Databáze standardu SQL, díl 6.

DML a aktualizace dat

Co se skrývá za zkratkou uvedenou v názvu dalšího pokračování seriálu, je tématem, jež nelze odbýt jen tak v rychlosti – je mu proto věnován celý šestý díl. A začneme hned „od podlahy“...

DML je zkratkou anglických slov Data Manipulation Language. Jazyk DML je částí SQL jazyka, která umožňuje vlastní manipulaci s daty. Budeme se nejprve zabývat tou částí DML, která slouží k aktualizaci obsahu tabulek. Ta hraje důležitou roli hned po definici tabulek a indexových souborů v DDL a představuje vlastní naplňování tabulek jednotlivými položkami, jejich opravami a rušením. Tabulky vytvořené jazykem DDL jsou totiž na počátku zcela prázdné a neobsahují ani jeden konkrétní řádek. Proto má jazyk DML celou řadu příkazů sloužících k aktualizaci dat.

SQL	význam
DELETE	zruš položky
FROM	z
INSERT	vložit položky
INTO	dovnitř do
SELECT *	vybrat všechny
SET	nastavit
UPDATE	aktualizovat položky
VALUES	hodnoty
WHERE	kde

Tabulka č. 1: Klíčová slova DML – aktualizace.

V tabulce č. 1 jsou uvedena důležitá klíčová slova jazyka DML pro aktualizaci dat a jejich český význam. Představme si situaci, kdy námi právě vytvořená tabulka OSOBA je prázdná a chtěli bychom do ní vložit několik řádků (položek) tak, aby na SQL serveru vznikla obdoba tabulky č. 2.

Budeme muset použít nový příkaz INSERT INTO s uvedením dvojic názvů sloupců a příslušných ukládaných hodnot. Nejprve je nutné vyjmenovat v libovolném pořadí sloupce tabulky OSOBA, do kterých budeme přidávat hodnoty.

Pak uvedeme klíčové slovo VALUES a za něj seznam konkrétních hodnot, které chceme přidat do právě vznikajícího nového řádku tabulky. Následující příkazy umožňují zaplnit tabulku OSOBA v duchu tabulky č. 2. Povšimněte si prosím práce s konstantami jednotlivých datových typů:

```
INSERT INTO OSOBA
        (RC, PRIJMENI, JMENO, MISTONAR, DATNAR, MISTOUMR, DATUMR,
        POHLAVI, LHAR, ZLODEJ, DOBRAK, IQ, JMENI, MENA) VALUES
        ("730411/3740", "Carlo", "Horatius", "Praha", "11.04.1973", NULL, NULL,
        "M", YES, NO, 30, 93, 10000000, "USD");

INSERT INTO OSOBA
        (RC, PRIJMENI, JMENO, MISTONAR, DATNAR,
        POHLAVI, LHAR, ZLODEJ, DOBRAK, IQ) VALUES
        ("730411/3751", "Carlo", "John", "Praha", "11.04.1973",
        "M", NO, NO, 100, 110);

INSERT INTO OSOBA
        (RC, PRIJMENI, JMENO, MISTONAR, DATNAR,
        POHLAVI, LHAR, ZLODEJ, DOBRAK, IQ, JMENI, MENA) VALUES
        ("730411/3762", "Carlo", "Mike", "Praha", "11.04.1973",
        "M", NO, YES, 10, 130, 1000, "DEM");
```

Dále následuje ukázka tří příkazů INSERT INTO, které SQL server odmítne provést:

```
INSERT INTO OSOBA
        (RC, PRIJMENI, JMENO, MISTONAR, DATNAR,
        POHLAVI, LHAR, ZLODEJ, DOBRAK, IQ, JMENI, MENA) VALUES
        ("730411/3740", "Carlo", "Horatius", "Praha", "11.04.1973", NULL, NULL,
        "M", YES, NO, 30, 93, 10000000, "USD");

INSERT INTO OSOBA
        (RC, JMENO, MISTONAR, DATNAR,
        POHLAVI, LHAR, ZLODEJ, DOBRAK, IQ) VALUES
        ("730411/3773", "Jonas", "Brno", "11.04.1973",
        "X", YES, NO, 30, 200);

INSERT INTO OSOBA
        (JMENO, POHLAVI, MENA) VALUES ("Oleg", "M", "GBP");
```

První chybný příkaz je zoufalým pokusem, jak dát do tabulky OSOBA podruhé stejného člověka. Zde tvrdě narazíme na ENTITNÍ INTEGRITU danou unikátností klíčového sloupce RC. Jiné důvody k odmítnutí přidání řádku nejsou. Druhý příkaz obsahuje hned několik chyb. Naráží se zde několikrát na DOMÉNOVOU INTEGRITU tabulky OSOBA, neboť Jonášovi chybí příjmení, je domýšlivý a jeho pohlaví je opředeno velkou záhadou. Třetí příkaz je ukázkou pokusu o zadání něčeho velmi nejasného do systému s velmi jasnými pravidly. Pokuste se sami napsat jiné tři korektní a nekorektní příkazy INSERT INTO.

Pokud bychom nebyli zcela spokojeni s tabulkou č. 2 po uvedených příkazech, nezbývá než některé řádky zrušit nebo modifi-

RC	PRIJMENI	JMENO	MISTONAR	DATNAR	MISTOUMR	DATUMR
730411/3740	Carlo	Horatius	Praha	11.4. 1973	NULL	NULL
730411/3751	Carlo	John	Praha	11.4. 1973	NULL	NULL
730411/3762	Carlo	Mike	Praha	11.4. 1973	NULL	NULL

POHLAVI	LHAR	ZLODEJ	DOBRAK	IQ	JMENI	MENA
M	YES	NO	30	93	10 000 000	USD
M	NO	NO	100	110	NULL	NULL
M	NO	YES	10	130	1 000	DEM

Tabulka č. 2: Obsah tabulky OSOBA.

kovat. K rušení jednotlivých řádků nebo celé skupiny řádků slouží příkaz DELETE s doplňkovým klíčovým slovem WHERE. Nejjednodušší je veškeré věty z tabulky OSOBA zrušit. To provedeme následujícím jednoduchým příkazem k hromadnému vyvraždění:

```
DELETE FROM OSOBA;
```

Výsledkem tohoto příkazu nebude úplné zrušení tabulky OSOBA, ale pouze odstranění všech jejích řádků. V tom je rozdíl mezi příkazem DELETE z DML, který likviduje pouze data, a příkazem DROP TABLE z DDL, který ničí i tabulku, ať už je plná dat nebo nikoli.

Náš příkaz DELETE bohužel neměl žádnou logickou podmínku za WHERE. Takový příkaz je poněkud nebezpečný. Typičtější je rušit jenom některé řádky. Představme si, že předchozí příkaz nebyl proveden. Místo něj raději provedeme příkaz, který zruší druhý řádek z tabulky OSOBA. To můžeme udělat velmi citlivě s využitím porovnávání rodného čísla s konkrétní hodnotou rodného čísla osoby ve druhém řádku:

```
DELETE FROM OSOBA WHERE RC="730411/3751";
```

Teď teprve oceníme výhody ENTITNÍ INTEGRITY, neboť sloupec RC je unikátním klíčem tabulky OSOBA. Pokud by neexistoval sloupec nebo skupina sloupců, která má charakter unikátního klíče, těžko bychom mohli cíleně rušit nebo aktualizovat jednotlivé řádky. Nyní je vhodné se ještě jednou vrátit k normálním formám tabulek. Tabulka OSOBA je v 5NF a už od 3NF je snadná jakákoli aktualizace jejího obsahu.

Pokud bychom se pod vlivem pietního aktu rozhodli u osoby v prvním řádku změnit datum a místo úmrtí na nějakou konkrétní hodnotu, použijeme UPDATE. Opět použijeme porovnávání hodnoty klíče s RC nebožtíka a klíčové slovo SET pro deklaraci všech plánovaných změn:

```
UPDATE OSOBA SET MISTOUMR="Pardubice", DATUMR="01.01.1997" WHERE RC="730411/3740";
```

I opravy konkrétních údajů stojí a padají na ENTITNÍ INTEGRITĚ.

Aktualizovat můžeme jednotlivá pole jak v jednotlivých řádcích, tak i ve skupinách řádků. Malé děti si dost často myslí, že není možné umřít, a neuznávají tento pojem. Pokud by skutečně umírání bylo zakázáno, pak je třeba zpětně dosadit neurčité hodnoty místa i data úmrtí. To se snadno provede následujícím hromadným příkazem UPDATE:

```
UPDATE OSOBA SET MISTOUMR=NULL, DATUMR=NULL;
```

Pokud bychom dospěli k závěru, že všechny osoby uvedené v tabulce mají IQ 140, provedeme následující příkaz:

```
UPDATE:
UPDATE OSOBA SET IQ=140;
```

Procvičte si databázovou představivost na po sobě jdoucích příkazech jazyka DML:

```
UPDATE OSOBA SET IQ=IQ/7;

UPDATE OSOBA SET ZLODEJ=YES WHERE POHLAVI="M" AND IQ>100;

ALTER TABLE OSOBA ADD COLUMN PADOUCH LOGICAL;

UPDATE OSOBA SET PADOUCH=(ZLODEJ OR LHAR) AND POHLAVI="M";
```

Tabulka č. 3: Obsah tabulky POHYB.

UCET	TYP	PORADI	DATUM	CASTKA	MENA
37683193	V	13	31.3. 1998	3700,00	USD
37683193	V	14	1.4. 1998	100,00	USD
37683193	P	15	10.4. 1998	5000,00	USD
12310338	P	3	10.4. 1998	6000,00	KC
12310338	V	4	16.4. 1998	7200,00	KC

U každého z nich je vhodné posoudit jeho korektnost a představit si jeho důsledky. Určete, na jaká integritní omezení narazí neplatné příkazy a jaký bude obsah tabulky po každém z uvedených příkazů.

Věnujme se tabulce č. 3, která představuje plánovaný obsah tabulky POHYB. Jde o pohyby na účtech a o jednotlivé typy a pořadí operací.

Pokud bychom chtěli naplnit jednotlivé položky, postupně provedeme následující příkazy jazyka DML pro přidávání jednotlivých řádků.

Příkazy jsou pro procvičení formulovány tak, aby jeden z nich byl SQL serverem odmítnut a jiný přidal omylem nadbytečný řádek:

```
INSERT INTO POHYB
        (UCET, TYP, PORADI, DATUM,
        CASTKA, MENA)
        VALUES ("37683193", "V", 13,
        "31.03.1998", 3700, "USD");

INSERT INTO POHYB
        (UCET, TYP, PORADI, DATUM,
        CASTKA, MENA)
        VALUES ("37683193", "V", 14,
        "01.04.1998", 100, "USD");

INSERT INTO POHYB
        (UCET, TYP, PORADI, DATUM,
        CASTKA, MENA)
        VALUES ("37683193", "P", 15,
        "10.04.1998", 5000, "USD");

INSERT INTO POHYB
        (UCET, TYP, PORADI, DATUM,
        CASTKA, MENA)
        VALUES ("37683193", "D", 13,
        "31.03.1998", 134500, "USD");

INSERT INTO POHYB
        (UCET, TYP, PORADI, DATUM,
        CASTKA, MENA)
        VALUES ("12310338", "P", 3,
        "10.04.1998", 6000, "KC");

INSERT INTO POHYB
        (UCET, TYP, PORADI, DATUM,
        CASTKA, MENA)
        VALUES ("12310338", "V", 4,
        "16.04.1998", 7200, "KC");

INSERT INTO POHYB
        (UCET, TYP, PORADI, DATUM,
        CASTKA, MENA)
        VALUES ("12310338", "V", 130,
        "31.03.1998", 3, "KC");

INSERT INTO POHYB
        (UCET, TYP, PORADI, DATUM,
        CASTKA, MENA)
        VALUES ("37683193", "V", 13,
        "31.03.1998", 134500, "USD");
```

Každý SQL server odpoví: „NE, TO NEJDE!!!", protože kombinace čísla účtu, pořadí a data by už nebyla jako složený klíč unikátní. Opět vidíme hlídací funkci indexových souborů v SQL.

V tabulce č. 4 je vidět plánovaný obsah tabulky POST, která se vrací k dříve analyzované úloze o postech a lidech, kteří na nich pracují. Tabulka má následující sloupce. Sloupec CISPOST představuje číslo postu, které je unikátním klíčem. Sloupec NAZEV přestavuje název pracovního zařazení na postu a nemusí být unikátní. Sloupec NADRIZENY neobsahuje nic jiného než číslo nadřízeného postu, které může být vynecháno pouze u vedoucího firmy nebo organizace. Sloupec RC je nepovinné rodné číslo konkrétní osoby, která právě teď na daném postu pracuje. Nejprve si pro zopakování a s využitím jazyka DDL vytvoříme tabulku POST, potom příslušné indexové soubory a na závěr tabulku naplníme příkazem INSERT INTO z DML:

```
CREATE TABLE POST
        (CISPOST INTEGER NOT NULL,
        NAZEV CHAR(30) NOT NULL,
        NADRIZENY INTEGER CHECK
        (NADRIZENY IS NULL
                OR NOT
        (NADRIZENY=CISPOST)),
        RC CHAR(11));

CREATE UNIQIE INDEX CIPOST ON
POST (CISPOST);
CREATE INDEX NAPOST ON POST (NA-
ZEV);
```

Tabulka č. 4: Obsah tabulky POST.

CISPOST	NAZEV	NADRIZENY	RC
10	GENERAL	NULL	730411/3740
36	TECHNICKY REDITEL	10	540327/3249
61	OBCHODNI REDITEL	10	730411/3751
57	PROJEKTANT	36	730411/3762
63	PROJEKTANT	36	NULL

Následujícím příkazem musíme nadbytečný řádek odstranit:

```
DELETE FROM POHYB WHERE
UCET="12310338" AND PORA-
DI=130 AND DATUM="31.03.1998";
```

Pokusme se do tabulky POHYB přidat řádek, který se na první pohled zdá být zcela normální :

```
CREATE INDEX BOSSPOST ON POST
(NADRIZENY);
CREATE INDEX RCPOST ON POST
(RC);

INSERT INTO POST
        (CISPOST, NAZEV, NADRIZENY,
        RC)
        VALUES (10, "GENERALNI",
        NULL, "730411/3740");
```

```
INSERT INTO POST
        (CISPOST, NAZEV, NADRIZENY,
        RC)
        VALUES (36, "TECHNICKY REDI-
        TEL", 10, "540327/3249");

INSERT INTO POST
        (CISPOST, NAZEV, NADRIZENY,
        RC)
        VALUES (61, "OBCHODNI REDI-
        TEL", 10, "730411/3751");

INSERT INTO POST
        (CISPOST, NAZEV, NADRIZENY,
        RC)
        VALUES (57, "PROJEKTANT", 36,
        "730411/3762");

INSERT INTO POST
        (CISPOST, NAZEV, NADRIZENY,
        RC)
        VALUES (63, "PROJEKTANT", 36,
        NULL);

INSERT INTO POST
        (CISPOST, NAZEV, NADRIZENY,
        RC)
        VALUES (10, "GENERAL", NULL,
        "540327/3249");
```

Pokud vám došlo, jaký je můj splnitelný sen a o co ani nechci usilovat, zajisté už víte, kdy jsem se narodil, jaké je organizační schéma firmy, která to může někam dotáhnout, a kolik je tam neobsazených fleků.

Hlavní výhodou SQL jazyka a tedy i příkazů DML je vysoký stupeň bezpečnosti práce s daty.

Může se totiž stát, že ze dvou klientských stanic zdánlivě současně probíhá přidávání identické osoby se stejným rodným číslem do tabulky OSOBA. Protože SQL server vždy jeden z dotazů přijme dřív, přednostně se ho pokouší splnit, a pak oznámí úspěšnost provedení operace.

Teprve potom se zabývá druhým identickým dotazem s tím, že vyrozumí druhou stanici, která dotaz formulovala, že není možné tímto DML příkazem přidat tutéž osobu kvůli ENTITNÍ INTEGRITĚ. Z hlediska bezpečnosti práce v síti SQL jazyk nepřináší žádné problémy.

Závěrem této kapitoly o aktualizaci dat bych rád uvedl několik příkladů rozmanitého použití příkazů INSERT, DELETE a UPDATE aplikovaných na již existující nebo fiktivní tabulky.

Pokud bychom chtěli všem zaměstnancům o 100 Kč zvýšit plat, provede se to příkazem:

```
UPDATE PRACOVNIK SET HRU-
BA=HRUBA+100;
```

Pokud bychom si potom uvědomili, že všem Karlům jsme chtěli zvýšit plat o 500 Kč, pak přídavek 400 Kč učiníme příkazem:

```
UPDATE PRACOVNIK SET HRU-
BA=HRUBA+400 WHERE JME-
NO="Karel";
```

Pokud naopak budeme chtít, aby všichni Karlové byli v databázi PRACOVNIK zrušeni pro svoji pracovní nespolehlivost, provedeme to příkazem:

```
DELETE PRACOVNIK WHERE JME-
NO="Karel";
```

Pokud budeme chtít přidat do tabulky OSOBA všechny osobní údaje z tabulky PRIJATI, provedeme to příkazem:

```
INSERT INTO OSOBA SELECT * FROM
PRIJATI;
```

Posledně zamýšlený příkaz je platný pouze za předpokladu, že tabulka PRIJATI obsahuje pole se shodnými názvy jako tabulka OSOBA a že přidáním všech osobních údajů z tabulky PRIJATI do tabulky OSOBA nevzniknou rozpory.

Formulace SELECT * FROM znamená: vyber všechno z tabulky. V případě existence alespoň jednoho konfliktu se nepřidá ani jeden z řádků z tabulky PRIJATI do tabulky OSOBA. Jde tedy opět o bezpečnou operaci. Podobně bychom mohli do tabulky POHYB přidat několik nových položek z tabulky DNESNI příkazem:

```
INSERT INTO POHYB SELECT * FROM
DNESNI;
```

Do tabulky POST přidáme nové oddělení mořeplavby z tabulky SEABOY příkazem:

```
INSERT INTO POST SELECT * FROM
SEABOY;
```

Pokud v tabulce SEABOY budou všichni přímo nebo nepřímo podřízeni postu VELITEL MOREPLAVBY a ten se bude odkazovat na nadřízeného s číslem 61, pak se ve výsledné tabulce POST začlení celá mořeplavba pod obchodního ředitele. Pokud se mořeplavba stane neperspektivním oborem, stačí napsat:

```
UPDATE POST SET NADRIZENY=NULL
WHERE NAZEV="VELITEL MOREPLAV-
BY";
```

Copak nepěkného věští poslední posloupnost příkazů?

```
UPDATE POST SET RC=NULL WHERE
NADRIZENY IS NOT NULL;
UPDATE POST SET RC="730411/3740";
DELETE POST WHERE RC="730411/
3740";
DELETE POST;
DROP TABLE POST;
```

Jaromír Kukal