Databáze a jazyk SQL

Archiv 4.8.2000 16

Pojem databáze dnes není zcela jistě nikomu cizí. Lidé mají potřebu evidovat a shromažďovat informace už odpradávna. Celá dnešní moderní společnost je postavena na databázových systémech, od evidence občanů, přes zdravotnictví, hospodářství, školství, až po letectví, výzkum, nebo síť mobilních telefonů.

Následující seriál článků věnujících se SQL, vás seznámí nejen se základními pojmy z oblasti databází, ale i s konkrétními vlastnostmi a použitím jazyka SQL jako nástroje pro vytvoření databáze a manipulaci s daty. Jazyk SQL je standardním nástrojem, ze kterého většina dnes používaných databázových systémů vychází a v různé míře tyto systémy tento standard dodržují.

V prvním článku vás seznámím se základní teorií databází, která je nesmírně důležitá pro jakoukoliv další práci v oblasti databázových systémů, ať už budete pracovat jako koncový uživatel, nebo návrhář, či programátor informačního systému.

Vývoj databází

Nyní se vraťme asi o 40 let zpět. Právě v 60. letech tohoto století vzniká současný pojem databáze, entita, atribut entity a vazba mezi entitami. To jsou základní pojmy, o kterých se na úvod zmíním.

Databázi si lze představit jako soubor dat, který slouží pro popis reálného světa (např. evidence školní knihovny, sklad chemikálií, evidence studentů). **Entita** je prvek reálného světa (např. člověk, stroj, vyučovaný předmět, město), který je popsán svými charakteristikami (vlastnostmi). Ty se většinou považují za **atribut** (např. jméno, příjmení, stav, plat, hmotnost).

Dalším důležitým pojmem je **vazba mezi entitami**. Jednotlivé entity odpovídající prvkům z reálného světa, mají mezi sebou určitý vztah. Např. každý člověk má právě jedny osobní údaje vedené na magistrátě, na oddělení občanských průkazů. To hovoříme o vazbě typu 1:1. Dalším typem je vazba 1:N, jíž bude odpovídat např. skutečnost, že jeden člověk může vlastnit více kreditních karet (ale jedna kreditní karta může být vlastněna pouze jedním člověkem). Posledním typem vazby je M:N. Zde není žádné omezení, příkladem by mohla být situace, že student na vysoké škole si může zapsat několik různých předmětů (ale jeden předmět může být zároveň zapsán více studenty).

V této době vzniká ještě jeden pojem, a to **databázový model**. Ten byl zaveden zejména matematiky, jako prostředek pro popis databáze. Zpočátku se používaly modely dvojího typu: **hierarchický** (založen na modelování hierarchie mezi entitami se vztahy podřízenosti a nadřízenosti) a dále **síťový** (vychází z teorie grafů, uzly v grafu odpovídají entitám a orientované hrany definují vztahy mezi entitami). V 70. letech se

uvedené databázové modely ukázaly být nedostatečné (objevily se problémy s realizací a implementací vazby M:N), a proto vznikl **relační model**, který se stal standardem a používá se dodnes.

Relační databáze

Podívejme se nyní blíže na relační model. Základním pojmem je **relace**. Relaci, aniž bych zaváděl jakékoliv matematické definice, si lze představit jako tabulku, která se skládá ze sloupců a řádků. Sloupce odpovídají jednotlivým vlastnostem (atributům) entity. Údaje v jednom řádku tabulky zobrazují aktuální stav světa. Budu mít např. tabulku

```
ZAMĚSTNANEC
```

, která bude popisovat entitu pracovníka ve firmě. Sloupce tabulky budou:

```
ČÍSLO
```

,

```
JMÉNO
```

,

```
PŘÍJMENÍ
```

,

```
DAT_NAR
```

,

```
PLAT
```

a

```
SMLOUVA_OD
```

. Atribut

```
DAT_NAR
```

značí datum narození pracovníka a

```
SMLOUVA_OD
```

uvádí datum, od kterého je pracovník v naší firmě zaměstnán. Představme si, že reálnému světu odpovídá následující naplnění tabulky:

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAR	PLAT	SMLOUVA_OD
1	jan	novák	15.10.75	15000	1.1.2000
2	petr	nový	1.4.78	21500	12.5.1999
3	jan	nováček	6.9.65	17500	7.7.1998

Tabulka je základním stavebním kamenem pro budování celé databáze. Relace tedy odpovídá celé tabulce a prvku relace odpovídá jeden konkrétní řádek. Jeden řádek bývá často nazýván **databázovým záznamem**. Soubor tabulek (relací) pak tvoří celou databázi (**relační schéma**).

U tabulek ještě chvíli zůstaňme. Jedna tabulka nám popisuje nějakou entitu. Za sloupce v tabulce zvolíme ty atributy, které o dané entitě chceme evidovat a které nás zajímají. Nyní je nejvyšší čas říci si něco o tvorbě relačních tabulek. Tvorba „dobře navržených“ tabulek je klíčový a důležitý úkol zejména z hlediska dlouhodobého. (Pokud bychom v nějakém databázovém systému, který již nějakou dobu běží v ostrém

provozu, nalezli nějakou chybu a zjistili bychom, že spočívá ve špatně navržené databázi, mělo by to katastrofální důsledky, a to nejen po finanční stránce.) Než budu pokračovat dál, vysvětlím další základní pojmy z oblasti relační databáze.

Základní pojmy

Hodnotou většinou rozumíme uživatelská data. Každý sloupec v tabulce má svůj **datový typ** (např. celé číslo, řetězec, datum, logická hodnota, apod).

Pro práci s databázovými tabulkami je užitečné (ne-li přímo nutné) mít alespoň jednu položku (sloupec), jejíž hodnota nám bude jednoznačně identifikovat záznam v tabulce. Pokud taková položka nebude příliš velká (např. v počtu bajtů), zvolíme ji za tzv. **primární klíč**. V našem příkladu tabulky

```
ZAMĚSTNANEC
```

lze za primární klíč zvolit položku

```
ČÍSLO
```

. Primární klíč má tu vlastnost, že jeho hodnota je jedinečná, tj. pro žádné dva řádky v tabulce nemůže nastat situace, že by hodnota primárního klíče byla totožná. Databázové systémy většinou umožňují definovat jako primární klíč n-tici položek, např. dvojici nebo trojici položek. V takovém případě se mohou některé položky v klíčích opakovat, ale nesmí být shodné všechny položky dvou primárních klíčů najednou.

Neméně důležitá je i **funkční závislost**. Funkční závislosti si lze představit jako tvrzení o reálném světě. Například plat zaměstnance závisí na tom, jakou vykonává funkci, tj. plat závisí na funkci, zapisujeme

```
FUNKCE->PLAT
```

. Druhým příkladem by mohla být výše jízdného, která závisí na délce vlakové trasy, tj.

```
DÉLKA_TRASY->VÝŠE_JÍZDNÉHO
```

. Takových příkladů z běžného života bychom našli mnoho.

Normální formy

Pojem normálních forem se používá ve spojitosti s dobře navrženými tabulkami. Správně vytvořené tabulky splňují 4 základní normální formy.

1. normální forma (1NF)

První, nejjednodušší, normální forma (značíme **1NF**) říká, že všechny atributy jsou atomické, tj. dále již nedělitelné (jinými slovy, hodnotou nesmí být relace). Mějme např. tabulku

```
ADRESA
```

, která bude mít sloupce

```
JMÉNO
```

,

```
PŘÍJMENÍ
```

a

```
BYDLIŠTĚ
```

. Naplnění tabulky nechť odpovídá reálnému světu:

JMÉNO	PŘÍJMENÍ	BYDLIŠTĚ
jan	novák	Ostravská 16, Praha 16000
petr	nový	Svitavská 8, Brno 61400
jan	nováček	Na bradlech 1147, Ostrava 79002

Pokud bychom v této tabulce chtěli vypsat všechny pracovníky, jejichž PSČ je rovno určité hodnotě, dostali bychom se do potíží, neboť bychom to nemohli zjistit přímo a jednoduše. A to proto, že atribut `BYDLIŠTĚ` není atomický, skládá se z několika částí: `ULICE`, `ČÍSLO`, `MĚSTO` a `PSČ`. Správný návrh tabulky, který bude respektovat **1NF** bude vypadat následovně:

JMÉNO	PŘÍJMENÍ	ULICE	ČÍSLO	MĚSTO	PSČ

jan	novák	Ostravská	16	Praha	16000
petr	nový	Svitavská	8	Brno	61400
jan	nováček	Na bradlech	1147	Ostrava	79002

Obecně bychom se měli snažit, aby obsahem jedné databázové položky byla právě jedna hodnota (určitého databázového typu).

2. normální forma (2NF)

Tabulka splňuje **2NF**, právě když splňuje **1NF** a navíc každý atribut, který není primárním klíčem je na primárním klíči úplně závislý. To znamená, že se nesmí v řádku tabulky objevit položka, která by byla závislá jen na části primárního klíče. Z definice vyplývá, že problém **2NF** se týká jenom tabulek, kde volíme za primární klíč více položek než jednu. Jinými slovy, pokud má tabulka jako primární klíč jenom jeden sloupec, pak **2NF** je splněna triviálně. Nechť máme tabulku

`PRACOVNÍK`

, která bude vypadat následovně (atribut

`ČÍS_PRAC`

značí číslo pracoviště, kde daný pracovník pracuje, atribut

`NÁZEV_PRAC`

uvádí jméno daného pracoviště):

ČÍSLO	JMÉNO	PŘÍJMENÍ	ČÍS_PRAC	NÁZEV_PRAC
1	jan	novák	10	studovna
2	petr	nový	15	centrála
3	jan	nováček	10	studovna

Jaký primární klíč zvolíme v této tabulce? Pokud zvolíme pouze

```
ČÍSLO
```

, je to špatně, neboť zcela určitě název pracoviště, kde zaměstnanec pracuje, není závislý na číslu pracovníka. Takže za primární klíč musíme vzít dvojici (

```
ČÍSLO,ČIS_PRAC
```

). Tím nám ovšem vznikl nový problém. Položky

```
JMÉNO
```

,

```
PŘÍJMENÍ
```

a

```
NÁZEV_PRAC
```

nejsou úplně závislé na dvojici zvoleného primární klíče. Ať tedy děláme, co děláme, nejsme schopni vybrat takový primární klíč, aby tabulka splňovala **2NF**. Jak z tohoto problému ven? Obecně převedení do tabulky, která již bude splňovat **2NF**, znamená rozpad na dvě a více tabulek, kde každá už bude splňovat **2NF**.

Takovému „rozpadu“ na více tabulek se odborně říká **dekompozice relačního schématu**. Správně navržené tabulky splňující **2NF** budou vypadat následovně (tabulka

```
PRACOVNÍK
```

a

```
PRACOVIŠTĚ
```

):

ČÍSLO	JMÉNO	PŘÍJMENÍ	ČIS_PRAC
1	jan	novák	10
2	petr	nový	15
3	jan	nováček	10

ČÍSLO	NÁZEV
10	studovna
15	centrála

Dále si všimněte, že pokud tabulka nesplňuje **2NF**, dochází často k redundanci. Konkrétně v původní tabulce

informace, že pracoviště číslo 10 se jmenuje „studovna“, byla obsažena celkem dvakrát. **Redundance** je jev, který obvykle nesplnění **2NF** doprovází. O tom, že redundance je nežadoucí, netřeba pochybovat. Zkuste si rozmyslet, jak byste postupovali v obou příkladech, kdyby ve vaší společnosti došlo ke změně názvu pracoviště číslo 10 ze „studovna“ na „klubovna“.

3. normální forma (3NF)

Relační tabulky splňují třetí normální formu (**3NF**), jestliže splňují **2NF** a žádný atribut, který není primárním klíčem, není tranzitivně závislý na žádném klíči. Nejlépe to opět vysvětlí následující příklad. Mějme tabulku `PLATY`, která bude vypadat takto:

ČÍSLO	JMÉNO	PŘÍJMENÍ	FUNKCE	PLAT
1	jan	novák	technik	15000
2	petr	nový	vedoucí	21500
3	jan	nováček	správce	17500

Pomineme zatím fakt, že tato tabulka nesplňuje ani **2NF**, což je základní předpoklad pro **3NF**. Chci zde jen vysvětlit pojem **tranzitivní závislost**. Nebudeme přemýšlet, co je primární klíč, na první pohled vidíme, že konkrétně atributy

`JMÉNO`

,

`PŘÍJMENÍ`

a

`FUNKCE`

závisí na atributu

`ČÍSLO`

(ten by nejspíš byl primárním klíčem). Dále můžeme vidět, že atribut

`PLAT`

zřejmě je funkčně závislý na atributu

`FUNKCE`

a pokud vememe v úvahu, že

`ČÍSLO->FUNKCE`

a

`FUNKCE->PLAT`

, dostaneme díky jevu nazývanému *tranzitivita*, že

`ČÍSLO->PLAT`

. Postup, jak dostat tabulky do **3NF**, je podobný jako v případě **2NF**, tj. opět provedeme dekompozici (tabulka

`FUNKCE`

a

`PLATY`

):

ČÍSLO	JMÉNO	PŘÍJMENÍ	FUNKCE
1	jan	novák	technik
2	petr	nový	vedoucí
3	jan	nováček	správce

FUNKCE	PLAT
technik	21500
vedoucí	17500
správce	15000

Z hlediska základních tří normálních forem, jsou tyto dvě tabulky již v pořádku. Z praktického hlediska je vhodnější použít nějaký číselník funkcí, abychom splnili podmínku, že primární klíč v tabulkách má být co nejkratší délky. Nejlepší zápis je tedy následující:

ČÍSLO	JMÉNO	PŘÍJMENÍ	CIS_FUN

1	jan	novák	121
2	petr	nový	156
3	jan	nováček	127

ČÍSLO	FUNKCE	PLAT
121	technik	21500
156	vedoucí	17500
127	správce	15000

Boyce-Coddova normální forma

Poslední prakticky užívanou formou je tzv. **Boyce-Coddova normální forma** (**BCNF**). Tabulka splňuje **BCNF**, právě když pro dvě množiny atributů

```
A
```

a

```
B
```

platí:

```
A->B
```

a současně

```
B
```

není podmnožinou

```
A
```

, pak množina

```
A
```

obsahuje primární klíč tabulky. Tato forma zjednodušuje práci s tabulkami, ve většině případů, pokud dobře postupujeme při tvorbě tabulek, aby splňovaly postupně **1NF**, **2NF** a **3NF**, forma **BCNF** je splněna.

Jazyk SQL

Po delším úvodu, který ale považuji za velmi důležitý pro pochopení základních principů a pojmů ze světa databází a který vám poskytuje návod, jak správně navrhnout databázové tabulky, se konečně dostávám k jazyku **SQL**.

Historie jazyka **SQL** spadá do 70. a 80. let. První standard byl přijat v roce 1986 (označován jako *SQL86*). Časem se však projevily některé nedostatky. Opravená verze je z roku 1992 a je označována jako *SQL92*. Ten je v oblasti relačních databází standardem dodnes. Zkratka SQL značí **Structured Query Language**. Jazyk v sobě zahrnuje nástroje pro tvorbu databází (tabulek) a dále nástroje na manipulaci s daty (vkládání dat, aktualizace, mazání a vyhledávání informací).

SQL patří mezi tzv. **deklarativní programovací jazyky**, což v praxi znamená, že kód jazyka SQL nepíšeme v

žádném samostatném programu (jako by tomu bylo např. u jazyka C nebo Pascal), ale vkládáme jej do jiného programovacího jazyka, který je již procedurální. Se samotným jazykem SQL můžeme pracovat pouze v případě, že se terminálem připojíme na SQL server a na příkazový řádek bychom zadávali přímo příkazy jazyka SQL.

Jak už jsem se zmínil, SQL se skládá z několika částí. Některé části jsou určeny pro administrátory a návrháře databázových systémů, jiné pak pro koncové uživatele a programátory. První částí jazyka SQL je jazyk *DDL* – **Data Definition Language**. Jedná se o jazyk pro vytváření databázových schémat a katalogů. Způsob ukládání tabulek definuje jazyk *SDL* – **Storage Definition Language**. Třetí částí pro návrháře a správce je jazyk *VDL* – **View Definition Language**, určující vytváření pohledů (pohled si lze představit jako virtuální tabulku složenou z různých jiných tabulek). Poslední částí, kterou se budu převážně zabývat, je jazyk *DML* – **Data Manipulation Language**, který obsahuje základní příkazy

```
INSERT
```

,

```
UPDATE
```

,

```
DELETE
```

a nejpoužívanější příkaz

```
SELECT
```

. S jazykem DML pracují nejvíce koncoví uživatelé a programátoři databázových aplikací.

Starší komentáře ke článku

Pokud máte zájem o starší komentáře k tomuto článku, naleznete je zde.

TAGY ČLÁNKY

TWITTER FACEBOOK GOOGLE +

PŘEDCHOZÍ ČLÁNEK
Tipy pro optimalizaci webových stránek

DALŠÍ ČLÁNEK
PHP - vytváříme vlastní funkce v PHP

Jaromír Skřivan

Mohlo by vás zajímat

Mozilla vytváří nový prohlížeč pro vývojáře

6.11.2014 *0*

Zoner vs. EET

15.5.2017 *0*

16 Příspěvků v diskuzi

Miranu

21.8.2009 at 11:15

Prosil bych o postup jak se přihlásit k nějakému serveru, nebo jak dělat na servru 2000, popř. 2008 (MS – MYSQL sever 2000, 2008) abych se mohl učit přímo na něm a né jen koukáním sem...
Jsem v tomto začátečník, avšak access ovládám na průměrné až ecpertní úrovni.
Díky mooc

Odpovědět

aattto

4.5.2010 at 15:01

trapnýýýýýýýýýý toje husty a mise topak musíme učit

Odpovědět

pecka

8.5.2010 at 14:26

tak nevim jestli autor opisoval ze skript VUT a nebo naopak jesltli nekdo z VUT opsal do skript tento clanek...

Odpovědět

Miroslav Kucera

8.5.2010 at 15:11

No, vzhledem k tomu, ze clanek je stary uz deset let, asi to jen stezi zjistime :-)

Odpovědět

majkl

9.6.2010 at 13:51

Sice stare, ale porad dobre!

Odpovědět

Anna

29.8.2010 at 21:41

Pěkně srozumitelně napsáno, díky! Ne všechny články o SQL počítaj i s úplnými laiky...

Odpovědět

Ondra

13.9.2010 at 13:03

Vyborny clanek... jasne a srozumitelne napsany.

Odpovědět

Anonym

5.11.2010 at 11:10

to myslíte vážně???

Odpovědět

Petr Vystyd

22.2.2011 at 11:26

Velice prinosne!

Odpovědět

Jirka

25.2.2011 at 14:06

Souhlasím s Annou-viz její komentář z 29.8.2010.

Odpovědět

Asitaka

18.3.2011 at 18:27

Velmi srozumitelné a přehledné skvělé pro začínající programátory.

Odpovědět

Anonym

7.1.2012 at 16:53

Autor komentáře: pecka

Datum vložení: 08. Květen 2010, 14:26:38

tak nevim jestli autor opisoval ze skript VUT a nebo naopak jesltli nekdo z VUT opsal do skript tento clanek...

myslím že autor skript a autor článku je jedna a ta samá osoba.

Odpovědět

Nabuu

15.1.2012 at 18:43

Diky hlavne za ty NF. Na škole jsem to nějak nepochytil..

Odpovědět

Killi

19.3.2012 at 11:27

Picovina

Odpovědět

Anonym

14.2.2014 at 0:45

Kurva, může někdo napsat postup, jak, nebo kde se to můžu začít učit a vyzkoušet si to?

Odpovědět

Kulíšek

3.1.2016 at 17:49

Super článek, kdyby se to takhle vykládalo i ve školách tak nemusím trávit u učiva dvojnásobek času

Odpovědět

Odpovědět

Komentář

Jméno

Email

Web

Vložit komentář