Databáze standardu SQL, díl 14.

Spojování tabulek v SQL

Možná že jste v minulých dílech seriálu zlořečili tabulkám v 5NF, protože jejich dokonalost je vykoupena nedostatkem sloupců. Pokud budeme chtít např. evidovat telefonní kontakty na lidi pracující u firmy, nezbude než pracovat s dvěma tabulkami v 5NF.

V první tabulce OSOBA bude unikátním klíčem CISLO_PRAC a ještě zde bude uvedeno JMENO zaměstnance. V druhé tabulce KONTAKT tvoří složený unikátní klíč dvojice sloupců CISLO_P a TELEFON. Dnes nás bude zajímat odezva na různé dotazy do tabulek. Po provedení příkazů

CISLO_PRAC	JMENO
6	JOE
30	JIM
7	TOM

Tabulka 1: OSOBA.

CISLO_P	TELEFON
30	5164
36	3674
30	6541
7	4714

Tabulka 2: KONTAKT.

```
SELECT * FROM OSOBA;

SELECT * FROM KONTAKT;
```

obdržíme údaje, které jsou uvedeny v tabulkách 1 a 2.
Jistě vám neuniklo, že pracovník JOE, chudák, nemá telefon a že telefonní číslo 3674 patří neznámé osobě. Dnes se naučíme pokládat dotazy do více než jedné tabulky. Ten, kdo chce vidět všechno, a nemít z toho nic, použije jako zdroj informace jak tabulku OSOBA, tak tabulku KONTAKT. Tomu odpovídá platný příkaz:

```
SELECT * FROM OSOBA,
KONTAKT;
```

Nikoho nepřekvapí, že výsledná tabulka 3 obsahuje tolik sloupců, kolik jich mají obě tabulky dohromady. Počet řádků je však roven součinu počtu řádků v jednotlivých tabulkách. Došlo totiž k vygenerování všech 3*4=12 možných dvojic řádků z obou tabulek. Jde o matematikům důvěrně známý kartézský součin množiny řádků jedné a druhé tabulky. Pokud si představíme střední firmu, kde počet zaměstnanců je roven 1000 a počet kontaktů je roven 500, dostaneme tabulku obsahující 500 000 řádků a 4 sloupce. V tabulce 3 je ale větší problém v chaotičnosti a nadbytečnosti dat. V praxi totiž zpravidla potřebujeme spojit JMENO a TELEFON prostřednictvím čísla pracovníka.

CISLO_PRAC	JMENO	CISLO_P	TELEFON
6	JOE	30	5164
6	JOE	36	3674
6	JOE	30	6541
6	JOE	7	4714
30	JIM	30	5164
30	JIM	36	3674
30	JIM	30	6541
30	JIM	7	4714
7	TOM	30	5164
7	TOM	36	3674
7	TOM	30	6541
7	TOM	7	4714

Tabulka 3: Maximalizmus.

Znamená to tedy propustit na výstup pouze ty řádky, ve kterých se shoduje CISLO_PRAC a CISLO_P. K tomu by mohl sloužit následující příkaz:

CISLO_PRAC	JMENO	CISLO_P	TELEFON
30	JIM	30	5164
30	JIM	30	6541
7	TOM	7	4714

Tabulka 4: Na koho není kontakt, ten neexistuje.

```
SELECT * FROM OSOBA, KONTAKT
WHERE CISLO_PRAC=CISLO_P;
```

V tabulce 4 s hrůzou zjistíme, že zmizely jakékoli stopy po Joeovi a navíc i telefonní linka 3674.
Pokud nás zajímají opravdu jenom platné kontakty, měli bychom mít naopak radost. Pokud nám vadí zcela zbytečný sloupec CISLO_P, musíme vyjmenovat jednotlivé sloupce:

```
SELECT CISLO_PRAC, JMENO, TELE-
FON FROM OSOBA, KONTAKT WHERE
CISLO_PRAC=CISLO_P;
```

Pro jednoduchost stačí požadovat všechny sloupce z tabulky OSOBA a navíc ještě TELEFON:

```
SELECT OSOBA .*, TELEFON
FROM OSOBA, KONTAKT WHE-
RE CISLO_PRAC=CISLO_P;
```

My ale už známe pravidla projekce, restrikce řádků, agregace, vytváření skupin, restrikce skupin, práce s NULL a zahnízďování, a tak snadno vytvoříme složitější dotazy do dvou tabulek. Chceme-li vědět, jak se dovolat Quidovi, stačí napsat:

```
SELECT TELEFON
FROM OSOBA, KONTAKT
WHERE CISLO_PRAC=CISLO_P AND
JMENO="QUIDO"
ORDER BY TELEFON;
```

Můžeme být překvapeni velkým počtem linek, které nelze na obrazovce ani spočítat. Proto raději necháme SQL server, aby nám to spočetl sám:

```
SELECT COUNT(TELEFON) POCET
FROM OSOBA, KONTAKT
WHERE CISLO_PRAC=CISLO_P AND
JMENO="QUIDO";
```

Možná je počet linek tak velký proto, že existuje několik různých osob se stejným jménem. Malá statistika počtu linek podle jednotlivých Quidů je vyvolána dotazem:

```
SELECT CISLO_P, COUNT(TELEFON)
POCET FROM OSOBA, KONTAKT
WHERE CISLO_PRAC=CISLO_P AND
JMENO="QUIDO"
GROUP BY CISLO_P
ORDER BY CISLO_P;
```

Ono je vůbec zajímavé vědět, kolik má který jedinec telefonních kontaktů:

```
SELECT CISLO_P, JMENO,
COUNT(TELEFON) POCET
FROM OSOBA, KONTAKT
WHERE CISLO_PRAC=CISLO_P
GROUP BY CISLO_P
ORDER BY CISLO_P;
```

Takhle nějak vypadá seznam úspěšných lidí:

```
SELECT CISLO_P, JMENO,
COUNT(TELEFON) POCET
FROM OSOBA, KONTAKT
WHERE CISLO_PRAC=CISLO_P
GROUP BY CISLO_P
HAVING COUNT(TELEFON)>=3
ORDER BY CISLO_P;
```

Mohlo by se stát, že budeme pracovat se dvěma tabulkami, které budou mít stejný název sloupce. Například tabulka CLOVEK i tabulka UCET obsahují sloupec RODNE; potom musíme jednoznačnost dotazu SQL zajistit uvedením názvu tabulky před název sloupce. Oddělovačem názvu tabulky od názvu sloupce je **tečka**. Přejmenovávat můžeme nejen výrazy v projekci, ale i dlouhé názvy tabulek tak, jak plyne z následujících příkazů jazyka SQL:

```
SELECT CLOVEK.*, UCET.*
FROM CLOVEK, UCET WHERE
CLOVEK.RODNE=UCET.RODNE;
```

```
SELECT X.*, Y.*
FROM CLOVEK X, UCET Y
WHERE X.RODNE=Y.RODNE;
```

Takto vypadá přehled zůstatků na jednotlivých účtech:

```
SELECT X.RODNE RODNE_CISLO,
JMENO, ROCNIK, UCET, BANKA,
ZUSTATEK
FROM CLOVEK X, UCET Y
WHERE X.RODNE=Y.RODNE
ORDER BY JMENO, X.RODNE,
BANKA,UCET;
```

Přehled majitelů účtů by měl obsahovat počet účtů a součet zůstatků na nich, a to pro každého majitele zvlášť:

```
SELECT X.RODNE RODNE_CISLO,
JMENO, ROCNIK,
COUNT(UCET) POCET_UCTU,
SUM(ZUSTATEK) PRACHY
FROM CLOVEK X, UCET Y
WHERE X.RODNE=Y.RODNE
GROUP BY X.RODNE
ORDER BY JMENO, X.RODNE;
```

Takhle nějak si lze vybrat v drsných krajích ženicha:

```
SELECT X.RODNE RODNE_CISLO,
JMENO, ROCNIK,
COUNT(UCET) POCET_UCTU,
SUM(ZUSTATEK) PRACHY
FROM CLOVEK X, UCET Y
WHERE X.RODNE=Y.RODNE AND
ROCNIK BETWEEN 1890 AND 1920
AND POHLAVI="MUZ" AND
NOT STAV="ZENATY"
GROUP BY X.RODNE
HAVING
SUM(ZUSTATEK)>=1000000
ORDER BY JMENO, X.RODNE;
```

Porovnejte si restrikci řádků realizovanou pomocí podmínky za WHERE a restrikci skupin uskutečněnou pomocí podmínky za HAVING. Následující, jen o málo odlišný dotaz SQL nerozpozná boháče s decentralizovaným jměním:

```
SELECT X.RODNE RODNE_CISLO,
JMENO, ROCNIK,
COUNT(UCET) POCET_UCTU,
SUM (ZUSTATEK) PRACHY
FROM CLOVEK X, UCET Y
WHERE X.RODNE=Y.RODNE AND
ROCNIK BETWEEN 1890 AND 1920
AND POHLAVI="MUZ"
AND NOT STAV="ZENATY"
AND ZUSTATEK>=1000000
GROUP BY X.RODNE
ORDER BY JMENO, X.RODNE;
```

Co je to JOIN

Pokud chceme odlišit definici vazby mezi tabulkami od ostatních podmínek výběru, spojíme dvě tabulky do jednoho datového zdroje pomocí slova JOIN, které popisuje spojení mezi levou a pravou tabulkou. Nejjednodušší je INNER JOIN. Následujícím příkazem vznikne také tabulka 4, ve které chybí JOE a linka 3674:

```
SELECT * FROM OSOBA INNER JOIN
KONTAKT ON CISLO_PRAC=CISLO_P;
```

Složitější dotaz by mohl vypadat následovně:

```
SELECT CISLO_P,
COUNT(TELEFON) POCET
FROM OSOBA INNER JOIN KONTAKT
ON CISLO_PRAC=CISLO_P
WHERE JMENO="QUIDO"
GROUP BY CISLO_P
ORDER BY CISLO_P;
```

Pokud nám nejde jenom o osoby u telefonů, ale o osoby jako takové, použijeme LEFT JOIN, který zachovává ty řádky levé tabulky, které nemají protipól v pravé tabulce. Chybějící údaje mají hodnotu NULL. Tabulka OSOBA musí pak ležet vlevo od LEFT JOIN. V odpovědi na následující SQL dotaz už JOE chybět nebude:

```
SELECT * FROM OSOBA LEFT JOIN
KONTAKT ON CISLO_PRAC=CISLO_P;
```

Pomocí LEFT JOIN máme také šanci vypsat osoby, které ještě nemají telefon:

```
SELECT CISLO_PRAC, JMENO
FROM OSOBA LEFT JOIN
KONTAKT ON CISLO_PRAC=CISLO_P
WHERE TELEFON IS NULL
ORDER BY JMENO, CISLO_PRAC;
```

JAROMÍR KUKAL