Databáze standardu SQL, díl 16.

Zpátky k DDL

Jedeme dál! Hlavním tématem dalšího pokračování našeho seriálu jsou virtuální tabulky.

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. Virtuální tabulka je totiž jenom pohledem do jiné tabulky nebo tabulek. Zatímco klasická tabulka CLOVEK je vytvářena příkazem CREATE TABLE CLOVEK s uvedeným seznamem sloupců, je virtuální tabulka PRACHAC vytvářena z už existující tabulky CLOVEK pomocí příkazu CREATE VIEW PRACHAC, uvnitř kterého je uveden vhodný příkaz SELECT. V našem konkrétním případě bude vhodné pojmenovat i sloupce nově vzniklé virtuální tabulky. Tak dostaneme příkaz pro její vytvoření:

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

Pokud v okamžiku formulace dotazu existovaly tabulky CLOVEK a UCET a neexistovala tabulka PRACHAC, vznikne virtuální tabulka PRACHAC bez problémů. V ostatních případech virtuální tabulka nevznikne. Pokud dojde ke změnám v obsahu tabulek CLOVEK nebo UCET, okamžitě se promítnou do virtuální tabulky PRACHAC, do které lze nahlédnout jako do každé jiné jedním z následujících příkazů:

```
SELECT * FROM PRACHAC
ORDER BY PRIJMENI,JMENO,RODNE;

SELECT * FROM PRACHAC
ORDER BY JMENI DESC, PRIJMENI,
JMENO, RODNE;

SELECT * FROM PRACHAC
WHERE JMENI>100000000;

SELECT PRIJMENI, MAX(JMENI)
FROM PRACHAC
GROUP BY PRIJMENI;
```

Nyní je zřejmé, že vytvoření VIEW se vyplatilo a že používání ORDER BY uvnitř VIEW není k ničemu. Pro úplnost je uveden obecný tvar příkazu pro vytvoření virtuální tabulky:

```
CREATE VIEW název virtuální tabulky
(seznam názvů sloupců)
AS
SELECT seznam výrazů
FROM seznam tabulek
WHERE logický výraz
GROUP BY seznam sloupců
HAVING logický výraz;
```

Je nutné respektovat následující omezení příkazu CREATE VIEW:

- Nesmí obsahovat ORDER BY.
- Nesmí obsahovat HAVING bez GROUP BY.
- Seznam výrazů projekce musí mít stejný počet členů jako seznam názvů sloupců.
- Seznam názvů sloupců je nepovinný.

Virtuální tabulka vznikne, je-li vnitřní příkaz SELECT platný a je připravena pro čtení jiným příkazem SELECT. Virtuální tabulka se může odkazovat na jiné tabulky včetně virtuálních tabulek. Odkazování virtuální tabulky na sebe samu není možné. Pokud budeme chtít virtualitu nové tabulky brát doslova a budeme ji chtít aktualizovat pomocí INSERT, UPDATE nebo DELETE, musíme se poněkud uskrovnit a přijmout další omezení. Při popisu projekce nesmí obsahovat výrazy GROUP BY, DISTINCT, více než jednu tabulku za FROM, JOIN ani zahnízděný dotaz. Pozornému čtenáři jistě neušlo, že v zájmu absolutní virtuality přišel téměř o všechny vymoženosti příkazu SELECT. Zdravý rozum velí naopak zapomenout na absolutní virtualitu a věnovat se naplno vymoženostem virtuálních tabulek určených jenom pro čtení. Konečně tak máme k dispozici nástroj k virtuálnímu, ale trvalému spojování několika tabulek v jeden celek. Teď bude velmi snadné dívat se na reálná data v několika tabulkách a zároveň nemít žádnou zbytečnou práci. Nad klasickými tabulkami tak můžeme vytvořit i několik hierarchických vrstev virtuálních tabulek, které pomohou přísně strukturovaný svět fyzicky uložených dat oddělit od světa lidí a jejich klientských aplikací. To slouží nejen k usnadnění čtení dat, ale mnohdy též k utajení základních datových struktur před konkurencí. Virtuální tabulka je velmi křehkým objektem, který relativně snadno zaniká. Klasickou cestou ke zrušení naší virtuální tabulky PRACHAC je použití příkazu

```
DROP VIEW PRACHAC;
```

Virtuální tabulka PRACHAC zanikne, zrušíme-li některou z tabulek CLOVEK, UCET nebo některý z používaných sloupců těchto tabulek například pomocí příkazů

```
DROP TABLE CLOVEK;
DROP TABLE UCET;
ALTER TABLE CLOVEK
DROP COLUMN RC;
ALTER TABLE UCET
DROP COLUMN STAV;
```

To je první a poslední případ, kdy si v SQL vzniklý objekt nechrání své zdroje před zničením a raději zruší sám sebe. Porovnejte to například s vytvářením, užitím a rušením indexů, které si svoji tabulku důsledně hlídají. Použijeme-li příkazy

```
DELETE FROM CLOVEK;
DELETE FROM UCET;
```

bude to patrně velká ztráta informací o lidech a účtech, ale virtuální tabulka PRACHAC to přežije ve zdraví a kdykoli

bude připravena zobrazit každého miliónáře, který bude znovu zadán do prázdných tabulek CLOVEK a UCET.

Virtualita se vyplatí

Mějme tabulku CLOVEK se sloupci RC, JMENO, PRIJMENI a tabulku KONTAKT se sloupci RC, TYP a SPOJENI. Mezi tabulkami existuje relace zprostředkovaná rodným číslem RC, která je typu N : 1 a má název KONTAKT NA CLOVEKA. Sloupec TYP představuje typ kontaktu jako TELEFON, FAX, EMAIL a podobně, zatímco sloupec SPOJENI obsahuje konkrétní hodnotu telefonního čísla, čísla faxu či počítačovou adresu. Pokud jsou obě tabulky zaplněny bezrozpornými daty, máme motivaci k vytvoření celé řady virtuálních tabulek. Především nás bude zajímat seznam lidí a příslušných spojení:

```
CREATE VIEW CLOVICEK(RC, JMENO,
PRIJMENI, TYP, SPOJENI)
AS
SELECT CLOVEK.RC, JMENO,
PRIJMENI, TYP, SPOJENI
FROM CLOVEK LEFT JOIN KONTAKT
ON CLOVEK.RC=KONTAKT.RC;
```

Od tohoto okamžiku je zbytečné číst data přímo z původních tabulek. Nevadí, že tabulka CLOVICEK je virtuální. Můžeme z ní vytvořit telefonní seznam příkazem:

```
CREATE VIEW TELEFON(RC, JMENO,
PRIJMENI, TEL)
AS
SELECT RC, JMENO, PRIJMENI,
SPOJENI FROM CLOVICEK
WHERE TYP="TELEFON";
```

Virtuální tabulka TELEFON umožní zobrazení telefonního seznamu příkazem

```
SELECT * FROM TELEFON ORDER BY
PRIJMENI, JMENO, RC, TEL;
```

Už jistě víte, jak vytvořit VIEW všech faxových kontaktů, všech e-mailových kontaktů nebo seznam všech lidí, na které není žádný kontakt. Představte si, že chcete v rámci marketingové akce zavolat všem lidem, na které existuje alespoň jeden telefonní kontakt a kteří ještě nemají internetovou adresu. Na to stačí vytvořit virtuální tabulku AKCE:

```
CREATE VIEW AKCE(PRIJMENI,
JMENO, RC, TEL)
AS
SELECT PRIJMENI, JMENO, RC,
SPOJENI FROM CLOVICEK
WHERE TYP="TELEFON"
AND RC NOT IN
(SELECT DISTINCT RC FROM
CLOVICEK WHERE TYP="EMAIL");
```

Kdykoli se pak lze zeptat, kterým klientům máme nabízet službu:

```
SELECT * FROM AKCE ORDER BY
PRIJMENI, JMENO, RC, TEL;
```

Pokud tvůrce databázového systému vhodně navrhne tabulky, relace mezi nimi, a nad tím vším vybuduje racionálně systém virtuálních tabulek, je vše připraveno k formulaci jednoduchých klientských dotazů SELECT do jednotlivých virtuálních tabulek s jednoduchou podmínkou za WHERE a definovaným tříděním pomocí ORDER BY. Tvůrce aplikace pak ani nemusí tušit, jak dokonalého systému se vlastně ptá. Přiberme si do našeho minisystému ještě další dvě tabulky. Třetí tabulka UCET bude mít jako klíč CISUCTU a navíc ještě dva sloupce RC a STAV, kde RC je rodné číslo majitele účtu a STAV je okamžitý stav účtu v korunách. Občas majitel účtu dovolí jiným lidem, aby z něj také směli vybírat peníze. Tento jev je popsán tabulkou PRAVO se složeným klíčem, tvořeným sloupci CISUCTU a RC. Zároveň přibylo několik relací N : 1 mezi tabulkami. Jde o relace UCET VLASTNI OSOBA, PRAVO PRO OSOBU a PRAVO NA UCET. Teď je třeba nové tabulky pokrýt užitečným systémem virtuálních tabulek. Především bude užitečné vytvořit virtuální tabulku lidí a jejich účtů:

```
CREATE VIEW CLOVEK_S_UCTY
(PRIJMENI, JMENO, RC, UCET, STAV )
AS
SELECT PRIJMENI, JMENO,
CLOVEK.RC, CISUCTU, STAV
FROM CLOVEK LEFT JOIN UCET
ON CLOVEK.RC=UCET.RC;
```

Pro vytipování nebo odstranění lidí, kteří nemají vlastní účet, budou potřebné další dvě virtuální tabulky. Navíc ušetříme na přejmenovávání sloupců:

```
CREATE VIEW IGNORANT_BANK
AS
SELECT PRIJMENI, JMENO, RC
FROM CLOVEK_S_UCTY
WHERE UCET IS NULL;
```

```
CREATE VIEW MAJITEL_UCTU
AS
SELECT PRIJMENI, JMENO, RC,
UCET, STAV FROM CLOVEK_S_UCTY
WHERE UCET IS NOT NULL;
```

Souhrnné údaje mají mnohdy větší cenu než podrobné výpisy, a tak máme o důvod víc k tomu, abychom pokračovali v návrhu další virtuální tabulky:

```
CREATE VIEW MAJITEL(PRIJMENI,
JMENO, RC, CELKEM)
AS
SELECT PRIJMENI, JMENO, RC,
SUM(STAV) FROM MAJITEL_UCTU
GROUP BY RC;
```

Nenechaví lidé počítají do svého jmění i stavy na účtech, na které mají právo výběru, což vyžaduje rafinovanější virtuální tabulku nad tabulkami CLOVEK, UCET a PRAVO:

```
CREATE VIEW NAROK(PRIJMENI,
JMENO, RC, MAJITEL, UCET, STAV)
AS
SELECT PRIJMENI, JMENO,
CLOVEK.RC, CLOVEK.RC=UCET.RC,
CISUCTU, STAV
FROM CLOVEK, UCET, PRAVO
WHERE CLOVEK.RC=UCET.RC OR
CLOVEK.RC=PRAVO.RC AND
PRAVO.CISUCTU=UCET.CISUCTU;
```

Výraz CLOVEK.RC=UCET.RC je přejmenován na sloupec MAJITEL, který má pak logickou hodnotu YES, nebo NO, podle toho, zda je daný člověk majitelem účtu, nebo má na něj jenom právo výběru. Celkový sumarizovaný pohled do dat:

```
CREATE VIEW SUMARIZACE
(PRIJMENI, JMENO, RC, CELKEM)
AS
SELECT PRIJMENI, JMENO, RC,
SUM(STAV) FROM NAROK
GROUP BY RC;
```

Chceme-li zjistit, kolikrát většími částkami disponují lidé v porovnání se skutečným vlastnictvím, vytvoříme si virtuální tabulku:

```
CREATE VIEW BLAMAZ(POMER)
AS
SELECT (SELECT SUM(CELKEM)
FROM SUMARIZACE)/
(SELECT SUM(CELKEM) FROM
MAJITEL);
```

Tolik pro dnešek, příště pokračujeme.

Jaromír Kukal