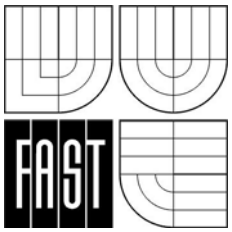
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STAVEBNÍ

MILUŠE KUTÍNOVÁ, JIŘÍ MACUR

INFORMAČNÍ TECHNOLOGIE A SYSTÉMOVÁ ANALÝZA



STUDIJNÍ OPORY
PRO STUDIJNÍ PROGRAMY S KOMBINOVANOU FORMOU STUDIA

OBSAH

Vstupní informace k modulu

Cíle

Cílem tohoto textu je objasnit problematiku tvorby a provozu informačních systémů, které se opírají o ukládání dat a získávání informací z databázových systémů. Práce s tímto typem programů je totiž odlišná od práce s programy, které většina uživatelů běžně používá, což jsou textové editory, grafické nebo různé výpočetní programy. Zde vždy pracujeme s určitým souborem, který upravujeme do konečné podoby.

Od informačního systému očekáváme, že nám pomůže nejen získat potřebnou informaci, ale bude zároveň vynucovat pravidla, která vznik požadované informace zajistí. U informačního systému také vždy očekáváme, že bude používán mnoha navzájem kooperujícími uživateli s různými rolemi. Někteří uživatelé mohou být odpovědni za pořizování a vkládání potřebných dat, jiní za provádění analýz a statistických přehledů, další pak používají poskytovaná data ke své práci nebo celkovému řízení instituce. Informační systém musí všem skupinám uživatelů zajistit správný systém oprávnění k jejich činnostem a zabránit operacím, které nedodržují stanovená pravidla.

Prostudování tohoto textu by nám mělo pomoci orientovat se v oblasti používaných postupů při návrhu informačních systémů a poskytnout základní přehled o možnostech dnešních technologií a architektur, na nichž jsou informační systémy založeny.

Požadované znalosti

Tento kurz je určen i pro ty, kteří ještě s žádným databázovým systémem nepracovali, tudíž se neočekávají, kromě běžné počítačové gramotnosti (uživatelská úroveň dovedností v prostředí operačního systému Windows), žádné předchozí znalosti v této oblasti. Proto je podstatná část textů věnována nezbytné databázové problematice.

Doba potřebná ke studiu

Doba potřebná ke zvládnutí tématu tohoto textu je závislá na zkušenostech, které s databázemi máte. Pokud si na otázku, co je to databáze odpovíte „nějaká data“ a otázkou, jak to vlastně funguje, jste se dosud nezabývali, budete patrně k pochopení textu potřebovat asi 8 až 10 hodin studia.

Klíčová slova

Data, informace, informační systém, informační technologie, třívrstvá architektura, databázová technologie, databáze, databázový systém, SŘBD, databázový model, relace, struktura, atribut, datové typy, tabulka, vztahy, primární klíč,

cizí klíč, databázové jazyky, DDL, DML, DCL, SQL, integrita, redundance, konzistence, transakce, E/R diagram.

Použitá terminologie

Diskutovaná problematika se neustále vyvíjí, takže česká terminologie nemusí být vždy exaktní. V textu je často uváděna i odpovídající anglická terminologie. Pro některá často užívaná spojení se v textu zavádějí zkratky, které jsou při prvním výskytu takového textu uvedeny v závorkách.

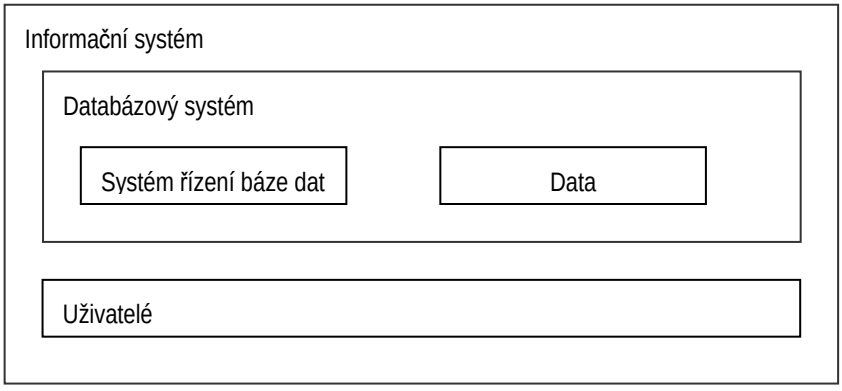
1. Úvod

Kdybychom měli charakterizovat současnost, asi by z mnoha úst zazněla při tomto popisu v nějakém tvaru slova jako komunikace, informace, globalizace: telekomunikace, informační technologie, informační systémy, globální rozšíření, ... Ano, dnešní svět je posedlý informacemi. Informační systémy dnes zajišťují chod téměř celé společnosti: najdeme je ve výrobní sféře, státní správě, finančních ústavech, zdravotnictví, ve školství i armádě. Nasazují se nejen ve velkých organizacích a institucích, začínají se uplatňovat i v menších firmách a provozech. Používá je stále větší počet lidí. Přínosy využití informačních technologií (IT) jsou obrovské: nové výrobní technologie, zvýšení produktivity práce, zvýšení kvality řízení. Vlivy IT však mohou být i negativní. Z pohledu pracovníků mohou ohrožovat jejich pracovní pozice, protože rychlost vývoje informačních technologií vede ke zvýšeným požadavkům na kvalifikaci lidí. Znalosti velmi rychle zastarávají, je nutná rekvalifikace a stálá potřeba sledovat nové trendy, mohou se projevit i vlivy na fyzické a psychické zdraví.

2. Informační systémy a informační technologie

Z obecného hlediska chápeme informační systém (IS) jako integrovaný systém zpracovávající data za účelem poskytování informací. Představuje soubor komponent, jež splňují určitý cíl. Tímto cílem mohou být např. informace o pacientech, pak jednoduchým informačním systémem může být kartotéka lékaře. Nebo informace o pohybu vozidel ve firmě, které se zaznamenávají do knihy jízd. V dnešní době si sice IS představujeme většinou v elektronické podobě, ale nutně to tak být nemusí. Důležité je, že IS *disponuje nástroji pro sběr, ukládání, údržbu a prezentaci dat*. V tomto textu však pokud budeme zmiňovat IS, budeme předpokládat, že pracují na počítačích – obsahují tedy automatizovanou část; v tomto textu se budeme šířeji zabývat problematikou databázových technologií.

V současnosti je každý IS postaven na databázovém systému, který představuje výše zmiňovaný nástroj pro sběr, ukládání a manipulaci s daty a můžeme jej znázornit například tak, jak ukazuje obrázek 2.1.

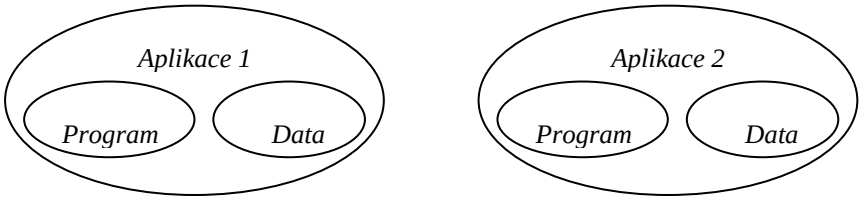


Obr. 2.1 Komponenty informačního systému

2.1. Vývoj technologie pro zpracování dat

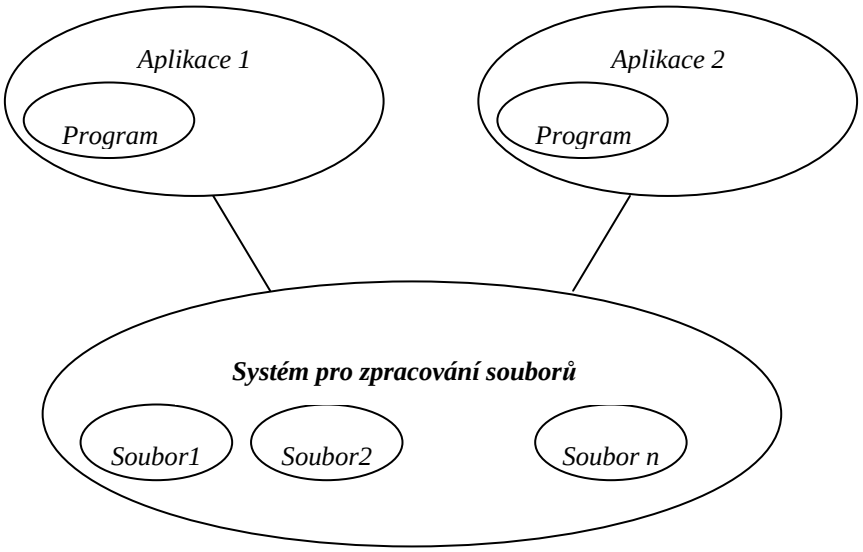
První počítače byly navrženy především pro řešení složitých numerických výpočtů (vývoj atomové bomby, dešifrování kódovaných zpráv), brzy však byly použity i pro zpracování hromadných dat. Umožnil to vývoj programovacích jazyků (FORTRAN, ALGOL, COBOL), které mnohonásobně zefektivnily psaní programů. Byly to především ekonomické agendy, které díky své masovosti způsobily, že se staly převládajícím typem použití počítačů.

První programy se vytvářely jako „šité na míru“ pro jednotlivé agendy, a tudíž vykazovaly nízkou míru datové i uživatelské flexibility. Datové soubory byly navrženy pro potřeby jednotlivých programů, což vylučovalo možnost využití v jiném programu bez „přeprogramování“. Rovněž při změně struktury datového souboru vznikala nutnost úpravy programu. Toto byly začátky zpracování dat spadající do období padesátých let, graficky odpovídají obrázku.2.2.



Obr. 2.2 Organizace dat v počátcích hromadného zpracování dat

Další vývoj vedl k vytvoření souborového systému, kdy se data oddělila od algoritmů do samostatných datových souborů, nicméně popis jejich struktury zůstal v programu. Stav ukazuje obrázek 2.3.

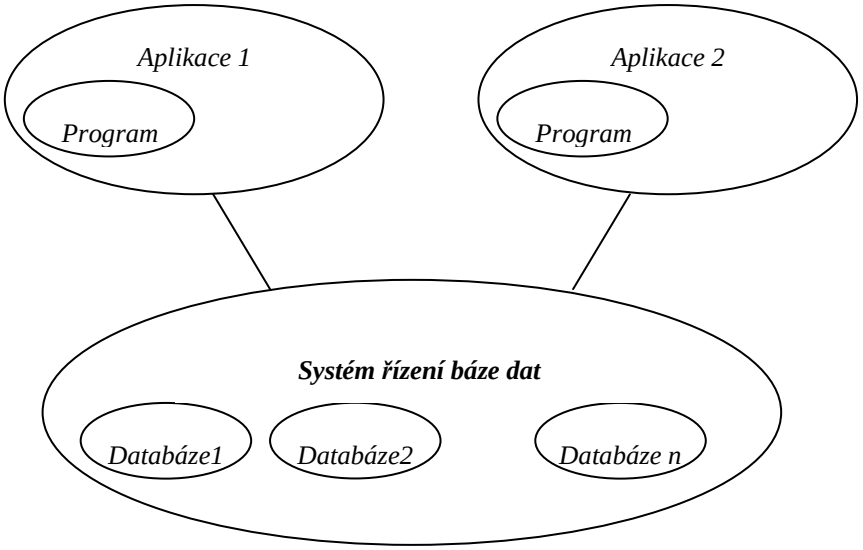


Obr. 2.3 Oddělení datových souborů od algoritmů (60. léta minulého století)

Je jasné, že v takovém systému existence řady datových souborů, které pokrývají potřeby několika programů provozovaných v jedné organizaci, musí tyto soubory obsahovat částečně stejná data. Vždyť ve zpracování osobních dat

zaměstnanců musí být informace o jejich jménech stejně jako v agendě mezd. Tento vícenásobný výskyt dat v systému nazýváme **redundancí**. Je to jev nežádoucí, neboť nejen že zvětšuje objem ukládaných dat, ale především je zdrojem nekonzistence dat. Je zřejmé, že data popisující jednu vlastnost objektu (v tomto případě jméno zaměstnance) by měla mít stejnou hodnotu ve všech svých výskytech, ať je to v jakémkoli datovém souboru příslušejícím jakémukoli programu. Jestliže se data změní, musí být opravena ve všech datových souborech, ve kterých jsou použita, jinak ztrácejí **konzistenci**. Souborové zpracování, které disponuje pouze prostředky operačního systému, má jen omezené možnosti při zajištění bezpečnosti a ochrany dat. Rovněž možnost sdílení dat více uživateli je omezená.

Tyto problémy se staly příčinou dalšího vývoje způsobu zpracování hromadných dat, jež byly završeny vytvořením prvních systémů řízení báze dat – tento způsob zpracování hromadných dat můžeme nazvat termínem databázová technologie. Znázorňuje ho obrázek 2.4.



Obr. 2.4 Uložení dat v databázi spravované SŘBD (současnost)

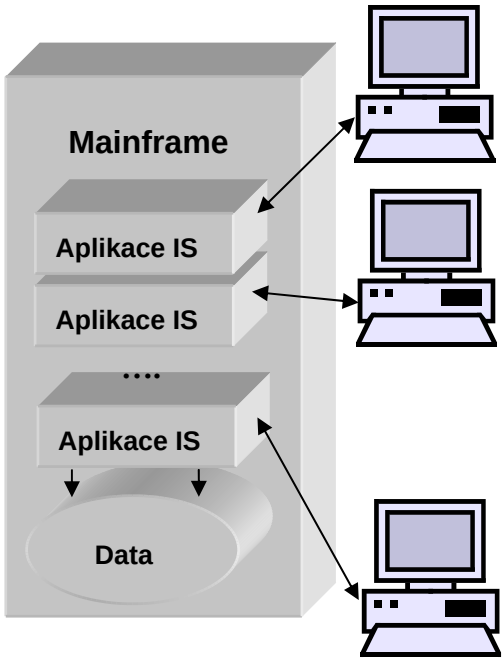
Data využívaná jednotlivými aplikacemi jsou uložena centralizovaně a spravována systémem řízení báze dat (SŘBD), který osvobodil aplikační programy od nutnosti definice a organizace dat. Zároveň poskytuje nástroje pro dosažení daleko lepší ochrany dat než operační systém a jelikož jsou data udržována jednotně, omezuje redundance a tím lze dosáhnout vyšší úrovně konzistence dat.

2.2. Vývoj technologií pro tvorbu informačních systémů

Již jsme uvedli, že informační systém vyžaduje sdílení pořizování a využívání informací mezi mnoha uživateli různého typu. Evoluci, která rozvíjí komfort tohoto sdílení, lze zjednodušeně popsat následujícím způsobem:

2.2.1. Nejstarší – centralizované systémy

Pro tyto systémy je typická silná „centralizovanost“. Konkrétní aplikace (vlastní logika informačního systému), programové aplikační rozhraní databáze a systému řízení báze dat využívají stejné zdroje (procesor, paměť a disk), tj. „běží na jednom počítači“ (tzv. mainframu), který je zdrojem veškerých výpočetních možností. Interakce a komunikace s koncovým uživatelem pak probíhá skrze terminály připojené k mainframu. Tato architektura neposkytuje fakticky žádný způsob použití možností grafického uživatelského prostředí a výpočetní síly eventuálního koncového počítače v roli terminálu. Terminálové protokoly limitují terminály na zobrazování výhradně textových informací. Výhodou této architektury je, že odpadají starosti s provozem sítě a údržbou více počítačů, aplikace a SŘBD úzce spolupracují, což přispívá k celkovému výkonu. Těsná vazba mezi aplikací a SŘBD se však projevují jako značná nevýhoda v případě potřeby přenosu aplikace na jiný SŘBD (tzv. portace).



Obr. 2.5 Schéma informačního systému v centralizované architektuře

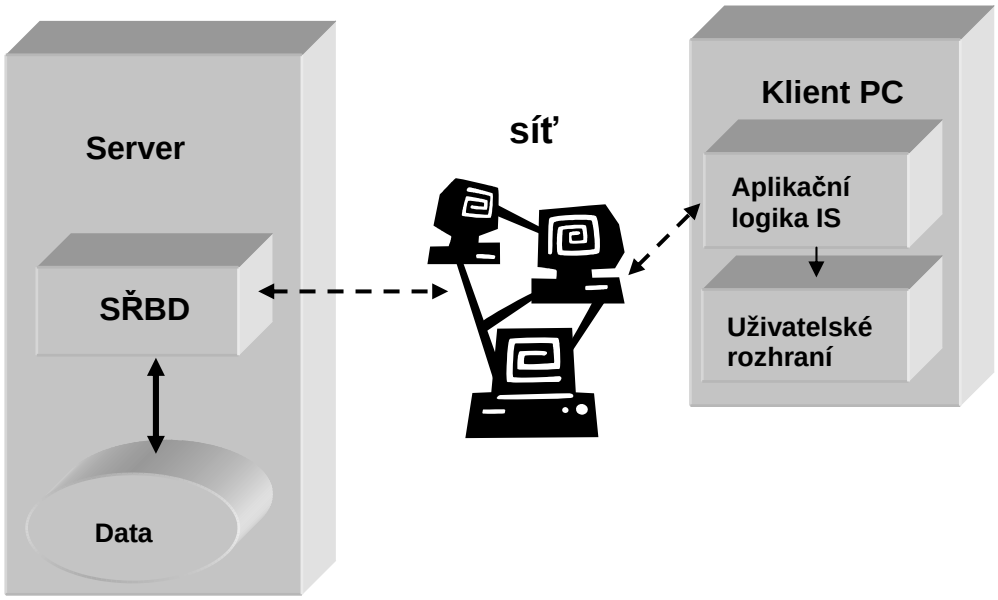
2.2.2. Úspěšné systémy Client/Server

Podobně jako u centralizovaných systémů, každý systém client/server používá SŘBD běžící na jednom vyhrazeném (dedikovaném) počítači – databázovém serveru. Podstatným rozdílem oproti centralizovaným systémům je však skutečnost, že aplikace (vlastní informační systém) běží na odděleném počítači uživatele a pro připojení k SŘBD využívá počítačovou síť. Aplikace je tedy oddělena od SŘBD, používá separátní paměť, vlastní procesor a má přístup na lokální disk počítače uživatele. Komunikace mezi aplikací a SŘBD (tedy dvěma různými počítači) probíhá prostřednictvím tzv. datového protokolu (data protocol), což je jednoduše zakódovaná forma SQL požadavku (viz dále). Počítač, na kterém běží aplikace, poskytuje grafické uživatelské rozhraní, a používá tzv. aplikační rozhraní (interface) databáze (API). Obvykle se označuje jako

klientský počítač (client), i když striktně vzato je v roli klienta pouze program na klientském počítači. Počítač, na kterém běží SŘBD, se nazývá server (opět, vlastním serverem je program realizující SŘBD).

Uvedený model umožnil využití nezanedbatelného výpočetního výkonu koncových uživatelských počítačů (nazývaných také frontend) a uvolnil servery (backend) tak, aby se mohly soustředit zejména na efektivní provádění datových operací. Zajišťování uživatelského rozhraní pro všechny připojené terminály totiž představuje u centralizované architektury značnou zátěž. Navíc je počet připojitelných terminálů u centrálního mainframu značně limitován.

Uvedené rozdělení (distribuce) úkolů tedy umožnilo rychlý nárůst výkonu a zvýšilo celkovou flexibilitu řešení. Výměna SŘBD již neznamená výměnu aplikací na všech klientských počítačích, postačí zachování vlastností aplikačního rozhraní u nového systému. Připojení klientů prostřednictvím sítě také dramaticky zvýšilo jejich možný počet.



Obr. 2.6 Schéma informačního systému v architektuře klient–server

2.2.3. Moderní vícevrstvé architektury

I architektura klient–server však má své limity. Ukázalo se, že mají-li být informační systémy efektivní, musí se neustále přizpůsobovat změnám v organizaci, technologickému vývoji a v neposlední řadě i požadavkům na komfort ze strany uživatelů. Výměna systému však znamená zásah do všech klientských počítačů, což například v prostředí internetu, kdy jejich celkový počet ani není znám, představuje velmi obtížnou úlohu.

Filozofie používání počítačové sítě v architektuře klient–server omezuje také počet současně připojených klientů. Po překročení relativně nízkého počtu (cca 100) dochází ke zhoršení výkonu tím, že server udržuje s klientem spojení pomocí speciálních zpráv (keep-alive messages), a to i v případě, že klient není aktivní.

Uvedené nedostatky částečně odstraňuje zobecnění směrem k tzv. vícevrstvým architekturám (Multi tier architecture).

Dvouvrstvá architektura (Two tier architecture) představuje klasický databázový model klient–server, tak jak byl popsán v předcházejícím odstavci. Jednu vrstvu představuje klient, který požaduje nějaké služby, druhou vrstvu pak server, který tyto služby poskytuje. Uživatelské rozhraní je umístěno výlučně na straně klienta, správa databáze je umístěna na straně serveru. Server může fungovat zároveň jako klient jiného serveru v hierarchické klient–server architektuře. Pak hovoříme o tzv. zřetězené dvouvrstvé architektuře (chained two tier architecture).

Třívrstvá architektura (Three tier architecture) byla vyvinuta v devadesátých letech jako nástupce architektury dvouvrstvé. Mezi vrstvu klienta a serveru byla přidána třetí (prostřední – middle tier) vrstva. Tato vrstva může být implementována různými způsoby a technologiemi například formou monitoru transakčního zpracování (transaction processing monitor), serveru zpráv (message server) nebo aplikačního serveru (application server). Může provádět akce jako řazení požadavků do front, přidávat možnosti plánování a prioritního zpracování. Zavedení střední vrstvy ulehčuje administraci a správu změn, protože případné změny se zapisují pouze jednou na server střední vrstvy, a tak se stávají automaticky dostupné ve všech systémech. U jiných modelů by změna funkce musela být zapsána do každé aplikace. Podařilo se překonat a odstranit výkonnostní omezení týkající se dvouvrstvé architektury. Je zajištěn výkon i pro velké skupiny současně pracujících klientů (řádově stovek až tisíců počítačů).

Třívrstvá architektura je základem v současné době tak populárních internetových aplikací. Vrstvu klienta, která má na starosti grafické uživatelské rozhraní, logicky přebírá internetový prohlížeč (web browser). Pro vytváření uživatelského prostředí máme k dispozici nástroje v podobě jazyka HTML, CSS, pomocí skriptovacích jazyků na straně klienta a objektových modelů prohlížeče můžeme klientu svěřit i část logiky informačního systému.

Střední vrstvu představuje nejčastěji tzv. aplikační server, poskytující více služeb, které lze rovněž formálně od sebe oddělit (proto se často hovoří o n-vrstvé architektuře). Základem střední vrstvy bývá webový server protokolu http, který navíc disponuje rozšířením pro:

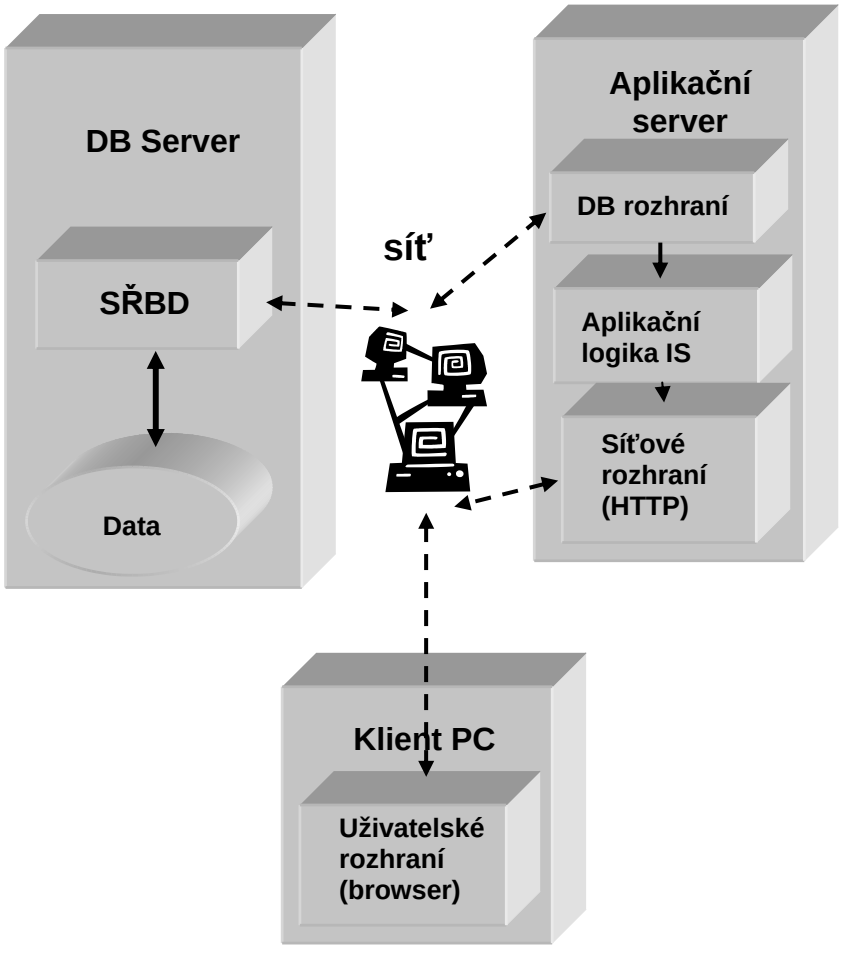
- udržování kontextu pro jednotlivé připojené uživatele
- poskytování zabezpečeného přístupu a ověřování oprávnění
- implementaci vlastní aplikační logiky informačního systému pomocí vestavěného prostředí pro vytváření a používání programových objektů
- realizaci transakčního zpracování
- prostřednictvím vestavěných objektů obsahuje aplikační rozhraní pro spolupráci se systémy SŘBD a jinými poskytovateli dat
- plus mnoho dalších možností

Zjednodušeným pohledem můžeme internetovou aplikaci vidět v podobě procesu, kdy

1. klient (prohlížeč internetu) požádá server o dokument s HTML formulářem

2. po vyplnění formuláře uživatelem jsou data odeslána zpět na webový server
3. ten předá data získaná od uživatele programu na aplikačnímu serveru, který na jejich základě zformuluje potřebný dotaz SQL
4. dotaz je předán databázovému serveru, který dotaz (příkaz) provede a vrátí nejčastěji databázovou relaci (tabulku)
5. aplikační server data získaná z databáze opatří značkami jazyka HTML a odpověď prostřednictvím webového serveru předá klientskému prohlížeči
6. klient – prohlížeč pak uživateli interpretuje data v patřičné grafické podobě

Uvedený proces se v různých obměnách opakuje, dokud uživatel práci s informačním systémem neukončí. Aplikační server musí přitom správně vyhodnotit fázi rozpracovanosti, v níž se uživatel systému nachází – jinými slovy, odezvy informačního systému závisí na historii jeho komunikace s uživatelem. Vzhledem k tomu, že používaný komunikační protokol HTTP je bezstavový a nepodporuje udržování takového historického kontextu, plní tuto úlohu aplikační server.



Obr. 2.7 Schéma informačního systému ve vícevrstvé architektuře

Také se zde řeší problémy oprávnění – aplikační server musí v každé etapě komunikačního procesu znát identitu uživatele a rozhodnout, zda je jeho požadavek legitimní. Rozhodování o oprávnění lze sice přesunout i na databázový server (ve dvouvrstvé architektuře jinou možnost nemáme), aplikační vrstva však obsahuje daleko bohatší nástroje, které jsou komfortnější i z hlediska uživatele (dostává srozumitelné odpovědi, co nemůže udělat, a proč).

Programy implementující informační systém z hlediska jeho pravidel a logiky jsou uloženy na aplikačním serveru a volají se podle potřeb uživatelů. Programy se často vytvářejí v interpretačním prostředí (jazyky VBscript, JavaScript, Perl apod.) kvůli rychlejšímu vývoji a snazšímu ladění. V tomto prostředí jsou pak k dispozici vestavěné objekty, resp. knihovny, které zpřístupňují požadavky klientů, rozhraní databázového systému, souborový systéme serveru aj. V současnosti jsou velmi používané platformy PHP (provozované zejména v operačních systémech typu unix), ASP nebo pokročilejší .NET (pro systémy orientované na MS Windows).

Výše nastíněná komplexnost aplikační vrstvy vedla k jejímu dalšímu rozdělení na vrstvy. Toto rozčlenění pomohlo definovat odpovědnosti jednotlivých vrstev v rámci aplikační vrstvy. Vrstvy jsou nejčastěji formálně definovány jako prezentační (presentation), aplikační (business) a datová (persistence). K ustálení terminologie v této oblasti však prozatím nedošlo.

Pozorný čtenář si mohl všimnout, že třívrstvá technologie představuje určitou formu návratu k vyšší centralizaci ve srovnání s architekturou klient–server. Jedná se však spíše o novou kvalitu, neboť na rozdíl od ryze centralistického pojetí nebývá aplikační server umístěn na stejném stroji jako databázový server. Skutečností však zůstává, že v případě architektury klient–server obsahuje klient také aplikační logiku informačního systému (tzv. tlustý klient), zatímco ve třívrstvé architektuře je úlohou klienta takřka výhradně poskytovat pouze komfortní uživatelské rozhraní (tenký klient). Z tohoto hlediska je také internetový prohlížeč považován za tenkého klienta, což je celkem paradoxní vzhledem k tomu, že je reprezentován velmi rozsáhlým programem.

2.2.4. Webové služby

Webové služby představují další zobecnění vícevrstvé architektury. Zatímco v případě třívrstvé architektury je výstupní odpověď aplikačního serveru určena zejména k zobrazení uživateli, v případě webové služby (web service) je výstup upraven do univerzální obecné podoby tak, aby ho mohla využít jiná aplikace k dalšímu zpracování.

Ilustrujme si uvedený mechanizmus na příkladu proslulé vyhledávací služby Google. Ta poskytovala své služby původně pouze prostřednictvím webové stránky – uživatel si načetl stránku s vyhledávacím formulářem, do něj zadal svůj dotaz a stránku odeslal. Zpět dostal odpověď od Googlu, opět v podobě webové stránky v jazyce HTML s lineárním seznamem odkazů na jiné stránky. Pokud měl uživatel jinou představu o výsledku (např. znázornit jej graficky, nebo jinak dále zpracovat, případně zkombinovat s jinými výsledky), musel si to zařídit sám.

V případě webové služby může uživatel vytvořit jinou aplikaci, která si sama vyžádá příslušná data od služby Google, zpracuje je podle svých potřeb, a s výsledkem naloží podle představ uživatele – zobrazí ho například v grafické podobě místo v podobě seznamu apod. Příklad se službou Google není zcela náhodný – právě Google totiž zavedení webových služeb silně podporuje.

Z dalších významných subjektů, poskytujících své služby na Internetu, se k podobnému kroku odhodlal také známý Amazon (původně orientován na elektronické knihkupectví). I zde je umožněno, aby si jiné aplikace samy prohledávaly jeho katalogy, vyhledávaly zde to, co potřebují a výsledky dále zpracovávaly podle svých představ a potřeb.

Základním znakem webové služby je zejména klient, který je nyní (na rozdíl od internetového prohlížeče) původní samostatnou aplikací s vlastní inteligencí a vlastní logikou zpracování dat. Server služby může být značně specializován, proti aplikačnímu serveru je obvykle daleko jednodušší. Proto se také v souvislosti s webovými službami někdy hovoří o modelu klient–služba, místo o modelu klient–server.

U webových služeb se dále významným způsobem mění charakter vztahu mezi klientem (aplikací) a službou (serverem poskytujícím službu). Zatímco dříve byly tyto dvě složky poměrně pevně svázány a jejich vazba měla statický charakter, v případě webových služeb je jejich vazba značně volnější a má spíše příležitostný charakter – klient si teprve na základě určité momentální potřeby vyhledá potřebnou službu pro spolupráci. K tomu jsou ovšem nutné nové standardy a protokoly.

Každá služba musí být standardizovaně popsána, co nabízí a jak se s ní dá komunikovat, a tyto informace musí být známým způsobem veřejně dostupné. Prostřednictvím takových veřejných katalogů pak budou moci konkrétní aplikace (klienti) vyhledávat konkrétní služby a také je využívat. Některé konkrétní standardy jsou již k dispozici, další se připravují či dokončují nebo vylepšují. Jde např. o jazyk XML (pro strukturovanou formulaci požadavků a odpovědí na ně), SOAP (pro vzájemnou komunikaci), WSDL (pro popis webových služeb), UDDI (pro realizaci katalogů webových služeb) a další.

Velké rozšíření webových služeb se teprve očekává, komplexní informační systémy dosud používají zejména dnes již klasickou třívrstvou technologii.

3. Databázové systémy

V předchozí kapitole jsme se pokusili ozřejmit vývoj způsobu zpracování hromadných dat. Na základě obrázků 2.1 a 2.4 můžeme vyvodit platnost následujícího zápisu:

> Systém řízení báze dat (SŘBD) + Databáze (DB) = Databázový systém (DBS)

Uvedené termíny představují základní pojmy, proto si je vysvětleme:

- *Systém řízení báze dat (SŘBD),*

  v anglické terminologii často užívaná zkratka DBMS (Data Base Management System), je program nebo soubor programů, který data organizuje: umožňuje je definovat, ukládat, měnit, mazat. SŘBD dodávají výrobci programového vybavení podobně jako operační systémy, textové editory nebo programy pro kreslení. Nejznámějšími produkty jsou např. dBase, MS Access, Oracle, Informix, Progress, MS SQL Server, Sybase a další.

- *Databáze (databázová aplikace)*

  Databáze je strukturovaná množina dat ve formě databázového souboru, tedy konkrétní data. Důležité je zde zejména slovo strukturovaná, protože databáze může být prázdná, tedy nenaplněná daty, a přesto můžeme říci, že existuje, pokud byl definován popis tvaru a struktury jejích dat. K těmto datům potom přistupují jednotliví uživatelé různé úrovně prostřednictvím aplikací, které jim umožňují zadávat data a dotazy pomocí uživatelsky příjemného rozhraní jako jsou formuláře nebo dialogová okna pro zadávání dat nebo tiskové sestavy pro zobrazení a tisk výsledků dotazů.

- *Databázový systém (DBS)*

  DBS je pojem, který spojuje vlastní údaje ukládané v databázi s nástroji (programy) umožňujícími jejich správu.

3.1. Charakteristika databázového systému

Z analýzy vývoje DB technologie lze formulovat základní charakteristiky a kritéria, která musí databázový systém splňovat. Jsou to zejména následující požadavky:

- *Struktury aplikačních programů a vlastních datových souborů musí být oddělené.*

  Tím je dosaženo vzájemné nezávislosti programů na datech a naopak. Pak je vytvořen předpoklad, aby bylo možné (s většími či menšími úpravami) spravovat data různými SŘBD.

- *K datům je možno přistupovat pouze prostřednictvím DB systému, nikoli přímo.*

  DB systém sám řídí přístup uživatelů k datům, tím je zajištěna ochrana dat. Na základě oprávnění přidělovaných uživatelům musí zabránit zneužití dat, ať už úmyslnému či neúmyslnému. Rovněž musí zajistit, že vkládaná data budou pravdivá, tedy odpovídající skutečnosti (v tom smyslu, že jsou pravděpodobná).

- *Dotazy lze tvořit i v průběhu práce se systémem, nemusí být formulovány předem.*

  Při vývoji systému budou jistě pro běžné použití sestaveny často kladené dotazy. Dotazy podle specifických požadavků je však možno formulovat i kdykoli později při rutinním provozu systému.

- *Je umožněn současný přístup k datům více uživatelům.*

  Moderní DB systém musí být víceuživatelský.

3.2. Úrovně pohledu na data

Předností databázového systému je to, že poskytuje uživateli abstraktní pohled na data bez znalosti implementačních podrobností, to znamená bez znalosti toho, jak se skutečně realizuje fyzické ukládání, vyhledávání a aktualizace dat uložených v souborech. Při pohledu na data můžeme mluvit o třech vrstvách, které jsou do určité míry nezávislé, což umožňuje provádět změny v těchto vrstvách, aniž by tím byly ovlivněny vrstvy ostatní.

Fyzická úroveň

nejnižší vrstva, na které je popsáno, jak jsou konkrétně data uložena. Na této úrovni pracuje s daty programátor – tvůrce SŘBD.

Konceptuální úroveň

popisuje logickou strukturu dat, která jsou v databázi obsažena. Na této úrovni abstrakce pracuje správce databáze, který popisuje strukturu dat a jejich vzájemné vazby, případně aplikační programátor, který vytváří aplikační programy s použitím prostředků jazyka DML. Popisu databáze na této úrovni říkáme **schéma databáze**.

Uživatelská úroveň

popisuje pro daného uživatele jen část databáze, s kterou je oprávněn pracovat. Popisu struktury databáze pro jednotlivé uživatele říkáme **subschéma** nebo též pohled a může jich být tolik, kolik je různých uživatelů. Uživatel na této úrovni může pracovat se systémem pouze prostřednictvím aplikačního programu (naivní uživatel) nebo může být schopen formulovat dotazy v databázovém jazyce (znalý uživatel).

3.3. Úkoly SŘBD

SŘBD musí mít prostředky pro popis dat, k jejich definici a vytvoření vlastní databáze. Rovněž je třeba nějakého mechanismu, který spravuje data na disku a ví, kde je uložen konkrétní prvek. Tyto **prostředky pro definici dat** bývají označovány jako jazyk typu **DDL** (Data Definition Language) a slouží jako nástroj pro vytvoření a definici databáze.

Dále musí být k dispozici prostředky, které dovolí přistupovat k již existujícím databázím, vybírat z nich data, třídit je, filtrovat a měnit. **Tyto prostředky pro manipulaci** s daty bývají označovány jako jazyk typu **DML** (Data Manipulation Language). Specifická část manipulace s daty – výběr dat, je realizována pomocí dotazovacího jazyka (viz. kapitoly 6.5 a 7).

SŘBD by měl poskytovat také možnost zobrazovat data na obrazovce nebo na tiskárně ve formě tiskové sestavy. Moderní SŘBD, tyto služby poskytují. O systému řízení báze dat, který poskytuje pouze základní služby jako je definice, údržba a manipulace s daty, mluvíme jako o **databázovém stroji**. Zobrazování a tisk dat pak je třeba zajistit aplikačním programem.

Když chceme s databázovým systémem pracovat, předpokládáme, že je možné se spoléhat na to, že data jsou správná, to znamená, že data odpovídají vlastnostem příslušného popisovaného objektu reálného světa. Této vlastnosti se říká **integrita**. Např. fakt, že únor má 28 dní, je třeba popsat takovými integritními podmínkami, které postihnou to, že datum 30. února je vždy porušením

integrity, zatímco datum 29. února jen někdy (není-li přestupný rok). Systém musí zajistit, že integrita nebude porušena ani v případě havárie (např. výpadek proudu) ani zásahem jiného uživatele nebo aplikace, ať již úmyslným nebo neúmyslným. Zajišťuje se jak na úrovni SŘBD, tak i zadáním určitých kritérií na úrovni definice dat, popřípadě na úrovni aplikační.

U rozsáhlých databázových aplikací se často stává, že některé informace jsou v systému obsaženy vícenásobně. Tato vlastnost se nazývá **redundance**. Správným návrhem databáze je třeba redundanci minimalizovat, neboť spotřebovává prostor na pevných pamětech a je zdrojem nekonzistencí. Nicméně pokud se vícenásobná data vyskytnou, SŘBD musí zajistit jejich **konzistenci**, to znamená, že data za všech okolností musí mít ve všech svých výskytech stejnou hodnotu. Problémy s konzistencí nastávají např. v době aktualizace, kdy oprava již započala, ale dosud nebyla dokončena. V této době jsou data jako celek nekonzistentní. Proto se takovéto operace provádějí po malých logicky uzavřených částech, tzv. **transakcích**, které je možno zopakovat, pokud nebyly provedeny správně. Transakce je tedy množina příkazů, které převádějí databázi z jednoho konzistentního stavu do jiného konzistentního stavu. **Prostředky pro popis ochrany přístupu k datům** se někdy označují jako jazyk typu **DCL** (Data Control Language). **Řízení transakcí**, jež poskytuje většina dnešních SŘBD, je zabezpečováno řídícími příkazy označovanými jako **TCC** (Transaction Control Commands).

Jestliže jsme výše označili prostředky pro zabezpečení úkolů SŘBD jako jazyky DDL, DML a DCL, neznamená to, že se jedná o nějak oddělené, samostatné jazyky, ale měli bychom je chápat u procedurálních jazyků jako množinu procedur zabezpečujících určité služby (definici dat, manipulaci s nimi, řízení a ochranu dat). V současnosti jsou tyto prostředky většinou integrovány do jednoho celku, takže moderní SŘBD poskytuje dostatečně silný databázový jazyk, který splňuje potřeby jak laických nebo příležitostných uživatelů, tak i správců dat, případně aplikačních programátorů.

Shrneme-li tedy úkoly SŘBD, musí poskytovat následující služby:

- Definice dat
- Údržba dat
- Manipulace s daty
- Zobrazení dat
- Zajištění integrity dat

3.4. Databázové aplikace

Databázová aplikace zajišťuje komunikaci uživatele s databázovým systémem. Umožňuje mu s daty pracovat, to znamená zadávat je, měnit, rušit, různě seskupovat a vytvářet z nich výpisy. Tyto aplikace mohou být vytvořeny v univerzálním nebo specializovaném programovacím jazyce. Dnešní SŘBD, zvláště systémy na PC, poskytují uživatelsky orientované nástroje pro přístup k databázi a snižují nutnost tradičního programování.

Jazyky používané pro tvorbu aplikací můžeme rozdělit do následujících skupin:

- Obecné procedurální jazyky jako je Pascal, Cobol, Basic, C. Procedurální znamená, že aplikace je zapsána ve formě procedur, to znamená algoritmů, které určují, jakým způsobem požadovaná data získat, změnit, atd. Tyto jazyky běžně používané pro tvorbu nedatatabázových aplikací, musí být rozšířeny o prostředky pro přístup k datům. Těmito prostředky jsou množiny funkcí, které bývají shromážděny do tzv. knihoven, které se dodávají spolu se SŘBD. Těmto univerzálním jazykům se říká jazyky 3. generace (**3GL**)
- Pro některé SŘBD byly vyvinuty specifické procedurální jazyky, které na rozdíl od univerzálních jazyků mohou být použity pouze v tomto systému, případně v podobných produktech, pokud se stanou dostatečným standardem, jako např. jazyk dBASE. Aby se odlišily od univerzálních jazyků, bývají označovány jako jazyky 4. generace (**4GL**).
- **Dotazovací jazyky** jsou prostředky umožňující formulovat požadavky uživatelů na výběr potřebných informací z databáze. Většinou se jedná o neprocedurální jazyky, to znamená, že příkazem se popisuje, co se má provést, jaká data potřebujeme, ale neurčuje se, jak se to má provést. Již v 70. letech byl firmou IBM vyvinut dotazovací jazyk SQL (Structured Query Language), určený pro komunikaci se SŘBD relačního typu. Vzhledem k tomu, že neposkytuje prostředky pro manipulaci s obrazovkou a pro uživatelský vstup a výstup, je spíš jakýmsi podjazykem. Jeho prostřednictvím lze interaktivně formulovat dotazy, nebo může být použit jako součást hostitelského jazyka pro přístup k datům, zatímco zbývající část aplikace je napsaná v jiném programovacím jazyce. SQL je tvořen v porovnání s jinými jazyky pouze několika příkazy, zato však velmi výkonnými. S jeho pomocí lze definovat data (definovat strukturu tabulky), aktualizovat data (přidávat a mazat řádky a sloupce, vkládat data z vnějšího souboru), ale hlavní síla je v příkazu SELECT pro formulaci dotazů. SQL dnes představuje standardní metodou přístupu k databázi. Obliba a rozšíření jazyka SQL vedlo k jeho standardizaci, nicméně implementace různých producentů SŘBD se od tohoto standardu vždy poněkud liší.
- Skupinu jazyků, která nezapadá do žádné z předchozích, bychom mohli nazvat ostatní jazyky. Patří sem např. jazyky maker, což ve skutečnosti není programovací jazyk, protože pouze automatizuje provádění různých akcí uživatele jako posloupnost stisků kláves. Nebo sem můžeme zahrnout jazyk QBE (Query By Example), přeloženo znamená dotaz prostřednictvím příkladu. Zase to není programovací jazyk, ale spíš uživatelské rozhraní, které dovoluje definovat dotazy intuitivním způsobem bez znalosti jazyka SQL. Mezi jiné jazyky než procedurální bychom mohli zařadit také objektově orientované jazyky, jako je C++, VisualBasic, Turbo Pascal, protože přistupují k programování ne přes definici procedur, ale popisem akcí, které jsou realizovány na různých objektech.

4. Základní pojmy

4.1. Data versus informace

Než se pustíme do databázové teorie a terminologie, zamysleme se nad rozdílem pojmů data a informace. Obecná definice počítače často zní: počítače jsou stroje na zpracování informací ve formě dat. Počítače tedy umožňují ukládat data a následně z těchto uložených dat získávat informace. **Data,** tedy představují statické hodnoty pocházející z jakékoli oblasti lidské činnosti, které je třeba uložit. Například údaje popisující rozměry, materiál a zatížení nějaké konstrukce nebo osobní údaje charakterizující určitou osobu. Z těchto uložených dat se různými způsoby manipulace s nimi získávají výstupní údaje, jež mají charakter **informací**. Jedním ze způsobů této manipulace je např. u numerických úloh zpracování vstupních dat programem podle nějakého algoritmu a zobrazení informací ve formě číselných nebo graficky zpracovaných výsledků. Jiným typem zpracování, typickým pro databázové systémy, jsou různé výběry z uložených dat, jejich seskupování nebo filtrování podle zadaných kritérií.

4.2. Co je to databáze?

Jak uvidíme dále, pojem databáze se někdy užívá v několika různých souvislostech. Lze však říci, že obecně si pod tímto pojmem představíme soubor uložených dat, jež popisují nějakou oblast lidské činnosti, na základě kterých jsme schopni o této oblasti čerpat informace. Může se např. jednat o databázi odběratelů zboží z nějaké velkoprodejny a údaje o nich mohou sloužit k rozesílání nabídek na různé reklamní akce nebo k vyhodnocování jejich nákupů. Přitom na formě databáze nezáleží. Většinou sice předpokládáme, že jde o data uložená v počítači, ale ještě i dnes existuje mnoho databází v papírové formě uložených v různých pořadačích. Např. mnoho obvodních lékařů si stále údaje o pacientech zapisuje do karet a jediným nástrojem pro efektivní vyhledávání informací je jejich řazení podle abecedy. To samozřejmě velmi urychlí vyhledání určitého pacienta, ale při požadavku zjistit pacienty se stejnou chorobou, se bude muset lékař spolehnout na svoji paměť nebo projít karty všech pacientů. Na databázi se tedy můžeme dívat jako na velkou skříň s informacemi a jde o to, aby práce s ní byla efektivní a umožňovala rychlé vyhledávání, filtrování a seskupování údajů podle požadavků uživatele.

4.3. Zdroje dat

Jestliže už víme, jak si představit databázi, měli bychom se zamyslet nad tím, co může být zdrojem dat, která shromažďuje. Stále častěji se setkáváme s informačními systémy, které zpřístupňují veřejné nebo privátní informace z nejrůznějších institucí a firem. Každý informační systém je vybudován na nějaké databázi. A jestliže tyto instituce a firmy se mohou zabývat jakýmikoli okruhy činnosti, pak právě zde leží zdroje dat, které je třeba shromažďovat pro potřeby získávání informací. Zdrojem dat jsou tedy informace z libovolných předmětových oblastí, což mohou být nejrůznější druhy výrobních, správních, vzdělávacích, vojenských nebo jiných organizací a institucí. Příklady uvádí následující tabulka:

Oblast	Zdroj dat
Výrobní podnik	Výrobky, suroviny, dodavatelé, odběratelé, zaměstnanci…
Letecká doprava	Lety, posádky, cestující, zaměstnanci, rezervace míst…
Realitní kancelář	Klienti, realitní objekty, poptávka, nabídka, zaměstnanci
Zdravotnické zařízení	Pacienti, diagnózy, lůžka, zdravotnický materiál, lékaři…
Banka	Účty, klienti, půjčky, úvěry, jiné služby, zaměstnanci…

Tab. 4.1 Příklady předmětových oblastí a datových objektů, které je popisují

4.4. Objekty dat

Z tabulky nahoře vidíme, že zdrojem dat jsou objekty (entity), které danou předmětovou oblast popisují. Tímto objektem může být člověk, předmět, událost, místo nebo pojem. Každý objekt je možno popsat datovými údaji, které jej charakterizují pro určitou předmětovou oblast. Některé objekty jsou specifické pro určitou předmětovou oblast (pacient pro zdravotnickém zařízení), jiné můžeme najít v mnoha oblastech (klienti, zaměstnanci). Přesto tyto stejné objekty pravděpodobně nebudou v různých předmětových oblastech popsány naprosto stejnými údaji. Např. u zaměstnance – pilota (letecká doprava) budeme sledovat kromě osobních údajů třeba počet nalétaných hodin, zatímco u zaměstnance – lékaře (zdravotnické zařízení) zase počet atestací a podobně.

4.5. Atributy dat

Datové údaje, které charakterizují objekt pro danou předmětovou oblast, nazýváme atributy (vlastnosti těchto objektů). Například zaměstnanec je specifikován svým jménem, příjmením, bydlištěm a dalšími údaji potřebnými např. pro zpracování mezd. Projekt zase svým názvem, identifikačním číslem, cenou, termínem.

4.6. Hodnota dat

Jestliže atributy představují vlastnosti objektů, které chceme v dané předmětové oblasti sledovat, pak tyto atributy pro každý prvek tohoto objektu nabývají konkrétních hodnot. Například atribut příjmení může nabývat konkrétních hodnot Novák, Vopršálek nebo Novotná, atribut plat 10 000, 15 000. Obecně mohou být hodnoty dat kvalitativní (popisné), jako je třeba právě příjmení zaměstnance či název projektu nebo kvantitativní (počitatelné), to znamená, že vyjadřují množství, počet nějakých jednotek. Tyto dva druhy hodnot představují dva základní datové typy ukládaných dat: textový řetězec a číslo. Většina databází rozšiřuje tyto základní datové typy ještě o typ datum a logickou hodnotu ano/ne (oba jsou ovšem podmnožinou typu číslo). Moderní databáze v souvislosti s velikostí ukládaných dat rozlišují množství podtypů těchto základních typů.

4.7. Struktura databáze

Strukturou databáze nazýváme množinu datových prvků, které požadovaným způsobem popisují sledovaný objekt. Např. objekt ZAMESTNANEC bude popsán řadou atributů různého typu jako např. jméno, příjmení, RČ, plat, atd., z nichž některé budou nabývat kvalitativní jiné kvantitativní hodnoty různého rozsahu. Definování struktury databáze, představuje vytipování objektů a určení atributů a jejich oborů hodnot potřebných pro popsání určité předmětové oblasti. Je velmi důležitou částí návrhu databáze a může mít podstatný vliv na její správné fungování.

4.8. Záznam dat

Datovým záznamem rozumíme množinu hodnot souvisejících prvků jednoho datového objektu. Tedy všechny údaje o konkrétním zaměstnanci, projektu nebo půjčce. Ve formě těchto datových záznamů jsou potom data většinou uchovávána na paměťových médiích.

4.9. Klíčové prvky dat

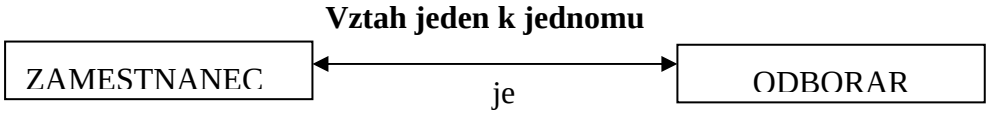
Některé prvky dat (atributy) mají zajímavé vlastnosti. Na základě znalosti jejich hodnot lze totiž zjistit i hodnoty ostatních souvisejících prvků dat. Tyto prvky, podle nichž lze určit i ostatní datové prvky téhož záznamu, nazýváme **klíčovými atributy**. Např. známe-li ISBN nějaké knihy, můžeme jednoznačně určit o jaký titul jakého autora se jedná, kdy a kým byla kniha vydána. Pokud bychom nepotřebovali sledovat informace o vydání knihy, mohla by k její identifikaci postačit znalost jejího názvu a autora, což by sice nebylo ideální, ale pro potřeby jednoduchého systému by to mohlo být postačující. Atributů, které jednoznačně identifikují záznam, může být tedy v objektu definováno více. Mluvíme o tzv. **kandidátních klíčích**. Je na projektantovi konkrétní databáze, aby rozhodl, který z nich bude použit pro přístup k datům záznamu.

4.10. Vazby mezi objekty

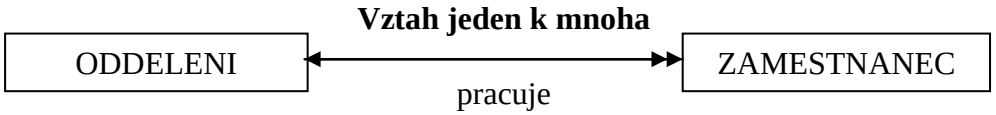
V tabulce 4.3.1 jsme viděli, že informace o určité předmětové oblasti jsou většinou popsány pomocí několika objektů. Každý objekt představuje ucelené informace o jeho vlastnostech. Tyto objekty však spolu vzájemně souvisí, vždyť při popisu reality říkáme: „Učitel vyučuje předměty, studenti navštěvují přednášky, přednáška probíhá v určité místnosti.“ Mezi daty tak mohou vznikat různé druhy vazeb. Rozlišujeme tři základní vztahy:

- Vztah jeden k jednomu
- Vztah jeden k mnoha
- Vztah mnohý k mnoha

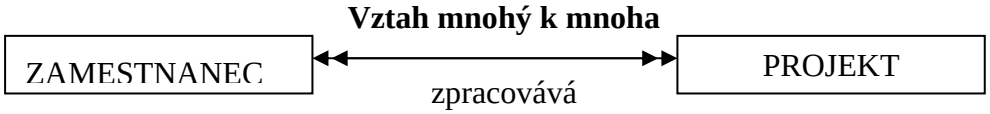
Např. v předmětové oblasti (FIRMA) mohou existovat datové objekty popisující tuto oblast, jako ZAMESTNANEC, ODDELENI, PROJEKT, ODBERATELE, a podobně. Vztahy, které mezi nimi existují, lze popsat slovně jako: zaměstnanec je zařazen do oddělení, zaměstnanec pracuje na nějakém projektu, projekt se vytváří pro určitého odběratele. Podrobíme-li tyto vztahy rozboru, můžeme určit jejich typ, jako u následujících objektů:



Vztah 1 : 1 je nejjednodušší, ale nejméně obvyklý vztah. Každému prvku jednoho objektu odpovídá pouze jeden prvek druhého objektu. Většinou se jedná o jakýsi „podobjekt“, jako v tomto případě, kdy o odborářích chceme sledovat další informace (jako např. funkce v odborech, datum vstupu apod.). Každý zaměstnanec však nemusí být odborářem. Je tedy zbytečné rozšiřovat objekt ZAMESTNANEC o atributy odboráře, protože u mnoha zaměstnanců nebudou mít význam a zůstanou nenaplněné.



Nejobvyklejší vztah, ve kterém s jedním prvkem jednoho objektu lze spojit více (jako zvláštní případ jeden nebo žádný) prvků druhého objektu. Zaměstnanec je standardně zařazen do jednoho oddělení a toto oddělení má většinou více zaměstnanců. Tento vztah však připouští, aby existovalo i oddělení s jedním zaměstnancem nebo dokonce bez zaměstnanců.



Tento typ vztahu je v reálném světě velmi běžný. Jak vidíme na příkladu, jeden zaměstnanec může pracovat na více projektech a projekt je současně zpracováván několika zaměstnanci. Nebo čtenář si může půjčit více knih a kniha může být půjčena více čtenářům. Jak uvidíme dále, v relačních databázích jej nelze realizovat přímo, ale je třeba jej rozložit na jednodušší vztah 1 : n pomocí tzv. vazebné tabulky.

Příklad 4.1

Pokuste se vytipovat objekty, které by bylo třeba popsat pro vytvoření databázového systému knihovny.

Základní objekty KNIHA a ČTENÁŘ jsou patrně jasné už z velmi lakonického popisu činností odehrávajících se v knihovně: čtenář si půjčuje knihy. Z této věty můžeme odvodit i nutnost dalšího objektu – VYPUJČKA, protože

je třeba vědět, kdo si jakou knihu půjčil, kdy to bylo a zda ji vrátil. Dál už bude záležet o jakou knihovnu se bude jednat. Pro naši domácí knihovnu to může být postačující, pro veřejnou knihovnu bude třeba se podrobněji seznámit s činnostmi, které se tu odehrávají, aby bylo možno splnit úkoly, které má systém plnit. Dalšími objekty, které tak bude třeba sledovat mohou být např. SKLAD pro popis umístění knih, VYDAVATELSTVÍ pro sledování kontaktů na vydavatele, ŽÁNR pro zatřídění knih, BESEDA pro organizaci besed s autory a další.

5. Datový model

Datový model popisuje strukturu databáze na konceptuální, tedy logické úrovni. Jedná se o formální popis uživatelské aplikace, tedy toho, jak databáze funguje. K zajištění úkolů, které má DB systém plnit a na základě analýzy daného prostoru problému, se provádí strukturalizace dat a popis vzájemných vztahů mezi nimi. Důležité je, že konceptuální schéma, které na této úrovni vzniká, je nezávislé na konkrétních implementačních nástrojích, tedy programech (SŘBD), které definované funkce a procesy realizují vlastními (poněkud odlišnými) prostředky.

Podle toho jak jsou data organizována a zpřístupňována uživateli a programátorovi, bylo rozpracováno několik DB modelů této organizace. Nutno podotknout, že databáze mají bohatou a dlouhou historii úzce spojenou s praxí. Vždy šlo o to, jak zpracovat data co nejefektivněji pro řešení konkrétního problému, a proto vznikaly aplikace založené na určitých modelech organizace dat, aniž by tyto byly vždy předem teoreticky definovány a popsány. Z historického hlediska byl pouze relační model teoreticky propracován dříve, než byl vyvinut SŘBD tohoto typu, ostatní modely pouze dodatečně teoreticky popisují DB systémy, které již byly prakticky několik let používány.

Dá se říct, že existující DB modely vycházejí ze tří základních přístupů organizace dat:

- *Soubory dat jsou vzájemně spojeny pomocí ukazatelů.*
  Na tomto principu jsou založeny síťové a hierarchické databáze, které představují základy databázové technologie. První SŘBD založené na těchto modelech se objevily v druhé polovině 60. let a jsou použity na velmi rozsáhlých projektech jako např. vesmírný program Apollo. Pracují efektivně a rychle při vyhledávání, lze pomocí nich dobře modelovat vztahy, i když u hierarchických databází je složitý vztah M : N problematický. Dodatečná změna jednou navržené struktury je však u nich velmi obtížná, protože vede k nutnosti předefinovat vztahy celého systému.

- *Soubory dat jsou vzájemně logicky nezávislé.*
  Tento přístup se uplatňuje u relačního modelu, který chápe data jako více méně nezávislé tabulky, se kterými se manipuluje pomocí operací relační algebry. Zakladatelem tohoto modelu je E. F. Codd, který na počátku 70. let publikoval články, v nichž popsal model založený na matematickém pojmu relačních množin. V průběhu 70. let vznikly první relační SŘBD, které prošly v dalších letech obdobím zdokonalování a v současnosti platí, že většina dnešních komerčních produktů jsou systémy založené na relačním modelu.

- *Soubory dat představují objekty.*
  Principy objektově orientovaného programování (OOP) pronikly i do oblasti databází. Objektový přístup nabízí možnost lépe popsat objekty reálného světa než relační model, který popisuje pouze vlastnosti objektů, tedy statická data. Na rozdíl od tabulek v relačním modelu mohou mít objekty kromě vlastností i metody, které představují funkce a procedury, které s nimi manipulují. Na základě společných rysů, lze vytvářet třídy objektů, které své vlastnosti a metody mohou dědit. Zdá se, že objektově orientované principy jsou pro modelování reality ideální, nicméně teoreticky objektově orientovaný model dosud není dostatečně popsán a standardizován, což je důvodem pomalejšího pronikání OOSŘBD do praxe. Spíše se projevují trendy rozšířit relační model o prvky a principy OOP, takže můžeme mluvit o objektově-relační technologii.

Každý datový model má pro určitou oblast úloh jisté přednosti, které opodstatňují jeho použití. Nicméně díky své efektivitě, flexibilitě a propracovanosti je relační model v současnosti nejpoužívanější, a proto se v tomto textu věnujeme právě a pouze relačním databázím.

6. Relační DB systémy

Relační DB model, jak už bylo řečeno výše, vychází z teorie množin. Související data tvoří množinu údajů, relaci. Relace je uspořádána v tabulce a tvoří její řádky. Od těchto relací je odvozen název DB modelu, nikoli od termínu relace, který se často v relačních databázových systémech používá pro vyjádření vztahu mezi tabulkami.

Jelikož relaci tvoří související údaje, je jasné, že k popsání určitého prostoru problému nebude stačit jediná relace, tedy tabulka. Jak jsme viděli v tabulce 4.1, je každá předmětová oblast popsána řadou objektů a minimálně údaje o těchto objektech budou uloženy v samostatných tabulkách. Většinou je v systému tabulek daleko víc, některé nemusejí reprezentovat žádný reálný objekt. Pro získání pohledu na data z různých úhlů, je většinou třeba pracovat s daty uloženými ve více tabulkách. Pro práci s takto chápanými daty jsou k dispozici prostředky relační algebry a kalkulu.

Shrneme-li předcházející odstavce, můžeme pro relační databázový model stanovit základní charakteristické vlastnosti:

- všechna data mají pravidelnou strukturu a ukládají se v tabulkách
- pro práci s daty existuje databázový jazyk, který umožňuje provádět operace relační algebry

6.1. Uložení dat

Právě jsme se dověděli, že v relačním DB modelu se data ukládají do tabulek. Rozeberme si jednotlivé její prvky na následujícím příkladu.

Příklad 6.1

Ukažme si příklad tabulky pro uložení údajů o zaměstnancích. Může vypadat např. takto:

ID *číslo*	***Příjmení*** *text, 20 z.*	***Jméno*** *text,10 z*	***Bydliště*** *text, 30 z*	***Plat*** *číslo*	***Narozen*** *datum*	***Odborář*** *log. hodn.*	***Oddělení*** *text, 30 z*	***Děti*** *číslo.*
1	*Prouza*	*Jan*	*Brno, Úzká 3*	*15000*	*2.4.1974*	*ano*	*Projekční*	*3*
2	*Novák*	*Adam*	*Brno, Cejl 24*	*18000*	*9.5.1960*	*ne*	*Osobní*	*1*
3	*Bílá*	*Dana*	*Kyjov, Trh 7*	*16000*	*3.7.1969*	*ne*	*Prodejní*	*2*
4	*Tesař*	*Ivo*	*Podomí 259*	*12000*	*6.6.1982*	*ano*	*Osobní*	*0*

Tab. 6.1 Data v tabulce ZAMĚSTNANEC

Tabulka (entita)

Data popisující vlastnosti jednoho druhu reálného objektu se shromažďují v tabulce. Při práci s datovým modelem mluvíme o tabulkách jako o entitách. Tabulky se většinou pojmenovávají jednotným číslem objektu, jehož data uchovávají. V rámci jednoho DB systému musí být názvy tabulek jedinečné.

Záhlaví

Jistě pozorujeme, že první řádek tabulky se liší od ostatních. To není u tabulek nic neobvyklého, první řádek tabulky obvykle obsahuje záhlaví, které vyjadřuje obecný popis údajů nacházejících se v jednom sloupci. U relační tabulky mluvíme o záhlaví jako o **struktuře tabulky,** která definuje **atributy** a obor hodnot, jichž mohou nabývat. Každá tabulka je vytvořena již definicí své struktury. Nemusí obsahovat žádné údaje, ale pokud jsou pojmenovány sloupce a stanoveno, jaký typ údajů bude do nich vkládán, tabulka již existuje.

Řádek (záznam, record)

Řádek tabulky obsahuje všechny údaje definované ve struktuře pro jeden prvek objektu, který tabulka představuje; v našem případě všechny sledované údaje o jednom zaměstnanci. Přitom na pořadí řádků nezáleží. Můžete podotknout, že byste chtěli vidět seznam zaměstnanců seřazený abecedně podle příjmení. Na to vám namítnu, že mě by zajímal jejich seznam podle stáří. Pro splnění těchto požadavků však existují jiné nástroje, než je uspořádání řádků v tabulce. Mnohem důležitější je vědět, jak si zpřístupním určitý řádek, který např. obsahuje údaje o osobě, která mě zajímá. V kapitole 4.9 jsme se dověděli, že toto umožňují klíčové atributy a že jich může existovat víc. V relační tabulce se atribut, který byl zvolen jako jednoznačný identifikátor řádku tabulky, nazývá **primární klíč**. V tabulce může být definován pouze jeden primární klíč, přičemž to neznamená, že nemůže být tvořen více atributy, pak mluvíme o složeném primárním klíči. Pokud nenajdeme mezi atributy vhodný přirozený primární klíč, můžeme pro identifikaci řádku použít systémem vygenerované unikátní číslo (umělý primární klíč). Později, až si budeme vysvětlovat, jak se realizují vztahy mezi tabulkami, uslyšíme ještě o **cizích klíčích**. Je zřejmé, že určení osoby „lidským“ způsobem podle příjmení, kdy musíme často upřesnit, že to není Franta Novák, ale Honza Novák, ale ne ten z Komína, ale ten ze Slatiny a ne ten starý, ale mladý, je pro databázový systém naprosto nevhodné. Proto je v našem příkladě tabulky zaměstnanců použit atribut ID, který nemá pro popis zaměstnance jiný účel, než jeho identifikaci.

Sloupec (atribut)

Jak je patrné z příkladu, je sloupec množina hodnot stejného typu. A jak vás zřejmě napadne, na pořadí sloupců, jak jsou definovány ve struktuře nezáleží. Je patrně jedno, jestli plat uvedu před nebo za datem narození, neboť pro různé účely si budu moci pomocí databázových nástrojů vybrat jen ty a v takovém pořadí, jak potřebuji. Naopak je důležité si uvědomit, že názvy atributů musí být v rámci jedné tabulky unikátní.

Doména

Jistě jste si všimli, že v záhlaví tabulky je pod názvem atributu uveden i jeho typ a rozsah. Na úrovni datového modelu představuje doména množinu hodnot stejného významového typu, v níž se budou hodnoty atributu pohybovat. Na této úrovni se tedy nejedná o specifický datový typ (např. integer, varchar(20), apod.), který definuje a implementuje konkrétní databázový stroj.

6.2. Normalizace

Podívejme se na tabulku z příkladu 6.1 a položme si otázky: „Musí obsahovat právě uvedené atributy?“ a „Proč neobsahuje např. data narození dětí zaměstnanců, aby mohly nezletilé děti dostat dárky k Vánocům nebo Dni dětí?“ Chceme-li, aby náš databázový systém odpovídal na určité otázky, je nutné tyto požadavky zohlednit při jeho návrhu, neboť žádný systém nedokáže poskytnout informace, pro něž nemá uložena potřebná data. Takže pokud chceme, aby náš systém vypsal seznam dětí, které dostanou dárky k Vánocům, musíme rozšířit tabulku zaměstnanců o potřebná data. Po úpravě by pak mohla vypadat např. takto:

Příklad 6.2

Alternativní návrh struktury tabulky pro uložení údajů o zaměstnancích, může vypadat např. takto:

ID *číslo*	***Příjmení*** *text, 20 zn.*	***Jméno*** *text, 10 zn.*	***Bydliště*** *text, 30 znaků*	***Plat*** *číslo*	***Narozen*** *datum*	***Oddělení*** *text, 30 znaků*	***RCděti*** *text, 60 znaků*
1	*Prouza*	*Jan*	*Brno, Úzká 3*	*15000*	*2.4.1974*	*Projekční*	*9802245512*
2	*Novák*	*Adam*	*Brno, Cejl 24*	*18000*	*9.5.1960*	*Osobní*	*8661112358,90111111*
3	*Bílá*	*Dana*	*Kyjov, Trh 7*	*16000*	*3.7.1969*	*Prodejní*	*0202021454,04511111*
4	*Tesař*	*Ivo*	*Podomí 259*	*12000*	*6.6.1982*	*Osobní*	

Tab. 6.2 Data v tabulce ZAMĚSTNANEC

Jaké informace bude poskytovat DB systém, záleží tedy na jeho úplnosti. Jak snadno a dobře bude na otázky odpovídat, však záleží především na jeho struktuře. Vezměme si například řešení požadavku vypsat všechny zaměstnance z jedné obce. Potřebná data jsou uložena ve sloupci [Bydliště]. Tento úkol lze v našem příkladě řešit, pokud předpokládáme jistou disciplínu při vkládání údajů do tabulky, a sice že při zadávání bydliště bude vždy zadáno nejdřív město, pak ulice a číslo. Při porušení tohoto pravidla patrně dostaneme chybné údaje. Víme, že relační model není v informatice žádná novinka, existují tedy jakési postupy a principy, které vedou k „správně“ navrženému datovému mo-

delu. Přičemž slovo správně je v uvozovkách proto, poněvadž nemůžeme například v případě našeho příkladu s adresou bydliště kategoricky říci: „Adresa musí být vždy zaznamenána jako pole jednotlivých jejích prvků tj.obec, ulice, číslo, PSČ.“ Ne, opravdu mohou existovat případy, kdy je výhodnější uložit adresu jako jeden řetězec.

Z popisu relačního modelu, jak jsme o něm doposud mluvili, plyne, že pracuje s dvourozměrnými tabulkami. Podívejme se opět na tabulku z příkladu 6.2 Chtěli jsme ukládat více údajů o dětech než jen jejich počet, proto jsme přidali sloupec [RCděti], abychom měli informaci i o jejich věku. Pokud by každý zaměstnanec měl jen jedno dítě, bylo by to v pořádku, ale to obecně neplatí. Narážíme zde na problém, jak do jednoho pole uložit více údajů, potřebovali bychom třetí rozměr a ten v relačním modelu nemáme. Zkušenosti ukázaly, že navržené řešení psát do jednoho pole více stejných údajů oddělených čárkou vede vždy k problémům. Stejně jako alternativa, která vás možná napadla, a sice pro každé dítě zavést vlastní sloupec [RCdítě1], [RCdítě2],... Je stejně špatná jako předchozí řešení, protože okamžitě se vynoří otázka kolik sloupců bude třeba? Nebo jak dlouhý má být řetězec pro uložení rodných čísel dětí? Je jasné, že navrhneme-li bezpečný počet dětí, např. 20, nenastane případ, kdy by sloupce chyběly, u většiny zaměstnanců však většina sloupců zůstane prázdná.

Jak tedy řešit tento „třírozměrný“ problém? Připomenu, že v příkladu 6.1, kde jsme popisovali jednotlivé prvky tabulky, bylo řečeno, že tabulka uchovává data o jednom druhu objektu. A nepředstavuje vlastně dítě zaměstnance jiný datový objekt? Jestliže vytvoříme tabulku DÍTĚ, pak jeden prvek této entity, tedy řádek tabulky, budou tvořit údaje o dítěti a nikoli o zaměstnanci a řádky na rozdíl od sloupců lze přidávat bez problémů. Navíc údajů o dětech mohu sledovat víc, např. jejich jméno a příjmení (to nemusí být nutně stejné jako příjmení rodiče).

Podívejme se na celou situaci na dalším příkladu:

Příklad 6.3

Další alternativa návrhu struktury tabulek pro uložení údajů o zaměstnancích:

ID *číslo*	***Příjmení*** *text, 20 znaků*	***Jméno*** *text, 10 znaků*	***Bydliště*** *text, 30 znaků*	***Plat*** *číslo*	***Narozen*** *datum*	***ZkratkaOd*** *text, 2 znaky*	***Oddělení*** *text, 30 znaků*
1	*Prouza*	*Jan*	*Brno, Úzká 3*	*15000*	*2.4.1974*	*PA*	*Projekční*
2	*Novák*	*Adam*	*Brno, Cejl 24*	*18000*	*9.5.1960*	*OO*	*Osobní*
3	*Bílá*	*Dana*	*Kyjov, Trh 7*	*16000*	*3.7.1969*	*PO*	*Prodejní*
4	*Tesař*	*Ivo*	*Podomí 259*	*12000*	*6.6.1982*	*OO*	*Osobní*

Tab. 6.3 Data v tabulce ZAMĚSTNANEC

ID *číslo*	***Příjmení*** *text, 20 znaků*	***Jméno*** *text,10 znaků*	***RCděti*** *text, 10 znaků*
1	*Prouza*	*Jan*	*9802245512*
2	*Nováková*	*Jana*	*8661112358*
3	*Novák*	*Adam*	*9011115555*
4	*Hanák*	*Dan*	*0202021454*
5	*Bílá*	*Ivona*	*0451194774*

Tab. 6.4 Data v tabulce DÍTĚ

Odstranili jsme tedy problém s vícenásobnými atributy, máme dvě tabulky, ale když se na ně díváte, musí vás napadnout otázka: „Jak vím, čí je které dítě?“ Jistě tušíte, že sázet na stejná příjmení by asi nebylo spolehlivé. Než se pustíme do řešení tohoto úkolu, udělejme ještě jednu úpravu ve struktuře tabulky ZAMĚSTNANEC.

Firma pro kterou navrhujeme tuto tabulku, může mít desítky zaměstnanců a pro každého ukládáme název oddělení, ve kterém pracuje. Tyto názvy se tedy u zaměstnanců téhož oddělení mnohokrát opakují a pro správné vyhledávání je třeba zajistit, aby měly stejné hodnoty. Zamyslíme-li se nad jednotlivými atributy tabulky ZAMĚSTNANEC, může nás napadnout, že název oddělení nepopisuje zaměstnance, ale samostatnou entitu ODDĚLENÍ. Přitom o odděleních nemusíme chtít shromažďovat žádné jiné informace než právě jeho název. Přesto bude efektivní, pokud vytvoříme tabulku ODDĚLENÍ, která bude mít jen tolik záznamů kolik je oddělení a každé bude identifikováno primárním klíčem, nejlépe číselným, neboť u čísla se vždy jedinečnosti dosáhne snadněji než u textu.

ID *číslo*	***Název*** *text, 30 znaků*	***Zkratka*** *text, 2 znaky*
1	*Projekční*	*PA*
2	*Osobní*	*OO*
3	*Prodejní*	*PO*
4	*Reklamační a vnějších vztahů*	*RV*

Tab. 6.5 Data v tabulce ODDĚLENÍ

O takových tabulkách, které obsahují pouze očíslované popisy nějakých položek, ať už jsou to oddělení jako v našem případě nebo škála barev, kterou si můžete vybrat ze vzorníku firmy dodávající třeba koberce, mluvíme jako o **číselnících**.

Tato kapitola nese název normalizace, ale zatím o ní nepadlo ani slovo. Na příkladu definice struktury tabulky ZAMĚSTNANEC jsme se pokusili uplatnit jakési principy a normy, jež by měly vést k dobře navrženému datovému modelu. Tento proces dekompozice dat na jednotlivé tabulky se nazývá **normalizace**. Odborníci zabývající se studiem databázového návrhu definovali tato pravidla jako tzv. **normální formy**, které se běžně uplatňují v několika úrovních od nejjednodušší první normální formy (zkráceně 1NF) až po 5NF. Vyšší forma vždy staví na normalizaci provedené na nižší úrovni. V našem příkladu

jsme uplatnili první a druhou normální formu a protože se jedná o základní formy, pokusme se shrnout jejich pravidla.

1NF

Tabulka je v první normální formě, pokud všechny její atributy jsou atomické, dále nedělitelné a obsahují skalární hodnoty.

2NF

Tabulka je v druhé normální formě, pokud splňuje podmínky 1NF a každý její atribut je závislý na primárním klíči. Jde o to, definovat v tabulce pouze atributy popisující jednu entitu.

3NF

Relace je v třetí normální formě, pokud je ve druhé normální formě a navíc všechny její neklíčové atributy jsou vzájemně nezávislé, tedy žádný atribut nezávisí na jiném atributu, kromě primárního klíče.

Příkladem porušení 3NF by v případě naší tabulky ZAMĚSTNANEC bylo např. přidání pole [Odměna], pokud by se její výše vždy a u všech zaměstnanců odvíjela od velikosti platu.

Ve většině případů postačí dodržení těchto základních principů, ale je možné jít v normalizaci i dál, to už je ale nad rámec našeho textu.

6.3. Vztahy mezi entitami

Jestliže jsme se v kapitole 4.10 dověděli, že při popisu reality najdeme mezi objekty tři základní typy vztahů, pak je zřejmé, že v datovém modelu, který realitu popisuje pomocí entit – tabulek, budou mezi nimi existovat stejné vztahy. Nestačí tedy popsat vlastnosti entit, ale je třeba popsat i vlastnosti vztahů. Neboť i vztahy mají vlastnosti. Ty tvoří především **účastníci**, kteří jsou určitým vztahem spojeni, dále jejich počet, který definujeme jako **stupeň** vztahu a **účast** dané entity ve vztahu, která může být úplná nebo částečná, podle toho, jestli entita může existovat i bez účasti ve vztahu.

Zápis entit a vztahů

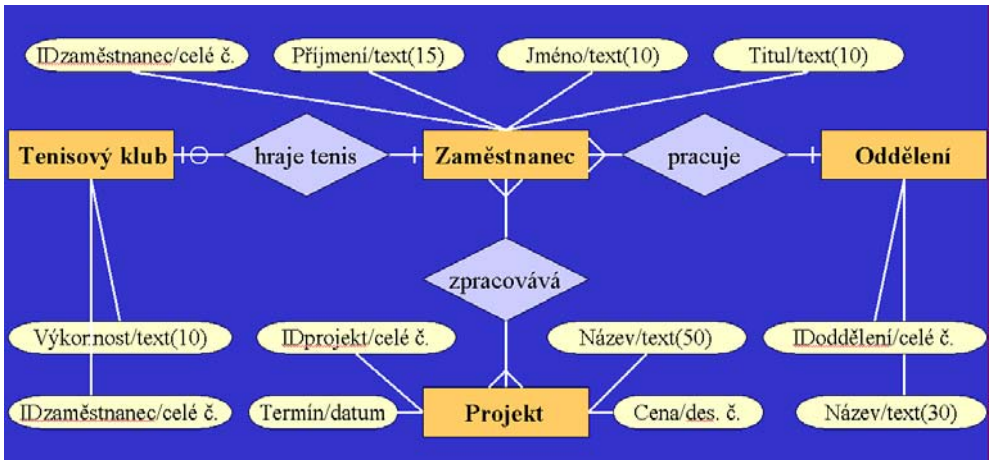
Jestliže bychom chtěli popsat dvě entity a jejich vzájemný vztah, mohli bychom to například udělat takto:

Entita: ZAMĚSTNANEC ([Id]/číslo, [Jméno]/text(15), [Příjmení]/text(20), [Kontakt] /text(20))

Entita: TENISKLUB ([Id]/číslo, [Výkonnost]/text(20), [ČlenOd]/datum)

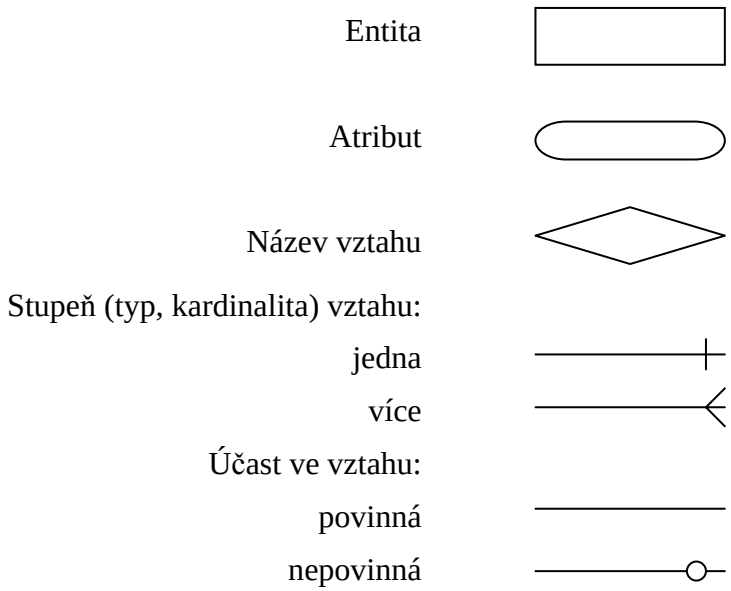
Vztah: *Hraje tenis* (ZAMĚSTNANEC /1/úplná, TENISKLUB /1/částečná*)*

Zápis je strukturovaný a snad i přehledný, ale jistě si dovedete představit, že při větším počtu entit a vzájemných vztahů, už se nám to tak jevit nebude. Proto se pro zápis entit a jejich vztahů používá grafické vyjádření označované jako **E/R diagram** (Entity Relationship). Grafické symboly používané k popisu nejsou vždy jednotné, E/R model může vypadat např. takto:

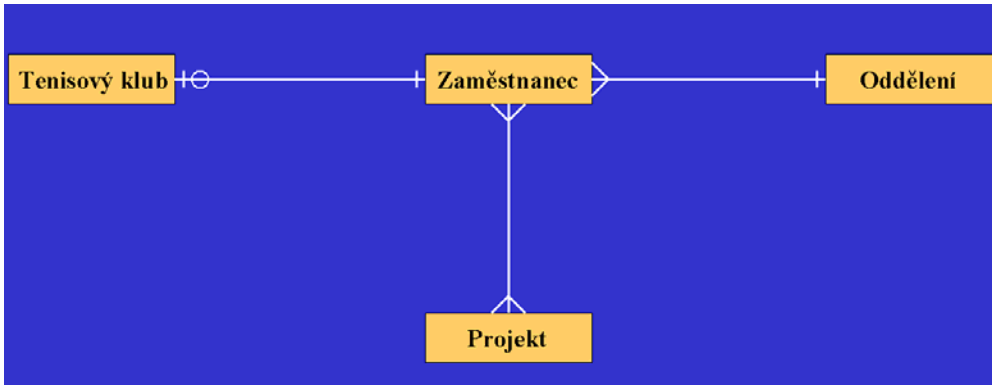


Obr. 6.1 E/R diagram včetně atributů, označovaný někdy jako ERA diagram

Z obrázku 66.3.1 můžeme odvodit, jaké symboly se pro kreslení E/R diagramů používají:



Ve složitějších případech se často vynechávají názvy vztahů a atributy, které mohou být popsány mimo diagram, takže diagram je pak přehlednější.



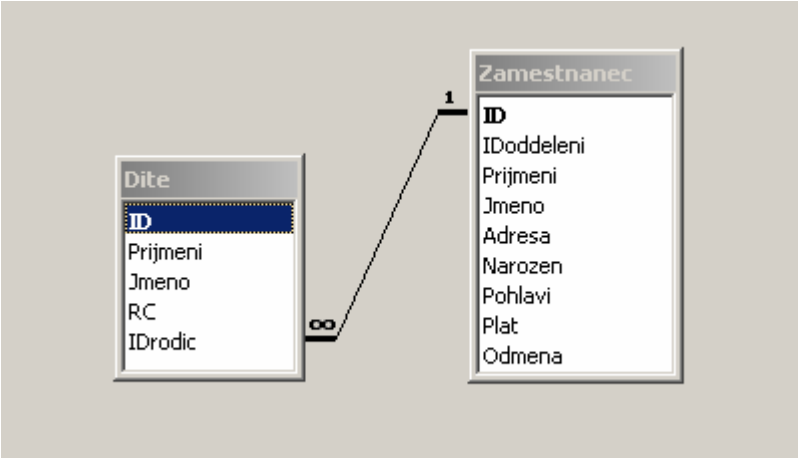
Obr. 6.2 Zjednodušený E/R diagram

Nyní, když jsme analyzovali a pomocí E/R diagramu přehledně znázornili entity a typy vztahů mezi nimi, zabývejme se tím, jakým způsobem je budeme v relační databázi implementovat.

Poznámka. Při implementaci datového modelu se často pro vztah mezi tabulkami používá termín relace. Ty se v každém systému nějakým způsobem definují. V této části výkladu jsou z praktických důvodů použity ke grafickému znázornění tabulek a jejich vzájemných vztahů otisky okna relací definovaných v MS Access®. Jsou z nich patrné všechny důležité informace: názvy tabulek, jejich atributy, primární klíče (označené tučně) a strany vztahu 1 : N (strana N je označena jako ∞)

Realizace vztahů

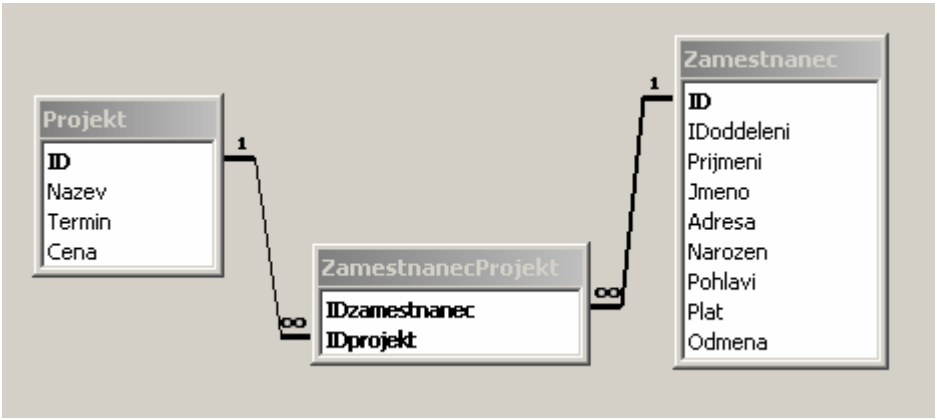
Typy vztahů mezi objekty jsme objasnili v kapitole 4.10. V relační databázi je základním typem vztahu vztah 1 : N. Ten se realizuje vložením primárního klíče z tabulky na straně 1 do tabulky na straně N ve formě tzv. **cizího klíče**. Na rozdíl od primárního klíče, který je pro jednu tabulku vždy jen jeden, cizích klíčů může být více, pokud je tabulka ve vztahu s více tabulkami. Podívejme se na naše tabulky ZAMĚSTNANEC a DÍTĚ v příkladu 6.26.3. Je evidentní, že vztah mezi nimi je typu 1 : N, neboť jeden zaměstnanec může mít více dětí, zatímco dítě patří k jednomu zaměstnanci (stav, kdy jsou oba rodiče zaměstnanci jedné firmy, neřešíme).



Obr. 6.3 Vazba mezi tabulkami ZAMĚSTNANEC a DÍTĚ

Jestliže pole [ID] slouží k identifikaci zaměstnance a je tedy v tabulce ZAMĚSTNANEC primárním klíčem, přidáme do tabulky DÍTĚ odpovídající pole pojmenované např. [IDrodic]. Cizí klíč vlastně vytváří odkaz z tabulky na straně N do tabulky na straně 1 a zpřístupňuje všechny hodnoty právě jednoho záznamu.

U vztahů M : N je to trochu složitější. V relační databázi je nelze realizovat přímo, ale modelují se pomocí tzv. **vazebné tabulky**. Tyto tabulky většinou slouží pouze k tomu, aby vztah M : N zjednodušily na dva vztahy 1 : N. Podívejme se na tuto situaci na následujícím obrázku:



Obr. 6.4 Realizace vztahu mezi tabulkami ZAMĚSTNANEC a PROJEKT

Vidíme, že vazebná tabulka obsahuje pouze primární klíče spojovaných tabulek (v tabulce ZAMĚSTNANECPROJEKT však mají funkci cizích klíčů). Budeme-li zjišťovat informace týkající se vzájemného vztahu zaměstnanec – projekt, jako např. kolik zaměstnanců pracuje na určitém projektu nebo na kterých projektech pracoval konkrétní zaměstnanec, odpovědí nám budou řádky právě této tabulky (spolu s podrobnostmi čerpanými z odpovídajících řádků spojovaných tabulek).

Integritní omezení

V procesu tvorby datového modelu je třeba nejen definovat entity a vztahy mezi nimi, ale i stanovit pravidla, na základě kterých databázový systém postavený na tomto modelu zajistí, že data uložená v systému jsou správná, tedy odpovídající skutečnosti. Toho se dosahuje definováním integritních omezení, to je pravidel, která datovou integritu zajistí. Integritní omezení jsou různého typu, podle toho, na jaké úrovni se definují.

- *Úroveň struktury tabulek (doménová integrita)*

  Doménová omezení jsou pravidla, která definují platné hodnoty atributů. Je to jednak určení oboru hodnot (datový typ) a jednak stanovení přípustnosti existence neznámých či nezadaných hodnot. Tyto hodnoty se označují jako tzv. hodnoty NULL. Ukažme si to na příkladu tabulky ZAMĚSTNANEC. Např. u pole [Narozen] můžeme stanovit, že smí obsahovat neznámé či nezadané hodnoty (NULL). Znamená to, že pak můžeme vložit nového zaměstnance a nemusíme zadat jeho datum narození. Na rozdíl od toho bychom požadovali, aby pole [Příjmení], bylo zadáno vždy.

- *Úroveň vztahů mezi tabulkami (referenční integrita)*

  V předchozím textu bylo vysvětleno, že vztahy mezi tabulkami se realizují prostřednictvím primárních a cizích klíčů. Jestliže dojde k narušení těchto vazeb, stane se systém nespolehlivým. Jak může k porušení vazeb dojít? Buď tím, že do tabulky na straně N vložíme záznam, jehož cizí klíč neodpovídá žádné hodnotě primárního klíče v tabulce na straně 1. Nebo se hodnota primárního klíče v tabulce na straně 1 změní, ale nezmění se odpovídající hodnota cizího klíče v tabulce na straně N. Nebo odstraníme z tabulky na straně 1 záznam, na nějž se odkazuje cizí klíč v tabulce na straně N. Jestliže má tedy systém zajistit referenční integritu, musí zabránit vzniku těchto tzv. sirotků, tj. záznamů v tabulce na straně N, které se odkazují na neexistující hodnoty primárních klíčů na straně 1. Prakticky v našich tabulkách

ZAMĚSTNANEC a DÍTĚ to znamená, že systém nedovolí vymazat záznam o zaměstnanci, pokud v tabulce DÍTĚ existuje záznam, který se na tohoto zaměstnance vztahuje. Abychom tedy mohli smazat záznam o zaměstnanci, musíme nejdříve odstranit záznamy o jeho dětech z tabulky DÍTĚ. Odstranění souvisejících záznamů může systém zajišťovat i automaticky. Obdobně v případě změny hodnoty primárního klíče by měl systém buď změně zabránit nebo provést automatickou aktualizaci hodnot souvisejících cizích klíčů.

- *Úroveň celé databáze (databázová integrita)*

  Omezení definovaná na této úrovni se definují s ohledem na více tabulek. Např. má-li zákazník víc než 50 realizovaných nákupů, je zařazen do kategorie A, pro kterou platí určitá procentní sleva z ceny nákupu.

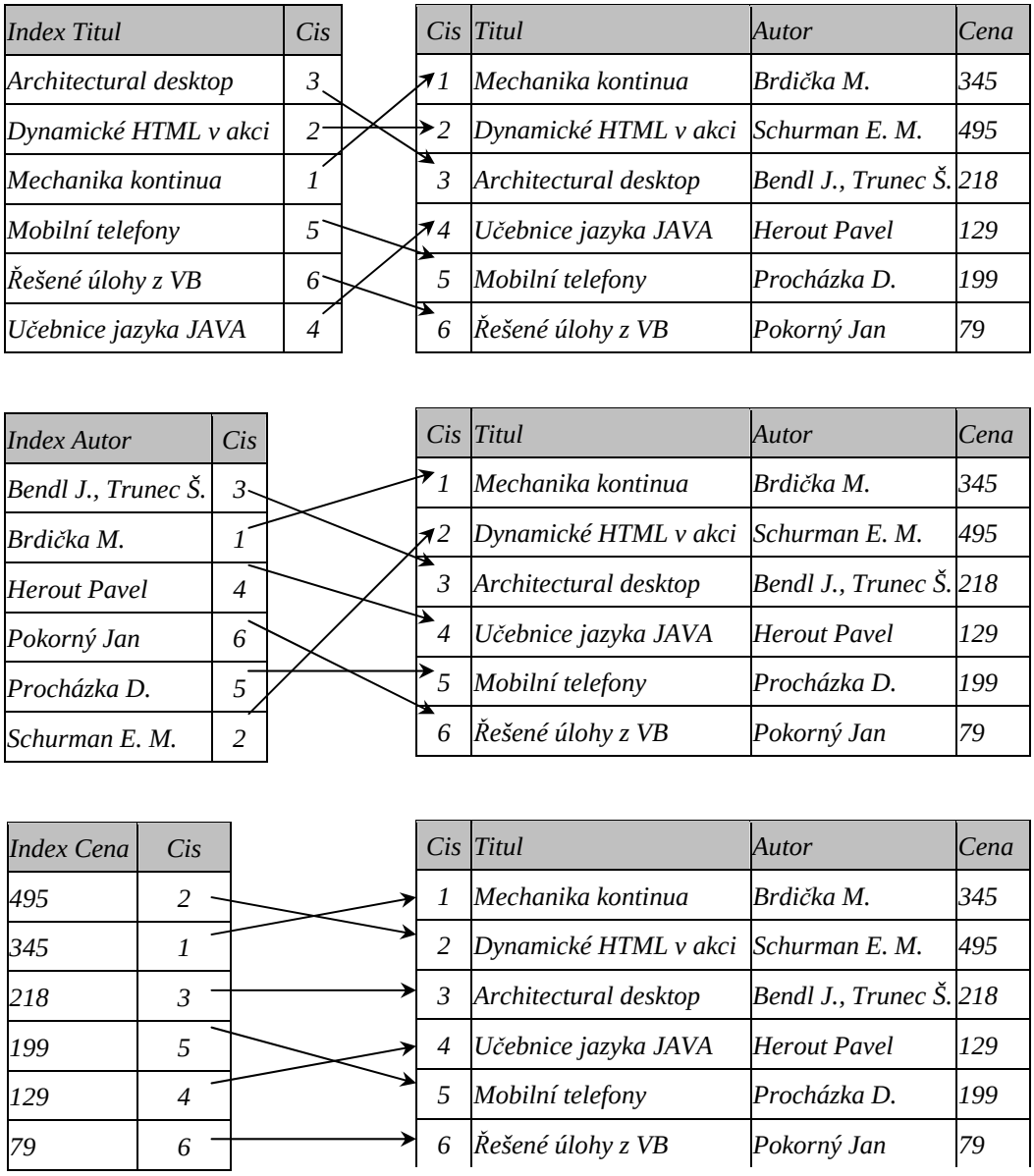
6.4. Objekty v databázi

Jistě jste z dosavadního výkladu pochopili, že při popisu a modelování reality se postupuje směrem jakéhosi „zjednodušení skutečnosti“ rozdělením dat do samostatných částí (tabulek). Jenže dotazy, na které má systém odpovídat, jsou většinou komplexní, a aby mohly být zodpovězeny, musí čerpat z dat uložených ve více tabulkách. Například z naší databáze bychom chtěli získat seznam zaměstnanců a jména jejich dětí včetně názvu oddělení v němž pracují. Pro získání požadované odpovědi potřebujeme z tabulky ODDĚLENÍ pole [Název] a pole [Příjmení] a [Jméno] z tabulek ZAMĚSTNANEC a DÍTĚ. Vidíme, že výslednou odpovědí je zase tabulka. Tato tabulka se však liší od tabulek ZAMĚSTNANEC nebo DÍTĚ v tom, že je pouze „virtuální“. Představuje totiž pouze určitý pohled na data uložená ve fyzických tabulkách. Pohledy tedy umožňují prohlížet si data různými způsoby podle požadavků uživatele vybíráním a kombinováním zdrojových dat pomocí operací relační algebry. K vytváření pohledů slouží **dotazovací jazyky**, kterými se relační operace vyjadřují. Operace relační algebry budou probrány v kapitole 6.5, kde si je předvedeme současně s příkazy jazyka SQL, které je realizují. Na tomto místě se zmíníme ještě o dalším typu objektu, který kromě tabulek a pohledů najdeme v každém databázovém systému. Jsou to tzv. indexy. Jedná se o pomocné informace, které však mohou velmi urychlit zpracování našich požadavků na informace ze systému. V kapitole 6.1 jsme si řekli, že záznamy v tabulkách jsou uloženy neuspořádaně. Jinak to ani není možné, protože existuje řada kritérií podle kterých lze záznamy třídit. Jestliže hledáme určitý záznam v tabulce, musíme přistupovat k diskovým pamětem a to je obvykle pomalé, proto je třeba počet přístupů minimalizovat. Indexové soubory by měli tuto činnost zefektivnit. O co jde, si můžeme vysvětlit na známém příkladu vyhledávání knih podle různých rejstříků v knihovně. Předpokladem je, že knihy umisťované do polic knihovny budou opatřeny štítkem s jedinečným evidenčním číslem (v databázové terminologii bychom řekli, že se jedná o primární klíč). Police se plní postupně, jak se knihy nakupují. Protože při hledání určité knihy nebudeme chtít prohledávat police od začátku knihu po knize, vytvoříme si kartotéku, kam budeme v abecedním pořadí místo knih ukládat lístky s informacemi o knize. S každou nově zařazovanou knihou je tedy třeba vyplnit i indexový lístek a zařadit do kartotéky. Budeme-li chtít urychlit vyhledávání nejen při znalosti názvu, ale i autora knihy, musíme vést kartotéky dvě. Z příkladu je jasné, že se zavedením indexů pro urychlení vyhledávání je spojena i určitá

režie (čas potřebný k vyplnění indexových lístků, místo pro umístění kartotéky). Obdobně pracují i databázové indexové soubory. Na dalším příkladu si ukažme, jak by indexové soubory pro příklad knihovny mohly vypadat.

Příklad 6.4

Pro tabulku kniha jsou vytvořeny 3 indexové soubory: TITUL – vzestupné řazení dle titulu, AUTOR – vzestupné řazení dle autora, CENA – sestupné řazení dle ceny knihy. Vidíme, že indexový soubor obsahuje požadované setříděné pole a hodnotu primárního klíče tohoto záznamu.



Index Titul	*Cis*
Architectural desktop	*3*
Dynamické HTML v akci	*2*
Mechanika kontinua	*1*
Mobilní telefony	*5*
Řešené úlohy z VB	*6*
Učebnice jazyka JAVA	*4*

Cis	*Titul*	*Autor*	*Cena*
1	*Mechanika kontinua*	*Brdička M.*	*345*
2	*Dynamické HTML v akci*	*Schurman E. M.*	*495*
3	*Architectural desktop*	*Bendl J., Trunec Š.*	*218*
4	*Učebnice jazyka JAVA*	*Herout Pavel*	*129*
5	*Mobilní telefony*	*Procházka D.*	*199*
6	*Řešené úlohy z VB*	*Pokorný Jan*	*79*

Index Autor	*Cis*
Bendl J., Trunec Š.	*3*
Brdička M.	*1*
Herout Pavel	*4*
Pokorný Jan	*6*
Procházka D.	*5*
Schurman E. M.	*2*

Cis	*Titul*	*Autor*	*Cena*
1	*Mechanika kontinua*	*Brdička M.*	*345*
2	*Dynamické HTML v akci*	*Schurman E. M.*	*495*
3	*Architectural desktop*	*Bendl J., Trunec Š.*	*218*
4	*Učebnice jazyka JAVA*	*Herout Pavel*	*129*
5	*Mobilní telefony*	*Procházka D.*	*199*
6	*Řešené úlohy z VB*	*Pokorný Jan*	*79*

Index Cena	*Cis*
495	*2*
345	*1*
218	*3*
199	*5*
129	*4*
79	*6*

Cis	*Titul*	*Autor*	*Cena*
1	*Mechanika kontinua*	*Brdička M.*	*345*
2	*Dynamické HTML v akci*	*Schurman E. M.*	*495*
3	*Architectural desktop*	*Bendl J., Trunec Š.*	*218*
4	*Učebnice jazyka JAVA*	*Herout Pavel*	*129*
5	*Mobilní telefony*	*Procházka D.*	*199*
6	*Řešené úlohy z VB*	*Pokorný Jan*	*79*

Tab. 6.6 Příklady nastavení indexových souborů na tabulku KNIHA

Jak vidíme z obrázku, je index rovněž tabulka, ale speciální. Nemůžeme k ní přistupovat přímo, ale je využívána k tomu, aby na základě požadovaného způsobu třídění poskytla ukazatele na záznamy v bázové tabulce. Indexů na jednu tabulku může být dle požadavků uživatele sestaveno více, databázový systém většinou automaticky vytváří index nad klíčovým atributem. Jak už bylo na-

značeno na příkladu reálné knihovny, vytváření a udržování indexů vyžaduje určitou systémovou režii. Nedá se tedy obecně říci, že vytvoření indexu musí vždy pozitivně ovlivnit práci s databází. Jestliže se vyhledává příliš mnoho údajů nebo se požaduje indexování pole, které se často mění, může být efektivnější pracovat bez indexu.

Shrňme a ujasněme si terminologii:

Tabulky, jejichž data jsou skutečně fyzicky uložena v databázi, se nazývají **bázové tabulky**.

Pohled představuje dotazovací výraz, na základě kterého je sestavena odvozená (virtuální) tabulka. Jednou formulovaný pohled lze pojmenovat a uložit v databázi. Práce s pohledy je obdobná jako práce s tabulkami: lze z nich vybírat jen určité údaje nebo je spojovat s jinými pohledy nebo tabulkami a vytvářet pohledy nové.

Index je pomocná informace, která slouží k urychlení vyhledávání v tabulkách.

Poznámka. Výše zmíněné objekty nejsou jediné, se kterými se v moderních databázových systémech setkáme, patří však mezi základní. Každý systém např. obsahuje informace, které popisují vlastní databázové struktury. Těmto metadatům (informacím o informacích) se říká **slovník nebo katalog dat**.

6.5. Dotazovací jazyky a relační algebra

Už víme, že dotazovací jazyky slouží k formulaci dotazů. Protože tabulky představují relace, a to jsou množiny, jsou nástrojem pro manipulaci s nimi množinové operace: kartézský součin, průnik, rozdíl a sjednocení. Kromě toho se uplatní i specificky databázové operace jako jsou projekce, selekce a spojení. Jelikož pracujeme s tabulkami a nikoli s obecnými množinami, je používaná terminologie a příklady množinových operací přizpůsobeny této skutečnosti. Příslušné operace lze provádět za předpokladu vzájemné kompatibility operandů: to znamená, stejného řádu relace (počet sloupců tabulky) a odpovídajících domén (souhlasné datové typy sloupců na stejné pozici). V kapitole 3.4 jsme získali základní informaci o jazyku SQL. Nyní probereme jednotlivé operace relační algebry a u každé si uvedeme příklad zápisu příkazu jazyka SQL, který ji realizuje. V této kapitole se jedná pouze o ilustrativní ukázky příkazů, které se budeme snažit vysvětlit, ale podrobným popisem syntaxe se v této kapitole zabývat nebudeme, tomu je věnována kapitola 7.

6.5.1. Projekce

Projekcí se vyberou z původní množiny A do výsledné množiny B pouze vybrané sloupce. Pokud vzniknou ve výsledné množině duplicitní řádky, jsou odstraněny. Ukažme si tuto situaci na příkladu:

Příklad 6.5

Máme tabulku ZÁKAZNÍK a pro potřeby tisku adres na informační letáky z ní potřebujeme vybrat jen údaje o jménech a adresách a ty seřadíme v rámci města podle abecedy.

ID *čís.*	***Příjme-ní,*** *text*	***Jméno*** *text*	***Město*** *text*	***PSČ*** *text*	***Ulice*** *text*	***Čp*** *čís.*	***Telefon*** *text*	***Mobil*** *text*	***Zápis*** *datum*
1	*Pilná*	*Jana*	*Brno*	*60200*	*Orlí*	*15*	*123456798*	*603123456*	*2.4.1994*
2	*Nový*	*Adam*	*Brno*	*60800*	*Cejl*	*18*	*111222111*	*707458789*	*9.5.2000*
3	*Takáč*	*Iano*	*Vyškov*	*68200*	*Oblá*	*1*	*789456123*	*603556899*	*6.6.2002*
4	*Blatná*	*Hana*	*Luleč*	*68700*	*Úzká*	*16*	*142563789*	*628456147*	*3.7.1999*

Tab. 6.7 Data v tabulce ZÁKAZNÍK

Příjmení *text*	***Jméno*** *text*	***Město*** *text*	***PSČ*** *text*	***Ulice*** *text*	***Čp*** *čís.*
Nový	*Adam*	*Brno*	*60800*	*Cejl*	*18*
Pilná	*Jana*	*Brno*	*60200*	*Orlí*	*15*
Blatná	*Hana*	*Luleč*	*68700*	*Úzká*	*16*
Takáč	*Ivan*	*Vyškov*	*68200*	*Oblá*	*1*

Tab. 6.8 Projekce tabulky ZÁKAZNÍK na výslednou množinu ADRESY

Realizace prostřednictvím příkazu SQL vypadá následovně:

```
SELECT Příjmení, Jméno, Město, PSČ, Ulice, Čp
FROM Zákazník
ORDER BY Město,Příjmení
```

Pokud umíte anglicky, měl by vám být celý zápis docela srozumitelný:

za klíčovým slovem SELECT (vyber) se uvede seznam polí, která chceme mít ve výstupní množině, za slovem FROM (z) je jméno zdrojové tabulky (případně pohledu), ze které data bereme a pokud chceme výstupní množinu setřídit, přidáme klíčové slovo ORDER BY (pořadí podle), za které uvedeme seznam polí, podle nichž chceme řádky třídit.

6.5.2. Selekce (restrikce)

Selekcí se omezuje množina záznamů. Na základě zadaného kritéria se z původní množiny vyberou jen ty řádky, které kritérium splňují. Co se týká sloupců, mohou být ve výstupní množině obsažena všechna pole původní množiny, nebo lze pomocí projekce vybrat jen některé sloupce. Pokud bychom z naší tabulky ZÁKAZNÍK chtěli vybrat pouze zákazníky z Brna, bude zápis v jazyce SQL vypadat takto:

```
SELECT *
FROM Zákazník
WHERE Město='Brno'
```

Hvězdička za klíčovým slovem SELECT znamená, že výstupní množina bude obsahovat všechny sloupce původní množiny. Objevilo se zde nové klíčové slovo WHERE (kde), za kterým se zapisují podmínky, které omezují záznamy ve výstupní množině. Pro formulaci těchto podmínek existují určitá pravidla, jejich výklad je však nad rámec našeho textu. Uveďme si ještě příklad kombinace obou operací, tedy případ, kdy bychom chtěli vypsat abecední seznam adres zákazníků z Brna:

```
SELECT Příjmení, Jméno, Město, PSČ, Ulice, Čp
FROM Zákazník
WHERE Město='Brno'
ORDER BY Příjmení
```

6.5.3. Spojení

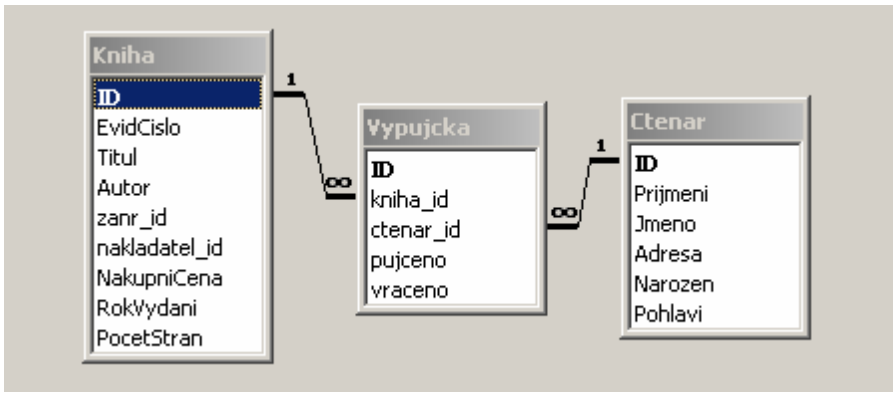
Operace spojení je patrně nejdůležitější a nejpoužívanější relační operací. Vždyť jestliže jsou v tabulkách uložena dekomponovaná data, musíme je umět zpětně spojit. A právě to provádí operace spojení: na základě porovnání jednoho nebo více polí spojovaných tabulek vrací sloučené řádky obou tabulek. Podle toho, na jakém operátoru porovnání je spojení založeno, se běžně rozlišují dva druhy spojení:

- Spojení na rovnost (equi-join) – nejobvyklejší způsob spojení pomocí operátoru =
- Theta spojení – spojení založené na jakémkoli jiném operátoru spojení (<, <=, >, >=, <>); tento typ spojení je v praxi poměrně vzácný.

Všechny výše uvedené typy spojení vrací jen ty záznamy, pro které je spojovací podmínka vyhodnocena jako pravdivá. O takovém typu spojení mluvíme jako o **vnitřním spojení** (inner join). Relační algebra podporuje ještě tzv. **vnější spojení** (outer join), které vrací odpovídající záznamy jako vnitřní spojení a navíc záznamy z jedné (left outer join) nebo druhé tabulky (right outer join), ve kterých jsou chybějící hodnoty doplněny hodnotou NULL (viz. kapitola 6.3). Vnější spojení může tedy být levé nebo pravé.

Příklad 6.6

Příklad spojení si ukážeme tabulkách KNIHA, VYPUJCKA a CTENAR, jejichž struktura a vzájemné vztahy jsou patrné z následujícího obrázku:



Obr. 6.5 Vztahy mezi tabulkami KNIHA, VYPUJCKA a CTENAR

Chceme-li vypsat informace o výpůjčkách, ale místo čísla čtenáře, které je v tabulce VYPUJCKA, bychom chtěli vidět příjmení a jméno čtenáře, budeme potřebovat spojit tabulku VYPUJCKA a CTENAR na základě rovnosti polí [ID] v tabulce CTENAR a [Ctenar_ID] v tabulce VYPUJCKA. Zápis příkazu v SQL bude následující:

```
SELECT Vypujcka.*, Ctenar.Prijmeni
FROM Ctenar INNER JOIN Vypujcka
ON Ctenar.ID = Vypujcka.Ctenar_id
```

Kromě nových klíčových slov INNER JOIN (vnitřní spojení) a ON (podle) v zápise příkazu vidíme ještě tzv. kvalifikované názvy polí (např. Ctenar.ID). Jde o to, že v jednotlivých tabulkách mohou být použity stejné názvy polí, a proto tam, kde se pracuje s více tabulkami, musí být jednoznačně řečeno o jaké pole z jaké tabulky se jedná. V souladu s objektovým způsobem zápisu tedy názvu pole předchází tečkou oddělený název tabulky, z níž pochází. Výsledkem tohoto příkazu je následující tabulka:

ID	*kniha_id*	*ctenar_id*	*pujceno*	*vraceno*	*Prijmeni*
4	*136*	*1*	*15.09.2003*	*16.09.2003*	*Bílá*
5	*204*	*1*	*16.09.2003*	*22.11.2003*	*Bílá*
12	*241*	*3*	*26.04.2004*		*Vodák*
11	*137*	*4*	*10.04.2004*	*31.03.2004*	*Sivá*
1	*285*	*5*	*12.02.2003*	*15.03.2003*	*Divý*
8	*312*	*5*	*18.02.2004*	*20.03.2004*	*Divý*
13	*369*	*5*	*01.05.2004*		*Divý*
2	*128*	*6*	*13.11.2003*	*20.12.2003*	*Novák*
6	*156*	*6*	*14.10.2003*	*02.01.2004*	*Novák*
10	*305*	*6*	*05.03.2004*	*15.03.2004*	*Novák*
3	*212*	*7*	*01.06.2003*	*31.08.2003*	*Novák*
14	*333*	*7*	*12.05.2004*		*Novák*
7	*239*	*9*	*28.12.2003*	*26.01.2004*	*Vacková*
9	*212*	*9*	*13.02.2004*	*29.02.2004*	*Vacková*

Tab. 6.9 Výsledek vnitřního spojení tabulky VYPUJCKA a CTENAR

Příkladem levého vnějšího spojení je příkaz, který vrací následující tabulku, v níž jsou všichni čtenáři, tedy i ti, kteří dosud nerealizovali žádnou výpůjčku:

```
SELECT Vypujcka.*, Ctenar.Prijmeni
FROM Ctenar LEFT JOIN Vypujcka
ON Ctenar.ID = Vypujcka.Ctenar_id
```

Prijmeni	Jmeno	ID	kniha_id	ctenar_id	pujceno	vraceno
Bílá	Eva	4	136	1	15.09.2003	16.09.2003
Bílá	Eva	5	204	1	16.09.2003	22.11.2003
Teplý	Jan					
Vodák	Kamil	12	241	3	26.04.2004	
Sivá	Věra	11	137	4	10.04.2004	31.03.2004
Divý	Tibor	1	285	5	12.02.2003	15.03.2003
Divý	Tibor	8	312	5	18.02.2004	20.03.2004
Divý	Tibor	13	369	5	01.05.2004	
Novák	Karel	2	128	6	13.11.2003	20.12.2003
Novák	Karel	6	156	6	14.10.2003	02.01.2004
Novák	Karel	10	305	6	05.03.2004	15.03.2004
Novák	Adam	3	212	7	01.06.2003	31.08.2003
Novák	Adam	14	333	7	12.05.2004	
Bláhová	Jana					
Vacková	Marie	7	239	9	28.12.2003	26.01.2004
Vacková	Marie	9	212	9	13.02.2004	29.02.2004
Gala	Filip					
Koláček	Emil					
Drahoš	Karel					
Novák	Karel					

Tab. 6.10 Výsledek levého vnějšího spojení tabulky VYPUJCKA a CTENAR

Případ pravého vnějšího spojení na těchto tabulkách by obdobně vracel všechny záznamy z tabulky VYPUJCKA, tedy i ty, pro které nebyl nalezen odpovídající čtenář. Správně fungující systém by však vzniku tohoto stavu měl zabránit (požadavek zachování referenční integrity) a nedovolit vložení výpůjčky pro neexistujícího čtenáře. Naše tabulka VYPUJCKA tomu odpovídá, proto výsledkem pravého vnějšího spojení našich tabulek VYPUJCKA a CTENAR by byla stejná množina záznamů jako v případě vnitřního spojení.

6.5.4. Sjednocení

Sjednocením dvou množin A a B ($A \cup B$) vznikne výsledná množina C, ve které budou všechny záznamy z množiny A i B. Předpokladem je, že z obou množin bude vybrán stejný počet atributů odpovídajícího datového typu. Na ukázku bychom chtěli např. pro účely rozeslání novoročních blahopřání našim zákazníkům i zaměstnancům získat jména, příjmení a adresy z tabulek ZÁKAZNÍK (viz. příklad 6.5) a ZAMESTNANEC (viz. příklad 6.1). Vidíme, že struktury obou tabulek nejsou stejné, najdeme v nich však údaje, které potřebujeme: pole [Příjmení] a [Jméno] jsou v obou tabulkách shodné, informace o adrese však je v tabulce ZAMĚSTNANEC uložena v poli [Bydliště] a v tabulce ZÁKAZNÍK ve čtyřech polích [Město], [PSČ], [Ulice] a [Čp]. Tato pole lze spojit a vložit do výsledné množiny jako jedno pole, které odpovídá adrese.

ID *číslo*	***Příjmení*** *text, 20 z.*	***Jméno*** *text,10 z*	***Bydliště*** *text, 30 z*	***Plat*** *číslo*	***Narozen*** *datum*	***Odborář*** *log. hodn.*	***Oddělení*** *text, 30 z*	***Děti*** *číslo.*
1	*Prouza*	*Jan*	*Brno, Úzká 3*	*15000*	*2.4.1974*	*ano*	*Projekční*	*3*
2	*Novák*	*Adam*	*Brno, Cejl 24*	*18000*	*9.5.1960*	*ne*	*Osobní*	*1*
3	*Bílá*	*Dana*	*Kyjov, Trh 7*	*16000*	*3.7.1969*	*ne*	*Prodejní*	*2*
4	*Tesař*	*Ivo*	*Podomí 259*	*12000*	*6.6.1982*	*ano*	*Osobní*	*0*

Tab. 6.11 Data v tabulce ZAMĚSTNANEC

ID *čís.*	***Příjme-ní,*** *text*	***Jméno*** *text*	***Město*** *text*	***PSČ*** *text*	***Ulice*** *text*	***Čp*** *čís.*	***Telefon*** *text*	***Mobil*** *text*	***Zápis*** *datum*
1	*Pilná*	*Jana*	*Brno*	*60200*	*Orlí*	*15*	*123456798*	*603123456*	*2.4.1994*
2	*Nový*	*Adam*	*Brno*	*60800*	*Cejl*	*18*	*111222111*	*707458789*	*9.5.2000*
3	*Takáč*	*Iano*	*Vyškov*	*68200*	*Oblá*	*1*	*789456123*	*603556899*	*6.6.2002*
4	*Blatná*	*Hana*	*Luleč*	*68700*	*Úzká*	*16*	*142563789*	*628456147*	*3.7.1999*

Tab. 6.12 Data v tabulce ZÁKAZNÍK

Zápis příkazu SQL a výsledná tabulka bude následující:

```
SELECT Příjmení,Jméno,Bydliště
FROM Zaměstnanec
UNION
SELECT Příjmení,Jméno,Město&', '&Ulice&' '& Cstr(Čp)
FROM Zákazník
```

Příjmení *text, 20 z.*	***Jméno*** *text,10 z*	***Bydliště*** *text, 30 z*
Prouza	*Jan*	*Brno, Úzká 3*
Novák	*Adam*	*Brno, Cejl 24*
Bílá	*Dana*	*Kyjov, Trh 7*
Tesař	*Ivo*	*Podomí 259*
Pilná	*Jana*	*Brno,Orlí 15*
Nový	*Adam*	*Brno, Cejl 18*
Takáč	*Iano*	*Vyškov, Oblá 1*
Blatná	*Hana*	*Luleč, Úzká 16*

Tab. 6.13 Sjednocení tabulek ZAMĚSTNANEC a ZÁKAZNÍK

6.5.5. Průnik a rozdíl

Operace průniku (A ∩ B) dvou množin se stejnými atributy vrátí ty záznamy, které jsou v obou množinách shodné. Tuto operaci bychom mohli označit jako hledání duplicit.

Operace rozdíl (A - B) dvou množin se stejnými atributy vrátí ty záznamy, které patří jen do jedné množiny a nejsou obsaženy ve druhé množině. Tuto operaci bychom mohli označit jako hledání sirotků.

Pro demonstraci těchto dvou operací použijeme tabulky KNIHA1 a KNIHA2, které obsahují seznam knih. Tabulky mohly být pořízeny různými systémy nebo aplikacemi a obsahují některé záznamy shodné, některé se vyskytují jen v jedné z nich.

Titul	***Autor***
Mechanika kontinua	*Brdička M.*
WEB DESIGN	*Satrapa Pavel*
Architectural desktop	*Bendl J., Trunec Š.*
Excel 97	*Kořínek M.*
Mobilní telefony	*Procházka D.*
Řešené úlohy z VB	*Pokorný Jan*
Dynamické HTML v akci	*Schurman E. M.*
Grafické formáty	*Sobota B., Milián J.*
Učebnice jazyka JAVA	*Herout Pavel*

Tab. 6.14 Data v tabulce KNIHA1

Titul	*Autor*
Mechanika kontinua	*Brdička M.*
Dynamické HTML v akci	*Schurman E. M.*
Architectural desktop	*Bendl J., Trunec Š.*
Učebnice jazyka JAVA	*Herout Pavel*
Mobilní telefony	*Procházka D.*
Řešené úlohy z VB	*Pokorný Jan*

Tab. 6.15 Data v tabulce KNIHA2

Pro obě operace je třeba nejprve obě tabulky spojit vnějším spojením, aby výsledná množina obsahovala všechny záznamy jedné z tabulek. U těch záznamů, které nemají odpovídající hodnoty v druhé tabulce bude doplněna hodnota NULL. Výsledná tabulka vytvořená pomocí následujícího příkazu SQL bude vypadat takto:

```
SELECT Kniha1.*, Kniha2.*
FROM Kniha1 LEFT JOIN Kniha2
ON Kniha1.Titul = Kniha2.Titul
AND Kniha1.Autor = Kniha2.Autor
```

Kniha1.Titul	*Autor*	*Kniha2.Titul*	*Kniha2.Autor*
Mechanika kontinua	*Brdička M.*	*Mechanika kontinua*	*Brdička M.*
WEB DESIGN	*Satrapa Pavel*		
Architectural desktop	*Bendl J., Trunec*	*Architectural desktop*	*Bendl J., Trunec Š.*
Excel 97	*Kořínek M.*		
Mobilní telefony	*Procházka D.*	*Mobilní telefony*	*Procházka D.*
Řešené úlohy z VB	*Pokorný Jan*	*Řešené úlohy z VB*	*Pokorný Jan*
Dynamické HTML v	*Schurman E. M.*	*Dynamické HTML v*	*Schurman E. M.*
Grafické formáty	*Sobota B., Milián*		
Učebnice jazyka JAVA	*Herout Pavel*	*Učebnice jazyka JAVA*	*Herout Pavel*

Tab. 6.16 Výsledek levého vnějšího spojení tabulky KNIHA1 a KNIHA2

Jak už bylo řečeno, operace průniku vrací shodné řádky v obou tabulkách. Pro získání této množiny záznamů, je třeba provést selekci řádků pouze na ty, které neobsahují hodnoty NULL. K výše uvedenému příkazu SQL je třeba toto omezení dodat v následující podobě:

```
WHERE (Kniha2.Titul Is Not Null)
```

Obdobně, jestliže operace rozdílu vrací řádky, které nejsou obsaženy v obou tabulkách, je třeba provést selekci řádků pouze na ty, které obsahují hodnoty NULL. Tvar podmínky v příkazu SQL bude následující:

```
WHERE (Kniha2.Titul Is Null)
```

6.5.6. Kartézský součin

Kartézský součin dvou množin (A x B) zkombinuje každý řádek množiny A s každým řádkem množiny B. Takže jestliže má jedna tabulka *m* řádků a druhá tabulka *n* řádků, výsledná tabulka bude mít *m* x *n* řádků. Jedná se většinou o poměrně rozsáhlé množiny, které však vznikají spíše omylem, pokud se spojí dvě tabulky bez uvedení podmínky spojení. Kartézský součin lze realizovat příkazem SQL:

```
SELECT Vypujcka.*, Ctenar.Prijmeni
FROM Ctenar, Vypujcka
```

7. Jazyk SQL

7.1. Druhy příkazů

V kapitole 3.4 jsme získali základní informaci o jazyku SQL, v předchozí kapitole jsme viděli příklady zápisu nejčastěji používaného příkazu SELECT. V této kapitole bychom rádi zdůraznili, že SQL je víc než dotazovací jazyk, že se vlastně jedná o kompletní jazyk pro správu databází. Obsahuje totiž příkazy všech komponent, které zabezpečují služby každého SŘBD (viz. kapitola 3.3). Tato kapitola uvádí výčet těch nejdůležitějších, zatím bez podrobné syntaxe, která bude probrána v následující kapitole.

Příkazy jazyka DDL – umožňují definovat objekty databáze, měnit a rušit je (tabulky, pohledy, indexy). Vytvoření nových objektů zajišťují příkazy začínající slovem CREATE (vytvoř), pro změnu objektů příkazy začínající slovem ALTER (změň) a pro jejich rušení slovem DROP (odstraň). Za ním vždy následuje typ objektu. a název objektu, jehož se příkaz týká. Jsou to například příkazy:

`CREATE TABLE`	`CREATE VIEW`	`CREATE INDEX`
`DROP TABLE`	`DROP VIEW`	`DROP INDEX`
`ALTER TABLE`	`ALTER VIEW`	

Účinky všech definičních příkazů se projeví změnami v systémovém katalogu. Příkazem `CREATE DATABASE` lze vytvořit samotnou databázi.

Příkazy jazyka DML – umožňují vybírat, modifikovat, mazat a přidávat data v tabulkách. Nejdůležitějším příkazem této skupiny je nám už známý příkaz SELECT. Zde je třeba si také uvědomit rozdíl mezi rušením a mazáním. Příkaz DELETE smaže řádky tabulky, ale nikoli tabulku, ta ač prázdná zůstává zachována. Naproti tomu příkaz DROP TABLE odstraní celou tabulku. Další příkazy této skupiny jsou:

`SELECT`	výběr dat z tabulek
`INSERT`	vložení nového řádku tabulky
`DELETE`	smazání dat v tabulce
`UPDATE`	oprava dat v tabulce

Tato skupina představuje nejčastěji užívané příkazy SQL. Zejména příkaz `SELECT` užívaný k formulaci dotazů zjišťujících informace z uložených dat, je patrně nejfrekventovanějším příkazem SQL.

Příkazy jazyka DCL – řídící příkazy, které umožňují řídit provoz a údržbu databáze, např. zavádění nových uživatelů a přidělování přístupových práv jednotlivým uživatelům. Jedná se například o příkazy:

`CREATE USER`	`CREATE ROLE`	`GRANT`
`DROP USER`	`DROP ROLE`	`REVOKE`
`ALTER USER`		

Příkazy jazyka TCC – zajišťují řízení transakcí. Mohou to být např. příkazy:

`ROLLBACK`	návrat dat do stavu před začátkem transakce
`COMMIT`	ukončení aktuální transakce
`SAVEPOINT`	uložení aktuálního stavu dat s možností pozdějšího návratu k tomuto stavu

Nutno podotknout, že provádění příkazů v databázi je vždy spojeno s oprávněním uživatele manipulovat s určitými objekty nebo provádět určité činnosti v závislosti na přidělených právech, případně vlastnictví objektů. Z tohoto pohledu běžný uživatel zpravidla řídící a transakční příkazy nedělá.

7.2. Zápis nejdůležitějších SQL příkazů

Zápis SQL příkazu, je jakousi formou programování a vyžaduje tudíž dodržení určitých pravidel, aby mohl být správně interpretován. Pro některé nejdůležitější příkazy si uvedeme popis jejich syntaxe, při kterém budou použity symboly uvedené v tabulce 7.1. Syntaktická pravidla nezahrnují formální úpravu zápisu příkazů, takže nezáleží na velikosti písma, ani počtu řádků, na kterých jsou zapsány. V tomto textu použijeme pro zápis klíčových slov jazyka SQL písmena velké abecedy, pro názvy tabulek malá písmena s počátečním velkým a pro označení sloupců písmena malé abecedy. Příkazy budeme členit na více řádků, aby zápis byl co nejpřehlednější. Vzhledem k tomu, že syntax jazyka SQL se vždy v jednotlivých implementacích poněkud liší od standardu, nebu-

deme se zabývat podrobnostmi žádného konkrétního databázového systému, ale budeme se držet základu normy SQL-92 bez detailních podrobností, které nejsou pro pochopení funkce příkazu podstatné. Aby však bylo možné vyzkoušet příklady v konkrétním DBS, jsou ukázky jednotlivých příkladů příkazů uvedeny pro MS Access, kde je lze spustit.

Poznámka: Chcete-li si příkazy v Accessu vyzkoušet, vyberte v databázovém manažeru objekt „Dotazy“ a zde položku „Vytvořit dotaz v návrhovém zobrazení“. Dialogové okno, které požaduje zadání tabulky, se kterou dotaz pracuje můžete zavřít bez výběru tabulky. Prostředí návrhového zobrazení dotazu přepněte z grafické podoby do textové výběrem volby „Zobrazení SQL“ na prvním tlačítku vlevo v nástrojové liště (rolovací tlačítko „Zobrazit“ s výběrem možností zobrazení dotazu).

Symbol	Popis
< >	Parametr, za který se dosadí konkrétní hodnota
[ ]	Volitelná, nepovinná část
|	Oddělovač variant, které lze použít
‘	Ohraničení textových konstant (apostrof)
...	Možnost opakování
{ }	Povinná volba jedné z uvedených možností.

Tab. 7.1 Tabulka symbolů použitých v popisu syntaxe příkazů

7.2.1. Příkaz CREATE TABLE

Vytvoří novou, *prázdnou* tabulku a popis uloží do katalogu dat.

```
CREATE TABLE <nazev_tabulky>
  (<definice_sloupce1> [,<definice_sloupce2> [, ...]]
  [<definice_omezeni_tabulky>]
  )
```

kde

```
< definice _sloupce> =
  <nazev_sloupce> <datovy_typ> [<def_omezeni_sloupce>]

<def_omezeni_sloupce> =
  [CONTRAINT <nazev_omezeni>]
  {[NULL | NOT NULL | UNIQUE | PRIMARY KEY |
   REFERENCES <nazev_tabulky> [(<seznam_sloupcu>) |
   CHECK (<podminka>)]}

<definice_omezeni_tabulky> =
  [CONTRAINT < nazev_omezeni >]
  {[UNIQUE (<seznam_sloupcu>) |
   PRIMARY KEY (<seznam_sloupcu>) |
   CHECK (<podminka>) |
   FOREIGN KEY (<seznam_sloupcu>)
   REFERENCES <nazev_tabulky> [(<seznam_sloupcu>)]]}
```

Názvy

Už bylo zmíněno, že názvy tabulek musí být v rámci systému unikátní, stejně jako názvy sloupců v rámci tabulek. Pro název sloupce navrženého jako cizí klíč je vhodné volit název tabulky, jejíž primární klíč vyjadřuje. Pro usnadnění orientace v rozsáhlejších systémech, je dobré přijmout určitá pravidla v označování objektů. Názvy by měly být smysluplné, výstižné a ne příliš dlouhé. Někdy se to může vylučovat, ale je třeba volit kompromisy. Například pro názvy tabulek a sloupců se volí podstatná jména vyjadřující druh dat do nich ukládaných. Pokud se použije víceslovné označení, je vhodné nepoužívat v názvech mezeru (i když to některé systémy připouštějí) a mezeru nahradit ji například podtržítkem (viz definice výše) nebo počátečním velkým písmenem druhého slova (viz příklady dále). Rovněž se nedoporučuje používat v názvech českou diakritiku.

Datový typ

Je obor hodnot, které mohou být do sloupce ukládány. Je možno použít pouze typy definované pro konkrétní databázový systém.

Omezení sloupce

Představuje možnost uživatele zadat integritní omezení týkající se především povolení nebo zakázání výskytu duplicitních hodnot a hodnot NULL. Dále je možno sloupec definovat jako primární klíč nebo cizí klíč, případně zadat podmínku, kterou musí splňovat všechny hodnoty zadávané do sloupce.

CONSTRAINT – omezení lze pojmenovat, což umožňuje pozdější modifikaci omezujících podmínek příkazem ALTER.

NULL – tuto hodnotu má pole, pokud do něj není zadána hodnota. Chceme-li tomu zabránit (vynutit zadání hodnoty do pole), můžeme v definici napsat:

```
konecHod      INTEGER    NOT NULL,
popisZavady   CHAR(20)   NULL,
```

Vysvětlení: do pole s názvem `konecHod` musejí být vždy zadány celočíselné hodnoty. Do pole `popisZavady` může, ale nemusí být vložen text o maximální délce 20 znaků.Ačkoli je to položka volitelná, doporučuje se explicitně uvést, jelikož různé systémy mohou mít různé výchozí nastavení.

UNIQUE – zabrání výskytu duplicitních hodnot ve sloupci.

```
cislo         INTEGER    NOT NULL   UNIQUE,
rodneCislo    CHAR(10)   NULL       UNIQUE
```

Vysvětlení: do pole s názvem `cislo` musejí být vždy zadány celočíselné unikátní hodnoty. Do pole `rodneCislo` může být zadáno 10 znaků, jejichž kombinace musí být pro každé pole unikátní.

PRIMARY KEY – nastaví pole jako primární klíč, tím je zabráněno výskytu duplicitních hodnot ve sloupci.

```
cislo    INTEGER    NOT NULL   PRIMARY KEY
```

Vysvětlení: pole s názvem `cislo` je pro tabulku nastaveno jako primární klíč a musejí být do něj vždy zadány celočíselné unikátní hodnoty.

REFERENCES – nastavuje cizí klíč jako odkaz do jiné tabulky. Systém pak zabrání odstranění záznamu z tabulky, na nějž se odkazuje pole v jiné tabulce. Nastavení cizích klíčů je prostředek k udržení konzistence databáze.

```
utvar_id      INTEGER    REFERENCES utvar      resp.
utvar_id      INTEGER    REFERENCES utvar (id)
```

Vysvětlení: pole s názvem `utvar_id` je pro tabulku nastaveno jako cizí klíč. Vytváří odkaz na pole primárního klíče v tabulce `utvar`. Mohou být do něj vkládány pouze celočíselné hodnoty, které odpovídají hodnotám pole primárního klíče v tabulce `utvar`.

CHECK – nastaví podmínku, která musí být splněna pro všechny hodnoty vkládané do pole. Způsob zápisu podmínky je podrobně popsán v rámci příkazu SELECT.

```
mesic    INTEGER    CHECK (mesic BETWEEN 1 AND 12)
```

Vysvětlení: do pole s názvem `mesic` mohou být vkládány pouze hodnoty z intervalu 1 až 12.

Omezení tabulky

Omezení tabulky jsou v podstatě shodná jako omezení sloupce, umožňují však založit omezení na více než jednom sloupci. Příklady z předchozího textu bychom mohli napsat také takto:

```
cislo         INTEGER    NOT NULL,
rodneCislo    CHAR(10)   NULL,
utvar_id      INTEGER,
mesic         INTEGER    NULL,
UNIQUE (cislo, rodneCislo),
PRIMARY KEY (cislo),
FOREIGN KEY (utvar_id)   REFERENCES utvar (id),
CHECK (mesic BETWEEN 1 AND 12)
```

Poznámka. Omezení CHECK v Accessu nelze vyzkoušet zápisem SQL, lze nastavit pouze v definici tabulky pomocí grafického rozhraní Accessu.

Příklad 7.1

Příkazem `CREATE TABLE` *vytvořte tabulky* `Zbozi` *a* `TypZbozi`. *Tabulka* `TypZbozi` *je číselník druhů zboží a má pole* `cislo` *, které je primárním klíčem tabulky a pole* `popis` *pro uložení textu popisu druhu zboží. Tabulka* `Zbozi` *bude obsahovat sloupec* `kod` *pro ukládání identifikátoru jednotlivých položek zboží a pro toto pole bude tudíž nastaven primární klíč. Dále má pole* `nazev` *pro popis zboží, pole* `cena` *a* `nakoupeno` *budou sloužit k uložení nákupní ceny a data nákupu, pole* `nabidka` *pro informaci zda se zboží nabízí v nabídkovém katalogu. Pole* `typ` *slouží k určení druhu zboží a představuje odkaz do tabulky druhů zboží s názvem* `TypZbozi` *a je tedy nastaveno jako cizí klíč. Do všech polí kromě pole* `nabidka` *musí být vložena hodnota (nepřipouští se hodnota* `NULL`*). Datové typy (*`INTEGER`, `CHAR`, `REAL`, `DATE` *a* `LOGICAL`*) jsou platné pro databázový systém Access.*

```
CREATE TABLE TypZbozi
(cislo      INTEGER   NOT NULL   UNIQUE   PRIMARY KEY,
 popis      CHAR(20)  NOT NULL
)

CREATE TABLE Zbozi
(kod        INTEGER   NOT NULL   UNIQUE,
 nazev      CHAR(30)  NOT NULL,
 cena       REAL,
 nakoupeno  DATE,
 nabidka    LOGICAL,
 typ        INTEGER   NOT NULL,
CONSTRAINT Zbozi_PK PRIMARY KEY (kod),
CONSTRAINT TypZbozi_FK FOREIGN KEY (typ)
  REFERENCES TypZbozi (cislo)
)
```

7.2.2. Příkaz ALTER TABLE

Umožňuje měnit *strukturu* již vytvořené tabulky. Tedy přidat nebo zrušit sloupec nebo integritní omezení nebo je upravit.

```
ALTER TABLE <nazev_tabulky>
  {[akce
  <definice_sloupce1> [,<definice_sloupce2> [, ...]]]
  }
```

kde

```
<definice_sloupce> = dtto jako u příkazu CREATE TABLE

akce =
  přidání sloupce/omezení:    ADD {COLUMN | CONSTRAINT}
  změna sloupce:              ALTER COLUMN
  zrušení sloupce/omezení:    DROP {COLUMN | CONSTRAINT}
```

Příklad 7.2

Následující příklady ukazují různé varianty příkazu ALTER TABLE.

```
ALTER TABLE Zbozi
ADD COLUMN dodavatel CHAR(20)
```

Vloží do tabulky Zbozi sloupec dodavatel textového typu s délkou 20 znaků.

```
ALTER TABLE Zbozi
ALTER COLUMN dodavatel INTEGER
```

Změní datový typ pole dodavatel v tabulce Zbozi na celočíselný. Pozor! Při změně datového typu pole, jež obsahuje data, může změna vést ke ztrátě dat. Bezztrátová je prakticky pouze změna na textový datový typ dostatečné

délky. Tímto příkazem nelze jednoduše změnit jméno sloupce. Je pouze možné vložit sloupec s novým jménem (ALTER TABLE ADD COLUMN) a původní odstranit (ALTER TABLE DROP COLUMN). Pokud by tabulka již obsahovala data, je pochopitelně nutno před zrušením původního sloupce data přesunout do nového příkazem UPDATE (viz dále).

```
ALTER TABLE Zbozi
ADD CONSTRAINT dodavatel_inique UNIQUE (dodavatel)
```

Vytvoří v tabulce Zbozi nové omezení s názvem dodavatel_inique, které zajistí, aby do pole dodavatel byly vkládány unikátní hodnoty.

```
ALTER TABLE Zbozi
DROP CONSTRAINT dodavatel_inique
```

Zruší v tabulce Zbozi omezení s názvem dodavatel_inique.

```
ALTER TABLE Zbozi
DROP dodavatel
```

Zruší v tabulce Zbozi sloupec s názvem dodavatel.

7.2.3. Příkaz DROP TABLE

Odstraní tabulku a všechny informace o ní (indexy, omezení) z databáze.

```
DROP TABLE <nazev_tabulky>
```

Tento příkaz je velmi jednoduchý a velmi nebezpečný. Je třeba s ním zacházet opatrně, protože *nevratně* odstraní všechny řádky i celou tabulku včetně indexů, které jsou případně definovány nad některými sloupci tabulky. Dotazy, které používají zrušenou tabulku nebudou funkční.

Příklad 7.3

Smažte tabulku `Zbozi` *vytvořenou v příkladu 7.1.*

```
DROP TABLE Zbozi
```

7.2.4. Příkaz SELECT

Příkaz SELECT je nejdůležitějším a nejpoužívanějším příkazem SQL. Nemění data v databázi, používá se pro zjišťování informací z databáze.

```
SELECT [ALL|DISTINCT]<seznam_polozek>
FROM <zdroj_dat>
[WHERE <podminka>]
[GROUP BY <seznam_polozek>]
  [HAVING <podminka>]
[ORDER BY <seznam_polozek> [ASC|DESC]]
```

kde:

SELECT – uvádí výstupní množinu (obsah a pojmenování sloupců). ALL resp. DISTINCT vybírá všechny resp. pouze jedinečné hodnoty uvedeného seznamu položek

FROM – uvádí zdroj dat. Může to být jedna nebo více tabulek či pohledů.

WHERE – omezuje řádky výstupní množiny. Budou vypisovány jen ty řádky, pro které je podmínka splněna.

GROUP BY – umožňuje seskupovat řádky se stejnou hodnotou v uvedeném seznamu položek, takže ve výstupní množině řádků jsou tyto řádky nahrazeny jediným.

HAVING – umožňuje omezit výstupní množinu seskupených řádků. Může být použito pouze pokud mu předchází GROUP BY. V jednom příkaze mohou být použita obě omezení (WHERE i HAVING). Zatímco WHERE omezuje řádky před seskupením, HAVING omezuje seskupené řádky.

ORDER BY – setřídí řádky výstupní množiny postupně podle uvedeného seznamu a to vzestupně (ASC) nebo sestupně (DESC).

Poznámka 1: Pořadí volitelných částí příkazu musí být dodrženo, tak jak je uvedeno v syntaxi příkazu.

Části <seznam_polozek> a <podminka> obsahují výrazy.

Výraz tvoří:

- Identifikátory
  - polí úplný: tabulka.pole, neúplný: pole
  - funkcí nazev_funkce(seznam argumentů)
    - skalární, např. LEFT(text, poč. znaků), SIN(číslo)
    - agregační, např. SUM(číselný výraz)
- Konstanty
  - číselné, např. 5,- 10.8
  - textové, např. “Jan”, “2003”
  - datumové, např. #1.1.2006#
  - logické: 0 (nepravda), 1 (pravda)
- Operátory
  - aritmetické: + , - , * , /
  - relační: = , < , <= , > , >=, <>
  - logické: AND, OR, NOT, (EXISTS, ANY, ALL)
- Klauzule
  - BETWEEN
  - IN
  - LIKE

Poznámka 1: znaky použité pro vymezení textových nebo datumových konstant se mohou v různých DB systémech lišit. Zde uvedené znaky “ a # používá MS Access.

Poznámka 2: skalárních fukcí je celá řada a jelikož se v různých DB systémech mohou poněkud lišit svým názvem, případně argumenty, nebudeme se pokoušet uvádět zde jejich výčet. Základních agregačních funkcí je pouze pět, proto si je uveďme:

```
SUM(výraz), AVG(výraz)
```

spočítá součet, resp. průměr hodnoty všech řádků výstupní množiny uvedeného výrazu, který musí být numerický.

```
MIN(výraz), MAX(výraz)
```

určí minimální, resp. maximální hodnoty ze všech řádků výstupní množiny uvedeného výrazu, který musí být numerický.

```
COUNT(*)
```

spočítá počet řádků výstupní množiny včetně záznamů obsahujících prázdné hodnoty. Uvede-li se místo hvězdičky konkrétní pole tabulky, pak spočítá počet řádků s neprázdnou hodnotou tohoto pole.

Jak je vidět, příkaz SELECT může být velmi jednoduchý, protože povinně obsahuje pouze dvě části – vybíraná data a zdroj dat, což při požadavku zobrazení všech sloupců a řádků jedné tabulky lze zapsat jako:

```
SELECT * FROM <nazev_tabulky>
```

Ale může být i velmi složitý; zvláště uvědomíme-li si, že kromě toho, že většinou děláme výběr určitých řádků a sloupců, třídíme a seskupujeme data, zdrojem dat je více tabulek nebo dotazů, můžeme také jeden příkaz SELECT spojit s jiným příkazem SELECT pomocí množinových operátorů (UNION – sjednocení, INTERSECT – průnik, MINUS – rozdíl) nebo můžeme tvořit vnořené dotazy (poddotazy), kdy hodnoty získané jedním příkazem jsou použity v jiném příkazu. Pro demonstraci si uveďme několik příkladů jednoduchých dotazů na jednu tabulku.

Příklad 7.4

Použijeme tabulku `Zbozi` *vytvořenou v příkladu 7.1. Budeme chtít omezit sloupce i řádky výstupní množiny, kterou vrací základní dotaz a budeme ji chtít seřadit.*

```
SELECT * FROM Zbozi
```

Základní příkaz, který vybere všechny sloupce i řádky tabulky.

```
SELECT nazev, cena, nakoupeno FROM Zbozi
WHERE YEAR(nakoupeno)=2006
ORDER BY nakoupeno DESC, cena
```

Zobrazí pouze název, cenu a datum zakoupení toho zboží, které bylo pořízeno v roce 2006. Seznam bude seřazen nejprve sestupně podle data nákupu a v rámci stejného data vzestupně podle ceny. Funkce `YEAR()` *vrací z argumentu typu datum číslo roku.*

```
SELECT nazev, cena, cena*1.05 AS DPH
FROM Zbozi
```

```
WHERE YEAR(nakoupeno)= YEAR(DATE())
AND MONTH(nakoupeno)=11
ORDER BY nakoupeno DESC, cena
```

Předchozí ukázka je rozšířena o výpis dalšího sloupce pojmenovaného DPH. Primárně se vypisují názvy výstupních sloupců tak, jak jsou definovány ve struktuře tabulky. Pro výstup lze název přejmenovat použitím klíčového slova AS. Vypočítaný sloupec DPH není součástí tabulky, ale vznikl vyhodnocením výrazu a objevuje se pouze ve výstupní tabulce. Dále jsme v tomto příkazu oproti předchozímu omezili řádky tím, že jsme upřesnili, že vypisované zboží bylo nakoupeno v 11. měsíci letošního roku (funkce `DATE()` *vrací systémově nastavené datum).*

Příklad 7.5

V tomto příkladu budeme chtít z tabulky `Zbozi` *získat některé souhrnné informace, což může být například zjištění celkové nebo průměrné ceny zboží, vyhledání nejdražšího nebo nejlevnějšího zboží určitého druhu nebo zjištění počtu položek nakoupených v jednom dni.*

```
SELECT SUM(cena) AS [Cena celkem], typ
FROM Zbozi
WHERE MONTH(nakoupeno) BETWEEN 1 AND 3
AND YEAR(nakoupeno)= YEAR(DATE()-1)
GROUP BY typ
ORDER BY typ
```

Tento příklad vypíše celkovou cenu jednotlivých typů zboží nakoupeného v prvních třech měsících loňského roku. U souhrnných dotazů je třeba si uvědomit, že na výstupu za slovem `SELECT` *mohou být uvedeny pouze agregační funkce a položky, podle nichž se provádí seskupení (uvedené za* `GROUP BY`*). Rovněž řadit není možné podle jiných položek než podle kterých je provedeno seskupení. Dále je v ukázce použita klauzule* `BETWEEN`*, která zjednodušuje zápis podmínky:*

```
MONTH(nakoupeno)>=1 AND MONTH(nakoupeno)<=3
```

```
SELECT COUNT(*) AS [Počet zboží], typ
FROM Zbozi
WHERE MONTH(nakoupeno) BETWEEN 1 AND 3 AND
YEAR(nakoupeno)= YEAR(DATE()-1)
GROUP BY typ
HAVING typ IN (1, 3, 5)
ORDER BY typ
```

V tomto příkladě se podobně jako v předchozím zajímáme o zboží nakoupené v prvním čtvrtletí loňského roku, chceme však zjistit počet nakoupených položek zboží jednotlivýcht typů. Oproti předchozímu příkladu, nás speciálně zajímá pouze druh zboží 1, 3 a 5. Podmínku formulovanou pomocí klauzule IN lze také zapsat následovně:

```
typ = 1 OR typ = 5 OR typ = 7
```

Pro zodpovězení komplexnějších dotazů je přirozeně třeba čerpat data z více než jedné tabulky. Pro práci s více tabulkami se pouze rozšiřuje určení zdroje dat za slovem `FROM`. Ostatní části (podmínka, seskupení, řazení) se použijí stejně jako u dotazu na jednu tabulku. Definice zdroje dat je následující:

```
FROM <tabulka1>
  {INNER | LEFT | RIGHT} JOIN <tabulka2>
  ON <definice_spojeni>
```

kde:

`INNER`, `LEFT`, `RIGHT` vyjadřují typ spojení viz kapitola 6.5.3

`<definice_spojeni>` je:
`<tabulka1.vazebne_pole> = <tabulka2.vazebne_pole>`

Poznámka: vazebná pole se nemusí vždy porovnávat pouze na rovnost, lze použít i ostatní operátory (<, <=, <>, ...), porovnání na rovnost je však nejběžnější. Stejně tak některé DB systémy mohou podporovat i jiné typy spojení (`CROSS`, `FULL`); nejběžnějším typem spojení je však vnitřní spojení (`INNER JOIN`), které vrací jen odpovídající řádky obou tabulek.

Podívejme se na konkrétní příklady, které nám pomohou situaci objasnit. V následujících příkladech se data čerpají z více než jedné tabulky.

Příklad 7.7

Mějme tabulky `Zbozi` *a* `TypZbozi` *vytvořené v příkladu 7.1. U každého zboží si budeme chtít zobrazit kromě jeho názvu i celý popis druhu zboží, který je v tabulce* `TypZbozi`.

```
SELECT Zbozi.nazev, TypZbozi.popis
FROM Zbozi INNER JOIN TypZbozi
ON Zbozi.typ = TypZbozi.cislo
```

Spojení obou tabulek je realizováno prostřednictvím pole `typ` *(cizí klíč) v tabulce* `Zbozi`*, které se odkazuje na pole* `cislo` *v tabulce* `TypZbozi` *(primární klíč). V příkladu si můžeme všimnout tzv. klasifikovaných názvů (např.* `Zbozi.typ`*) které jednoznačně určují z jaké tabulky pole pochází, jelikož názvy polí v různých tabulkách mohou být stejné. Dále si můžeme všimnout, že odkazovaná pole nemusí mít stejný název, musi však mít odpovídající datový typ, to znamená musí ukládat stejné hodnoty.*

```
SELECT Zbozi.nazev, TypZbozi.popis
FROM Zbozi INNER JOIN TypZbozi
ON Zbozi.typ = TypZbozi.cislo
WHERE Zbozi.nakoupeno = #1.11.2006#
ORDER BY TypZbozi.popis, Zbozi.nazev
```

Tato varianta ukazuje, že další části příkazu `SELECT` *se pro spojené tabulky použijí naprosto shodně jako pro jednu tabulku*

Příklad 7.8

Přidejme k tabulkám `Zbozi` *a* `TypZbozi` *ještě tabulky* `Skladnik` *a* `Vydejka`*, které budou mít alespoň pole uvedená níže.*

```
CREATE TABLE Skladnik
(ID         INTEGER   NOT NULL   UNIQUE,
 prijmeni   CHAR(30)  NOT NULL,
 jmeno      CHAR(15),
 CONSTRAINT Skladnik_PK PRIMARY KEY (ID),
)
CREATE TABLE Vydejka
(cislo      INTEGER   NOT NULL   UNIQUE,
 IDskladnik INTEGER   NOT NULL,
 IDzbozi    INTEGER   NOT NULL,
 vydano     DATE,
 CONSTRAINT Vydejka_PK PRIMARY KEY (cislo),
 CONSTRAINT Skladnik_FK FOREIGN KEY (IDskladnik)
   REFERENCES Skladnik (ID)
 CONSTRAINT Zbozi_FK FOREIGN KEY (IDzbozi)
   REFERENCES Zbozi (kod)
)
```

Budeme-li požadovat při výpisu výdejky uvést plné jméno skladníka a zboží, musíme připojit tabulky, které tyto údaje obsahuj, jelikož v tabulce `Vydejka` *jsou pouze pole představující identifikátory skladníka a zboží..*

```
SELECT Zbozi.nazev, Vydejka.vydano, Skladnik.prijmeni
FROM Zbozi
INNER JOIN Vydejka ON Zbozi.kod = Vydejka.IDzbozi
INNER JOIN Skladnik ON Skladnik.ID = Vydejka.IDskladnik
```

Vidíme, že potřebujeme-li spojit více tabulek, stačí jednoduše přidat další spojení a vazebnou podmínku. Takže v případě, že chceme vidět ještě popis druhu zboží, bude příkaz vypadat následovně:

```
SELECT TypZbozi.popis Zbozi.nazev, Vydejka.vydano,
       Skladnik.prijmeni, Skladnik.jmeno
FROM TypZbozi
INNER JOIN Zbozi ON TypZbozi.cislo = Zbozi.typ
INNER JOIN Vydejka ON Zbozi.typ = Vydejka.IDzbozi
INNER JOIN Skladnik ON Vydejka.IDskladnik = Skladnik.ID
```

Upozornění
Pokud si tyto dva příkazy zkoušíte provést v Accessu, budou odmítnuty. Acces vyžaduje jasné vymezení hierarchie spojovaných tabulek pomocí závorek, což v případě spojení více jak tří tabulek přestává být přehledné a proto zde tento zápis neuvádíme. Pro ověření funkčnosti předchozích dvou příkladů v Accessu můžete použít jinou variantu definice spojení tabulek pomocí klauzule WHERE:

```
SELECT Zbozi.nazev, Vydejka.vydano, Skladnik.prijmeni
FROM Zbozi, Vydejka, Skladnik
WHERE Zbozi.kod = Vydejka.IDzbozi AND
      Skladnik.ID = Vydejka.IDskladnik
```

a

```
SELECT TypZbozi.popis Zbozi.nazev, Vydejka.vydano,
       Skladnik.prijmeni, Skladnik.jmeno
FROM TypZbozi, Zbozi, Vydejka, Skladnik
WHERE TypZbozi.cislo = Zbozi.typ AND
      Zbozi.typ = Vydejka.IDzbozi AND
      Vydejka.IDskladnik = Skladnik.ID
```

7.2.5. Příkaz INSERT

Vloží do tabulky jeden nebo více řádků.

```
INSERT INTO <nazev_tabulky>
[(<nazev_sloupce1> [,<nazev_sloupce2> [, ...]])]
{VALUES (<vyraz1> [,<vyraz2>[, ...]]) | <prikaz_select> }
```

Příklad 7.9

Mějme tabulku `Zbozi` *definovanou jako v příkladě 7.1, má tedy pole:*

```
(kod       INTEGER   NOT NULL   UNIQUE,
nazev      CHAR(30)  NOT NULL,
cena       REAL,
nakoupeno  DATE,
nabidka    LOGICAL,
typ        INTEGER   NOT NULL,
```

Následující příklady ukazují různé varianty příkazu `INSERT`*, jimiž lze do této tabulky přidat jeden nebo více řádků.*

Vloží do tabulky Zbozi nový řádek s hodnotami definovanými za slovem `VALUE`

```
INSERT INTO Zbozi
VALUES(2,”jogurt”,7.5,DATE(),1,1)
```

Tuto variantu s vynechání seznamu sloupců můžeme použít, pokud jsme si jisti pořadím sloupců definovaných ve struktuře tabulky a vkládáme hodnoty všech polí.

Kdybychom nechtěli vkládat všechna pole (vynechat lze pouze ta, pro která není vytvořeno omezení `NOT NULL`*) nebo pokud si nejsme jisti pořadím sloupců, měli bychom sloupce explicitně vyjmenovat.*

```
INSERT INTO Zbozi
(kod, nazev, cena, typ)
VALUES(2,"jogurt",20.5,1)
```

Výše uvedenými příklady vložíme do tabulky vždy pouze jeden řádek. Pokud chceme najednou vložit celý blok dat (více řádků), použijeme vnořený příkaz `SELECT`*, který vrací víceřádkovou sadu hodnot. Seznam výstupních polí tohoto příkazu musí odpovídat co do počtu a datového typu (nikoli jména) seznamu polí příkazu* `INSERT`.

```
INSERT INTO Zbozi (kod, nazev, cena, typ )
SELECT ID, popis, cena, druh
FROM Zbozi1
WHERE (((Zbozi1.druh)=2))
```

Vybere z tabulky `Zbozi1` *všechny záznamy s druhem zboží 2 a jejich identifikátor (*`ID`*), popis, cenu a druh zboží vloží do tabulky* `Zbozi`.

Pozor! Pokud je v tabulce `Zbozi`*, tak jako v našem případě, definováno omezení sloupce kod (*`UNIQUE`*), může systém odmítnout vložit řádek, jestliže hodnota pole* `ID` *vkládaného řádku již v tabulce* `Zbozi` *existuje.*

7.2.6. Příkaz UPDATE

Umožňuje měnit *hodnotu* polí tabulky. Změna se může týkat jednoho nebo více polí v celé tabulce nebo pouze ve vybraných řádcích.

```
UPDATE <nazev_tabulky>
  SET <nazev_sloupce1>=<vyraz>
     [,<nazev_sloupce2>=<vyraz> [, ...]]
  [WHERE <podminka>]
```

Podmínka a výraz se tvoří stejně, jak bylo popsáno v kapitole u příkazu `SELECT`. Výrazem může být i vnořený příkaz `SELECT`, který vrací odpovídající hodnotu.

Příklad 7.10

Použitím příkazu `UPDATE` *aktualizujte data v tabulce* Zbozi.

a) Pro celou tabulku nastavte hodnotu pole `nabidka` *na 1 (ano).*

```
UPDATE Zbozi SET nabidka=1
```

b) U zboží typu 2 navyšte cenu o 10%.

```
UPDATE Zbozi SET cena=1.1*cena
WHERE (typ=2)
```

c) U zboží nakoupeného 1.11.2006 změňte pole `typ` *na hodnotu 3 a cenu nastavte na nulu.*

```
UPDATE Zbozi SET typ = 3, Zbozi.cena = 0
WHERE (nakoupeno = #1.11.2006#)
```

7.2.7. Příkaz DELETE

Maže *celé* řádky tabulky. Pokud bychom chtěli smazat jen některé hodnoty v řádku, je třeba použít příkaz `UPDATE`.

```
DELETE FROM <nazev_tabulky>
[WHERE <podminka>]
```

Podmínka se tvoří stejně, jak bylo popsáno v kapitole u příkazu `SELECT`. Při provádění příkazu `DELETE` je třeba mít na paměti, že pokud jsou na tabulku vytvořeny odkazy z jiných tabulek (jsou v nich definovány omezení cizí klíče) a my chceme odstranit řádek, na nějž se odkazuje řádek v jiné tabulce, databázový systém může odmítnout příkaz `DELETE` vykonat neboť by došlo k porušení integrity. Smazání požadovaného řádku by mělo předcházet odstranění řádků v odkazované tabulce. Systém může být nastaven tak, že tuto akci provede sám. Situaci popisuje příklad.

Příklad 7.11

Zrušte v tabulce `TypZbozi`, *kterou jsme vytvořili v příkladu xx, druh zboží číslo 10.*

Jelikož jsme pro tabulku `Zbozi` *vytvořili omezení cizího klíče, který se odkazuje na pole číslo v tabulce* `TypZbozi`, *musíme nejprve v tabulce* `Zbozi` *buď všechny řádky, které se na druh zboží číslo 10 odkazují odstranit nebo je přeřadit do jiného druhu (např. 11):*

```
DELETE FROM Zbozi WHERE typ=10
```

případně

```
UPDATE Zbozi SET typ=11 WHERE typ=10
```

Teprve potom můžeme z tabulky `TypZbozi` *typ zboží číslo 10 odstranit:*

```
DELETE FROM TypZbozi WHERE cislo=10
```

7.2.8. Příkaz CREATE VIEW

Vytvoří virtuální tabulku (pohled, dotaz) založenou na jedné nebo více databázových tabulkách nebo pohledech. Ukládá a pojmenovává se definice dotazu, data se sestavují až při odkazu na pohled.

Příklad 7.12

Vytvořte pohled na tabulku `Zbozi` *s názvem* `Zbozi2006`*, který zobrazí jen zboží, které bylo pořízeno v roce 2006 seřazené sestupně podle data nákupu.*

```
CREATE VIEW Zbozi2006
AS
SELECT * FROM Zbozi
WHERE YEAR(nakoupeno)=2006
ORDER BY nakoupeno
```

Poznámka. Příkaz `CREATE VIEW` v Accessu nelze vyzkoušet zápisem SQL, lze pouze vytvořit nový dotaz pomocí grafického rozhraní Accessu.

7.2.9. Příkaz CREATE INDEX

Vytvoří index nad jedním nebo více poli tabulky.

```
CREATE [UNIQUE] INDEX <nazev_indexu>
  ON <nazev_tabulky>
  (<sloupec1> [ASC|DESC] [,<sloupec2> [ASC|DESC][, ...]])
```

Index obsahuje seznam odkazů na řádky vybraného sloupce ve zvoleném pořadí (vzestupné – ASC, sestupné – DES). Vytvoření indexu nad poli podle kterých se často třídí značně urychluje zpracování dotazu.

Příklad 7.13

V tabulce `Zbozi` *vytvořte indexy s názvem* `ZboziNazev` *a* `ZboziCena`*, které seřadí pole* `nazev` *vzestupně a pole* `cena` *sestupně.*

```
CREATE INDEX ZboziNazev
ON Zbozi (nazev)

CREATE INDEX ZboziCena
ON Zbozi (cena DESC)
```

8. Autotest

Otestujte si, zda jste probranou látku pochopili na následujícím testu. Pokud se vám nepodařilo odpovědět správně, vraťte se zpět a prostudujte příslušnou kapitolu. V otázkách je vždy správná pouze jedna odpověď.

1. *Co rozumíme pod pojmem „struktura databázové tabulky“?*
   - *(a) Hodnoty prvků dat jednoho sledovaného objektu.*
   - *(b) Množinu vlastností, které popisují sledovaný objekt.*
   - *(c) Takové údaje, podle nichž lze určit hodnoty všech položek téhož záznamu.*
   - *(d) Množinu hodnot souvisejících datových prvků.*
2. *O databázi můžeme říct, že NENÍ konzistentní, když*
   - *(a) některé hodnoty atributů nejsou pravdivé (data neodpovídají vlastnostem objektů reálného světa).*
   - *(b) neobsahuje primární a cizí klíče, které by umožnily vytvořit vazbu mezi tabulkami.*
   - *(c) některé atributy objektů datového modelu jsou uloženy vícenásobně (na různých místech databáze).*
   - *(d) hodnoty atributů objektů datového modelu, které jsou uloženy vícenásobně (na různých místech databáze), jsou různé.*
3. *Vytvoření datového modelu na konceptuální úrovni znamená, že*
   - *(a) se řeší způsob konkrétního (fyzického) ukládání potřebných dat v souborech.*
   - *(b) se popisuje logická struktura dat a vztahy mezi nimi.*
   - *(c) se definuje část databáze pro konkrétního uživatele.*
   - *(d) jsou předběžně vytipovány objekty pro návrh řešení v tabulce.*
4. *Kterou ze služeb NEPOSKYTUJE jádro SŘBD (tzv. databázový stroj)?*
   - *(a) Definici dat*
   - *(b) Manipulaci s daty.*
   - *(c) Zobrazení dat.*
   - *(d) Údržbu dat.*
5. *Co rozumíme v databázové terminologii pod pojmem „klíčový atribut“?*
   - *(a) Takový záznam v databázi, kterým lze charakterizovat celou databázi.*
   - *(b) Takovou vlastnost datového prvku, podle jejíž hodnoty lze jednoznačně identifikovat záznam.*
   - *(c) Takový atribut, který má výlučné postavení mezi ostatními, a proto pro jednu databázi může vždy existovat pouze jeden klíčový atribut.*
   - *(d) Takové číslo, které určuje pozici záznamu v souboru.*

6. *Situaci, kdy se nám podaří vložit do databáze datum 30.února, označujeme jako*
   (a) *chybnou transakci.*
   (b) *problém redundance.*
   (c) *porušení konzistence.*
   (d) *porušení integrity.*
7. *Který datový model je v oblasti DB systémů v současnosti nejrozšířenější?*
   (a) *Klient/server*
   (b) *Síťový*
   (c) *Relační*
   (d) *Objektový*
8. *Který jazyk se v prostředí DB systémů nejčastěji používá k získávání informací z databáze?*
   (a) *Programovací jazyk COBOL*
   (b) *Jazyk typu DCL (Data Control Language).*
   (c) *Jazyk 4GL (4 Generation Language).*
   (d) *Dotazovací jazyk SQL (Structured Query Language).*
9. *K čemu slouží v databázi primární klíč?*
   (a) *Primární klíč slouží k jednoznačné identifikaci databáze.*
   (b) *Primární klíč slouží k řazení dat podle požadovaných kritérií a rychlému přístupu k nim.*
   (c) *Primární klíč slouží k jednoznačnému určení pozice záznamu v souboru.*
   (d) *Primární klíč slouží k jednoznačné identifikaci záznamu v tabulce.*
10. *Co NEPLATÍ o „cizím klíči“ v tabulce relační databáze?*
    (a) *Je to primární klíč jiné tabulky.*
    (b) *Cizí klíč může být v tabulce jen jeden.*
    (c) *Je to prostředek pro určení vazeb mezi tabulkami.*
    (d) *Cizích klíčů může být v tabulce několik.*
11. *Jaký vztah mezi entitami NELZE v relační databázi realizovat pouze dvěma tabulkami?*
    (a) *Mnohý k mnoha (m : n).*
    (b) *Jedna k jedné (1 : 1).*
    (c) *Jedna k více (1 : n).*
    (d) *Žádný.*

12. *Co se v relační databázi označuje termínem „normalizace“?*
    (a) *Proces výběru objektů popisujících předmětnou oblast pro vytvoření DB systému.*
    (b) *Proces tvorby návrhu datového modelu.*
    (c) *Uplatnění určitých pravidel (norem) v procesu návrhu struktur tabulek.*
    (d) *Sjednocení použitých primárních a cizích klíčů v tabulkách.*
13. *Jestliže prohlásíme, že data v databázi jsou redundantní, znamená to, že*
    (a) *databáze neobsahuje správná, realitě odpovídající data.*
    (b) *databáze je zatím pouze definovaná, ale ještě nenaplněná daty.*
    (c) *pro data zatím nebyly definovány vzájemné vztahy.*
    (d) *databáze obsahuje stejná data uložená vícenásobně.*
14. *Jestliže SŘBD umožňuje transakční zpracování, pak*
    (a) *umožňuje uživatelské požadavky na výběr dat (transakce) realizovat pomocí dotazovacího jazyka SQL.*
    (b) *umožňuje ukládat data pomocí transakčního protokolu.*
    (c) *zajišťuje uložení dat bez redundancí.*
    (d) *umožňuje operace, při nichž hrozí porušení konzistence (např.při aktualizaci) provádět bezpečně jako celek.*
15. *Přidat atribut do již definované tabulky relační databáze*
    (a) *lze realizovat přidáním sloupce do tabulky.*
    (b) *lze realizovat přidáním řádku do tabulky.*
    (c) *lze realizovat přidáním řádku a sloupce do tabulky.*
    (d) *nelze realizovat.*
16. *Specifické procedurální jazyky označované jako 4GL*
    (a) *mohou být použity pro tvorbu jakékoli aplikace.*
    (b) *nemohou být použity pro tvorbu databázové aplikace.*
    (c) *mohou být použity pro tvorbu databázové aplikace pouze v určitém SŘBD.*
    (d) *mohou být použity pro tvorbu databázové aplikace v jakémkoli SŘBD.*
17. *Co NEPLATÍ o indexech v databázích?*
    (a) *Slouží k urychlení vyhledávání v databázích.*
    (b) *Jsou to virtuální tabulky, které vznikají na základě uložených dat v paměti počítače.*
    (c) *Pro data uložená v databázi lze vytvořit více indexů.*
    (d) *Jsou to pomocné informace, které se ukládají v databázi.*

18. *K čemu slouží v prostředí databází E-R diagram?*
    (a) *Ke grafickému popisu dané předmětné oblasti pomocí entit a vztahů.*
    (b) *Ke grafickému znázornění struktury tabulek.*
    (c) *Ke grafickému popisu toku dat.*
    (d) *Ke grafickému znázornění datového modelu.*
19. *Co představuje v databázi hodnota NULL?*
    (a) *Číselná data s hodnotou nula.*
    (b) *Textová data představující prázdný řetězec.*
    (c) *Atributy, které musí zůstat prázdné (nezadané).*
    (d) *Neznámé nebo nezadané hodnoty.*
20. *Ve vztahu k relačním operacím v DB systému selekcí rozumíme:*
    (a) *Operaci relační algebry, pomocí níž se omezí sloupce tabulky.*
    (b) *Operaci relační algebry, pomocí níž se omezí řádky tabulky.*
    (c) *Operace, které vyberou vhodné objekty pro popis datového modelu.*
    (d) *Operaci (výběr tabulek), která definuje relace mezi tabulkami.*

Závěr

Shrnutí

Pokusili jsme se objasnit problematiku spojenou s ukládáním dat a získáváním informací z databázových systémů. Po prostudování tohoto textu by se měl čtenář orientovat v základních pojmech, které souvisí s teorií a návrhem databází. Měl by mít dostatečný základ pro rozšiřování teoretických znalostí, tak i poznatky prakticky využitelné při práci s konkrétním databázovým systémem.

Studijní prameny

Seznam základní literatury

[1] Kříž, J.:Databázové systémy, CERM, 2005

[2] Pokorný, J.:Databázová abeceda, Science, 1998

[3] Pokorný, J.: Dotazovací jazyky, Science, 1994

[4] Pokorný, J.: Databázové systémy a jejich použití v informačních systémech, Academia, 2000

Seznam doplňkové studijní literatury

[5] Riordan M. R.: Vytváříme databázové aplikace, Computer Press 2000

[6] Král, J.: Informační systémy, Science, 1998

Odkazy na elektronické studijní zdroje a prameny

[7] http://www.fit.vutbr.cz/study/courses/DSI/public/ - Heading2

[8] http://www.dbsvet.cz/

[9] http://www.dbs-intro.com/

Klíč k autotestu

Správné odpovědi k testovým otázkám:

1b, 2d, 3b, 4c, 5b, 6d, 7c, 8d, 9d, 10b, 11a, 12c, 13d, 14d, 15a, 16c, 17b, 18a, 19d, 20b