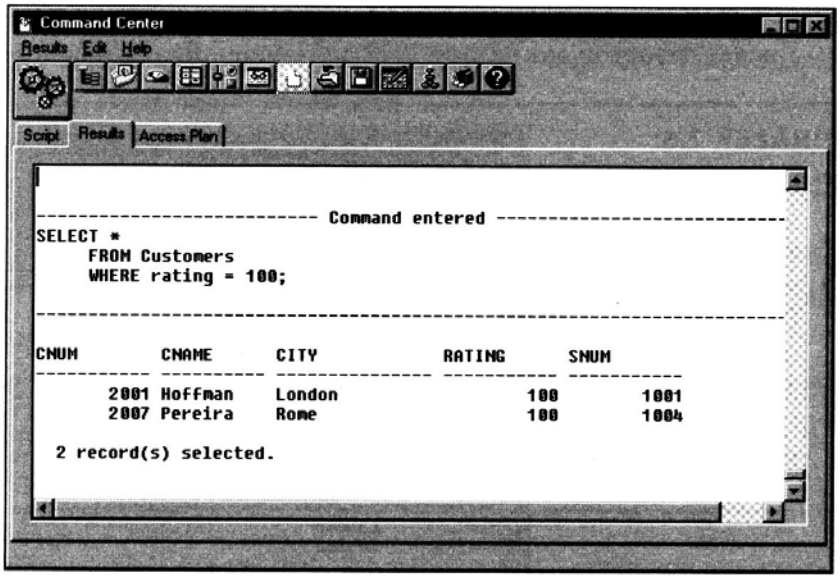
Zkusme příklad s číselným polem v klauzuli WHERE. Pole hodnocení (rating) tabulky „Customers" slouží k třídění zákazníků do skupin založených na kritériích vyhodnotitelných číslem, jako např. úvěrové hodnocení nebo hodnocení založené na předcházejících nákupech. Takové číselné kódy mohou být v relačních databázích užitečné jako způsob sumarizující komplexní informace. Všechny zákazníky s hodnocením 100 můžeme vybrat následujícím příkazem:

```
SELECT *
   FROM Customers
   WHERE rating = 100;
```

Nepoužili jsme jednoduché uvozovky, protože pole rating je číselné. Výsledek dotazu je ukázán na obrázku 7.6.

OBRÁZEK 7.6:

SELECT s číselným polem v predikátu



Použití porovnání v predikátech

Relační operátor je matematický symbol, který označuje jistý typ porovnání dvou hodnot. Relační operátor rovnosti používáme intuitivně; např. `2 + 3 = 5` nebo `city = 'London'`. Existují i další relační operátory. Předpokládejme, že chcete zobrazit všechny prodejce s provizí převyšující určitou hodnotu. Použijete porovnání typu větší než. Následuje výčet všech relačních operátorů, které SQL rozeznává:

=	rovnost
>	větší než
<	menší než
>=	větší než nebo rovno
<=	menší než nebo rovno
<>	nerovnost

Pokud se vyskytuje ve srovnání hodnota NULL, výsledkem porovnání je hodnota UNKNOWN.

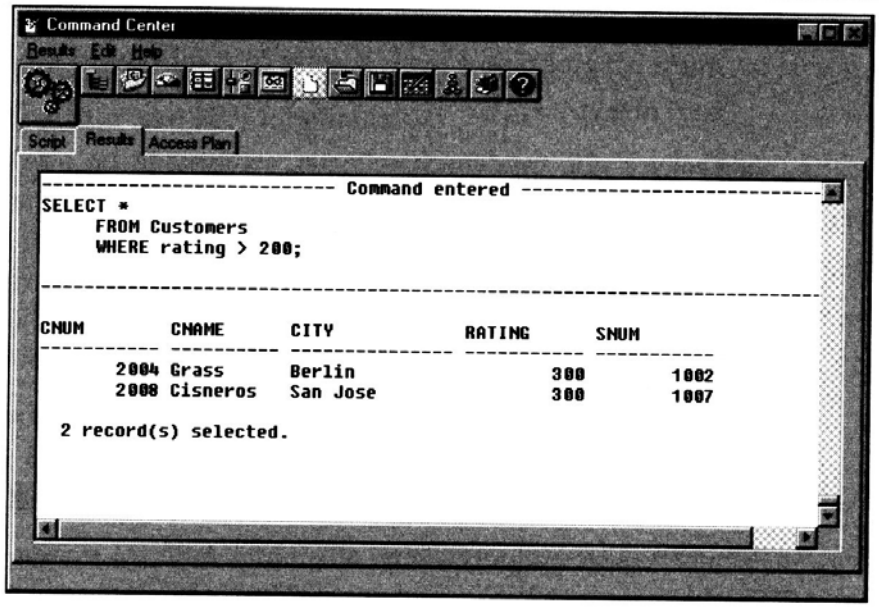
Tyto operátory mají standardní význam pro číselné hodnoty. Např. předpokládejme, že chcete vidět všechny zákazníky s hodnocením nad 200. Jelikož 200

je skalární hodnota stejného typu jako hodnoty ve sloupci rating, můžete použít relační operátor k jejich porovnání:

```
SELECT *
   FROM Customers
   WHERE rating > 200;
```

Výstup tohoto dotazu ukazuje obrázek 7.7.

OBRÁZEK 7.7: Dotaz používající nerovnost



Chcete-li vidět i zákazníky s hodnocením rovným 200, pochopitelně použijete predikát:

```
rating >= 200
```

Pro znakové hodnoty význam relačních operátorů záleží na řazení s následujícími možnostmi:

- Když DBMS plně vyhovuje SQL92, podporuje také *řadicí posloupnosti (collating sequences).* Řadicí posloupnost je databázový objekt, který specifikuje, jak se má množina znaků řadit. Můžete vytvářet a přiřazovat vlastní řazení. DBMS, který podporuje řadicí posloupnosti, většinou obsahuje implicitní způsob řazení znakových entit, jejichž podpora je zabudovaná do programu. Řadicí posloupnosti jakožto objekty umožňují uživatelské rozšíření DBMS o znakové sady, které nebyly původní součástí systému.

- Pokud DBMS plně nevyhovuje SQL92, může řadit znaky podle binárních čísel, které je interně reprezentují. Pro anglický jazyk se bude řazení většinou řídit ASCII kódem eventuálně jeho rozšířeními. ASCII kód odpovídá abecednímu pořadí, ovšem všechna velká písmena předcházejí všechna malá písmena. Tudíž 'a' < 'b', ale 'a' > 'B'. Lze si přestavit i systém založený na kódu EBCDIC. V tomto kódování se řadí rovněž podle anglické abecedy, ale všechna malá přímena předcházejí všechna velká písmena.
- DBMS mohou poskytovat řazení nezávislé na velikosti písmen, které nemůže být změněno a je tudíž spíše částí funkčnosti produktu než databázový objekt.

Jakékoliv z výše uvedených schémat neodpovídá na otázku, jak řadit znaky neobsažené v abecedě (tj. čísla, interpunkci, speciální znaky atd.) v rozsahu znakového souboru. Beze sporu bude váš systém mít nějaké definované pořadí, které by mělo být specifikované v systémové dokumentaci, ale neexistují žádná obecná pravidla, protože si lze jen těžko představit nějaká smysluplná. Co by například mělo být výsledkem porovnání '*' < 'a'?

Jiný standardní datový typ, na který se může použít relační operátor, je DATETIME. Ten ovšem může obsahovat datum, čas nebo kombinaci obojího a tudíž jej není možno řadit vždy. Např. '11/24/99' > '11:52:43' nemá smysl.

Mimochodem hodnoty, který se mohou řadit, se nazývají *skalární hodnoty*. SQL predikáty typicky porovnávají skalární hodnoty s použitím buď relačních operátorů nebo speciálních SQL operátorů, aby zjistili, zda je výsledkem porovnání TRUE. Některé speciální SQL operátory si vysvětlíme v příští kapitole.

Nejméně pro dva datové typy SQL nemá porovnání smysl:

- Pro vysloveně binární data, která jsou často označovaná jako BLOB (Binary Large Objects – rozsáhlé binární objekty). Předpokládejme, že jste uložili obrázky vašich zaměstnanců do databáze ve formátu JPEG. Řazení těchto binárních reprezentací by bylo založeno na barvě jejich pixelů. Takové řazení asi nemá smysl.
- SQL99 dovoluje uživatelům vytvářet abstraktní strukturované datové typy (UDT). Jejich řazení nemůže být také smysluplné.

Vývojáři aplikací někdy vytváří vlastní řazení i těchto datových typů, je-li to užitečné. Sémantika SQL takové řazení dovoluje.

POZNÁMKA Více o SQL99 viz část VI.

Práce s NULL hodnotami

Často tabulka obsahuje řádky, které nemají hodnoty v každém sloupci, protože je informace neúplná anebo protože se sloupec jednoduše nepoužije pro každý případ. SQL povoluje v takové situaci vložit do sloupce NULL. Když je ve sloupci NULL, znamená to, že databázový program speciálně označil, že tento sloupec nemá žádnou hodnotu. Jelikož NULL není technicky hodnota, nemá datový typ a může být umístěno do jakéhokoliv typu sloupce. Nicméně NULL se v SQL často používá jako NULL hodnota.

Předpokládejme, že máte nového zákazníka, kterému ještě nebyl přiřazen prodejce. Místo čekání na prodejce, který mu bude přiřazen, chcete vložit zákazníka do databáze ihned, aby se v této nevyjasněné situaci neztratil. Můžete vložit řádek zákazníka s NULL pro snum a doplnit hodnotu do tohoto sloupce později, když bude prodejce přiřazen. NULL hodnoty ovšem podstatně mění způsob, kterým pracuje SQL logika, jak si ukážeme v následujícím oddíle.

Použití booleovských operátorů v predikátech

Základní booleovské operátory použité ve většině programovacích jazyků se vyskytují i v SQL, ale kvůli NULL hodnotám pracují trochu odlišně než obvykle. Ve většině programovacích jazyků booleovské výrazy nabývají hodnotu buď TRUE (pravda) nebo FALSE (nepravda). V SQL mohou také nabýt hodnotu UNKNOWN (neznámá hodnota). Z tohoto důvodu SQL používá třístavovou logiku spíše než tradiční dvoustavovou. Booleovské operátory se vztahují k jedné nebo více booleovským hodnotám a vytváří jedinou booleovskou hodnotu: TRUE, FALSE nebo UNKNOWN. Booleovské operátory používané v SQL jsou standardní AND, OR a NOT. Existují i složitější booleovské operátory (např. „exkluzivní or" často zapisované XOR), ale ty můžete vytvořit kombinací našich tří jednoduchých. V tradiční (dvoustavové) booleovské logice operátory pracují následovně:

- NOT vezme jednoduchý booleovský výraz (ve tvaru NOT A) jako argument a mění jeho hodnotu z FALSE na TRUE a z TRUE na FALSE.

- AND vezme dva booleovské výrazy (ve tvaru A AND B) jako argumenty a vyhodnotí je na TRUE, když jsou oba TRUE, jinak na FALSE.
- OR vezme dva Booleovské výrazy (ve tvaru A OR B) jako argumenty a vyhodnotí je na TRUE, když alespoň jeden z nich je TRUE. Ve zbývajícím případě je vyhodnotí na FALSE.

Ovšem třístavová logika SQL je o něco složitější. Tabulky 7.1, 7.2 a 7.3 ilustrují, jak NOT, OR a AND pracují v třístavové logice.

TABULKA 7.1: Pravdivostní hodnoty používající NOT v třístavové logice

Výraz	Pravdivostní hodnota
NOT TRUE	FALSE
NOT FALSE	TRUE
NOT UNKNOWN	UNKNOWN

Jak vidíte, rozdíl je v tom, že NOT nemění hodnotu UNKNOWN. Je stále UNKNOWN. To je důležitý rozdíl v SQL predikátech, poněvadž UNKNOWN se většinou chová stejně jako FALSE. Pokud si neuvědomíte rozdíl, budete svedeni na scestí. Následující dvě tabulky jsou formátovány podobně jako tabulky násobení, které jste měli tak rádi na základní škole. Výsledek operace OR (v Tabulce 7.2) nebo AND (v tabulce 7.3) dvou pravdivostních hodnot najdete, podíváte se na průsečík řádku jedné hodnoty se sloupcem hodnoty druhé.

TABULKA 7.2: Pravdivostní hodnoty používající OR ve třístavové logice

OR	TRUE	FALSE	UNKNOWN
TRUE	true	true	true
FALSE	true	FALSE	UNKNOWN
UNKNOWN	true	UNKNOWN	UNKNOWN

Zapamatujte si následující pomůcku:

- TRUE OR (jakákoliv pravdivostní hodnota) je TRUE.
- UNKNOWN OR (jakákoliv pravdivostní hodnota jiná než TRUE) je UNKNOWN.
- FALSE OR FALSE je FALSE.

TABULKA 7.3: Pravdivostní hodnoty používající AND v třístavové logice

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

Můžete si zapamatovat následující pomůcku:

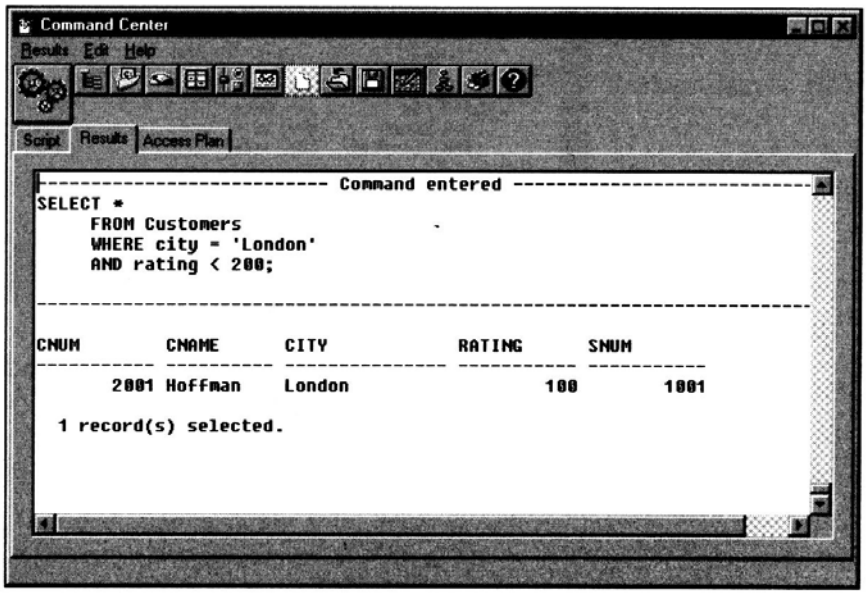
- FALSE AND (jakákoliv pravdivostní hodnota) je FALSE.
- UNKNOWN AND (jakákoliv pravdivostní hodnota jiná než FALSE) je UNKNOWN.
- TRUE AND TRUE je TRUE.

Zavedením booleovských výrazů s booleovskými operátory můžete značně zvýšit sofistikovanost predikátů. Předpokládejme, že chcete vidět všechny zákazníky v Londýně, kteří mají hodnocení (rating) menší než 200:

```
SELECT *
   FROM Customers
   WHERE city = 'London'
   AND rating < 200;
```

Výstup tohoto dotazu ukazuje obr. 7.8., kde je jen jeden zákazník, který splňuje požadavek.

OBRÁZEK 7.8: SELECT s použitím Booleovského operátoru



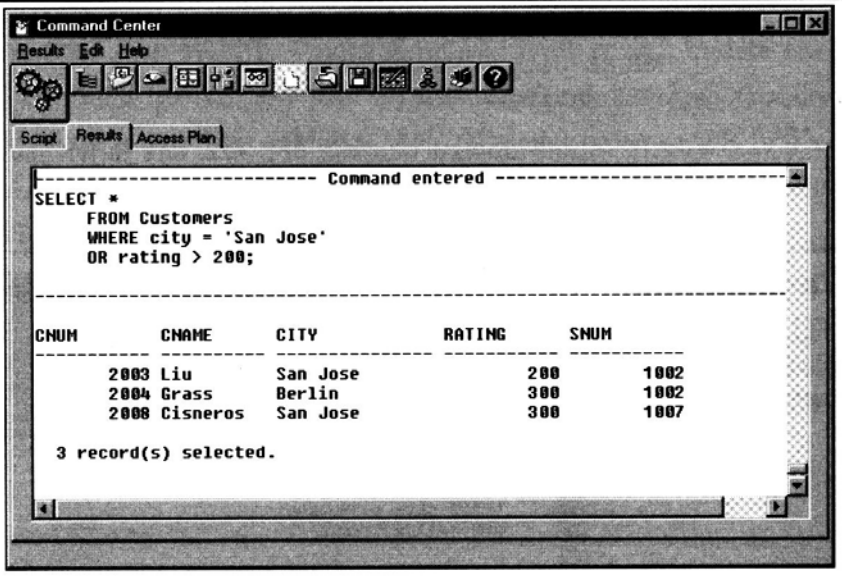
A co třeba takový Clemens, zákazník číslo 2006? Je z Londýna, ale má hodnocení NULL. Prozkoumejme tento výraz. Pro Clemense výraz city = 'London' nabývá hodnotu TRUE a rating < 200 je UNKNOWN (NULL < 200). Výsledkem predikátu TRUE AND UNKNOWN je hodnota UNKNOWN. Jelikož predikát vybírá jen řádky, které jsou TRUE, Clemens není vybrán. Všimněte si že, kdybychom stanovili rating > 200, nebyl by stále vybrán. SQL obsahuje speciální operátor pro práci s NULL hodnotami, který si představíme v příští kapitole.

Kdybyste použili OR, dostali byste všechny zákazníky, kteří splnili jednu z těchto podmínek. Následující dotaz vybírá všechny, kteří jsou buď umístěni v San Jose nebo mají hodnocení výšší než 200:

```
SELECT *
   FROM Customers
   WHERE city = 'San Jose'
   OR rating > 200;
```

Výstup tohoto dotazu ukazuje obr. 7.9.

OBRÁZEK 7.9:
SELECT s použitím OR

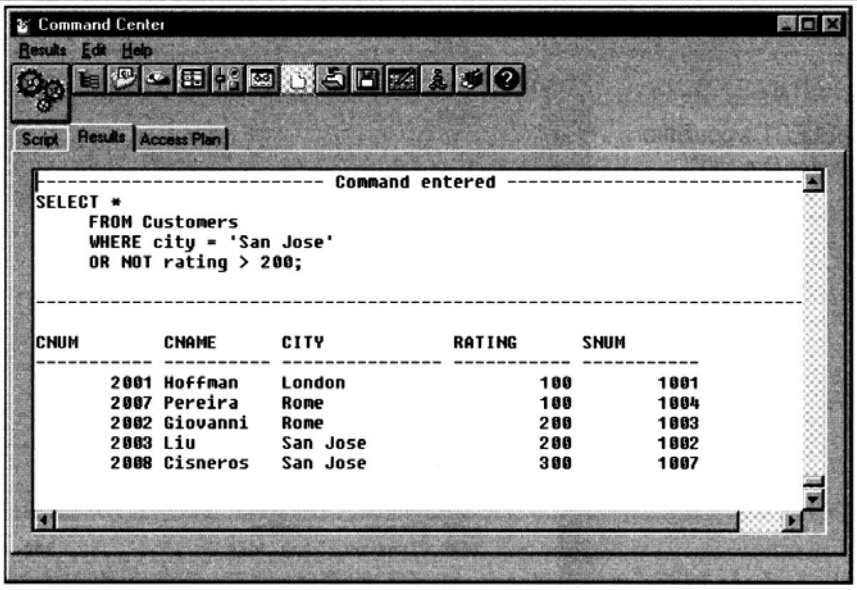


Můžete použít operátor NOT k obrácení booleovské hodnoty. Zde je příklad dotazu s NOT:

```
SELECT *
   FROM Customers
   WHERE city = 'San Jose'
   OR NOT rating > 200;
```

Výstup tohoto dotazu ukazuje obr. 7.10.

OBRÁZEK 7.10: SELECT s použitím NOT



Dotaz vybral všechny věty s výjimkou Grasse a Clemense. Grass není v San Jose a jeho hodnocení je vyšší než 200, takže neprošel oběma testy. Clemens neprošel v testu na město a jeho hodnocení je UNKNOWN. Všechny ostatní řádky splnily jedno nebo druhé kritérium (případně obě najednou). Všimněte si, že NOT operátor musí předcházet booleovskou hodnotu, kterou má měnit. Nesmí být umístěn před relační operátor. Následující příklad ukazuje *nesprávné* použití:

```
rating NOT > 200
```

Tento zápis se v SQL hodnotí jako chybný, přestože v lidském jazyce by byl správný.

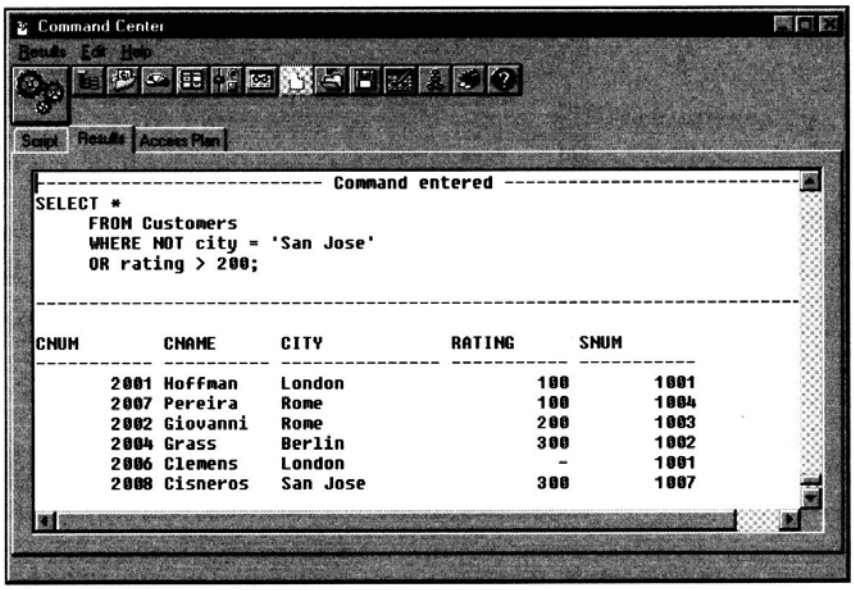
Následující příklad obrací pozornost na další problém. Jak by SQL vyhodnotil následující příkaz?

```
SELECT *
   FROM Customers
   WHERE NOT city = 'San Jose'
   OR rating > 200;
```

Zneguje se jen výraz `city = 'San Jose'` nebo oba výrazy anebo dokonce pouze výraz rating > 200? Jak již bylo napsáno, správná odpověď je ta prvně

jmenovaná. SQL použije NOT jen k bezprostředně následujícímu booleovskému výrazu. Výstup tohoto dotazu ukazuje obr. 7.11.

OBRÁZEK 7.11: SELECT s použitím složeného NOT

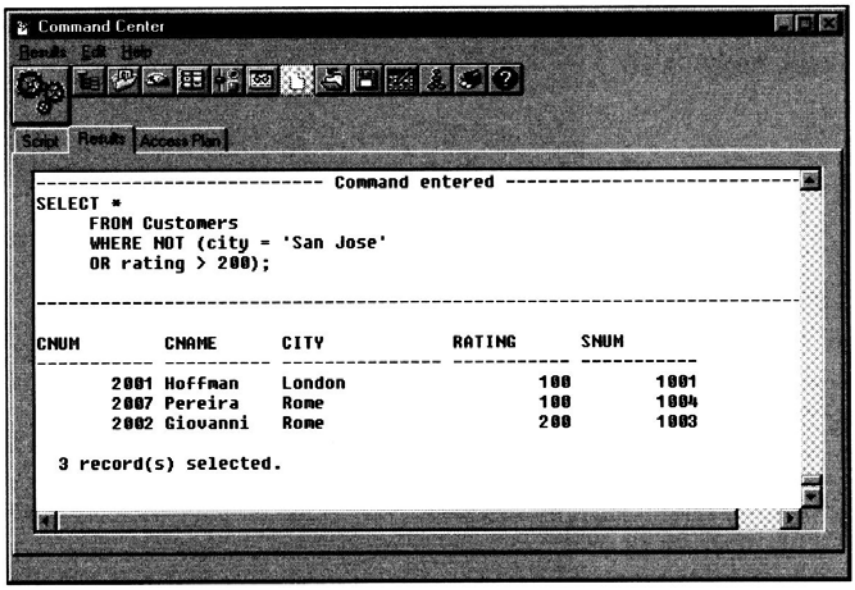


S následujícím příkazem obdržíte jiný výsledek:

```
SELECT *
   FROM Customers
   WHERE NOT (city = 'San Jose'
   OR rating > 200);
```

Závorky slouží v SQL ke změně priority operátorů. SQL chápe závorky tak, že cokoliv uvnitř nich je vyhodnocováno dříve a bere se jako jediný výraz s čímkoliv mimo nich (což je standardní interpretace booleovské logiky a mnohých programovacích jazyků). Jinými slovy SQL vezme každý řádek a určuje, zda `city = 'San Jose'` nebo `rating > 200`. Když alespoň jedna z podmínek nabývá hodnotu TRUE, booleovský výraz uvnitř závorek je také TRUE. Ovšem, když booleovský výraz uvnitř závorek je TRUE, predikát jako celek nabývá hodnotu FALSE, protože NOT obrací TRUE na FALSE a naopak. Výstup tohoto dotazu je ukázán na obr. 7.12.

OBRÁZEK 7.12:
SELECT s NOT a závorkami



Následuje záměrně složitý příklad. Podívejte se, jestli jste schopni sledovat jeho logiku (výstup je ukázán na obr. 7.13):

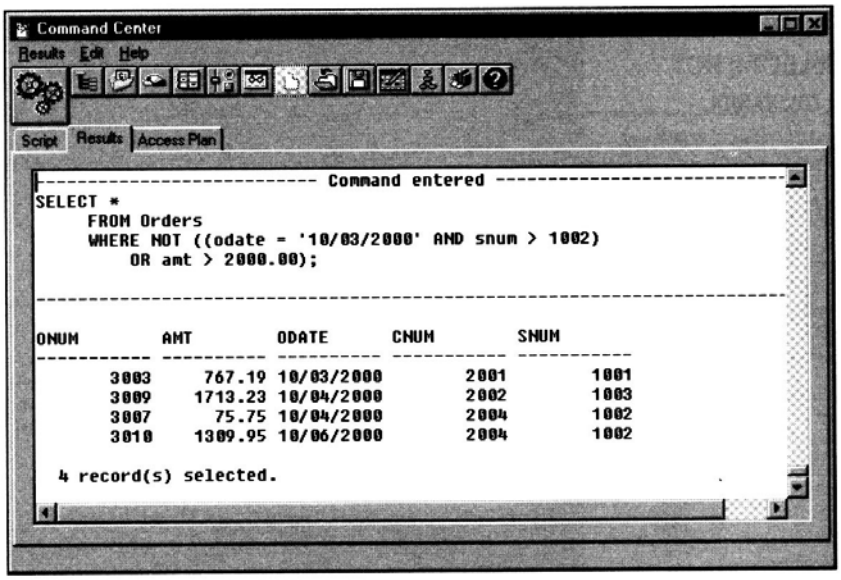
```
SELECT *
   FROM Orders
   WHERE NOT ((odate = '10/03/2000' AND snum > 1002)
   OR amt > 2000.00);
```

Ačkoliv jsou jednotlivé booleovské operátory jednoduché, situace se zkomplikuje, když se zkombinují do složitých výrazů. Cesta jak vyhodnotit složitý booleovský výraz vede přes přednostní vyhodnocení booleovských výrazů nejvíce vnořených do závorek. Po obdržení jednoduché booleovské hodnoty pokračujte s výrazy o jednu závorku méně vložených výrazů až postupně vyhodnotíte celý příkaz.

Zde je podrobné vysvětlení, jak vyhodnocujeme příklad z obr. 7.13.

- Napřed jsme zjistili, že žádný z testovaných sloupců (odate, snum a amt) náhodou neobsahuje NULL hodnoty. Tudíž zde nebudou žádné predikáty s výsledkem UNKNOWN a můžeme se omezit na konvenční (dvoustavovou) booleovskou logiku, která bude v tomto příkladu i tak dost složitá. V reálné situaci byste mohli jen použít tento předpoklad pouze, pokud by všechny testované sloupce měly NOT NULL omezení.

OBRÁZEK 7.13:
Složitý dotaz



- Nejvíce vnořené booleovské výrazy v predikátu — odate = '10/03/2000' a snum > 1002 — jsou spojené pomocí AND, se vyhodnotí na TRUE pro všechny řádky, které splňují obě podmínky. Nazvěme tento složený booleovský výraz B1.
- B1 je spojené s výrazem amt > 2000.00 (B2) operátorem OR a vytváří třetí výraz B3.
- B3 je TRUE pro daný řádek, jestliže buď B1 nebo B2 je TRUE.
- B3 je zcela obsažené v závorkách jimž předchází NOT. Tím je tvořen konečný booleovský výraz — B4, který je podmínkou predikátu. Tudíž predikát B4 je TRUE, kdykoliv B3 je FALSE a naopak.
- B3 je FALSE kdykoliv B1 a B2 jsou obě FALSE.
- B1 je FALSE, když datum objednávky řádku není 10/03/2000 nebo jeho hodnota snum není větší než 1002.
- B2 je FALSE pro všechny řádky s množstvím nižším nebo rovným 2000,00. Jakýkoliv řádek s množstvím vyšším než 2000,00 by zapříčinil, že by B2 byl TRUE. Výsledek B3 by byl TRUE a B4 FALSE. Tudíž se všechny takové řádky vynechávají z výstupu.

- Ze zbývajících řádků se ještě vyřadí ty s 3. říjnem a snum větším než 1002, protože B1 je na nich TRUE, tudíž B3 TRUE a výsledný predikát dotazu je FALSE. Výstup tvoří zbylé řádky.

Zde je o trochu jednodušší formulace výše uvedeného dotazu:

```
SELECT *
   FROM Orders
   WHERE (odate <> 10/03/2000 OR snum <= 1002)
   AND amt <= 2000.00;
```

POZNÁMKA V složitějších booleovských výrazech je lepší si vytvořit pravdivostní tabulku. První sloupce obsahují jednoduché výroky, další sloupce tvoří složitější výroky a poslední sloupec je cílový predikát. Řádky tabulky obsahují u jednoduchých výroků kombinace všech možných hodnot, u výroků složených pak výsledek. Počet kombinací je N^M, kde N je počet stavů logiky a M je počet jednoduchých výroků (proměnných vstupujících do predikátu). V našem příkladu tedy $2^3 = 8$.

V1: odate = '10/03/2000'

V2: snum > 1002

V3: amt > 2000,00

V4: V1 AND V2, tj. odate = '10/03/2000' AND snum > 1002

V5: V4 OR V3, tj. (odate = '10/03/2000' AND snum > 1002) OR amt > 2000,00

V6: NOT V5, tj. NOT ((odate = '10/03/2000' AND snum > 1002) OR amt > 2000,00)

V1	V2	V3	V4	V5	V6
TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	TRUE	FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	TRUE	TRUE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

Příkaz zobrazí všechny kombinace při nichž V6 nabývá hodnotu TRUE.

Shrnutí

Nyní znáte několik způsobů, jak přimět tabulku, aby místo prostého odhalení obsahu vydala informace, které chcete. Umíte přeuspořádat sloupce tabulky a vynechat některé z nich. Můžete se rozhodnout, zda chcete vidět duplicitní hodnoty.

Nejdůležitější osvojená dovednost spočívá v umění definovat podmínku zvanou predikát, která určuje, který z tisíců možných řádek tabulky bude vybrán na výstup. Predikáty se mohou stát sofistikovanou možností jak řídit, které řádky se vyberou dotazem. Je to právě schopnost rozhodovat přesně, co chcete vidět, co dělá SQL dotazy tak účinnými. Příštích několik kapitol bude z větší části věnováno popisu vlastností, které zvětšují sílu predikátů.

Umíte najít různými relačními operátory záznamy, jejichž sloupce odpovídají dané hodnotě. Můžete také použít booleovské operátory AND a OR, abyste vytvořili vícenásobné podmínky, z nichž každá může tvořit jednoduchý predikát. Jak jsme viděli, booleovský operátor NOT převrací význam podmínky nebo skupiny podmínek. Dovedete nastavit účinek booleovských a relačních operátorů použitím závorek, které určují pořadí prováděných operací. Poradíte si již s těmito operacemi na jakékoliv úrovni složitosti. Vyzkoušeli jste si budování komplexních podmínek z jednoduchých částí.

Nyní když jsme si ukázali, jak se používají standardní matematické operátory, můžeme se přesunout k operátorům, které jsou výlučné pro SQL. Více se dozvíte v příští kapitole.

Práce s SQL

1. Napište příkaz SELECT, který zobrazí čísla objednávek, množství a datum pro všechny řádky tabulky „Orders".
2. Napište dotaz, který zobrazí všechny řádky z tabulky „Customers", pro které je číslo prodejce rovno 1001.
3. Napište dotaz, který zobrazí tabulku „Salespeople" se sloupci v následujícím pořadí: city, sname, snum, comm.

4. Napište dotaz, který zobrazí hodnoty snum všech prodejců s objednávkami současně v tabulce „Orders" aniž dojde k opakování.

5. Napište dotaz, který zobrazí jména a města prodejců z Londýna s provizí > 0,10.

6. Napište dotaz do tabulku „Customers", jehož výstup bude vylučovat všechny zákazníky s hodnocením <= 100, pokud nejsou umístění v Římě.

7. Jaký bude výstup následujícího dotazu?

```
SELECT *
   FROM Orders
   WHERE (amt < 1000
     OR
     NOT (odate = '10/03/2000'
     AND cnum > 2003));
```

8. Jaký bude výstup následujícího dotazu?

```
SELECT *
   FROM Orders
   WHERE NOT ((odate = '10/03/2000' OR snum > 1006)
     AND amt > = 1500);
```

9. Jak jednodušším způsobem napsat tento dotaz? Předpokládejme, že sloupec comm nemůže obsahovat NULL hodnoty.

```
SELECT snum, sname, city, comm
   FROM Salespeople
   WHERE (comm >= .12
     OR
          comm < .14);
```

KAPITOLA 8

POUŽITÍ IN, BETWEEN, LIKE, IS NULL A AGREGAČNÍCH FUNKCÍ

- Operátor IN
- Operátor BETWEEN
- Operátor LIKE
- Operátor IS NULL
- Sumarizace dat agregačními funkcemi

Kromě relačních a booleovských operátorů, které jsme probrali v předešlé kapitole, používá SQL skupinu speciálních operátorů, která zahrnuje IN, BETWEEN, LIKE a IS NULL. V této kapitole se naučíte, jak s nimi zacházet, jako by to byly relační operátory. Znalost nových operátorů vám umožní vytvářet promyšlenější a účinnější predikáty.

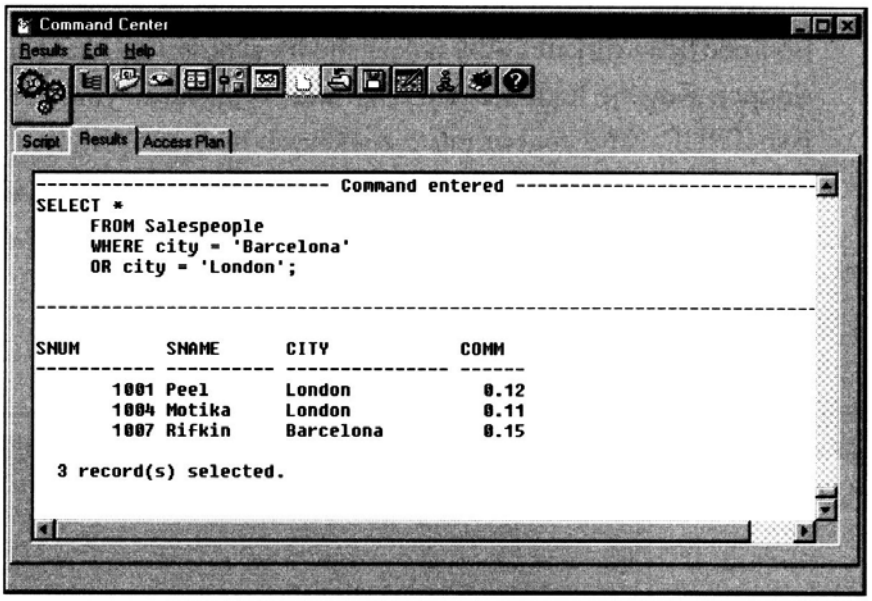
Dostanete se nad rámec jednoduchého vybírání dat z databáze a objevíte, jak můžete odvozovat nové informace z původních hodnot. Provedete to agregačními nebo sumarizačními funkcemi, které zpracují skupinu hodnot a zredukují ji na jednoduchou hodnotu. Naučíte se, jak používat tyto funkce, jak definovat skupiny hodnot ke kterým se použijí tyto funkce a jak určit, které z výsledných seskupených hodnot se vyberou na výstup. Uvidíte také za jakých podmínek lze použít sloupcové hodnoty s odvozenou informací v jediném dotazu.

Operátor IN

IN explicitně definuje soubor, ze kterého daná hodnota může nebo nemůže být zahrnuta. Z dosavadních poznatků víte, že najít všechny prodejce umístěné buď v Barceloně nebo v Londýně, znamená použít následující dotaz (jeho výstup je ukázán na obr. 8.1):

```
SELECT *
     FROM Salespeople
     WHERE city = 'Barcelona'
     OR city = 'London';
```

OBRÁZEK 8.1: Nalezení prodejců v Barceloně a Londýně

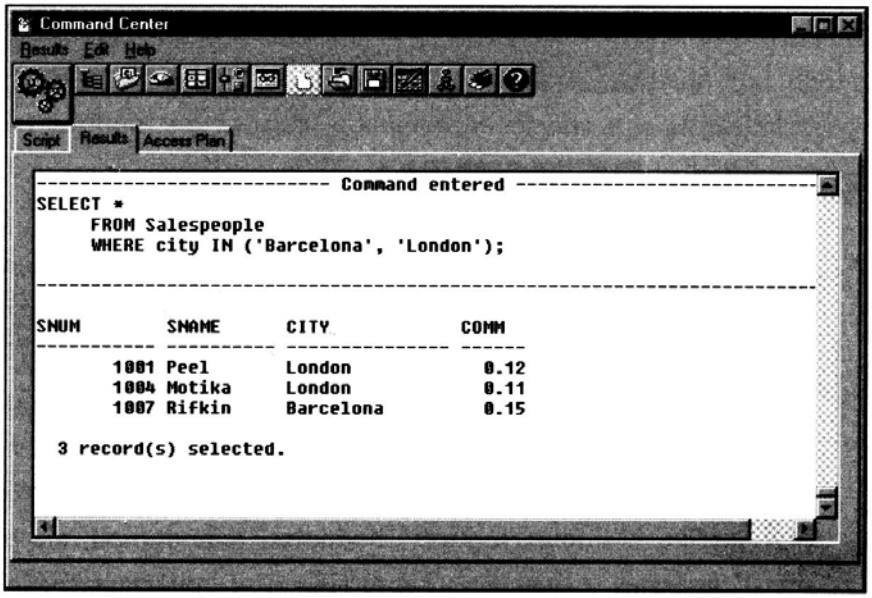


Zde je jednodušší způsob, jak dostat stejnou informaci:

```
SELECT *
     FROM Salespeople
     WHERE city IN ('Barcelona', 'London');
```

Výstup tohoto dotazu je zobrazen na obr. 8.2.

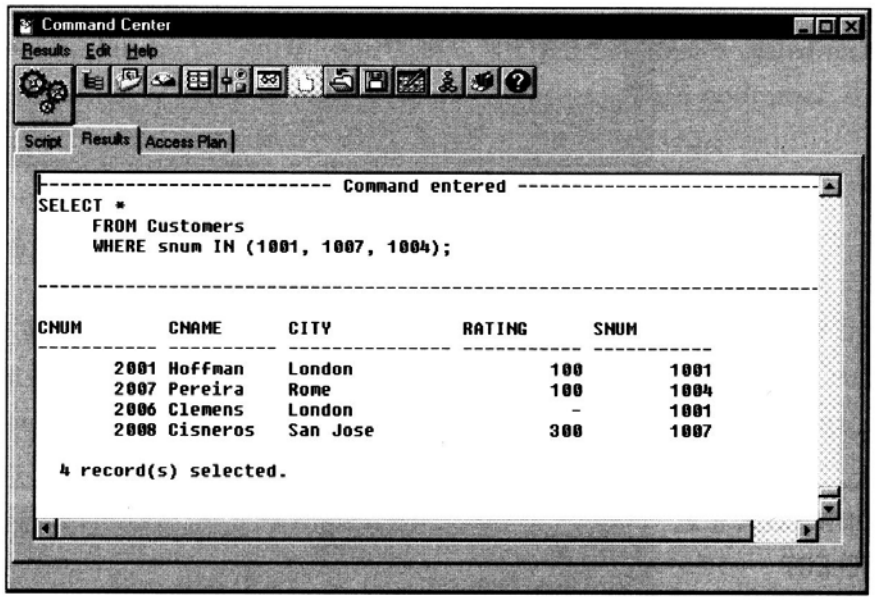
OBRÁZEK 8.2: SELECT s použitím IN



Jak vidíte IN explicitně definuje členy souboru v závorkách. Jednotlivé prvky jsou odděleny čárkami. Aby nalezl shodu, zkontroluje různé hodnoty uvedeného sloupce. Když je hodnota nalezena mezi vyjmenovanými, predikát se vyhodnotí jako TRUE. Když soubor místo znakových hodnot obsahuje čísla, vynechávají se jednoduché uvozovky. Najděme všechny zákazníky odpovídající prodejcům 1001, 1007 a 1004. Výstup následujícího dotazu ukazuje obr. 8.3:

```
SELECT *
     FROM Customers
     WHERE snum IN (1001, 1007, 1004);
```

OBRÁZEK 8.3: SELECT používající IN s čísly



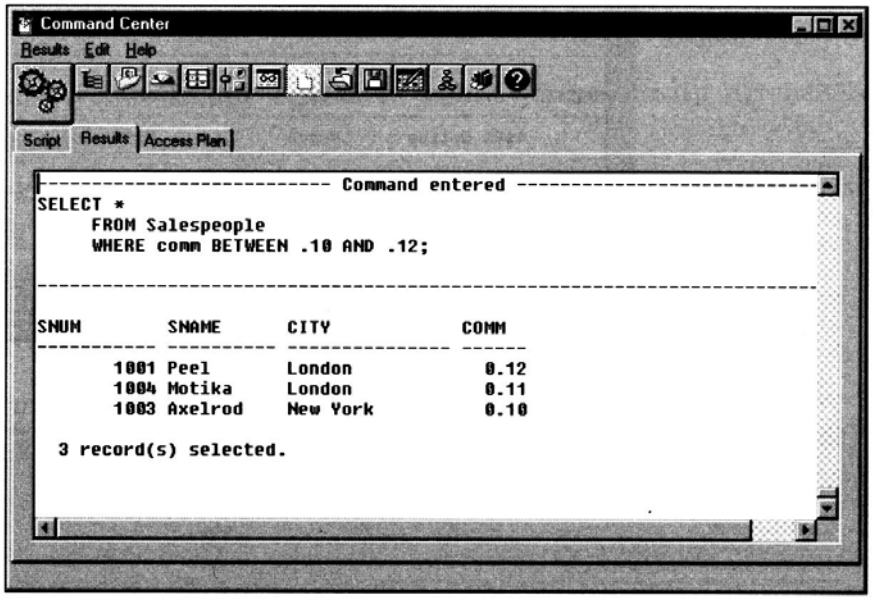
Operátor BETWEEN

Operátor BETWEEN se podobá IN. Místo výčtu hodnot jako v případě IN ale BETWEEN definuje interval do něhož musí spadat hodnoty, aby byl výsledek predikátu pravdivý. Zápis se provádí takto: klíčové slovo BETWEEN následované počáteční hodnotou, klíčovým slovem AND a koncovou hodnotou. Na rozdíl od IN je BETWEEN citlivý na pořadí, tudíž první hodnota v klauzuli musí být první v abecedním nebo číselném pořadí (všimněte si, že na rozdíl od angličtiny, SQL neříká hodnota „je mezi" hodnotou a hodnotou, ale jednoduše *hodnota* „mezi" *hodnotou a hodnotou*). Tento formát se používá rovněž u LIKE operátoru. (Klíčové slovo IS je rezervované pro použití v IS NULL operátoru). Následující

dotaz vybírá z tabulky „Salespeople" všechny prodejce s provizí mezi 0,10 a 0,12 (výstup ukazuje obr. 8.4):

```
SELECT *
     FROM Salespeople
     WHERE comm BETWEEN .10 AND .12;
```

OBRÁZEK 8.4: SELECT s použitím BETWEEN

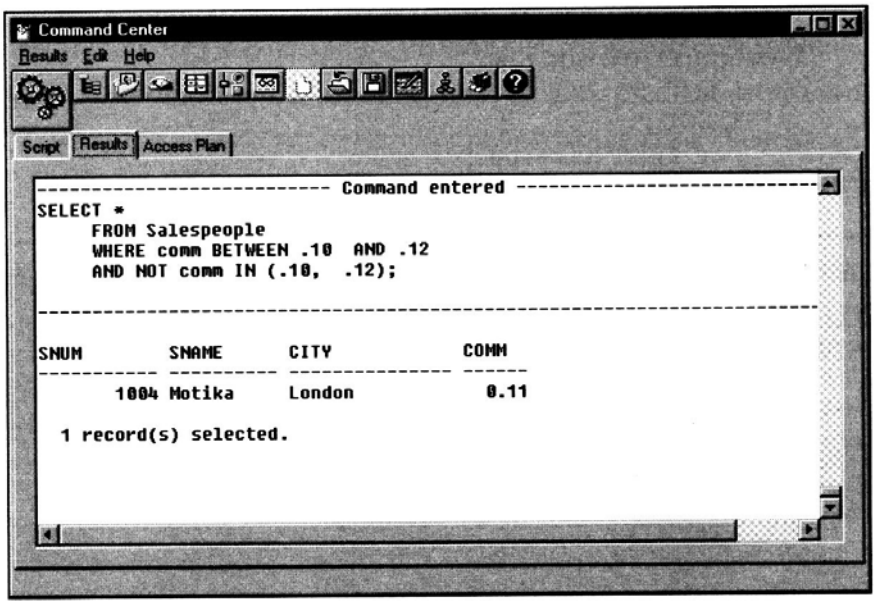


Všimněte si, že operátor BETWEEN je inkluzívní. Tj. hodnoty odpovídající jedné z dvou hraničních hodnot (v tomto případě .10 a .12) jsou součástí intervalu. SQL přímo nepodporuje neinkluzívní BETWEEN. Musíte buď definovat vaše hraniční hodnoty tak, že jejich inkluzívní interpretace je přijatelná nebo zadat něco takového:

```
SELECT *
     FROM Salespeople
     WHERE (comm BETWEEN .10, AND .12)
     AND NOT comm IN (.10, .12);
```

Výstup tohoto dotazu ukazuje obr. 8.5.

OBRÁZEK 8.5: Provedení BETWEEN neinkluzívním



Ano, tento příkaz je trochu neobratný, ale zároveň ukazuje, jak kombinovat nové operátory s booleovskými a tím vytvořit složitější predikát. V podstatě použijete IN a BETWEEN právě tak, jak to děláte s relačními operátory, chcete-li porovnat hodnoty.

U BETWEEN je důležité zdůraznit pořadí, v kterém jsou hraniční hodnoty. Vezměte si např. následující dotaz :

```
SELECT *
     FROM Salespeople
     WHERE comm BETWEEN .12 AND .10;
```

Tento zápis není ekvivalentní předcházejícímu příkladu. Ve skutečnosti se tato verze nikdy nevyhodnotí jako TRUE, protože 0,12 > 0,10. BETWEEN se interpretuje tak, že první hodnota předchází nebo se rovná cílové a druhá následuje nebo se rovná počáteční a nikdy ne naopak. Doslova se předcházející příklad s BETWEEN přeloží následovně:

```
SELECT *
     FROM Salespeople
     WHERE comm >= .12 AND comm <= .10;
```

To ovšem není pravda pro jakékoliv reálné číslo. Ačkoliv se to může zdát dost zřejmé, může to způsobit zmatek v aplikacích, kde čísla použitá v SQL pří-

kazu mohou být proměnné namísto konstant. Princip, který si musíte zapamatovat, praví, že první porovnávaná hodnota v klauzuli BETWEEN musí být vždy menší než druhá.

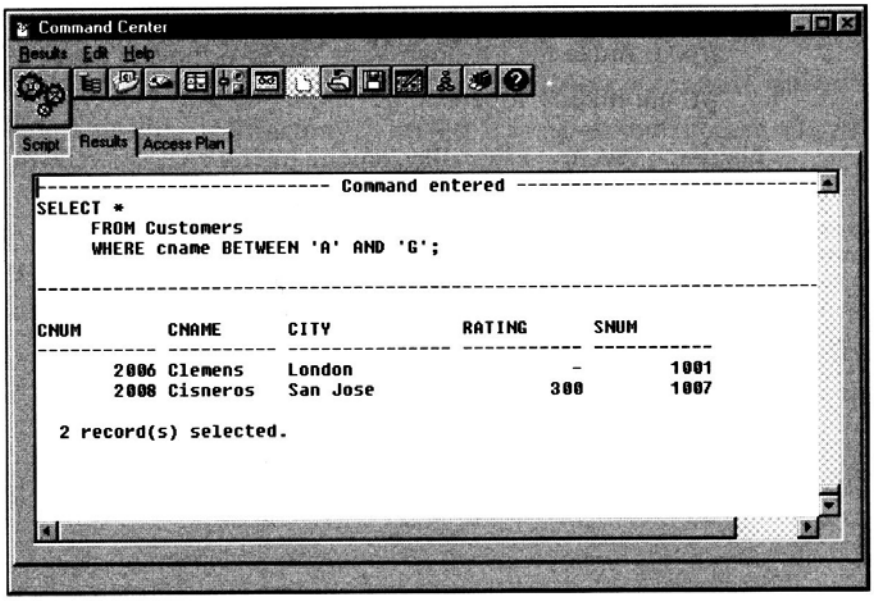
Operátor BETWEEN pracuje jako relační také se znakovými sloupci podle podmínek řazení. To znamená, že jej můžete použít k výběru rozsahů z abecedního řazení. Je důležité, abyste to udělali v souladu s použitím velkých písmen.

Tento dotaz vybírá všechny zákazníky, jejichž jména spadají do určitého abecedního rozsahu:

```
SELECT *
     FROM Customers
     WHERE cname BETWEEN 'A' AND 'G';
```

Výstup tohoto dotazu je ukázán na obr. 8.6.

OBRÁZEK 8.6: Použití BETWEEN alfabeticky



Všimněte si, že Giovanni a Grass jsou vynecháni, ačkoliv je BETWEEN inkluzívní. Je to kvůli způsobu jakým BETWEEN porovnává řetězce různé délky. Řetězec 'G' je kratší než řetězec 'Giovanni', takže BETWEEN doplní 'G' mezerami. Mezery předcházejí písmena abecedy (ve většině běžných řazení), takže se Giovanni nevybere. To samé platí pro Grasse. Je důležité si to zapamatovat, pokud používáte BETWEEN k výběru znakových řetězců. Obvykle zapíšete pís-

meno bezprostředně následující za posledním, které chcete zahrnout nebo přidat do vaší druhé hraniční hodnoty. Jestliže váš systém plně vyhovuje SQL92, konkrétní řazení bude obsahovat PAD ATTRIBUTE, který určuje, zda dojde k doplnění mezerami, jak jsme popsali výše. Více informací viz Řazení v referenční příručce.

Operátor LIKE

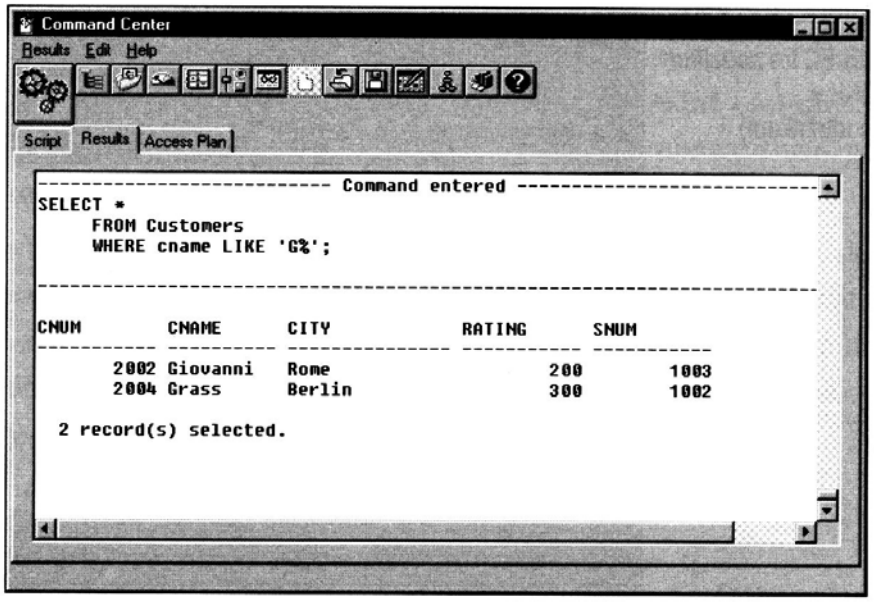
LIKE se používá s textovými datovými typy a slouží k vyhledávání podřetězců. Jinými slovy hledá text, který odpovídá části řetězce. Přitom se používají speciální *zástupné znaky (wild cards).* S operátorem LIKE se používají dva zástupné znaky:

- Podtržítko „_" zastupuje jakýmkoliv jednoduchý znak. Např. 'b_t' odpovídá 'byt' nebo 'bit', ale neodpovídá řetězci 'bajt'.
- Znak procento „%" zastupuje posloupnost libovolného počtu znaků. Počet znaků může být i nulový. Vzor '%p%t' odpovídá řetězců 'pat', 'opat' nebo 'pt', ale nikoliv 'lopata'.

Najděme všechny zákazníky, jejichž jména začínají velkým *G* (výstup ukazuje obr. 8.7):

```
SELECT *
     FROM Customers
     WHERE cname LIKE 'G%';
```

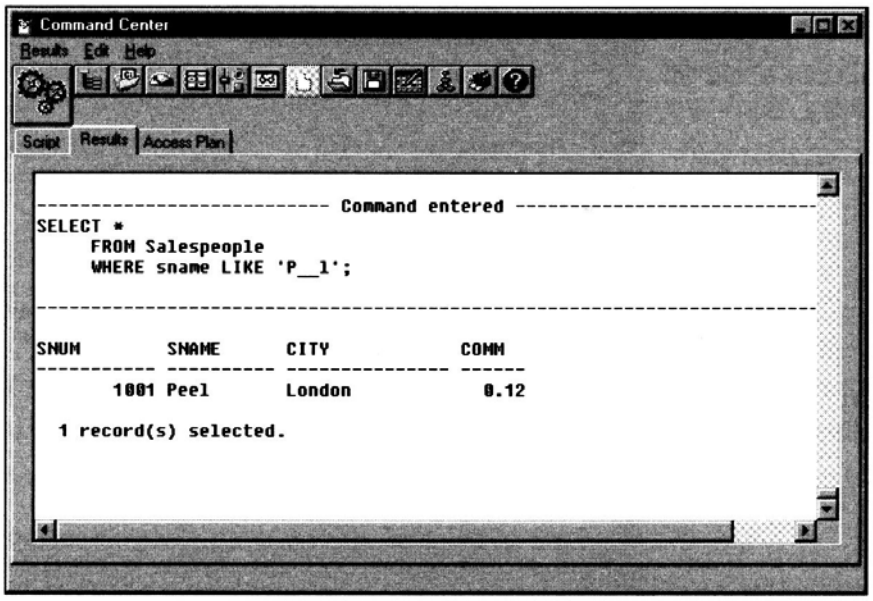
OBRÁZEK 8.7:
SELECT s použitím LIKE s „%“



LIKE může být užitečný při hledání např. jména nebo jiných hodnot, když si nemůžete vzpomenout na celé znění. Předpokládejme, že si nejste jisti, zda se jméno jednoho vašeho prodejce píše Peal nebo Peel. Můžete jednoduše aplikovat vaši novou znalost týkající se zástupných znaků (výstup tohoto dotazu ukazuje obr. 8.8):

```
SELECT *
     FROM Salespeople
     WHERE sname LIKE 'P__l';
```

OBRÁZEK 8.8:
SELECT s použitím LIKE s „_“ (podtržítkem)



Každé podtržítko představuje pouze jediný znak, takže např. jméno Prettel by se nevybralo. V mnoha implementacích je nutno na konec řetězce ještě dopsat zástupný znak „%“, pokud délka sloupce sname přesahuje počet znaků ve jménu Peel a sloupec sname má textový datový typ pevné délky (CHAR). V takovém případě je totiž sloupcová hodnota sname ve skutečnosti uložena jako jméno Peel následované řadou mezer tak, aby celkový počet znaků dosáhl pevné délky specifikované pro sloupec. Tudíž znak 'l' se nepovažuje za konec řetězce. Zástupný znak „%“ jednoduše akceptuje všechny doplněné mezery. To by nebylo nutné, jestliže by sloupec sname byl typu VARCHAR.

Co dělat, když potřebujte mít ve vzoru znak procento nebo podtržítko? V predikátu LIKE můžete definovat jakýkoliv znak jako *escape znak*. Escape znak se používá bezprostředně před zástupným znakem a říká, že se má zástupný znak interpretovat doslovně. Např. bychom mohli náš sloupec sname následovně otestovat na přítomnost podtržítek:

```
SELECT *
     FROM Salespeople
     WHERE sname LIKE '%/_%' ESCAPE '/';
```

Ze současných dat se nic nevybere, protože jsme do jmen našich prodejců nezahrnuli žádná podtržítka. Klauzule ESCAPE definuje „/“ jako escape znak. Escape znak se používá v LIKE řetězci následován procentem, podtržítkem nebo

sám sebou (jak hned vysvětlíme). Kdykoliv se v řetězci vyskytne takový znak, vyhledá se jako literál místo zástupného znaku s obvyklým významem. Escape znak musí být pouze jeden a mění význam pouze bezprostředně následujícího znaku. V předchozím příkladu mají znaky procenta stále zástupný význam, pouze podtržítko zobrazuje samo sebe.

Jak jsme se zmínili, escape znak může být použit sám na sebe. Jinými slovy, když chcete ve sloupci najít escape znak, vložíte jej jednoduše dvakrát. První výskyt escape znaku znamená „vezmi následující znak doslovně jako znak" a druhý je samotný escape. Zde je předcházející příklad přeformulovaný tak, aby hledal výskyty řetězce _/ ve sloupci sname:

```
SELECT *
     FROM Salespeople
     WHERE sname LIKE '%/_//%' ESCAPE '/';
```

V současných datech se opět nic nenajde. Vzor hledá jakoukoliv posloupnost znaků (%), následovanou podtržítkem (/_), escape znakem (//) a jakoukoliv posloupností koncových znaků (%).

Operátor IS NULL

Jak jsme probrali v předcházející kapitole, když se NULL porovnává s jakoukoliv hodnotou, dokonce i s jinou NULL hodnotou, výsledek není ani TRUE ani FALSE, ale *UNKNOWN*. Často potřebujete rozlišit mezi FALSE a UNKNOWN. Tj. oddělit řádky obsahující sloupcové hodnoty, které neuspějí v predikátové podmínce, a ty, které obsahují NULL. Z tohoto důvodu SQL poskytuje speciální operátor IS s klíčovým slovem NULL, který slouží k nalezení NULL hodnot.

Abychom nalezli všechny věty v naší tabulce „Customers" mající ve sloupci city NULL hodnoty, mohli bychom zadat:

```
SELECT *
     FROM Customers
     WHERE city IS NULL;
```

Tento dotaz momentálně nevytvoří výstup, protože nemáme žádné NULL hodnoty v sloupci city. NULL hodnoty jsou ale velmi důležité a vrátíme se k nim později.

Použití NOT se speciálními operátory

Speciální operátory, které jsme probrali v této kapitole, mohou být následovány booleovským operátorem NOT. Oproti tomu relační operátory musí mít NOT před celým výrazem. Např. když chceme vyloučit NULL hodnoty z našeho výstupu, místo toho abychom je hledali, použijeme NOT k obracení významu operátoru:

```
SELECT *
     FROM Customers
     WHERE city IS NOT NULL;
```

V nepřítomnosti NULL hodnot (což je momentálně náš případ) tento dotaz zobrazí celou tabulku „Customers". Ekvivalentní zápis:

```
SELECT *
     FROM Customers
     WHERE NOT city IS NULL;
```

je také přijatelný.

V prostřední úrovni shody se SQL92 umí IS NULL kromě jednoduchého testování jednotlivých hodnot testovat i soubory hodnot. Některé produkty podporují tuto funkčnost, ač oficiálně prostřední úrovně shody se standardem nedosahují. V takovém případě *NOT hodnota IS NULL* nemusí nutně odpovídat výrazu *hodnota* IS NOT NULL. Na podrobnosti se podívejte do referenční příručky do oddílu Predikáty.

Můžeme také použít NOT s IN:

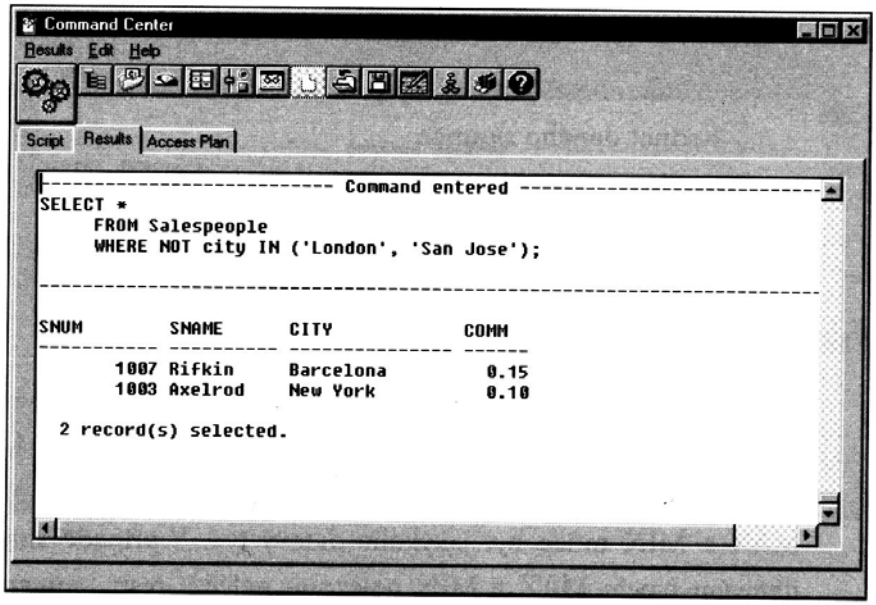
```
SELECT *
     FROM Salespeople
     WHERE city NOT IN ('London', 'San Jose');
```

To samé jinak:

```
SELECT *
     FROM Salespeople
     WHERE NOT city IN ('London', 'San Jose');
```

Výstup tohoto dotaz ukazuje obr. 8.9.

OBRÁZEK 8.9:
Použití NOT s IN



Stejným způsobem můžete použít NOT BETWEEN a NOT LIKE. Mimochodem tyto dotazy nezobrazí žádné řádky, ve kterých má sloupec city hodnotu NULL. Predikát NULL IN ('London', 'San Jose') má totiž hodnotu UNKNOWN. Tudíž predikát NOT (NULL IN ('London', 'San Jose')) je také UNKNOWN, jak vyplývá z pravdivostní tabulky třístavové logiky z předcházející kapitoly. Znovu opakujeme, že, když musíte pracovat s NULL hodnotami, používejte IS NULL operátor. Když byste chtěli zahrnout NULL hodnoty sloupce city mezi ty, které nejsou ani v San Jose ani v Londýně, mohli byste dotaz přeformulovat takto:

```
SELECT *
     FROM Salespeople
     WHERE NOT city IN ('London', 'San Jose')
     OR city IS NULL;
```

Sumarizace dat agregačními funkcemi

Dotaz může vytvářet zobecnění skupiny hodnot stejně jako nové sloupcové hodnoty. Činí tak použitím *agregačních funkcí*. Agregační funkce vytváří jednu hodnotu z celé skupiny záznamů. Zde je seznam těchto funkcí:

- COUNT zobrazí počet řádků nebo ne-NULL sloupcových hodnot, které dotaz vybral.
- SUM zobrazí součet všech vybraných hodnot daného sloupce.
- AVG zobrazí aritmetický průměr (střední hodnotu) všech vybraných hodnot daného sloupce.
- MAX zobrazí největší ze všech vybraných hodnot daného sloupce.
- MIN zobrazí nejmenší ze všech vybraných hodnot daného sloupce.

Jak používat agregační funkce?

Agregační funkce se v dotazech používají podobně jako jména sloupců v klauzuli SELECT. Jako argumenty funkcí figurují jména sloupců. SUM a AVG mohou používat pouze sloupce s číselnými datovými typy. Argumentem COUNT, MAX a MIN může být jakýkoliv datový typ. V případě sloupců se znakovým datovým typem MAX a MIN naleznou největší resp. nejmenší znakový řetězec na základě abecedního pořádku (více informací viz Řazení v referenční příručce). NULL hodnoty se všech případech ignorují.

Abychom zobrazili součet všech našich nákupů z tabulky „Orders", mohli bychom zadat následující dotaz, jehož výstup je ukázán na obr. 8.10:

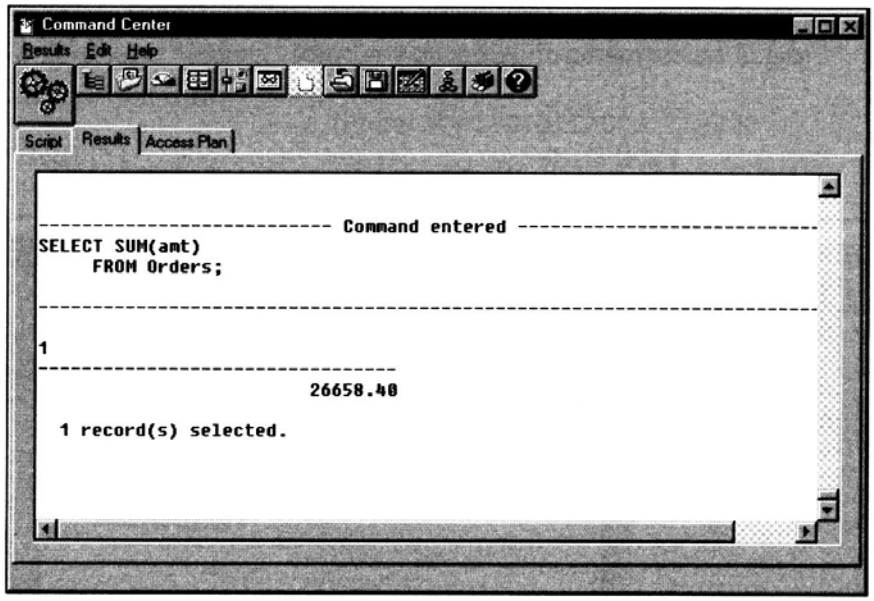
```
SELECT SUM(amt)
     FROM Orders;
```

Tento dotaz se od výběru sloupce liší v tom, že vrací jedinou hodnotu, bez ohledu na to, kolik řádků je v tabulce. Stručně řečeno vzhledem k tomuto omezení nemůžete použít agregační funkce a s neagregovanými sloupci najednou, pokud zároveň nezadáte klauzuli GROUP BY.

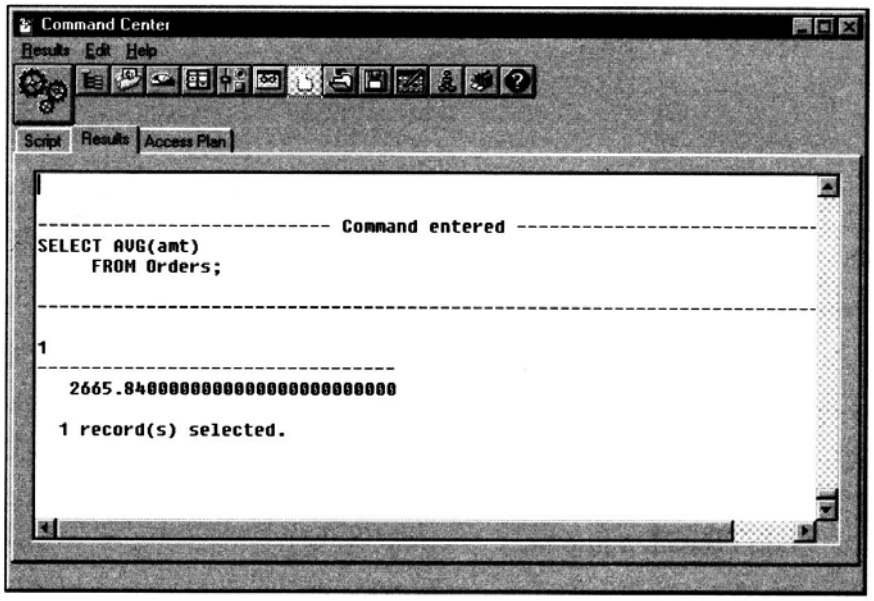
Nalezení průměrného množství je podobná operace (výstup následujícího dotazu je ukázán na obr. 8.11):

```
SELECT AVG(amt)
     FROM Orders;
```

OBRÁZEK 8.10: Výběr SUM (součtu)



OBRÁZEK 8.11: Výběr AVG (průměru)



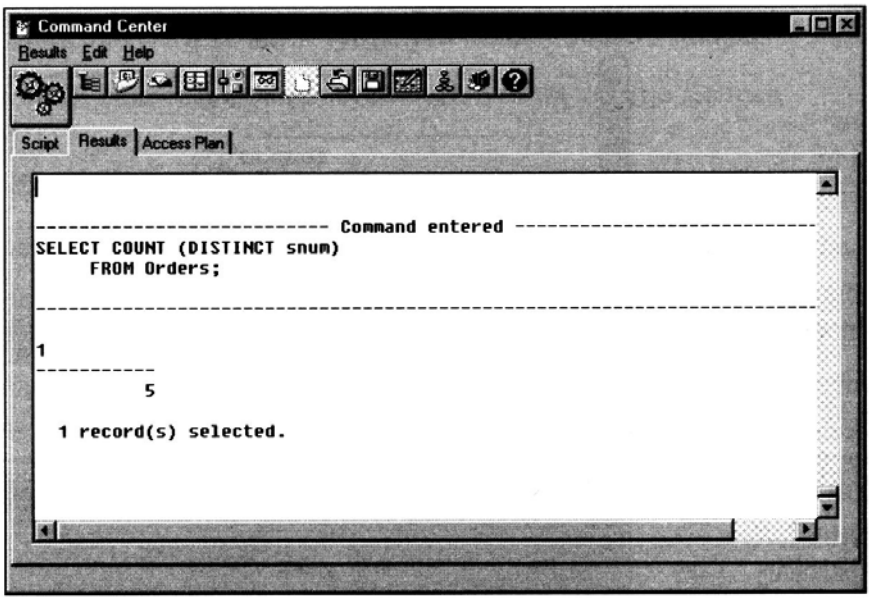
Speciální vlastnosti COUNT

Funkce COUNT je trochu jiná. Počítá množství hodnot v daném sloupci nebo počet řádků v tabulce. Jestliže počítáte sloupcové hodnoty, můžete ji kombinovat s DISTINCT, abyste získali počet různých hodnoty v daném sloupci.

Výpočet hodnot Mohli bychom COUNT použít např. ke zjištění počtu prodejců na seznamu objednávek v tabulce „Orders" (výstup je ukázán na obr. 8.12):

```
SELECT COUNT(DISTINCT snum)
     FROM Orders;
```

OBRÁZEK 8.12: Počítání sloupcových hodnot



Pokud bychom nespecifikovali DISTINCT, obdrželi bychom počet všech řádků, které neobsahují NULL hodnotu v sloupci snum. To by byl počet objednávek přiřazených jakémukoliv prodejci, který se ale pravděpodobně liší od počtu prodejců.

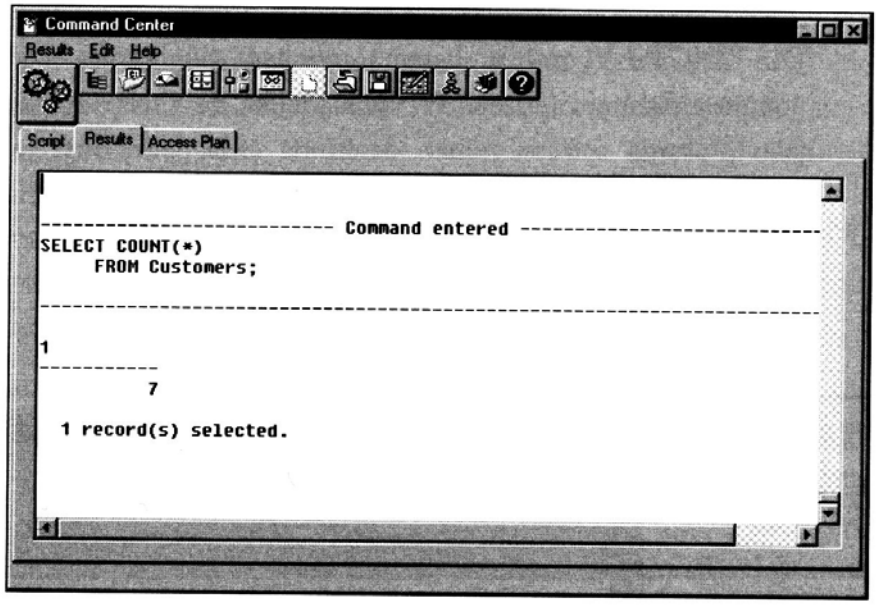
Všimněte si, že jsme v předchozím příkladu umístili DISTINCT spolu se jménem sloupce do závorek nikoliv bezprostředně po klíčovém slovu SELECT, jak tomu bylo dosud. V jednom dotazu můžete aplikovat COUNT na více sloupců. Zároveň můžete dle potřeby míchat agregační funkce z nichž některé použijí DISTINCT a jiné ne.

Tímto způsobem lze DISTINCT použít s jakoukoliv agregační funkcí, ale nejčastěji to asi bude s COUNT. Na MAX a MIN nemá žádný vliv. V SUM a AVG obvykle chcete zahrnout i opakované hodnoty, protože ovlivňují celkový součet a průměr všech hodnot.

Počítání řádků Chcete-li zjistit celkový počet řádků tabulky, použijte funkci COUNT s hvězdičkou namísto jména sloupce jako v následujícím příkladě, jehož výstup ukazuje obr. 8.13:

```
SELECT COUNT(*)
     FROM Customers;
```

OBRÁZEK 8.13: Počítání řádků namísto hodnot



COUNT s hvězdičkou zahrnuje jak NULL hodnoty tak duplikáty, tudíž nemůžete použít DISTINCT. Z tohoto důvodu může vytvořit číslo větší než COUNT konkrétního sloupce, který vylučuje řádky, které v tomto sloupci obsahují NULL hodnoty a může také eliminovat duplikáty.

Zahrnutí duplikátů do agregačních funkcí Agregační funkce akceptují argument ALL, který se umísťuje před jméno sloupce jako DISTINCT, ale znamená opak – zahrnout duplikáty. Rozdíly mezi ALL a „*" v použití s COUNT jsou následující:

- ALL stále akceptuje jméno sloupce jako argument
- ALL nezapočítá NULL hodnoty

Hvězdička je jediný argument, který zahrnuje NULL hodnoty a používá se s COUNT. Ostatní agregační funkce si NULL hodnot nevšímají. Následující příkaz zobrazí počet všech hodnocení (rating) v tabulce „Customers" včetně duplikátů vyjma NULL hodnot:

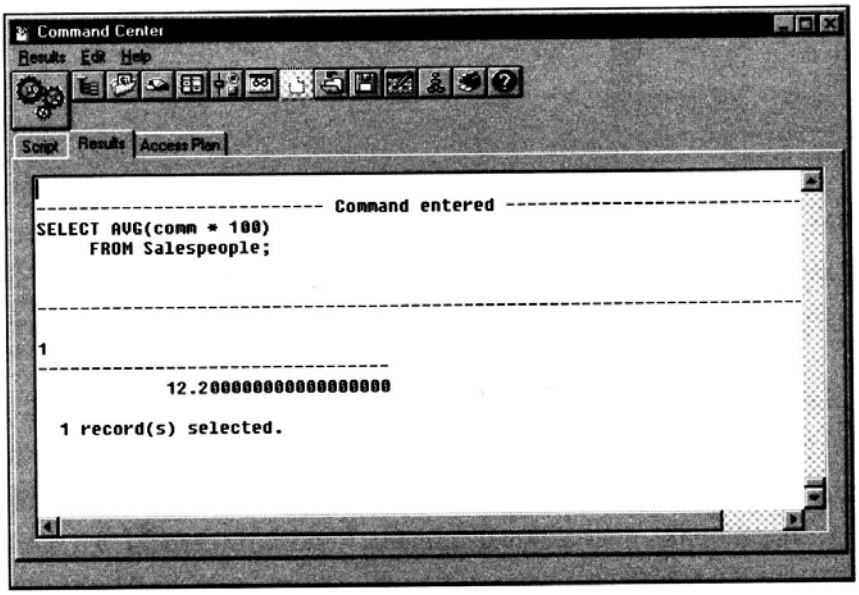
```
SELECT COUNT(ALL rating)
    FROM Customers;
```

Agregace založené na výrazech

Až dosud jste používali agregační funkce s jednotlivými sloupci jako argumenty. Argument agregační funkce může být složen ze skalárních výrazů zahrnujících jeden nebo více sloupců. (V takovém případě není povolen modifikátor DISTINCT.) Výrazy probereme mnohem důkladněji v příští kapitole. Nyní použití výrazů jen okusíte. Předpokládejme, že chcete zobrazit hodnoty provizí jako procenta spíše než jako desetinná čísla. Jednoduše je vynásobíme stem. Když chcete najít průměrnou provizi a vyjádřit ji jako procento, mohli byste použít následující dotaz, jehož výstup ukazuje obr. 8.14:

```
SELECT AVG(comm * 100)
    FROM Salespeople;
```

OBRÁZEK 8.14: Agregace založené na výrazu



Předcházející dotaz byl přepracován, aby ilustroval použití výrazů v agregačních funkcích. Jak se ovšem stává, není zvoleno ideální místo. Bylo by vhodnější napsat

```
AVG(comm) * 100
```

než napsat

```
AVG(comm * 100)
```

V druhém případě se matematický výpočet provádí pouze jedenkrát. Ačkoliv je možné, že váš DBMS bude dost chytrý, aby to pochopil a provedl druhou verzi i když jste zapsali první, nespoléhejte na to.

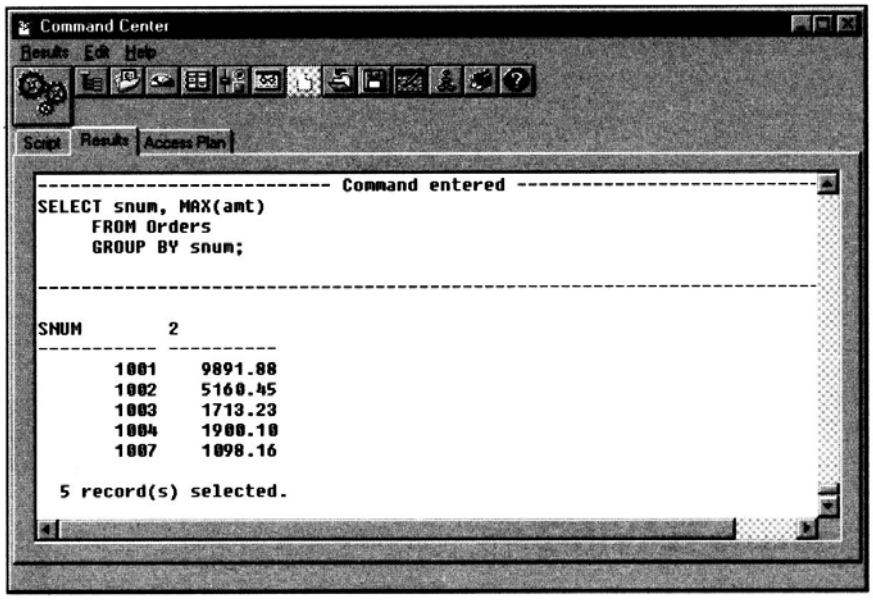
Klauzule GROUP BY

Klauzule GROUP BY umožňuje definovat podmnožinu hodnot v konkrétním sloupci a použít agregační funkci na tuto podmnožinu. Podmnožina je skupina hodnot jednoho sloupce, která má společnou hodnotu jiného sloupce (popř. jiných sloupců). Např. předpokládejme, že chcete nalézt největší objednávku každého prodejce. Mohli byste vytvořit separátní dotaz pro každého prodejce zvlášť výběrem MAX(amt) z tabulky „Orders" postupně pro všechny hodnoty snum. GROUP BY to ovšem udělá za vás v jednom příkazu. Seskupíte dotaz podle hodnot snum a potom vypočítáte MAX odděleně pro každou skupinu. Zde je kód:

```
SELECT snum, MAX(amt)
     FROM Orders
     GROUP BY snum;
```

Výstup tohoto dotazu ukazuje obr. 8.15.

OBRÁZEK 8.15: Nalezení maximálního množství pro každého prodejce



Jak vidíte GROUP BY použije agregační funkce nezávisle na každou skupinu, která má společnou sloupcovou hodnotu. V našem příkladu skupinu tvoří

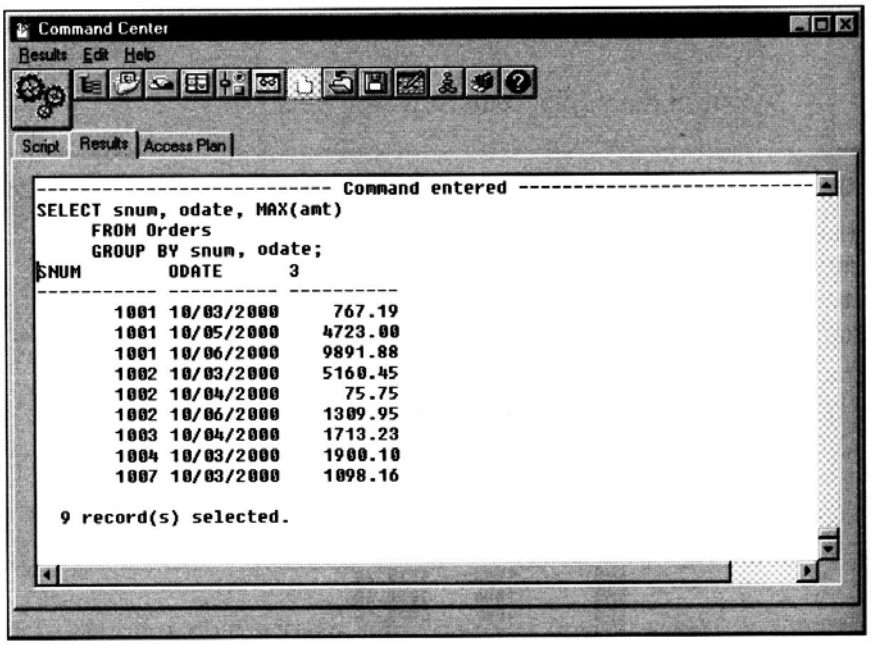
všechny řádky se stejnou hodnotou snum. Funkce MAX se použije odděleně na každou takovou skupinu. To znamená, že sloupec, na který se použije GROUP BY, má z definice jen jednu hodnotu za celou výstupní skupinu stejně jako agregační funkce. Výsledkem je kompatibilita mezi agregovanou hodnotou a tímto způsobem upraveným sloupcem.

Také můžete použít GROUP BY s několika sloupci. Předěláme předchozí příklad tak, že předpokládáme, že chcete vidět největší objednávku každého prodejce v každém dni. Seskupíte tedy tabulku „Orders" podle kombinace data a prodejce a použijete funkci MAX na každou skupinu:

```
SELECT snum, odate, MAX(amt)
     FROM Orders
     GROUP BY snum, odate;
```

Výstup tohoto dotazu je ukázán na obr. 8.16. Ovšem prázdné skupiny, tj. data, kdy zpracovávaný prodejce neměl žádné objednávky, se nezobrazí.

OBRÁZEK 8.16: Nalezení největších objednávek prodejce za jednotlivé dny



Klauzule HAVING

Předpokládejme, že byste v předcházejícím příkladu chtěli vidět pouze maximální nákupy, které přesahují částku 3000 dolarů. Nezapíšete-li poddotaz, který bude vysvětlen později, nemůžete ve WHERE klauzuli použít agregační funkce, protože

predikáty se vyhodnocují pro každý řádek zvlášť, zatímco agregační funkce pro skupinu řádků. To znamená, že zápis podobný následujícímu se odmítne jako chybný:

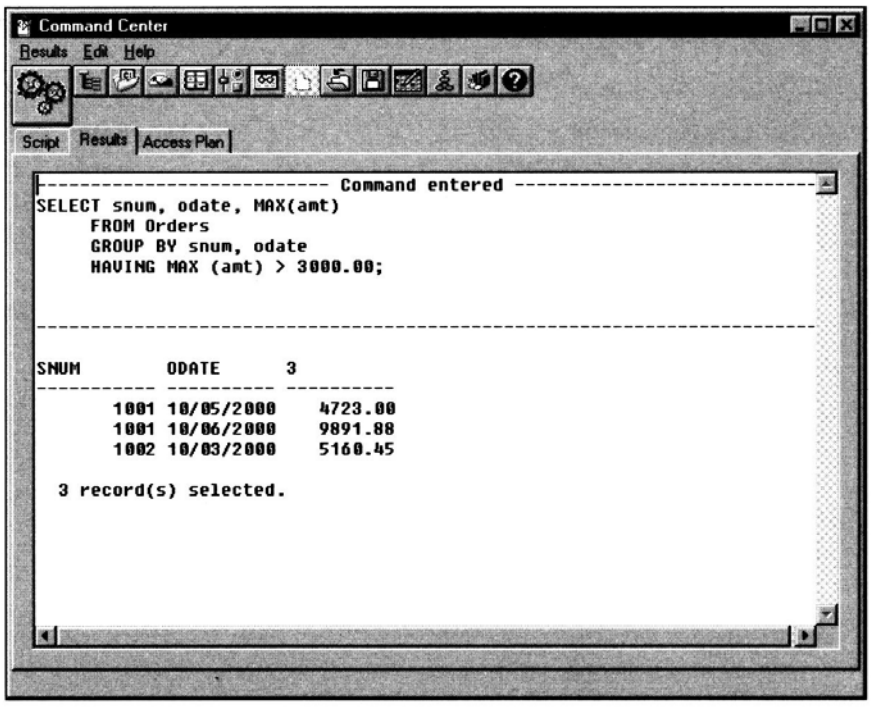
```
SELECT snum, odate, MAX(amt)
     FROM Orders
     WHERE MAX(amt) > 3000.00
     GROUP BY snum, odate;
```

Ve striktní ISO interpretaci SQL by byl tento dotaz odmítnut. Každopádně je neobratný a nelogický. Klauzulí WHERE se pokoušíme filtrovat řádky *před* tím než byly zformovány do skupin pomocí GROUP BY. Abyste např. zobrazili maximální nákupy přes 3000 dolarů, vytvořte filtr použitím klauzule HAVING. Tato klauzule definuje kritéria použitá k vyloučení určitých skupin z výstupu, podobně jako to klauzule WHERE dělá s individuálními řádky. Následuje správný příkaz:

```
SELECT snum, odate, MAX(amt)
     FROM Orders
     GROUP BY snum, odate
     HAVING MAX (amt) > 3000.00;
```

Výstup tohoto dotazu ukazuje obr. 8.17.

OBRÁZEK 8.17: Vylučování skupin agregovaných hodnot



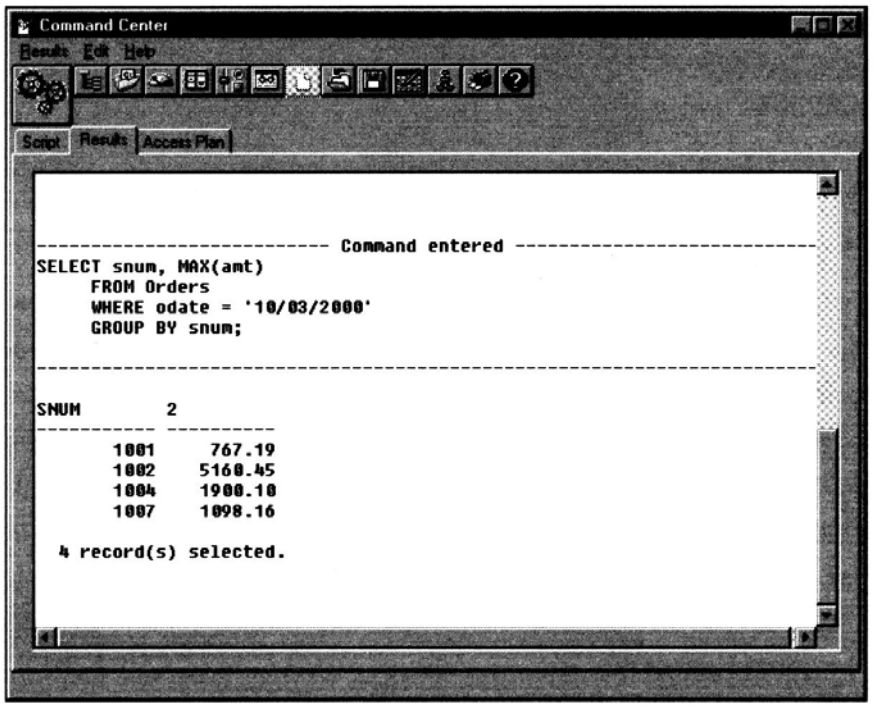
Argumenty klauzule HAVING dodržují pravidla původně zavedená pro klauzuli GROUP BY v příkazu SELECT. Musí mít jednu hodnotu za každou výstupní skupinu. Následující příkaz by byl nedovolený:

```
SELECT snum, MAX(amt)
     FROM Orders
     GROUP BY snum
     HAVING odate = '10/03/2000';
```

Klauzule HAVING se nemůže odkazovat na sloupec odate, protože ten může mít (a skutečně má) víc než jednu hodnotu na výstupní skupinu. Pro danou skupinu může sloupec odate být jak roven, tak různý od hodnoty '10/03/2000', v závislosti na členovi skupiny, o kterého se zajímáte. Abychom se vyhnuli této situaci, konstruujeme klauzuli HAVING tak, aby odkazovala jen na samotné agregáty a na sloupce vybrané pomocí GROUP BY. Níže je uveden správný způsob, jak formulovat předchozí dotaz (výstup je ukázán na obr. 8.18):

```
SELECT snum, MAX(amt)
     FROM Orders
     WHERE odate = '10/03/2000'
     GROUP BY snum;
```

OBRÁZEK 8.18: Maximum za každého prodejce za 3. října

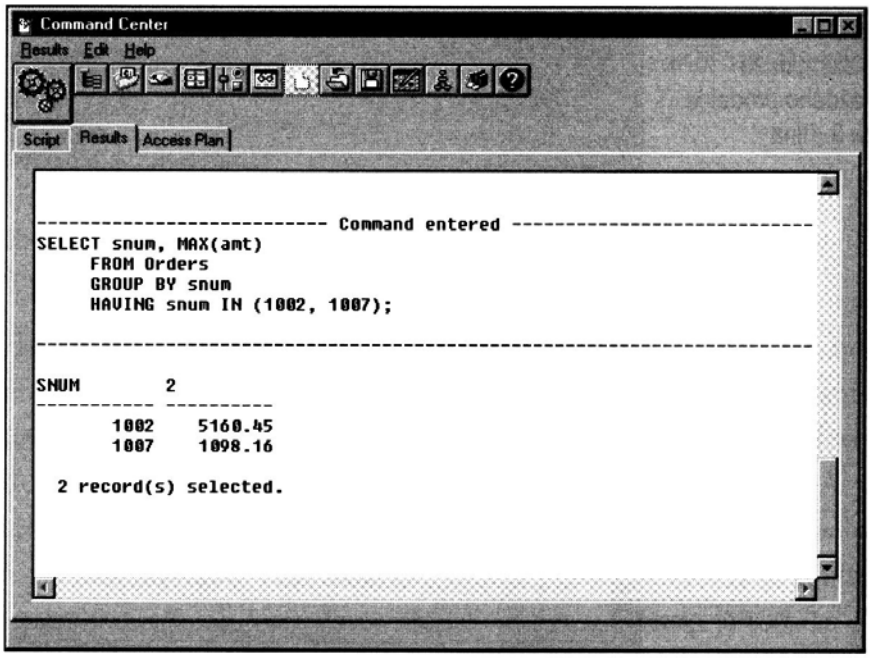


Jak jsme se již zmínili HAVING akceptuje pouze argumenty, které mají jednu hodnotu pro každou výstupní skupinu. V praxi jsou sice odkazy na agregační funkce nejobvyklejší, ale výběr sloupců klauzulí GROUP BY je také povolen. Např. bychom mohli zjistit největší objednávky Serrese a Rifkina následujícím příkazem:

```
SELECT snum, MAX(amt)
     FROM Orders
     GROUP BY snum
     HAVING snum IN (1002, 1007);
```

Výstup dotazu ukazuje obr. 8.19.

OBRÁZEK 8.19: Použití HAVING s GROUP BY



Nevnořujte agregáty

Ve striktní interpretaci ISO SQL nemůžete vytvořit agregát agregátu. Předpokládejme, že chcete najít den, ve kterém bylo dosaženo nejvyššího součtu objednávek. Jestliže se pokusíte zadat:

```
SELECT odate, MAX( SUM(amt))
     FROM Orders
     GROUP BY odate;
```

bude váš příkaz pravděpodobně odmítnut. (Některé implementace toto omezení nevynucují. Může to být výhodné, protože vnořené agregáty mohou být navzdory své problematičnosti užitečné.) V předchozím příkazu se má SUM použít pro každou skupinu hodnot sloupce odate a MAX má vybrat nejvyšší součet. Klauzule GROUP BY naznačuje, že by měl existovat jeden řádek výstupu pro každou skupinu hodnot sloupce odate.

Shrnutí

Nyní umíte konstruovat predikáty na základě vztahů speciálně definovaných pomocí SQL. Umíte vyhledávat hodnoty v určitém rozsahu (BETWEEN) nebo ve

výčtovém souboru (IN) nebo hledat znakové hodnoty, které odpovídají textu v rozsahu parametrů, které definujete (LIKE). Také jste se naučili něco o speciální funkčnosti, kterou SQL potřebuje k řešení chybějících dat, které jsou realitou databázového světa. K tomuto účelu se hodí NULL hodnoty. Umíte vybírat nebo vylučovat NULL hodnoty z vašeho výstupu operátorem IS NULL (nebo IS NOT NULL).

Jste také schopni používat dotazy trochu odlišně k odvozování dat místo jejich pouhého vyhledávání. Tento silný prostředek sahá nad rámec popisu agregačních funkcí uvedeného v této kapitole. To znamená, že není nezbytné sledovat určité informace, když umíte formulovat dotaz, který vám je dokáže odvodit. Dotaz vám zobrazí výsledky během chvilky, zatímco tabulky součtů nebo průměrů obsahují věrohodnou informaci, pouze jsou-li pravidelně aktualizovány. Nedoporučuje se, aby agregační funkce zcela nahradily nezávislé sledování takové informace. Nejlepší řešení spočívá ve vhodné kombinaci obou prostředků.

Umíte používat agregáty skupin hodnot definovaných klauzulí GROUP BY. Tyto skupiny mají společnou sloupcovou hodnotu a mohou se nacházet v rámci jiných skupin majících společnou hodnotu jiného sloupce. Predikáty stále slouží k určení, které řádky použije agregační funkce. Kombinace dosud uvedených vlastností vám umožňuje vytvořit agregáty založené na úzce definovaných podmnožinách hodnot ve sloupci. Dále umíte definovat další podmínky k vyloučení určitých výsledných skupin klauzulí HAVING.

Nyní se z vás stali v mnoha stránkách zkušení mistři v tvorbě hodnot dotazem. V příští kapitole vám ukážeme některé věci, které můžete s vytvořenými hodnotami dělat.

Práce s SQL

1. Napište dva různé dotazy, které zobrazí všechny objednávky ze 3. nebo 4. října 2000.
2. Napište dotaz, který zobrazí všechny zákazníky, jejichž jméno má počáteční písmeno v intervalu od A do G.
3. Napište dotaz, který vybere všechny zákazníky, jejichž jména začínají buď malým nebo velkým písmenem C.

4. Napište dotaz, který vybere všechny objednávky, které mají ve sloupci amt uvedenu hodnotu nula nebo NULL. Tj. vyberte objednávky s nulovým množstvím.
5. Napište dotaz, který zobrazí počet záznamů objednávek za 3. října.
6. Napište dotaz, který zobrazí počet záznamů v tabulce „Customers". Vyberte pouze záznamy, které ve sloupci city nemají hodnotu NULL.
7. Napište dotaz, který zobrazí nejmenší objednávku každého zákazníka.
8. Napište dotaz, který vybere prvního zákazníka v abecedním pořadí se jménem začínajícím na G.
9. Napište dotaz, který vybere nejvyšší hodnocení (rating) v každém městě (city).
10. Napište dotaz, který za každý den zobrazí počet prodejců, kteří měli objednávky. (Jestli má prodejce víc než jednu objednávku, započítá se jen jednou.)