Výuková prezentace 3

6MDBS1

Databázové systémy

Ing. Vladimír Přibyl, Ph.D.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Databázový návrh

Logický návrh

- Převod konceptuálního modelu na logickou strukturu odpovídající zvolenému typu DBMS, v našem případě tedy na relační strukturu
  - Entita = relace = tabulka
  - Výskyt entity = n-tice = záznam = řádek tabulky
  - Atribut = sloupec tabulky (pole)
  - Vztahy = vztahy mezi relacemi (tabulkami)

Vlastnosti relací (tabulek)

- jedinečná v rámci databáze
- atributy (sloupce) mají jedinečná jména
- atributy uchovávají jen jednu hodnotu
- všechny hodnoty v jednom sloupci jsou ze stejné domény
- pořadí atributů (sloupců) není významné
- pořadí záznamů (řádků) není významné
- záznamy jsou jedinečné (tabulka neobsahuje duplicitní řádky)
  - Tato vlastnost se v určitých situacích v reálných databázových systémech striktně nevyžaduje

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Typy relací

- entitní relace
  - n-tice atributů popisující vlastní entity(objekty)
- vztahové relace
  - n-tice atributů tvořících klíče entit vstupujících do vztahu
  - n-tice atributů popisujících samotný vztah

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Realizace vztahových relací

- přímo
  - využití primárních klíčů jako cizích klíčů
  - lze takto realizovat vazby 1:1 a 1:N
- pomocí pomocné tabulky
  - takto lze realizovat všechny typy bez omezení

Normalizace datového modelu

- Technika používaná pro vytvoření sady tabulek (relací) s minimální redundancí, která podporuje všechny požadavky návrhu databáze.
  - Je obvyklé, že díky normalizaci mohou vzniknout další tabulky
- Bylo definováno několik tzv. normálních forem. (1. NF, 2. NF …)
- Každá vyšší normální forma zahrnuje splnění podmínek předchozí a přidává další podmínky.
- Normalizace je postavená na analýze funkčních závislostí mezi atributy.

1. Normální forma

- Základní forma, každá tabulka, která splňuje podmínky pro relaci je v 1. NF.
- Podmínky 1. NF
  - Každý atribut musí obsahovat jen jediný údaj.
  - Hodnoty atributů jsou atomické (dále významově nedělitelné).
- Příklad tabulky, která není v 1NF.

Pracovník	Adresa	Telefony
Jan Novák	Havlíčkova 2 Praha 3	240957235, 728365421
Eva Horáková	Svatoplukova 15 Brno	395425647, 731456741

1. Normální forma

- Náprava rozdělením do více sloupců

Jméno	Příjmení	Adresa	Telefon pevný	Telefon mobilní
Jan	Novák	Havlíčkova 2 Praha 3	240957235	728365421
Eva	Horáková	Svatoplukova 15 Brno	395425647	731456741

- Flexibilnější náprava rozdělením do více tabulek

ID	Jméno	Příjmení	Adresa
1	Jan	Novák	Havlíčkova 2 Praha 3
2	Eva	Horáková	Svatoplukova 15 Brno

ID pracovnika	Tel.	Typ tel.
1	728365421	Mobilní
1	240957235	Pevný
2	731456741	Mobilní
2	395425647	pevný

2. Normální forma

- Relace je v 2. NF pokud je v 1. NF a platí, že každý neklíčový atribut je funkčně závislý na všech částech primárního klíče a žádné jeho podmnožině.
- Má smysl jen tehdy, je-li primární klíč složený.
- Řešením je rozklad na dvě relace.

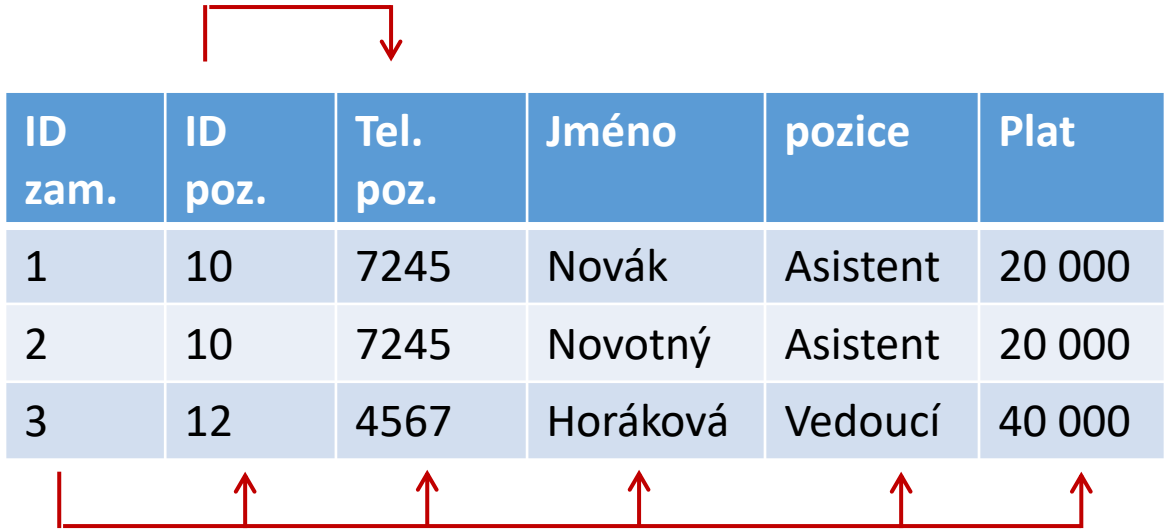
ID zam.	ID poz.	Jméno	pozice	Hod. týdně
1	10	Novák	Asistent	20
1	11	Novák	Asistent	20
2	12	Horáková	Vedoucí	40

ID zam.	Jméno	pozice
1	Novák	Asistent
1	Novák	Asistent
2	Horáková	Vedoucí

ID zam.	ID poz.	Hod. týdně
1	10	20
1	11	20
2	12	40

3. Normální forma

- Relace je v 3. NF pokud je v 1NF a 2NF a platí, že každý neklíčový atribut je funkčně závislý jen na primárním klíči a žádném jiném atributu či atributech.
- Řeší tzv. tranzitivní závislosti.



ID zam.	ID poz.	Tel. poz.	Jméno	pozice	Plat
1	10	7245	Novák	Asistent	20 000
2	10	7245	Novotný	Asistent	20 000
3	12	4567	Horáková	Vedoucí	40 000

ID zam.	ID prac.	Jméno	pozice	Plat
1	10	Novák	Asistent	20 000
2	10	Novotný	Asistent	20 000
3	12	Horáková	Vedoucí	40 000

ID poz.	Tel. poz.
10	7245
12	4567

Fyzický návrh

- Realizace logického modelu v konkrétním DBMS
- Nastavení datových typů
- Nastavení omezení polí a záznamů
- Volba a nastavení indexů
- Zajištění referenční integrity

Nastavení omezení

- Omezení jednotlivých polí tabulky (column constraints)
  - Kontrola rozsahu hodnot
  - Kontrola formátu hodnot
  - Nastavení parciality/totality pole
  - Kontrola duplicity (většinou prostřednictvím indexů)
- Omezení uplatňovaná na úrovni celých záznamů
  - Každý záznam musí splňovat omezení ve tvaru podmínky WHERE z příkazu SELECT z SQL
- Referenční omezení
  - Definice cizích klíčů (foreign key) a závislostí záznamů, resp. akcí prováděných při updatech, či mazáních ve svázaných tabulkách.
  - Cílem je dosáhnout referenční integrity.

Referenční integrita

- Její porušení jsou chyby ve vazbách
- Definice: hodnota cizího klíče musí existovat jako hodnota primárního klíče některé relace nebo musí být hodnota cizího klíče prázdná.
- Součástí správy ref. integrity může být také mazání, resp. aktualizace svázaných polí v kaskádě.
- Kontrola:
  - Automatizovaně – většinou malé dat. systémy (např. MS Access)
  - Součást definice omezení tabulky
  - Prostřednictvím speciálních, automaticky spouštěných, ale plně editovatelných, procedur – tzv. TRIGGERů
    - vyšší flexibilita
    - možnost výpočtů a porovnání dat

Indexy

- Index je datová struktura, která umožňuje DBMS rychleji lokalizovat konkrétní záznamy v datovém souboru tabulky. Tím zrychluje odezvu na dotazy.
- Typy indexů
  - Primární – souvisí s primárním klíčem a často (např. MS Access) vzniká automaticky
    - maximálně jeden
  - Sekundární (vedlejší) – index definovaný na základě neklíčového a obvykle nesetříděného pole (případně polí)
    - může jich být víc
  - Indexy mohou být jednoduché (obvykle) nebo složené, podobně jako klíče.
- Pokud se jedná o jedinečné indexy, jsou zároveň prostředkem pro zamezení duplicit v polích.

Volba sekundárních indexů

- Údržba indexů vyžaduje dodatečný výkon a prostor v paměti, proto vytváříme indexy uvážlivě s ohledem na přínos a jeho poměr vůči zvýšeným nárokům.
- Obecná pravidla pro vytváření indexů
  - Indexovat primární klíč (pokud není automaticky)
  - Sekundární indexy kandidátně pro pole, která jsou často využívána pro vyhledávání, spojení, seskupování a řazení.
  - Neindexovat pole, která se často aktualizují.
  - Neindexovat malé tabulky (malý užitek z indexů).
  - Neindexovat pole obsahující dlouhé textové řetězce.

Administrace databází

Administrace databází

- Soubor činností, jejichž cílem je zajistit vývoj, použitelnost a bezpečnost databáze.
- Nějaký druh a rozsah administrace vyžaduje každý typ a velikost databáze, ale rozsah a složitost mohou být velmi rozdílné:
  - malý jednouživatelský systém – administrace je obvykle jednoduchá a dělá ji často uživatel sám.
  - velký systém pro mnoho uživatelů, obsahující důležitá data – administrace je náročný proces, na jehož realizaci jsou zapotřebí často i celá oddělení.
- V další části se budeme zaměřovat především na:
  - zajištění konzistence a integrity dat
  - kontrolu souběžnosti
  - zabezpečení a zálohování s obnovením

Konzistence a integrita dat

- **Integrita** vyjadřuje korektnost dat, tedy stav, kdy data obsahují jen přípustné hodnoty, které odpovídají realitě.
- **Integrita dat v tabulkách**
  - korektnost hodnot je zajišťována omezeními
    - datový typ, kontrola prázdných hodnot (NOT NULL), omezení polí a tabulek (CONSTRAINT, CHECK)
    - viz příkaz CREATE TABLE
- **Referenční integrita**
  - korektnost vazeb
  - viz příkaz CREATE TABLE (CONSTRAINT FOREIGN KEY)
- Ve složitějších případech slouží k udržování integrity **triggery** (spouště)

Trigger

- Procedura, která se **automaticky spouští** v souvislosti s vykonáním příslušné akce na tabulce, pro kterou je definovaná.
  - triggery mohou být definovány pro akce **vkládání, aktualizace a odstraňování**
- V rámci triggeru je možné definovat v podstatě libovolně složitou operaci, podobně jako v případě uložené procedury.
  - může tedy řešit jak referenční integritu, tak integritu dat, případně konzistenci dat v rámci databáze (např. aktualizaci počítaných, záměrně duplicitních dat)
- Triggery byly implementovány v databázových systémech dříve, než byly standardizovány v rámci SQL, takže je jejich implementace v různých systémech částečně odlišná.

Trigger

- Syntaxe standardu SQL:

  CREATE TRIGGER <název triggeru>

  {**BEFORE**|**AFTER**}

  {**INSERT**|**UPDATE**|**DELETE** [OF <seznam sloupců>]}

  ON <název tabulky> [**REFERENCING** <možnosti odkazování>]

  [**FOR EACH {ROW|STATEMENT}**]

  [**WHEN** (<podmínka>)]

  <prováděné příkazy SQL>

Trigger

- Příklad pro vkládání:

```
CREATE TRIGGER Log
AFTER INSERT ON hodnoceni
FOR EACH ROW
BEGIN ATOMIC
        INSERT INTO logy (tabulka, čas, uživatel, akce)
        VALUES ('hodnoceni',CURRENT_TIMESTAMP, CURRENT_USER,'insert');
END;
```

Kontrola souběžnosti

- Kontrola souběžnosti má význam jen v případě, že s databází může pracovat více uživatelů v jediný okamžik.
  - Jednou z možností, jak předcházet dále popsaným problémům se souběžností, je tedy zajistit, aby v daný okamžik pracoval s databází jen jediný uživatel. U jednouživatelských databází je to splněno automaticky u víceuživatelských by bylo potřeba řadit požadavky jednotlivých uživatelů do fronty a vykonávat je postupně. To by ovšem vedlo k nízké propustnosti systému a obvykle se tento přístup nepoužívá.
- kontrola souběžnosti pak má za cíl zajistit, aby práce jednoho uživatele nevhodně neovlivnila práci jiného uživatele i v případě, že pracují souběžně.
  - v některých případech to znamená, že výsledek musí být stejný, jako by uživatel pracoval sám
  - jindy ho mohou ostatní ovlivnit, ale předvídatelným způsobem.

Kontrola souběžnosti

- Každý způsob zajištění kontroly souběžnosti má pro určitou situaci výhody i nevýhody.
- Samozřejmě platí, že čím přísnější pravidla kontroly souběžnosti, tím menší výkon. Je tedy jasné, že nastavení pravidel je vždy nějakým kompromisem mezi výkonem (efektivitou, propustností) a úrovní kontroly souběžnosti (tím úrovní zajištění konzistence změn v databázi a integrity dat)
- Požadavky uživatelů na akce (změny) v databázi jsou obvykle složeny z více dílčích akcí (změn), které ovšem dávají smysl jen jako celek. Je proto potřeba, aby byly provedeny buď všechny, nebo žádná. Proto jsou ve většině databázových systémů zpracovávány požadavky uživatelů ve formě **transakcí** (ucelených posloupností akcí)
- !! Použití transakcí samo o sobě problémy souběžnosti neřeší.!!

Kontrola souběžnosti - transakce

- Sada příkazů, které se provádí jen jako celek. Tedy pokud dojde v rámci transakce k chybě, která způsobí nedokončení některé části posloupnosti, je celá transakce zrušena a všechny dosud provedené změny jsou vráceny.
- Transakce musí splňovat 4 podmínky, označované ACID
  - A – **atomická** (nedělitelná, „všechno, nebo nic“)
  - C – **konzistentní** – dílčí akce musí vést ke konzistentnímu stavu databáze (transakce nezpůsobí chyby v integritě nebo ve struktuře dat)
  - I – **izolovaná** – není ovlivněna jinými transakcemi a data, která by mohla být v průběhu transakce dočasně v nekonzistentním stavu, by neměla být dostupná jiným transakcím, dokud nebudou opět konzistentní
  - T – **trvalá** – výsledky zůstanou uchovány

Kontrola souběžnosti - transakce

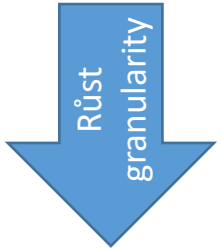
- Jazyk SQL má pro realizaci (řízení průběhu) transakcí sedm příkazů, které patří do skupiny DCL:
  - SET TRANSACTION
  - **START (BEGIN) TRANSACTION**
  - SET CONSTRAINTS
  - SAVEPOINT
  - RELEASE SAVEPOINT
  - **ROLLBACK**
  - **COMMIT**

Kontrola souběžnosti - transakce

- Zpracování souběžných transakcí, tedy těch, které jsou z pohledu uživatelů zpracovávány zároveň, probíhá tak, že jednotlivé dílčí akce transakcí jsou prokládány.
  - čas procesoru je postupně předáván jednotlivým transakcím
  - přepínání je tak rychlé, že to za normálního stavu není navenek vidět a uživatelům se zdá, že probíhá vše současně
- při souběžném zpracování transakcí by bez další opatření docházelo k problémům
  - problém ztracené aktualizace
  - problém nečistého (falešného) čtení (dirty reading)
  - problém neopakovatelného čtení (nonrepeatable reading)
  - problém fantomového čtení (phantom reading)

Uzamykání prostředků

- obvyklá cesta k řešení nekonzistencí vzniklých souběžným zpracováním transakcí
- zabraňuje sdílení dat, která jsou uzamčena za účelem aktualizace
- **granularita** zámku – velikost uzamčené části databáze
  - celá databáze
  - celá tabulka
  - konkrétní řádek
  - konkrétní pole



- s rostoucí granularitou roste náročnost pro správu, ale klesá počet konfliktů
- Zámky mohou být nastavené databází (**implicitní**) nebo pomocí příkazů (**explicitní**)
- Mohou také být **výhradní** (zakázáno je všechno), nebo **sdílené** (čtení je povolené)

Uzamykání prostředků

- Takové schéma souběžného zpracování transakcí, při kterém dostaneme stejné výsledky, jako bychom dostali při sériovém zpracování transakcí se označuje jako **serializovatelné**.
- Jeden ze způsobů jak toho dosáhnout je využití **dvoufázového zamykání**.
  - Transakce může získávat zámky podle potřeby (pokud tomu nebrání zámek od jiných transakcí), ale po uvolnění prvního zámku již další získat nemohou.
- **Uváznutí** (deadlock) – situace, kdy jedna transakce čeká na možnost uzamčení, která je blokovaná jinou transakcí, která zase čeká na uzamčení, které je blokované první transakcí
  - prevence (např. všechny zámky je potřeba uplatnit najedou)
  - uváznutí je povoleno, ale existují prostředky na detekci, po které dojde k přerušení jedné z transakcí a tím k uvolnění uváznutí

Uzamykání prostředků

- **Optimistické uzamykání**
  - Předpokládáme, že ke konfliktu nedojde, takže provedeme transakci a na jejím konci požádáme o zámek a změny potvrdíme. Pokud dojde ke konfliktu, je transakce opakována.
  - Výhodou je, že zámek netrvá tak dlouho (zejména v situaci zámků s nízkou granularitou).
  - Nevýhodou je možné násobné opakování v situaci, kdy se s požadovaným prvkem pracuje často.
- **Pesimistické uzamykání**
  - Zámek aplikujeme hned (není potřeba transakci opakovat), ale trvá dlouho

Zabezpečení databáze

- Cílem zabezpečení databáze (v užším smyslu) je zajistit, aby operace s databází dělali jen autorizovaní uživatelé.
- proces lze rozdělit na 2 fáze:
  - autentifikace – ověřování uživatele
  - autorizace – přidělování oprávnění (permission)
    - různé typy práv pro jednotlivé objekty (čtení , zápis, změny, odstranění …)
- k tomuto účelu mají databázové systémy možnost definovat uživatelské účty
  - v některých případech (SQL Server - Windows) je možné propojení autentifikace s operačním systémem, kdy autentifikace probíhá prostředky OS
- práva se jednotlivým uživatelům přiřazují obvykle **prostřednictvím rolí.** Role jsou přidělovány autentifikovaným uživatelům.
- v rámci SQL jazyka slouží k tomuto účelu příkazy z DCL
  - CREATE ROLE, GRANT, REVOKE

Zabezpečení databáze

- Součástí zabezpečení na úrovni databázového systému je, kromě správy uživatelských účtů, rolí a oprávnění, také ochrana databáze před přístupem zvenčí a zabezpečení dat proti chybám vzniklým selháním systému.
  - HW poruchy
  - výpadky napájení
  - chyby v programech nebo procedurách
  - lidské chyby
- Obvyklá opatření
  - umístění DB serveru za firewallem
  - aplikace nejnovějších aktualizací (databázový systém i operační systém)
  - vhodná omezení nepoužívaných funkcí databázového systému
  - šifrování citlivých dat při ukládání nebo přenosu
  - zálohování

Zabezpečení databáze

- Zabezpečení na úrovni aplikace
  - důvodem může být zjednodušení administrace účtů na úrovni databáze
  - zajištění opatření, která nejde zajistit na úrovní přidělování práv uživatelským účtům či rolím
    - např. pokud je potřeba zajistit, že z tabulky (ke které má uživatel právo ke čtení) může aktuální uživatel číst jen některá data. Je to tedy potřeba zajistit modifikací dotazu, resp. příslušným parametrem procedury.
- Je potřeba si uvědomit, že čím dále od dat se zabezpečení provádí (v případě aplikace je to dál než v případě samotné databáze), tím větší je riziko infiltrace.

Zálohování databáze

- důležitá součást zabezpečení databáze, kterou je potřeba dělat pravidelně
- je východiskem pro obnovení databáze po selhání
- obnovení databáze je možné dvěma postupy
  - obnovení založené na opakovaném zpracování
  - obnovení pomocí vrácení zpět (rollback) nebo postupu vpřed (rollforward)

Obnovení založené na opakovaném zpracování

- nejlepší teoretická možnost obnovy
- vycházíme ze známého bodu (záloha databáze) a znovu provedeme všechny změny (transakce) od tohoto bodu
  - kromě pravidelných záloh databáze je tedy potřeba ještě záznam všech transakcí zpracovaných od poslední zálohy
- bohužel tuto možnost není většinou možné použít
  - opakovaná zpracování transakcí trvá přibližně stejně dlouho jako původně, takže by nemuselo být možné z důvodu zátěže serveru dohnat
  - automatizované provádění transakcí by v důsledku souběhu nemuselo vést k naprosto shodnému výsledku (nebudou v něm zahrnuty náhodné doby způsobené aktivitou uživatelů)

Obnovení pomocí vrácení zpět nebo postupu vpřed

- Stejně jako v předchozím případě je nutné vytvářet **pravidelné zálohy**
- na rozdíl od předchozího postupu je nutné vytvářet **protokol o změnách** v databázi (journal file)
- transakce se neprovádějí znovu, ale aplikují se přímo změny zapsané v protokolu
- **postup vpřed** (rollforward) tedy aplikuje všechny změny z protokolu od známého bodu (zálohy)
  - protokol v tomto případě musí obsahovat všechny stavy po zpracování transakce – tzv. **následné snímky** (after-image)
- vrácení zpět (rollback) naopak odstraňuje potenciálně nedokončené změny až ke známému stavu
  - protokol v tomto případě musí obsahovat všechny stavy před zpracováním transakce – tzv. **předchozí snímky** (before-image)
  - pro obnovení transakcí je ovšem potřeba posléze znát také následné snímky

Protokol (Journal file)

- Obsahuje záznamy o změnách provedených v rámci transakcí
- vytváří se chronologicky a obsahuje
  - identifikátor záznamu v protokolu
  - čas akce
  - identifikátor transakce
  - ukazatele na předchozí a další změnu v rámci stejné transakce (kvůli snazší orientaci)
  - typ operace
  - objekt operace
  - **předchozí snímek** (stav před změnou)
  - **následný snímek** (stav po změně)

  } Pokud operace způsobí změny
- **Změny se do protokolu zapisují dříve, než jsou skutečně provedeny**
  - aby nedošlo ke ztrátě, pokud dojde k havárii mezi zápisem a provedením změny

Kontrolní body

- synchronizační bod mezi databází a protokolem
- v určitou chvíli se přeruší zpracování nových požadavků, dokončí se všechny již zahájené (počká se na potvrzení o úspěšném zápisu na disk a do protokolu) a do protokolu se zapíše **kontrolní bod**
- kontrolní body obvykle vytváří databázový systém automaticky i několikrát za hodinu.
- pokud tedy dojde k havárii, která nezpůsobí ztrátu dat na disku, ale jen není jasné, které transakce jsou v pořádku dokončeny, stačí vrátit zpět a posléze obnovit jen změny od posledního kontrolního bodu v protokolu. To obvykle vede ke značné úspoře času.

Protokol

- pokud dochází při zpracování transakcí k rušení změn (v důsledku zámků a souběhu), děje se tak obvykle na základě stejného principu jako je dříve popsané vracení zpět.
  - transakce si pro tento účel může vytvářet vlastní dočasný protokol
  - v tomto případě musí docházet k umazávání záznamů o změnách z hlavního protokolu

Zdroje

- Kroenke, David a Auer, David J. Databáze. 1. vyd. Brno: Computer Press, 2015. 496 s. ISBN 978-80-251-4352-0.
- Sheldon, Robert. SQL: začínáme programovat. 1. vyd. Praha: Grada, 2005. 499 s. Průvodce. ISBN 80-247-0999-6.