Každý jen trochu chytrý nepořádník okamžitě odpoví své hrdé ANO. Navíc na přilepšenou dodá, pro koho je ten pořádek určen. Pokud by byl opravdu chytrý, bude raději mlčet, neboť tím podle psychologů deklaruje svůj vysoký individualismus, a nikoli vysoké IQ.

Zvládají inteligenti chaos?

Posadíte-li do jedné kanceláře nebo k jednomu projektu dva chytré zastánce chaosu na několik dnů, ani jeden z nich nebude hovořit o ráji. Podobná situace vzniká i na SQL serveru. Pokud jsme jediným individuem, které aktualizuje data, a vybičujeme své duševní schopnosti na maximum, podaří se nám udržet v datech pořádek a nikdo zvenku nepozná, že máme nepořádek ve struktuře tabulek. Jednou však realizátora databáze může přestat pořizování dat bavit, jeho chaos se dostane do rukou jiných jedinců, kteří ho pak už nikdy nezvládnou, a to tím spíš, čím jich bude více.

V předchozích dílech seriálu jsem se příliš nestaral o zvládání chaosu, neboť v SQL DML k tomu nebyla příležitost a výklad DDL jsem trochu odlehčil pro snadnější porozumění. Nyní nastal čas tvrdého zásahu proti chaosu v datech. Hlavní smysl takového počínání je v tom, aby s daty mohli pracovat i ti lidé, kteří chaos prostě nezvládají. Vedlejší efekty jsou tři. Zabráníme předem rozvleklému reklamačnímu řízení při předávání systému. Dále zabráníme těm uživatelům systému, kteří by chtěli na našem chaosu vydělat prachy, aby tak učinili okamžitě. V neposlední řadě tím převychováme svou chaotickou analytickou duši.

SQL server se brání

A proto je tak prima, jak se to zpívá ve známé písni. V databázové teorii nalezneme základní definici databázového systému jako množiny entit (tabulek) a integritních omezení. Integrita po česku je celistvost neboli schopnost tvořit jeden bezesporný celek. Pokud má SQL server současně sloužit více klientům, musí buď všem dovolit všechno (a tak umožnit dokonalý chaos), nebo odmítat všechny rozporuplné požadavky na změnu dat (a tak tvrdě ochromit práci všech bordelářů). SQL server má obecné možnosti ochrany celistvosti dat a vy je budete muset v souladu se svou analýzou konkrétně naplnit. Přitom riskujete, že vás okolí bude kamenovat za to, že nelze vydat zboží, které není na skladě, že nelze poslat dopis firmě, jejíž adresa není známa, nebo že peníze nejdou poslat na účet, jehož majitel není znám. Nebojte se jich. Stejní lidé vás odsoudí i v případě, že se podaří zadat prodej milionu kusů nákladního automobilu, že budou muset psát adresu podruhé i kvůli faktuře nebo že dolary zmizí třeba do Alp. Navíc bych se k nim v druhém případě přidal i já. Pokud tedy plánujete, co všechno chcete zakázat, nezabouchněte sami sobě zadní vrátka. Proto plánujte vždy minimální počet integritních omezení, který již umožňuje obranu proti chaosu. Bude--li omezení příliš mnoho, bude v nich možná také chaos, který neunesete. Lepší heslo by možná bylo: Chaos v SQL může vzniknout i bez integritních omezení – rozumný analytik drží nejen uživatele, ale i sám sebe na uzdě.

Doménová integrita

Doménovou integritu už napůl znáte z CREATE TABLE. U každého sloupce tabulky se můžeme vyjádřit o doméně (množině dovolených hodnot), v níž se musí nalézat zadávané hodnoty. Pokud JMENO nesmí být NULL, CISLO_BOT nesmí být menší než 40 a OKRES může být pouze DC, UL nebo TP, stačí napsat uvnitř příkazu CREATE TABLE tři vnitřní deklarace sloupců:

```
JMENO VARCHAR(30) NOT NULL,
CISLO_BOT DECIMAL(2,0)
CHECK (CISLO_BOT>=40),
OKRES VARCHAR(2)
CHECK (SPZ IN(”DC”,”UL”,”TP”));
```

Jen si uvědomte, v kolika tabulkách se může vyskytnout rodné číslo s povinným lomítkem a nepovinnou poslední cifrou. Proto je lepší nejprve vytvořit tzv. doménu a v ní na jednom místě hlídat pravidla zápisu. Na takovou doménu se pak můžeme odvolávat jejím jménem jako na nový datový typ, který vznikl omezením jiného datového typu. Jediný háček je v tom, že při vytváření domény nevíme, jak se budou jmenovat sloupce při konkrétním a rozmanitém použití v CREATE TABLE. Podobně jako někteří muži říkají všem ženám BERUŠKO, aby se to nepletlo, budeme uvnitř domény nazývat její sloupec jako VALUE. Následuje vytvoření několika jednoduchých domén:

```
CREATE DOMAIN SLOVO
AS VARCHAR(30) NOT NULL;

CREATE DOMAIN RODNE
AS VARCHAR(11)
CHECK (VALUE LIKE ”_____/___%”);

CREATE DOMAIN NAZEV AS SLOVO CHECK (VALUE NOT LIKE ” %”);

CREATE DOMAIN MERITKO
AS DECIMAL(3,0) CHECK
```

(VALUE BETWEEN 30 AND 300);

První doména SLOVO je určena pro hlídání maximálně třicetiznakových neprázdných slov. Druhá doména RODNE hlídá jedenáctiznakový text, zda po šesti znacích následuje lomítko a za ním tři nebo čtyři znaky. Z toho automaticky plyne, že hodnota VALUE nemůže být NULL, neboť výsledek operátoru LIKE by nebyl YES. Třetí doména NAZEV vychází z domény SLOVO a upřesňuje, že VALUE nesmí začínat mezerou. Podobný trik se vám bude velmi často hodit. Poslední doména MERITKO bude prospěšná při ukládání hodnot výšky lidí v centimetrech. Máme-li domény připraveny, stačí je stručně a přehledně použít při zajišťování doménové integrity a zároveň ke stručnějšímu vytváření tabulky. Následující tabulka CLOVEK sice ještě nemá klíč, ale ještě chvíli vydržte do výkladu entitní integrity:

CREATE TABLE CLOVEK

(JMENO NAZEV, PRIJMENI NAZEV,

RC RODNE, VYSKA MERITKO);

Pokud nějakou doménu nepotřebujeme, můžeme ji zlikvidovat příkazem

DROP DOMAIN MERITKO;

Také vám vadí, že někteří lidé mají stejné křestní jméno jako příjmení? V království EXTRA s tím zatočili hned od začátku, a pak si marně lámali hlavu, jak pomocí domény, která zná jen tu svou VALUE, porovnat hodnoty ve dvou různých sloupcích. Pomocí domén to nejde. Druhý způsob realizace doménové integrity v SQL DDL spočívá v uvedení podmínky CHECK uvnitř CREATE TABLE, ale až za definicí všech sloupců tabulky. Obecně totiž platí, že integritní omezení, která nebyla napsána přímo při definici sloupců, lze psát až na konec příkazu CREATE TABLE. Týká se to nejen CHECK, ale i PRIMARY KEY, UNIQUE a FOREIGN KEY. Problém z EXTRA království lze řešit následovně:

CREATE TABLE EXTRA

(JMENO NAZEV, PRIJMENI NAZEV,

CHECK (JMENO<>PRIJMENI));

Entitní integrita

Entitní integritu také známe z počátečních dílů seriálu. Chceme-li zajistit jednoznačný přístup k řádku v rámci jedné entity (tabulky), musí existovat jednoduchý nebo složený klíč jako jeden sloupec tabulky nebo skupina sloupců, které nabývají unikátních hodnot. Při definici sloupce RC, který má být

unikátní, v tabulce CLOVEK stačí kombinovat doménovou a entitní integritu uvnitř příkazu CREATE TABLE:

RC RODNE UNIQUE;

Pokud si vzpomeneme na unikátnost dodatečně, umíme vytvořit unikátní indexový soubor příkazem

CREATE UNIQUE INDEX RCCLO

ON CLOVEK(RC);

Unikátní klíč má pro entitu (tabulku) dvojí význam. Prvotní význam pro tabulku v 5NF je v přímém přístupu k datům. Pak též hovoříme o primárním klíči, anglicky PRIMARY KEY. Každá tabulka v 5NF musí mít právě jeden primární klíč. Druhý význam unikátních klíčů je v kontrolní funkci dalších souvislostí mezi hodnotami v různých řádcích jedné tabulky. Pak používáme tradiční název UNIQUE. Takových unikátních klíčů může mít tabulka několik, tedy i žádný. Pokud by v království EXTRA používali kombinaci jména a příjmení k jednoznačné identifikaci osob, stačí v CREATE TABLE přidat PRIMARY KEY:

CREATE TABLE EXTRA

(JMENO NAZEV, PRIJMENI NAZEV,

CHECK (JMENO<>PRIJMENI),

PRIMARY KEY (JMENO, PRIJMENI));

V říši SUPER musel mít ještě každý občan tajnou přezdívku, nesouvisející ani se jménem, ani s příjmením, a navíc kombinace křestního jména a přezdívky musela vést k jednoznačnému oslovení agenta. Zde proto využijeme jak doménu NAZEV, tak tři přídavné doménové integrity, jeden primární klíč a jedno unikátní omezení. Jde o skloubení doménové a entitní integrity:

CREATE TABLE SUPER

(JMENO NAZEV, PRIJMENI NAZEV,

PREZDIVKA NAZEV,

CHECK (JMENO<>PRIJMENI),

```
CHECK (JMENO<>PREZDIVKA),
CHECK (PREZDIVKA<>PRIJMENI),
PRIMARY KEY (JMENO, PRIJMENI),
UNIQUE (JMENO, PREZDIVKA));
```

Referenční integrita

Tabulka SUPER je evidentně číselníkem, který se na nic neodkazuje. Jinak tomu bude s tabulkou MZDA, ve které plánujeme sloupec RC z domény RODNE, jehož hodnoty musí být ve vztahu k hodnotám sloupce RC z domény RODNE v tabulce JEDINEC, která má RC jako PRIMARY KEY. Nejde o nic jiného než o realizaci relace A : 1 MZDA PRO CLOVEKa. Tabulka JEDINEC je dokonalou databází zaměstnanců v 5NF s potřebnými osobními údaji. Tabulka MZDA, obsahující pouze rodné číslo, měsíc, rok a mzdové údaje, se pomocí rodného čísla spojuje s cizí tabulkou JEDINEC, na kterou se odkazuje. Právě představě o omezeních při spojení tabulek se říká referenční integrita. Omezení tohoto typu pomáhá zajistit nekonfliktnost obsahu dvojice tabulek v rámci každé jednotlivé relace. Nejlépe bude formulovat nejpřísnější možnou představu o nespornosti obsahu tabulek MZDA a JEDINEC. Především je zcela zbytečné hlídat referenční integritou rušení jednotlivých mezd, neboť tím, že někdo někdy nedostane mzdu, nevznikne rozpor v databázi. Přidávání nového zaměstnance do tabulky JEDINEC stačí hlídat pomocí doménové a entitní integrity a referenční integrita nehraje roli. Přidávání nové položky do tabulky MZDA je hlídáno referenční integritou. Existuje-li stejné rodné číslo v tabulce JEDINEC, proběhne přidání řádku bez problémů. Pokud stejné rodné číslo v tabulce JEDINEC neexistuje, referenční integrita je povinna zajistit, aby nedošlo ke sporu v datech. Z toho plyne, že nepřidá pomýlenou položku do tabulky MZDA.

To je užitečné pro zamezení výplaty mrtvým duším. Opačná situace nastává při rušení zaměstnanců v tabulce JEDINEC. Referenční integrita opět pomáhá hlídat následky našeho počínání. Rušíme-li zaměstnance, který ještě nikdy nedostal výplatu, je situace jasná a zrušení v tabulce JEDINEC proběhne. Na rušení zaměstnance, který již výplatu alespoň jednou dostal, můžeme nahlížet různě extrémně. Nejtvrdší je takový postup zakázat. Jedině tak lze zamezit anonymitě příjmů bývalých zaměstnanců. O dalších mírnějších možnostech bude pojednáno dále. Na opravu rodného čísla v tabulce JEDINEC nebo v tabulce MZDA musíme pohlížet jako na kombinaci rušení a vytváření. Změna rodného čísla v tabulce JEDINEC je možná, pokud dotyčný ještě nikdy nic nebral. Změna rodného čísla v tabulce MZDA je možná, pokud nové rodné číslo existuje v tabulce JEDINEC. Realizace takto přísné referenční integrity se provede při vytvoření tabulky MZDA, kde na konec přidáme

```
FOREIGN KEY(RC)
REFERENCES JEDINEC(RC);
```

Tím je definován vztah mezi sloupcem RC z naší tabulky MZDA a sloupcem RC z cizí tabulky JEDINEC; jde o nejtvrdší formu referenční integrity. Je-li v cizí tabulce UCET tvořen primární klíč

sloupci CU a CODE, představujícími číslo účtu a kód banky, pak se z tabulky STAV, obsahující CUCTU, BANKA, DATUM, PRIJEM a VYDEJ, musíme odkazovat deklarací v posledním řádku CREATE TABLE:

```
FOREIGN KEY(CUCTU,BANKA)
REFERENCES UCET(CU,CODE);
```

Extrémní tvrdost není vždy na místě. Co když někomu spletou rodné číslo v tabulce JEDINEC a zjistí se to až po mnoha měsících vyplácení mzdy? Pak požadavek zákazu změny RC v tabulce JEDINEC asi prakticky neobstojí. Lépe je si přát, aby se to všude přečíslovalo samo. Následující deklarace v tabulce MZDA ji činí mírně nesvéprávnou, neboť změna RC v cizí tabulce JEDINEC v ní nedobrovolně vynutí změny RC. K tomu slouží klauzule ON UPDATE CASCADE:

```
FOREIGN KEY(RC) REFERENCES
JEDINEC(RC) ON UPDATE CASCADE;
```

Pokud budeme ještě benevolentnější, je na místě přemýšlet, jak umožnit rušení položek v tabulce JEDINEC. Tak třeba firma v tabulce ZAKAZKA vede i RC zaměstnance, který za ni odpovídá, a může se stát, že pracovník dá výpověď. Pak zrušení pracovníka může být dovoleno s tím, že na zakázce bude uvedena místo rodného čísla hodnota NULL. Tím zahladíme stopy po zaměstnanci, ale jeho zakázky se neztratí. To se vyplatí i kvůli možnosti vést evidenci zakázek, které visí ve vzduchu. Do tabulky ZAKAZKA musíme tedy napsat odkaz na tabulku JEDINEC, a to s dvěma klauzulemi ON UPDATE CASCADE a ON DELETE NULL:

```
FOREIGN KEY(RC) REFERENCES
JEDINEC(RC) ON UPDATE CASCADE ON DELETE NULL;
```

Nestojíme-li o zakázky, které zaměstnanec měl na krku, provedeme i rušení kaskádně:

```
FOREIGN KEY(RC) REFERENCES
JEDINEC(RC) ON UPDATE CASCADE ON DELETE CASCADE;
```

Oznamka – malý projekt

Seriál o SQL se blíží ke svému rozuzlení. Proto by neškodilo zabývat se až do konce seriálu jednou názornou úlohou od stadia analýzy přes domény, tabulky, integrity a view až po uložené procedury, které budou předmětem jeho poslední části. Seznamku neboli seznamovací kancelář analyzovat nebudeme. Mnohem více se toho dá naučit analýzou “oznamky”, tj. instituce, kam se chodí splňovat čestná oznamovací povinnost za úplatu. Cílem mé analýzy je přesvědčit vás, že jádro oznamky tvoří tři tabulky: CLOVEK, CIN a UDANI. Pokud vám chybí tabulka UDAVAC, pak asi nevíte, že udavač je také jenom (bohužel) člověk a lidé se mohou udávat vzájemně u jedné a téže instituce. Bylo by analytickou chybou to nedovolit. Každý typ činu má svůj název a cenu pro případ běžného udání. Udá-li konkrétní osoba jinou konkrétní osobu, že v určitý den spáchala sledování hodný čin, může dostat i vyšší odměnu, jde--li o prominentního udavače nebo prominentní oběť udání. Z toho potom plyne, že u každého člověka musí být znám jeho koeficient udavače a koeficient oběti jako čísla větší nebo rovná jedné. Jedničky by se měly samy doplňovat v případě neznámých koeficientů. Odměna za jedno konkrétní udání je rovna součinu koeficientu udavače, koeficientu oběti a základní ceny činu. Celková měsíční odměna se týká pouze udavačů a je rovna součtu cen za měsíc. U každé oběti se naopak sleduje intenzita sledování ve finančních jednotkách. Výkonnost agentů, mohutnost činů či sledovanost obětí lze sledovat nejen po stránce ekonomické, ale i statistické. Existují však některá zásadní omezení oznamovacího režimu. Nikdo nesmí udávat sám sebe a navíc nesmí v jednom dni udat jinou konkrétní osobu v souvislosti s jedním typem činu víc než jedenkrát. Anonymní udání, udání neznámé osoby a udání nespecifikovaného činu jsou zakázány. Analýza končí dalšími databázovými omezeními spojenými s konkrétním návrhem tabulek.

Tabulka CIN má sloupce CIC, NAZEVC a CENAC. Primárním klíčem je CIC jako číslo činu. Název činu NAZEVC musí být unikátní a základní cena činu CENAC nesmí být záporná. Tabulka CLOVEK má jako primární klíč rodné číslo osoby RC. Následují sloupce JMENO, PRIJMENI, KOEUD a KOEOB, kde poslední dva představují koeficient udavače a koeficient oběti. Poslední tabulka UDANI se odvolává na údaje z předchozích dvou tabulek. Obsahuje denní záznamy ve sloupcích DEN, RCUD, RCOB a CICINU.

Primární klíč je určen celou čtveřicí sloupců. Není divu, když tabulka UDANI je spojovací entitou v 5NF a zároveň má umožňovat opakované udání jiný den. Sloupce RCUD a RCOB nesou informaci o rodném čísle udavače a oběti, a proto musí být cizími klíči pro dvojité spojení s tabulkou CLOVEK. Sloupec CICINU obsahuje číslo oznámeného činu a je cizím klíčem pro spojení s tabulkou CIN. Sloupec DEN obsahuje pouze datum udání. Mezi tabulkami tedy existují tři relace A : 1, a to UDANI POSKYTL CLOVEK, UDANI O CLOVEKu a UDANI O CINu. Analýza nepředpokládá evidenci míst, kde byly činy spáchány nebo hlášeny.

```
CREATE DOMAIN KOEFICIENT

AS DECIMAL(6,3) DEFAULT 1

CHECK (VALUE >=1);


CREATE DOMAIN PRACHY

AS DECIMAL(10,2) DEFAULT 0

CHECK (VALUE>=0);
```

```
CREATE DOMAIN CISLO
AS INTEGER CHECK (VALUE >0);
```

Další dvě domény RODNE a NAZEV máme již hotové z předchozího textu. Nyní s využitím doménové a entitní integrity vytvoříme oba číselníky CIN a CLOVEK:

```
CREATE TABLE CIN (CIC CISLO, NAZEVC NAZEV, CENAC PRACHY,
PRIMARY KEY (CIC), UNIQUE (NAZEVC));

CREATE TABLE CLOVEK (RC RODNE,
JMENO NAZEV, PRIJMENI NAZEV,
KOEUD KOEFICIENT,
KOEOB KOEFICIENT,
PRIMARY KEY (RC));
```

Při vytvoření tabulky UDANI vyjdeme z existence tabulek CIN A CLOVEK a použijeme všechny typy integrit:

```
CREATE TABLE UDANI
(DEN DATETIME, RCUD RODNE,
RCOB RODNE, CICINU CISLO,
PRIMARY KEY
(DEN, RCUD, RCOB, CICINU),
CHECK (RCUD<>RCOB),
FOREIGN KEY (RCUD)
REFERENCES CLOVEK(RC),
FOREIGN KEY (RCOB)
REFERENCES CLOVEK(RC),
```

```
FOREIGN KEY (CICINU)
REFERENCES CIN(CIC));
```

Pokračujeme příště.

Jaromír Kukal

```
CREATE VIEW OBET(RC, JMENO, PRIJMENI, KOEFICIENT, POCET, CELKEM, UPRAVENO)
AS SELECT RCO, JMENOO, PRIJMENIO, KOEO, COUNT(*), SUM(PRACHY),SUM(FINAL)
FROM UDANICKO GROUP BY RCO;

CREATE VIEW PRECIN(CISLO, NAZEV, CENA, POCET, CELKEM,UPRAVENO)
AS SELECT CIC, NAZEVC, PRACHY, COUNT(*), SUM(PRACHY),SUM(FINAL)
FROM UDANICKO GROUP BY CIC;
```

Pod vlivem minulých dílů seriálu snadno vytvoříme pohledy na rodná čísla udavačů a obětí:

```
CREATE VIEW RCUDAV(RC) AS SELECT RCU FROM UDANI GROUP BY RCU;

CREATE VIEW RCOBET(RC) AS SELECT RCO FROM UDANI GROUP BY RCO;
```

Poslední view je pro vás malým rébusem:

```
CREATE VIEW MEDAILE(RC, JMENO, PRIJMENI) AS
SELECT RC, JMENO, PRIJMENI FROM OBET
WHERE RC NOT IN RCUDAV;
```

Pro završení trpkého humoru se vraťme do DML a zadejme několik příkazů SELECT, které prověří důkladnost předchozí přípravy:

```
SELECT TOP 10 PERCENT * FROM UDAVAC ORDER BY POCET DESC;

SELECT * FROM UDANICKO WHERE RCU IN RCOBET OR RCO IN RCUDAV;

SELECT COUNT(*) POCET_KUSU FROM MEDAILE;
```