Databáze standardu SQL, díl 15.

Patnáctý díl našeho seriálu je z větší části věnován dokončení tématu spojování tabulek.

Ve spojení je síla

Nejde-li nám o lidi, ale naopak o kontakty, přijde JOE zkrátka, ale linka 3674 bude zachráněna. Budeme muset buď prohodit tabulky, nebo použít RIGHT JOIN. V obou případech získáme tabulku 1 s bezprizorním telefonem:

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

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

Pak snadno získáme přehled o anonymních linkách příkazem:

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

Poslední možností spojování tabulek je FULL OUTER JOIN, který neztratí nic. Rozumní analytici používají typicky LEFT JOIN k realizaci relace 1 : N mezi levou a pravou entitou. Buďme ještě chvíli nerozumní a vygenerujme tabulku 2 dotazem:

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

Pro procvičení bude zajímavé realizovat seznam všech osob s uvedením počtu kontaktů. Osoby, které se dříve ztrácely, budou mít vedle sebe číslo nula:

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

Teprve teď máme šanci realizovat seznam neúspěšných lidí příkazem:

```
SELECT CISLO_P, JMENO,
COUNT(TELEFON) POCET
FROM OSOBA LEFT JOIN KONTAKT
ON CISLO_PRAC=CISLO_P
GROUP BY CISLO_P
HAVING COUNT(TELEFON)<=2
ORDER BY CISLO_P;
```

Bez LEFT JOIN by nám ti nejméně úspěšní chyběli. Šlo by pak jen o seznam neúspěšných lidí, kterým to můžete říct do telefonu.

Dotaz do tří tabulek

Zvědavosti se meze nekladou. Proto se můžeme současně podívat do tří a více tabulek najednou. Jsou-li tabulky spojeny relacemi, má takový pohled smysl a každé relaci odpovídá jedna vazební podmínka za WHERE. Mějme číselník TRPASLIK a číselník VLASTNOST, obsahující typické trpasličí

vlastnosti. Pokud bude ve spojovací entitě ZNALOST uvedena dvojice CISTRP a CISVLA jako unikátní klíč, máme podchyceny drby o vlastnostech konkrétních trpaslíků. Je--li unikátní klíč CIT v tabulce TRPASLIK a tabulka VLASTNOST má klíč CIV, pak již můžeme vypsat všechny drby o všech trpaslících pomocí dotazu:

```
SELECT JMENO, POPIS
FROM TRPASLIK, VLASTNOST,
ZNALOST
WHERE CIT=CISTRP
AND CIV=CISVLA
ORDER BY JMENO, POPIS;
```

SQL server je natolik inteligentní, že z unikátnosti klíčů CIT a CIV pochopí, že stačí systematicky prohledat tabulku ZNALOST a v tabulce TRPASLIK rychle vyhledat příslušné JMENO, respektive v tabulce VLASTNOST její slovní POPIS. Bohužel se ale nic nedozvíme o trpaslících, o kterých se nic neví, ani o vlastnostech, které trpaslíci ještě nemají. Pokud nám to zatím nevadí, zkusme několik SQL dotazů:

```
SELECT DISTINCT JMENO
FROM TRPASLIK, VLASTNOST,
ZNALOST
WHERE CIT=CISTRP
AND CIV=CISVLA
AND POPIS IN (”LENOST”,”DRZOST”)
ORDER BY JMENO;
```

```
SELECT POPIS
FROM TRPASLIK, VLASTNOST,
ZNALOST
WHERE CIT=CISTRP
AND CIV=CISVLA
```

```
AND JMENO=”BRUMLA”
ORDER BY POPIS;

SELECT JMENO, COUNT(POPIS)
POCET_VLASTNOSTI
FROM TRPASLIK, VLASTNOST,
ZNALOST
WHERE CIT=CISTRP
AND CIV=CISVLA
GROUP BY JMENO
ORDER BY JMENO;
SELECT POPIS, COUNT(JMENO)
POCET_TRPASLIKU
FROM TRPASLIK, VLASTNOST,
ZNALOST
WHERE CIT=CISTRP
AND CIV=CISVLA
GROUP BY POPIS
ORDER BY POPIS;
```

Zahnízděný JOIN

Je na čase zabránit ztrátám na trpaslících a jejich vlastnostech použitím zahnízděného LEFT JOIN. Stačí si uvědomit, že TRPASLIK LEFT JOIN ZNALOST je informační zdroj obsahující všechna data o trpaslících, a jako takový jej stačí spojit s tabulkou VLASTNOST. Tabulka VLASTNOST stojí vlevo od LEFT JOIN. Vpravo je pak v závorce zahnízděný LEFT JOIN. Nepoužité reference na trpaslíky a na vlastnosti se pak neztratí a budou jim odpovídat jednotlivé řádky výsledné tabulky. Vylepšený základní dotaz potom zní:

```
SELECT JMENO, POPIS
FROM VLASTNOST LEFT JOIN
(TRPASLIK LEFT JOIN ZNALOST ON CIT=CISTRP) ON CIV=CISVLA
```

```
ORDER BY JMENO, POPIS;
```

Chytráky také snadno zjistíme:

```
SELECT JMENO FROM VLASTNOST LEFT JOIN
(TRPASLIK LEFT JOIN ZNALOST ON CIT=CISTRP) ON CIV=CISVLA
WHERE POPIS=”CHYTROST”
ORDER BY JMENO;
```

Následující dotaz vypíše trpaslíky, o kterých se zatím nic neví. Takový dotaz bez LEFT JOIN není možný:

```
SELECT JMENO FROM VLASTNOST LEFT JOIN
(TRPASLIK LEFT JOIN ZNALOST ON CIT=CISTRP) ON CIV=CISVLA
WHERE POPIS IS NULL
ORDER BY JMENO;
```

Pokud nás zajímá pouze jejich počet, stačí napsat:

```
SELECT COUNT(JMENO) NEZNAMY FROM VLASTNOST LEFT JOIN
(TRPASLIK LEFT JOIN ZNALOST ON CIT=CISTRP) ON CIV=CISVLA
WHERE POPIS IS NULL;
```

Úleva zvaná VIEW

Žádná kaše není tak horká...

Pokud seriál trvá příliš dlouho, mohou podle klasika nastat dvě možnosti. Buď roste složitost nových předkládaných fakt nade všechny meze únosnosti pro čtenáře, nebo klesá procento nových a současně užitečných informací až téměř k samé nule. Jistě jste již přesyceni rozvinutými možnostmi příkazu SELECT z minulých dílů a v tomto díle očekáváte další složitosti, nebo již jen opakování známých fakt. Dnes nás čeká nové téma, které je současně nové, snadné, užitečné, a navíc navazuje na předchozí témata. Chtě nechtě musíme udělat velký návrat do DDL SQL a věnovat se těm jeho

partiím, které by před rokem vyzněly naprázdno. Ano, bude řeč o vytváření, používání a rušení virtuálních tabulek zvaných VIEW.

Velké pokušení

Snad každý, komu se povedl elegantní příkaz SELECT, si ho okamžitě opíše do poznámkového bloku, aby se pojistil pro příští podobné situace. Technické podobě “bloku” se meze nekladou. Jindy je pro nás cennější odpověď na dotaz, která není ničím jiným než tabelárním přehledem dat. Příkaz SELECT můžeme snadno doplnit o slovo INTO a název cílové tabulky, a tak v souladu s předchozími díly seriálu vytvoříme novou reálnou tabulku obsahující požadované konkrétní hodnoty. Proto je celá řada programátorů v pokušení skladovat za každou cenu koncentráty informací v nových a nových tabulkách tak, jak mohou vzniknout po příkazech typu:

```
SELECT RC, PRIJMENI, JMENO,
SUM(STAV)
INTO PRACHAC FROM CLOVEK, UCET
WHERE CLOVEK.RC=UCET.RC
GROUP BY CLOVEK.RC
HAVING SUM(STAV)>1000000
ORDER BY PRIJMENI, JMENO, RC;
```

Co je možné, není povinné a nemusí být efektivní. Předchozí postup nezaslouží následování, neboť má většinou tři nevýhody. Nová tabulka PRACHAC v prvé řadě zabírá místo na disku. Navíc tabulka neobsahuje informaci o svém okamžiku vzniku. V okamžiku čtení tabulky PRACHAC již mohou být bohatí i jiní lidé a občas někdo zemře nebo zchudne. To, že nová tabulka není aktuální, je podstatná vada na kráse. Třetí nevýhodu okusíme při každé aktualizaci tabulky. Nejprve ji budeme muset zrušit příkazem DROP TABLE PRACHAC, pak se podívat do notesu a znovu napsat příkaz SELECT. Dost často se projeví čtvrtá nevýhoda. Nová tabulka nemusí být v 5NF. Předchozí příkaz tuto nevýhodu nedemonstruje. Stačilo by na STAV účtu neaplikovat agregační funkci SUM a vynechat části GROUP BY a HAVING. Taková tabulka neobsahuje údaje o bohatých lidech, ale spíše o jednotlivých stavech na neznámých účtech a o jejich majitelích. Bohužel nemá unikátní klíč, tedy je pouze v 1NF. Přidáním sloupce CISUCTU do příkazu SELECT si mnoho nepomůžeme. Nová tabulka má sice unikátní klíč CISUCTU a je užitečná, ale je zatížena závislostí mezi neklíčovými sloupci RC a PRIJMENI, respektive RC a JMENO. Proto bude pouze v 2NF. Pro databázi je typické, že obsahuje pouze tabulky v 5NF a neobsahuje nadbytečná data. Pro uživatele je naopak normální vidět data v lidštější podobě, která se nadbytečností a nenormalizovaností přímo pyšní, neboť názornost si žádá své oběti.

Virtuální svět

Rok 2000 se kvapem blíží a moderní člověk dává často přednost virtuální realitě před tou trapně obyčejnou. Na našich obrazovkách a tiskárnách běžně vídáme čtyřmístný letopočet jako součást informace o datu, kdy se něco podstatného stalo. Někteří lidé nejpozději počátkem ledna příštího roku zjistí, že to byla pouze virtuální realita, která zastřela obyčejnou realitu dvoucifernou. Moderní virtuální doba přináší mnoho nového. Proto je nebezpečné rozvíjet intelekt předškolních dítek pomocí jednoduchých hádanek typu: Má to rohy, kopyta a ocas a není to zvíře. Odpověď je prostá: virtuální zvíře. Pokud bych měl srozumitelně definovat pojem virtuality, neobejdu se bez naivity vyjádření:

VIRTUÁLNÍ OBJEKT X MÁ STEJNÉ VLASTNOSTI JAKO X, ALE NENÍ TO X.

Přestože bych nikomu nepřál, aby se musel odprásknout virtuálním revolverem po delší virtuální symbióze s virtuální čarodějnicí, budu naopak velmi propagovat používání virtuálních tabulek místo těch normálních. Virtuální tabulka z pohledu uživatele i jazyka SQL vypadá jako klasická tabulka, ale není to ona. Pro nás je podstatné, že vytváření, používání a rušení virtuálních tabulek je jednoduchou záležitostí.

Zpátky k DDL

DDL-Data Definition Language jako součást SQL umožňuje vytváření a rušení virtuálních tabulek. Ty jsou anglicky označovány jako VIEW, což česky neznamená nic jiného, než pohled na něco či do něčeho.

Jaromír Kukal