Otázka 17 – A7B36DBS

Zadání 1
Slovníček pojmů 1
Datové modely ..... **Chyba! Záložka není definována.**
Konceptuální datový model 3
Databázové modely 6
Fyzický pohled na data 8
Relační datový model 12
Relace 30
Relační algebra 31
Normální formy 40

Zadání

Datové modely. Konceptuální datový model, Databázové modely, Fyzický pohled na data. Relační model, Relace, Relační algebra, Normální formy. (A7B36DBS)

Slovníček pojmů

- **entita** - významný prvek ve zkoumané oblasti. Entitou může být zaměstnanec, oddělení, výplata apod. Entity se v diagramu vyznačují jako obdélníky s vepsaným názvem entit
- **databáze (data)** je logicky uspořádaná (integrovaná) kolekce navzájem souvisejících dat**,** je sebevysvětlující, protože data jsou uchovávána společně s popisy, známými jako metadata (také schéma databáze)
- **relační model** je nejrozšířenějším způsobem uložení dat v databázi. Jedná se způsob uložení v logickém smyslu
- **relace** je množina vztahů mezi jednotlivými prvky domén
- **doména** je množina datových hodnot stejného typu. Tyto hodnoty popisují nějakou vlastnost objektu
- **atribut** je pojmenování pro každé užití hodnoty z domény v relaci
- **záhlaví relace** obsahuje jméno relace a jména atributů v relaci. Je v čase neměnné
- **tělo relace** obsahuje v čase proměnnou množinu n-tic hodnot, jejichž pořadí je dáno záhlavím relace
- **stupeň relace** je počet atributů relace
- **kardinalita relace** je počet řádků relace
- **dotazovací jazyk** umožňuje ovládat databázi prostřednictvím příkazů – dotazů, za pomoci vyhledávacích operátorů. Příkazy je možno rozdělit na příkazy pro manipulaci s daty, příkazy pro definici dat a příkazy pro řízení dat
- **index** (někdy též označovaný jako klíč - KEY) je databázová konstrukce, sloužící ke zrychlení vyhledávacích a dotazovacích procesů v databázi, definování unikátní hodnoty sloupce tabulky nebo optimalizaci fulltextového vyhledávání
- **predikátová logika** je formální odvozovací systém používaný k popisu matematických teorií a vět. Je rozšířením výrokové logiky (ta nedokáže vyjádřit některá složitější tvrzení o matematických strukturách). Do této logiky přidává kvantifikátory a vztah predikát-individuum. Individuum je prvek z nějaké množiny (univerza) a predikát je relace na této množině

- **referenční integrita** je nástroj databázového stroje, který pomáhá udržovat vztahy v relačně propojených databázových tabulkách. Referenční integrita se definuje cizím klíčem, a to pro dvojici tabulek, nebo nad jednou tabulkou, která obsahuje na sobě závislá data (například stromové struktury). Tabulka, v niž je pravidlo uvedeno, se nazývá podřízená tabulka (používá se také anglický termín slave). Tabulka, jejíž jméno je v omezení uvedeno je nadřízená tabulka (master). Pravidlo referenční integrity vyžaduje, aby pro každý záznam v podřízené tabulce, pokud tento obsahuje data vztahující se k nadřízené tabulce, odpovídající záznam v nadřízené tabulce existoval. To znamená, že každý záznam v podřízené tabulce musí v cizím klíči obsahovat hodnoty odpovídající primárními klíči nějakého záznamu v nadřízené tabulce, nebo NULL. Pokud je více atributů, které splňují pravidlo pro primární klíč, jeden zvolíme jako primární. Ostatní jsou alternativní klíče
- **primární klíč** je sloupec, který jednoznačně určuje řádky v tabulce. Pokud je třeba použít více sloupců pro jednoznačné určení řádků, potom hovoříme o tzv. složeném klíči.
- **tranzitivní relace** V logice a matematice se binární relace R na množině X nazývá tranzitivní, pokud pro každé α, β a γ z X platí, že pokud α je v relaci s β a β je v relaci s γ, je i α v relaci s γ

Formálně zapsáno:

$$\forall \alpha, \beta, \gamma \in X,\ \alpha R \beta \wedge \beta R \gamma \ \Rightarrow \alpha R \gamma$$

Například, „je větší než“ a „je rovno“ jsou tranzitivní relace: pokud a = b a b = c, platí i a = c. Na druhou stranu, „je matkou“ není tranzitivní relace, protože když Alice je matkou Břetislavy a Břetislava je matkou Cecílie, není Alice matkou Cecílie.

Datové modely

Definice (schéma, model)
Typicky pro každou databázi existuje strukturální popis druhu dat v ní udržovaných, ten nazýváme schéma. Schéma popisuje objekty reprezentované v databázi a vztahy mezi nimi. Je několik možných způsobu organizace schémat (modelování databázové struktury), známých jako modely. V modelu jde nejen o způsob strukturování dat, definuje se také sada operací nad daty proveditelná. Relační model například definuje operace jako „select“ nebo „join“. I když tyto operace se nemusejí přímo vyskytovat v dotazovacím jazyce, tvoří základ, na kterém je jazyk postaven.
Poznámka:
Většina databázových systému je založena na jednom konkrétním modelu, ale čím dál častější je podpora více přístupů. Pro každý logický model existuje více fyzických přístupů implementace a většina systému dovolí uživateli nějakou úroveň jejich kontroly a úprav, protože toto má velký vliv na výkon systému. Příkladem nechť jsou indexy, provozované nad relačním modelem.

„Plochý“ model
Toto sice nevyhovuje úplně definici modelu, přesto se jako triviální případ uvádí. Představuje jedinou dvoudimensionální tabulku, kde data v jednom sloupci jsou považována za popis stejné vlastnosti (takže mají podobné hodnoty) a data v jednom řádku se uvažují jako popis jediného objektu.

Relační model
Relační model je založen na predikátové logice a teorii množin. Většina fyzicky implementovaných databázových systému ve skutečnosti používá jen aproximaci matematicky definovaného relačního modelu. Jeho základem jsou relace (dvoudimensionální tabulky), atributy (jejich pojmenované sloupce) a domény (množiny hodnot, které se ve sloupcích mohou objevit). Hlavní datovou strukturou je tabulka, kde se nachází informace o nějaké konkrétní třídě entit. Každá entita té třídy je potom reprezentována řádkem v tabulce – n-ticí atributu. Všechny relace (tj. tabulky) musí splňovat

základní pravidla – poradí sloupců nesmí hrát roli, v tabulce se nesmí vyskytovat identické řádky a každý řádek musí obsahovat jen jednu hodnotu pro každý svůj atribut. Relační databáze obsahuje více tabulek, mezi kterými lze popisovat vztahy (všech různých kardinalit, tj. 1 : 1, 1 : n apod.). Vztahy vznikají i implicitně např. uložením stejné hodnoty jednoho atributu do dvou řádku v tabulce. K tabulkám lze přidat informaci o tom, která podmnožina atributu funguje jako klíč, tj. unikátně identifikuje každý řádek, některý z klíčů muže být označen jako primární. Některé klíče mohou mít nějaký vztah k vnějšímu světu, jiné jsou jen pro vnitrní potřeby schématu databáze (generovaná ID).

Hierarchický model

V hierarchickém modelu jsou data organizována do stromové struktury – každý uzel má odkaz na nadřízený (k popisu hierarchie) a setříděné pole záznamu na stejné úrovni. Tyto struktury byly používány ve starých mainframeových databázích, nyní je můžeme vidět např. ve struktuře XML dokumentu. Dovolují vztahy 1 : N mezi dvěma druhy dat, což je velice efektivní k popisu různých reálných vztahu (obsahy, řazení odstavců textu, tříděné informace). Nevýhodou je ale nutnost znát celou cestu k záznamu ve struktuře a neschopnost systému reprezentovat redundance v datech (strom nemá cykly).

Síťový model

Sítový model organizuje data pomocí dvou hlavních prvku, záznamu a množin. Záznamy obsahují pole dat, množiny definují vztahy 1 : N mezi záznamy (jeden vlastník, mnoho prvku). Záznam může být vlastníkem i prvkem v několika různých množinách. Jde vlastně o variantu hierarchického modelu, protože sítový model je také založen na konceptu více struktur nižší úrovně závislých na strukturách úrovně vyšší. Už ale umožňuje reprezentovat i redundantní data. Operace nad tímto modelem probíhají „navigačním“ stylem: program si uchovává svoji současnou pozici mezi záznamy a postupuje podle závislostí, ve kterých se daný záznam nachází. Záznamy mohou být i vyhledávány podle klíče.

Fyzicky jsou většinou množiny – vztahy – reprezentovány přímo ukazateli na umístění dat na disku, což zajišťuje vysoký výkon při vyhledávání, ale zvyšuje náklady na reorganizace. Smysl sítové navigace mezi objekty se používá i v objektových modelech.

Objektový model

Objektový model je aplikací přístupu známých z objektově-orientovaného programování. Je založen na sbližování programové aplikace a databáze, hlavně ve smyslu použití datových typu (objektu) definovaných na jednom místě; ty zpřístupňuje k použití v nějakém běžném programovacím jazyce. Odstraní se tak nutnost zbytečných konverzí dat. Přináší do databází také věci jako zapouzdření nebo polymorfismus. Problémem objektových modelu je neexistence standardu (nebo spíš produktu, které by je implementovaly).

Kombinací objektového a relačního přístupu vznikají objektově-relační databáze – relační databáze, dovolující uživateli definovat vlastní datové typy a operace na nich. Obsahují pak hybrid mezi procedurálním a dotazovacím programovacím jazykem.

Konceptuální datový model

V tomto modelu se snažíme popsat předmětnou oblast pomocí všech entit, které se v ní vyskytují, a všech vztahů mezi těmito entitami. V žádném případě však nebereme v úvahu pozdější způsob implementace a do jisté míry ani pozdější omezení technologického charakteru. Tím, že neuvažujeme o pozdějším způsobu implementace v konkrétním databázovém systému, můžeme věnovat veškerou energii na pochopení vlastního problému.

Nakonec získáme i obecně platný popis dané oblasti, který můžeme použít pro implementaci v odlišných databázových systémech bez nutnosti opětovné analýzy.

Cíle konceptuálního modelu:

- vytvořit obraz reality ve formalizované podobě nezávislý na pozdějším způsobu implementace
- formalizovat požadavky uživatelů a dát návrhářům snadno pochopitelný prostředek pro komunikaci s uživateli, kterému budou i uživatelé rozumět
- vytvořit podklad pro návrh datové základny

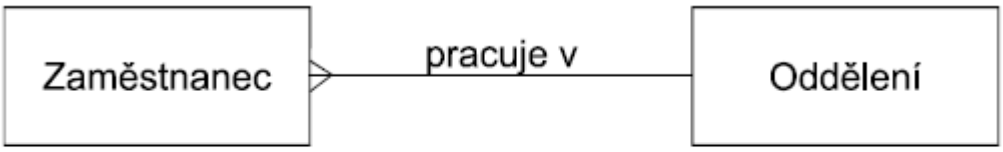
E-R diagramy, ERA diagramy

K formálnímu popisu reality slouží, tzv. E-R diagramy z anglického Entity-Relationship (do češtiny překládané jako diagramy entit a vztahů) nebo ERA diagramy z anglického Entity- Relationship-Attribute. V ERA diagramech jsou navíc u každé entity uvedeny i její atributy.

Pro kreslení obou typů diagramů existuje velké množství notací, které se liší množstvím značek vyjadřující vlastnosti popisované oblasti. Z praktických zkušeností vyplývá, že ani nejvíce graficky bohaté notace nedokážou popsat všechny situace z reálného světa a je nutné doplnit každý diagram slovním popisem. Velké množství značek používaných v diagramech navíc činí tyto diagramy nepřehlednými a ztěžuje jejich pochopení.

Prvky v datových modelech

- **Entita** Významný prvek ve zkoumané oblasti. Entitou může být zaměstnanec, oddělení, výplata apod. Entity se v diagramu vyznačují jako obdélníky s vepsaným názvem entit.
- **Atribut** Vlastnost entity podstatná z hlediska zkoumané oblasti. Atributem entity **Zaměstnanec** bude jeho jméno, výše platu apod. Atributy nemusíme v diagramu vyznačovat. Stačí, budou-li uvedeny v textovém komentáři k tomuto diagramu. (V přednáškách z DB1 se atributy značily hůlkou s kolečkem na konci a popiskem).
- **Vztah** Libovolný vztah, ve kterém mohou být dvě (nebo více) entit. Věta „Zaměstnanec pracuje v oddělení“ je vyjádřením vztahu **pracuje v** mezi entitami **Zaměstnanec** a **Oddělení**. Vztah je vhodné pojmenovat, protože mezi dvěma entitami může existovat více různých vztahů. Vztah je v diagramu vyznačen jako čára, která spojuje entity vystupující v tomto vztahu.



E-R diagram

Kardinalita vztahu

„Vidlička“ na straně zaměstnance ve vztahu s oddělením vyjadřuje tzv. kardinalitu vztahu zaměstnance s oddělením. Pod kardinalitou rozumíme počet výskytů obou entit, které se vztahu účastní. Víme, že v jednom oddělení pracuje obvykle více zaměstnanců. Naproti tomu jeden zaměstnanec pracuje v jeden okamžik pouze v jednom oddělení. Jedná se o příklad vztahu **n:1** (**n** zaměstnanců pracuje v jednom oddělení) nebo opačně **1:n** (v jednom oddělení pracuje **n** zaměstnanců).

Typy kardinalit vztahů:

- **1:1**

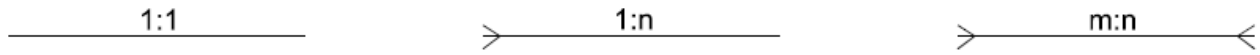
Vztah, ve kterém na obou stranách vystupuje pouze jeden objekt dané entity. Tyto vztahy se v realitě vyskytují pouze zřídka a jejich existence v diagramu bývá někdy způsobena chybou v popisu reality. Příkladem vztahu 1:1 může být vztah **manželé** mezi entitou **Muž** a entitou **Žena** (v případě monogamní společnosti) nebo vztah **třídní učitel** mezi entitami **Učitel** a **Třída.**

- **1:n**

Na jedné straně je jediný objekt, který je ve vztahu s jedním nebo více objekty na straně druhé. Jedná se o typ vztahu, který se vyskytuje velmi často. Kromě již uvedeného vztahu zaměstnance a oddělení to je například vztah **nadřízený – podřízený**, **zaměstnanec – výplata**, ale také **třída – žák**.

m:n

Specifickým typem vztahu jsou vztahy, ve kterých vystupuje více objektů na obou stranách. Ve vztahu **zaměstnanec – úkol** může více zaměstnanců řešit jeden úkol a zároveň může jeden zaměstnanec řešit více úkolů



Notace kardinality vztahu

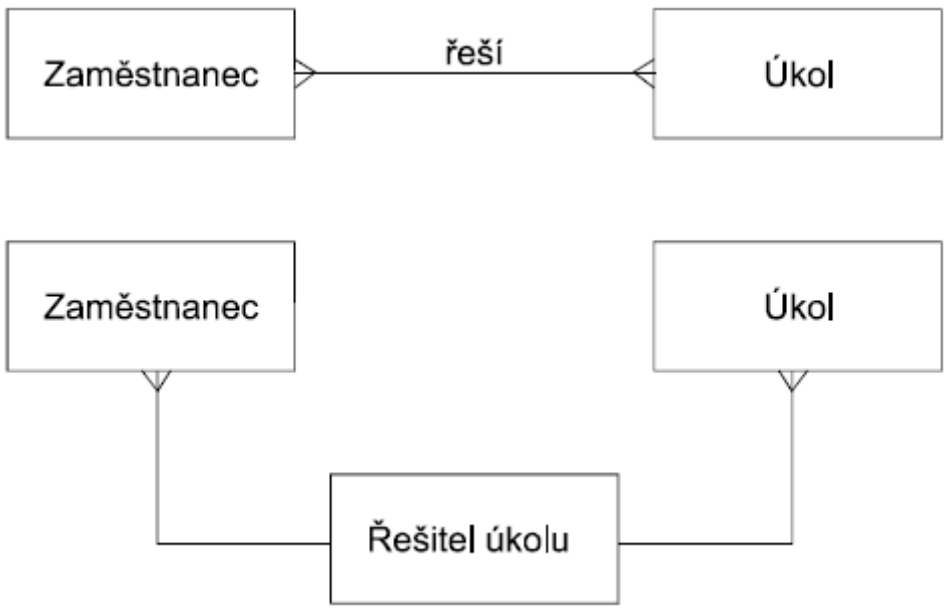
Povinnost výskytu

Kromě kardinality vztahu můžeme ještě rozlišovat povinnost a volitelnost jeho existence. Je nutné, aby každý zaměstnanec byl zařazen do určitého oddělení? Nebo může existovat zaměstnanec, který v současné době nepatří do žádného oddělení? Musí mít každý muž manželku a žena manžela? Povinnost výskytu se může značit různě. Obvykle se používá prázdných a plných značek (obvykle šipek), např. prázdné kolečko (vyjadřuje volitelnost na straně entity, která nemusí existovat). Značení se liší v jednotlivých CASE nástrojích.

Dekompozice vztahů m:n

Vztah **m:n** je z hlediska další práce velmi komplexní a je nutné pokusit se o jeho zjednodušení. Neděláme tak pouze kvůli jeho implementaci na další úrovni návrhu, ale i pro případ, že v tomto vztahu je „schována“ další entita, která zatím naší pozornost unikla.

Součástí této entity mohou být i atributy, o kterých jsme tušili, že existují, ale nevěděli jsme, ke které entitě je přiřadit. Dekompozicí vztahu **m:n** rozumíme vytvoření nové, tzv. vazební entity, která bude mít vztahy typu **n:1** na obě původní entity vztahu **m:n**. Nově vzniklá entita **Řešitel úkolu** vyjadřuje fakt, že zaměstnanec může být najednou řešitelem více úkolů najednou a jeden úkol může být řešen najednou více řešiteli.



Součástí této entity mohou být atributy, které nelze přiřadit žádné z entit **Zaměstnanec** a **Úkol**. Příkladem je atribut popisující hodnocení zaměstnance za odvedenou práci na daném úkolu. Protože zaměstnanec může pracovat na více úkolech a být hodnocen za každý jinak, nemůže být tento atribut

umístěn do entity **Zaměstnanec**. Zároveň nemůže být umístěn ani do entity Úkol, protože na úkolu může pracovat více zaměstnanců. Jediným správným umístěním atributu **Hodnocení** je entita **řešitel úkolu**, protože se vztahuje vždy k dvojici **{zaměstnanec; úkol}**.

Databázové modely

Síťový databázový model

Síťová databáze byla vyvinuta hlavně jako pokus o vyřešení problémů hierarchické databáze. Struktura síťové databáze je vyjádřena v pojmech *uzlů* (někdy také označovaných jako záznamy) a *množinových struktur*.

Na obrázku můžete vidět, že vysoká škola (VŠ) zaštiťuje několik fakult a specializovaných pracovišť případně ústavů. Každá fakulta má libovolný počet zaměstnanců, kteří učí žáky dané fakulty. Pod VŠ, ale také patří specializovaná pracoviště, či ústavy, kde se provádí výzkumy, kde jsou mimo zaměstnanců i studenti VŠ.

Uzel reprezentuje soubor záznamů a množinová struktura reprezentuje a zřizuje vztah v síťové databázi. Je to snadno pochopitelná konstrukce, která vytváří vztah mezi dvěma uzly tak, že jeden uzel je definován jako **vlastník** a druhý jako **prvek** (tato metoda je značným vylepšením vztahu rodič/potomek). Množinová struktura podporuje vztah 1: N, neboli jeden záznam v uzlu vlastník může být v relaci k jednomu, nebo více záznamům v uzlu člen. Na druhé straně, jeden záznam v uzlu člen je ve vztahu pouze k jednomu záznamu typu vlastník. Záznam v uzlu typu člen navíc nemůže existovat, aniž by byl ve vztahu k nějakému záznamu v odpovídajícím uzlu typu vlastník.

Mezi uzly může být definována jedna, nebo více spojení (množin) a libovolný počet může být součástí dalších množin s jinými uzly v databázi. Například uzel *učitel* je ve vztahu k uzlu *student* prostřednictvím množinové struktury *vyučuj*. *Specializované pracoviště* je také ve vztahu k uzlu studenti, ale prostřednictvím množinové struktury *zaměstnávej*.

Uživatel má možnost k datům v síťové databázi přistupovat pomocí procházení odpovídajících množinových struktur. Na rozdíl od hierarchické databáze, ve které musí k datům přistupovat z

kořenové tabulky, může uživatel v síťové databázi začít přistupovat k datům z libovolného uzlu a procházet přidruženými množinami. Vztaženo k příkladu, chcete zjistit, jakou vysokou školu studuje daný student. Začnete tedy od uzlu *student*, až se dostanete k uzlu *Vysoká škola* .

Výhodou síťové databáze je rychlý přístup k datům. Umožňuje uživatelům vytvářet dotazy, které jsou mnohem komplexnější než dotazy v hierarchickém modelu. Hlavní nevýhodou síťové databáze je, že uživatel musí znát strukturu databáze, aby mohl pracovat s množinovými strukturami.
Hierarchický databázový model
Data jsou strukturována hierarchicky a obvykle se znázorňují v podobě obráceného stromu. Přičemž jedna z tabulek slouží jako tzv. kořen tohoto obráceného stromu a ostatní tabulky jako větve vycházející z kořene, což můžete vidět na obrázku níže ve vysvětlující analogii.

Na obrázku vidíte, že nejvyšším prvkem jsou stromy, které se dělí na 2 základní druhy a každý druh má zase své podruhy a ty zase své podruhy, až se dostanete k jednotlivým instancím. Například u ovocných stromů lze uvést jabloň, u jehličnatých stromů smrk, u dřevin tis apod.

Vztah je v hierarchické databázi reprezentován termíny *rodič* a *potomek*. V tomto typu vztahu může být tabulka rodiče přidružena k jedné, nebo více tabulkám potomků, ale tabulka potomka může být přidružena pouze k jedné tabulce rodiče. Tyto tabulky jsou zřetelně propojeny šipkami, nebo prostorovým rozvržením záznamů v tabulce. Uživatel pak může k záznamům přistupovat ve směru hierarchie, tedy od kořenové tabulky, přičemž postupuje dále přes stromovou strukturu až ke hledaným datům (v našem příkladě jednotlivým stromům).

Výhodou hierarchické databáze je, že uživatel může získat data velmi rychle, protože mezi tabulkami existuje přímé propojení. Další výhodou je zabudování a automatické prosazování referenční integrity, což zajišťuje, že záznamy v tabulce potomka musí být napojeny na již existující záznamy v tabulce rodiče. Tím se docílí toho, že pokud odstraníte záznam v tabulce rodiče, budou smazány i propojené záznamy v tabulkách potomků.

Hierarchická databáze nepodporuje tvorbu komplexních vztahů. Můžeme se tedy se setkat s redundantními (nadbytečnými) daty. Například v praxi se běžně vyskytují vztahy mezi jehličnatými stromy a listnatými neovocnými stromy, vytvářejí společně smíšené lesy. Toto nelze v hierarchické struktuře modelovat přímo, ale lze to například vyřešit přidáním další tabulky (tabulek), kde se však již budou vyskytovat redundantní data.

Hierarchická databáze byla hojně využívaná zejména v době ukládání dat na magnetické pásky, zejména proto že přístup k datům byl pouze sekvenční. S příchodem magnetických médií a narůstajícím počtem redundantních dat se od používání hierarchického modelu značně upustilo.

Fyzický pohled na data

Po návrhu datového modelu je potřebné implementovat příslušnou aplikaci ve vhodném systému, který svými vlastnostmi vyhovuje zadaným požadavkům. Od doby, kdy se začaly integrovat datové soubory do bází dat, byla vyvinuta řada systémů pro různé operační i technické zázemí. Jednotlivé prostředky rozdělíme do dvou kategorií podle účelu a výkonnosti.

1. Malé „stolní systémy"

První kategorii vyplňují programové systémy, které jsou svým pojetím určeny spíše jednomu uživateli, přičemž se nepředpokládá velké množství dat. Pracovní označení „stolní systém" tedy koresponduje s technickým vybavením "na stole" – nejčastěji se jedná o osobní počítač. V poslední době se však výkony osobních počítačů a s nimi i příslušných programových systémů zvyšují natolik, že se hranice mezi oběmi kategoriemi poněkud prolínají. Typickým rysem stolního systému je také většinou plná otevřenost a přístupnost dat – to platí v plné míře pro systémy provozovanými například pod operačním systémem MS DOS. Vzhledem k tomu, že s těmito systémy přichází do styku většina uživatelů, všimneme si jejich vlastností podrobněji. Všechny stolní systémy jsou vybaveny funkcemi pro základní ovládání jednotlivých relací, a to jak interaktivními (pomocí tabulkové vizualizace relace), tak i pomocí jednoduchých příkazů. Jednotlivé ovládací jazyky jsou však vždy pevně spjaty s daným systémem. Z dalších služeb je možné zmínit systém vkládání a editace údajů relací pomocí nejrůznějších formulářových schémat a systém tvorby výstupních sestav včetně základního ovládání tiskových zařízení.

Jedním z prvních systémů pracujících na osobním počítači byl systém **dBase** firmy *Ashton Tate***.** Jeho základem byla relace zobrazená jako datová struktura v souboru, kde jednotlivé prvky relace tvořily samostatné záznamy. Soubor byl opatřen hlavičkou, v níž byla kromě jiných údajů definována struktura záznamů, tedy datové typy atributů a jejich pořadí. Tento typ souboru, známý pod zkratkou **DBF** (DataBase File) je použitelný pro uložení relací. Dosud představuje jeden z nejrozšířenějších formátů, zpracovatelný řadou jiných aplikací. Na svou dobu měl systém dBase ohromující parametry a výkon – bylo možné zpracovávat soubory až velikosti 2 GB. Každý záznam mohl obsahovat až 128 položek, každá položka pak až 254 znaků. Současně bylo možné pracovat až s 10 relacemi a 10 dalšími soubory. Systémem indexací byl zrychlen přístup k datům, jejich vyhledávání i řazení čímž byly dosahovány přijatelné výkony i na poměrně málo výkonném technickém vybavení. Oproti teoretickému modelu však vykazovala tato implementace relací jednu podstatnou odlišnost – prvky bylo možné v jedné relaci libovolněkrát opakovat, čímž byl porušen množinový charakter relace. S tím byly spojeny i operace, které se odvolávaly na fyzické uložení záznamů a respektovaly jejich posloupnost v souboru.

Stejný typ souboru používal i další systém s názvem **FoxBase** firmy *Fox Software*. Později se z něj vyvinul systém **FoxPro** a po pohlcení výrobce firmou Microsoft se dále vyvíjí pod názvem Visual FoxPro. Parametry byly oproti systému dBase dále zvýšeny a s postupujícími verzemi se začal poněkud odlišovat i formád DBF souboru. Pro aplikace vytvářené na osobních počítačích byl systém Fox velmi populárním vývojovým nástrojem a bylo v něm implementováno mnoho systémů.

Jinou cestou se vydala firma *Borland International* a vytvořila systém **Paradox.** Odlišný datový formát a jiná technika ovládání relací byla i přes poměrně značnou výkonnost patrně příčinou malého rozšíření tohoto produktu. Relace byly implementovány sice podobně jako u systémů typu FoxBase, dBase, ale implicitně se s nimi pracovalo jako s množinami. Velmi výkonný systém interaktivního ovládání obsahoval i funkci "Query by example", kterou bylo možno jednoduše zadat projekci a restrikční podmínky pouhým vizuálním výběrem sloupců tabulky a naznačením požadových hodnot atributů pro restrikci. Odlišným prvkem byla také existence tzv. pohledu – dočasná relace, která vznikla operacemi nad jinými relacemi (spojením, projekcí, restrikcí), s níž bylo možno pracovat podobně jako s relacemi ostatními, ale nebyla trvale ukládána na disk a byla modifikovatelná změnami hodnot v původních relacích.

Posledním produktem, o němž se v oblasti stolních databázových systémů zmíníme, je **Access,**

součást balíku **Office** firmy *Microsoft*. Je pochopitelné, že odráží dobu svého vzniku a implementuje všechny nejnovější technologie databázových systémů. Příznačnou metodou hrubé síly vytlačuje postupně všechny ostatní systémy z prostoru osobních počítačů, zvláště pak v posledních verzích, kdy je vřazen do základní sestavy MS Office. Implementované služby zahrnují základní práci s relacemi, formulaci dotazů, z nichž jsou vytvářeny pohledy, práci se vstupními zadávacími formuláři, výstupními sestavami a práci s makropříkazy, usnadňující rutinní činnosti. Zatímco v předchozích uvedených produktech pojem databáze představoval vždy jen jeden soubor záznamů, zde se již pojem databáze užívá pro souhrn tabulek, dotazů, formulářů a podobně. Tento komplex je ukládán do souborů s rozšířením *.MDB.
Systémy pro osobní počítače můžeme ještě doplnit o vývojové nástroje které umožňují vytvářet aplikace přistupující k nějaké bázi dat. Příkladem takového prostředí je systém Deplhi firmy Borland, jež umožňuje poměrně snadno vytvořit program, manipulující s daty v různých tvarech – od jednoduchých datových souborů typu DB nebo DBF až po velké databázové systémy.

2. Velké systémy
Význačnými vlastnostmi velkých systémů jsou především řadově větší kapacity a rychlosti, daleko vyšší úroveň bezpečnosti dat, schopnost práce s více uživateli – implementace přístupových práv v několika hierarchických úrovních, komunikace prostřednictvím dotazovacího jazyka atd.
Jádrem těchto systémů je tzv. databázový stroj (implementace SŘBD) komunikující s datovými soubory na jedné straně a s vnějším světem na straně druhé. Vzhledem k tomu, že tento programový systém poskytuje databázové služby na vzdálené stroje, často je nazýván také databázový server. Uživatel databázového stroje je pak v roli klienta, jež může být realizován nejrůznějšími prostředky. Komunikace mezi databázovým serverem a klientem se odehrává v dotazovacím jazyce. Klient posílá textové zprávy – příkazy, server reaguje rovněž textovými zprávami – odpověďmi. Textový tvar odpovědí je základní, i když se tvar odpovědí někdy rozšiřuje o binární data (například uložené obrázky), která vyžadují na straně klienta zvláštní interpretační modul.
Některé velké systémy umožňují práci v tzv. transakcích. Pod pojmem transakce zde rozumíme určitou posloupnost příkazů, jejichž činnost se realizuje pouze v dočasných datových strukturách, což umožňuje vrátit některé operace zpět. Teprve po úspěšném dokončení posledního příkazu a po uzavření transakce se veškerá data modifikují v originálních souborech.
Transakční přístup také umožňuje zcela bezpečné sdílení dat z různých míst, protože zabraňuje situaci, kdy se současně provádí modifikace dat ze dvou nezávislých aplikací. Mezi velké systémy lze zařadit například **Oracle**, systém firmy **Informix**, **Sybase**, **Interbase**, **Progres**, **mySQL**, **PostgreSQL**.
[1, Okruh 15]

Architektura klient–server
Klient-server
Je to síťová architektura, která odděluje klienta (často aplikaci s grafickým uživatelským rozhraním) a server, kteří spolu komunikují přes počítačovou síť. Klient-server aplikace obsahují jak klienta, tak i server. Opakem architektury klient-server je Peer-to-peer (zkráceně P2P), kde každý hostitel nebo instance programu může fungovat zároveň jako klient i jako server (mají rovnocenné postavení i zodpovědnost)..
Klient-server popisuje vztah mezi dvěma počítačovými programy, v nichž první program, klient, žádá o služby jiný program zvaný server. Na tomto modelu je založen například přístup na E-mail, Web, přístup k databázi… Například Webový prohlížeč, to je klientský program na uživatelském počítači, který může přistupovat k informacím na libovolném webovém serveru na světě. Dotaz zadaný přes prohlížeč se předá konkrétnímu webovému serveru, tento server předá dotaz databázovému programu, který pošle dotaz databázovému serveru. Odtud se odešle odpověď zpět do

databázovému programu, ten ji zase pošle zpět do Vašeho webového prohlížeče a ten výslednou odpověď zobrazí.

Distribuovaný databázový systém

Distribuovaný databázový systém je nejčastějším případem logicky centralizovaného a fyzicky decentralizovanéhoo systému. Hlavními výhodami jsou zlepšení zpracování dat a jejich lepší přístupnost. Data jsou fyzicky distribuována v počítačové síti. Uživatelé k nim přistupují pomocí jednoho logického schématu, společného pro celý systém. Je-li některý z počítačů v síti nedostupný, lze data získat z jiného. Distribucí lze předcházet možnému výpadku celé sítě, který by nastal v případě havárie centrálního systému.

Pro uživatele musí distribuovaný systém vypadat stejně jako nedistribuovaný. Uživatelé nesmí na první pohled rozpoznat, kde jsou data fyzicky uložena. Distribuovaný systém zahrnuje distribuované zpracování, distribuované komunikační prostředky, distribuovanou databázi a řídící funkce systému. Vhodné komunikační spojení představuje počítačová síť, dnes stále častěji založená na internetových technologiích. Každý počítač je v distribuovaném systému samostatnou jednotkou a musí být při vykonávání svých funkcí nezávislý na centrálním systému a ostatních počítačích. Z důvodů výkonnosti celého systému by nejdříve měla být využívána lokální data a teprve poté by mělo být učiněno navázání komunikace se vzdáleným systémem.

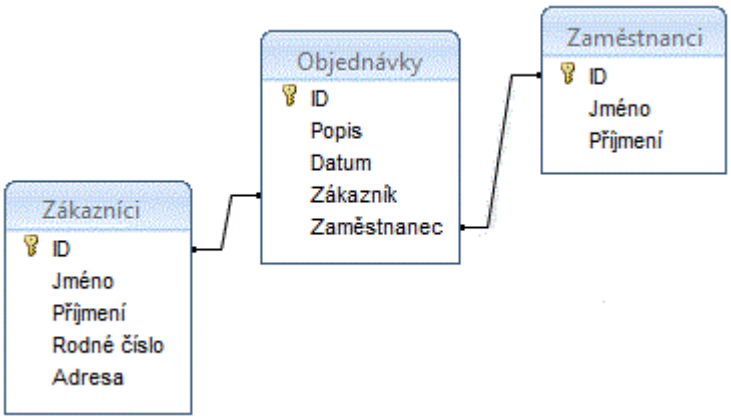
S lokálními daty úzce souvisí vytváření replik, na původním zdroji nezávislých kopií dat. Změny v replikovaných datech musí být zaznamenány do původního datového zdroje a přeneseny do ostatních replik. Podle toho, které části databázového systému jsou distribuovány rozlišujeme distribuovaný SŘBD a distribuovanou databázi. Je-li několik databází spojeno dohromady, hovoříme o nepravé distribuci. V případě, že několik databází má společně organizovaná data, hovoříme o pravé distribuci.

Podle toho, zda DBS obsahuje jednotné prostředí (typ SŘBD a databáze je ve všech uzlech shodný), označujeme takový systém jako homogenní, resp. heterogenní distribuovaný DBS. Distribuce dat přináší problémy s údržbou systému a je třeba zajistit konzistenci dat.

Klient–server je zvláštním případem distribuovaného systému. Veškerá data systému jsou umístěna na serverech. Všechny aplikační programy jsou spouštěny na klientech. Je porušen požadavek kladený na distribuované systémy – počítače nejsou navzájem nezávislé. Uživatel pracuje na počítači typu klient, na kterém jsou nainstalovány aplikační programy. Ty komunikují s programy na serverech. Servery umožňují zpracovat najednou požadavky mnoha klientů.

Relační model

Relační databázový model je z uvedených nejmladší a zároveň nejpoužívanější. V roce 1970 byl popsán Dr. Coddem. V současnosti je nejčastěji využíván u komerčních SŘBD. Model má jednoduchou strukturu, data jsou organizována v tabulkách, které se skládají z řádků a sloupců. V těchto tabulkách jsou prováděny všechny databázové operace.



Obr. 3 - Relační model

Relační datový model

Relační model byl popsán v roce 1970.

Základní ideje relačního modelu:

- RMD důsledně odděluje data, která jsou chápána jako relace, od jejich implementace
- přístup k datům je symetrický, tj. při manipulaci s daty se nezajímáme o přístupové mechanizmy k datům
- pro manipulaci s daty jsou k dispozici dva silné prostředky - relační kalkul a relační algebra
- pro omezení redundance dat v relační databázi jsou navrženy pojmy umožňující normalizovat relace

Základní definice RMD

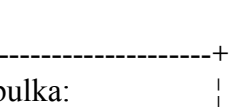
RMD má jediný konstrukt - databázovou relaci.
Mějme množiny D1,D2,D3 ,.....Dn. Z každé vybereme 1 prvek. Tím vytvoříme uspořádanou n-tici. Kartézský součin D1xD2... je množina šech posloupností (x1,x2,...) kde x1 je prvkem D1 ...
Relace je každá podmnožina kartézského součinu. Z hlediska databázových systémů jsou množiny D1,D2,.. množina hodnot atributů a označují se jako domény.

Od matematické relace se liší v několika aspektech:

- relace je vybavena pomocnou strukturou, které se říká schéma elace. Schéma relace se skládá ze jména relace, jmen atributů a domén
- prvky domén, ze kterých se berou jednotlivé komponenty prvků elace, jsou atomické (dále nedělitelné) hodnoty. Tomuto omezení e říká 1.normální forma relací (1NF).

Schéma relace R se vytvoří nad množinou atributů A1:D1,...AN:DN i jsou jména atributů a Di jsou domény. Dvojici Ai:Di se říká tribut relace.
schéma relace lze zapsat R(A1:D1,....An:Dn) relace R nad množinou A je libovolná podmnožina kartézského oučinu domén D1x...xDn. Doména náležící atributu C se označuje jako dom(C). Domény jsou obvykle primitivní typy dat (STRING, INTEGER...).
Prvkům relace se říká n-tice, přičemž n určuje řád relace. Relační schéma databáze je dvojice (R,I), kde R je množina schémat relací a I je množina integritních omezení.
Jedno z významných IO na relaci R(A) je existence primárního líče.

Primární klíč je množina atributů K z A, jejichž hodnoty jednoznačně určují n-tice relace R. K je minimální v tom smyslu, že z ní nelze odebrat žádný atribut, protože by to narušilo identifikační vlastnost.
Atribut, který je součástí nějakého klíče se nazývá klíčový.
Atributy, které nejsou součástí žádného klíče se nazývají klíčové. Z podstaty RMD vyplývá, že každá relace má klíč. protože relace jsou množiny, nesmí relace obsahovat duplicitní prvky.
Dalším důležitým IO je referenční integrita. Toto omezení opisuje vztahy mezi daty ve dvou relacích. Atribut, kterého se referenční integrita týká se nazývá cizí klíč (foreign key).
Takové dvě relace se obvykle nazývají master, detail nebo parent, ependent. Česky obvykle hlavní, závislá.

Příklad:
Hlavní tabulka:
UCITELE(CISLO,JMENO,PLAT,PRIPLATEK,...)

Závislá tabulka:
PREDMETY(ZKRATKA,NAZEV,.....,GARANT)

CISLO z relace UCITELE je primární klíč a objevuje se v relaci
PREDMETY jako položka GARANT. Položka GARANT je tzv. cizí klíč.

Přípustnou relační databází se schématem (R,I) nazýváme množinu relací R1,...Rk takových, že jejich prvky vyhovují I. O takové množině relací říkáme, že je konzistentní.

Formou reprezentace relací může být dvojrozměrná tabulka.

Podmínky, které musí splňovat relační tabulka:
- všechny hodnoty v tabulce musí být elementární - tzn. Dále nedělitelné - podmínka 1.NF
- sloupce mohou být v libovolném pořadí
- řádky mohou být v libovolném pořadí
- sloupce musí být homogenní = ve sloupci musí být údaje stejného typu
- každému sloupci musí být přiřazeno jednoznačné jméno (tzv. tribut)
- v relační tabulce nesmí být dva zcela stejné řádky. Tzn., že každý řádek je jednoznačně rozlišitelný.

Shrnutí pojmů (a jejich zjednodušení)

Doména: je množina datových hodnot stejného typu. Tyto hodnoty popisují nějakou vlastnost objektu.

Relace: je množina vztahů mezi jednotlivými prvky domén

Atribut: je pojmenování pro každé užití hodnoty z domény v relaci

Záhlaví relace: obsahuje jméno relace a jména atributů v relaci. Je v čase neměnné.

Tělo relace: obsahuje v čase proměnnou množinu n-tic hodnot, jejichž pořadí je dáno záhlavím relace.

Stupeň relace: je počet atributů relace

Kardinalita relace: je počet řádků relace

Primární klíč: je sloupec, který jednoznačně určuje řádky v tabulce. Pokud je třeba použít více sloupců pro jednoznačné určení řádků, potom hovoříme o tzv. složeném klíči.
Pokud je více atributů, které splňují pravidlo pro primární klíč, jeden zvolíme jako primární. Ostatní jsou alternativní klíče.

Definice primárního klíče:

Primární klíč je podmnožina atributů relace, která
1) jednoznačně identifikuje každý prvek relace
2) není redundantní, tj. žádný její atribut nelze vynechat, aniž by podmínka 1) přestala platit

Tabulky: jsou konktrétními instancemi relačního schématu

Kandidáti primárního klíče: primární klíč - pouze jeden
alternativní klíče - více

Relační algebra
je nástrojem pro manipulaci relací. Je to jazyk, který pracuje s celými relacemi. Operátory relační algebry se aplikují na relace a výsledkem jsou opět relace.

Základní operace relační algebry
- projekce
- selekce
- spojení

Projekce:
umožňuje potlačit označené atributy v relaci. Umožňuje přejít z relace o n sloupcích na relaci o p sloupcích, přičemž p < n. Nově vzniklá relace bude obsahovat p sloupců. Může obsahovat i méně řádků než původní relace, protože duplicitní řádky se v relaci nesmějí vyskytovat.

Selekce:
někdy se vyskytuje termín restrikce Operací selekce vznikne nová relace odstraněním nepotřebných řádků na základě logické podmínky. Podmínka je zadána Booleovským výrazem (pomocí logických spojek and, or, not), jehož atomické formule mají tvar t1 ń t2, kde ń je <,>,=,<=,>=,<>, ti je buď konstanta nebo jméno atributu

Spojení:
Spojení dvou relací vytvoří třetí relaci. Výsledná relace vždy obsahuje všechny kombinace, které vyhovují zadané podmínce. Podmínka vyjadřuje vztah mezi dvěma relacemi

atribut1 operátor atribut2
¦ ¦
patří 1.relaci patří 2.relaci

Spojení relací se zajišťuje pomocí společného atributu. Jednotlivé řádky z 1.relace se spojí s příslušnými řádky z 2.relace. Relace se nespojují podle názvů atributů, ale podle jejich hodnot.

Definice:
Spojení relací R a S podle podmínky f na atributu A z R a atributu B z S je relace
R[A f B]S = {rs|r ¸ R & s ¸ S & rR[A] f rS[B]}, kde rR[A],rS[B] jsou hodnoty atributů A, resp. B relací R, resp. S.

3 druhy spojení:

1. spojení na rovnost
atribut1 = atribut2

2. spojení na nerovnost
atribut1 <> atribut2

3.vnější spojení - INKLUZE

Funguje stejně jako přirozené spojení, do výsledné relace se však přidají i nespojené řádky z první relace
(ev. z druhé relace, ev. z obou relací). Pak příslušné atributy nejsou vyplněny.

Příklad:
Relace 1

Číslo	Jméno	Město	Ulice	Věk
1	Jiří	Kladno	Pražská 32	19
2	Karel	Praha	Evropská 1	17
3	Jan	Brno	Tichého 43	16
4	Karel	Brno	Dlouhá 21	19
5	Tomáš	Praha	Široká 7	20

Relace 2

ČísloS	Č-typu	Rok-výroby	Cena
1	101	1988	100
1	701	1989	105
2	101	1990	200
7	701	1991	300

Podmínka pro spojení na rovnost:
Číslo->relace 1 = ČísloS->Relace 2

Vznikne nová relace:

Číslo	Jméno	Město	Ulice	Věk	Č-typu	Rok-výroby	Cena
1	Jiří	Kladno	Pražská 32	19	101	1988	100
1	Jiří	Kladno	Pražská 32	19	701	1989	105
2	Karel	Praha	Evropská 1	17	101	1990	200

Zavedeme jednoduchý jazyk pro vyjádření operací relační algebry:

Příklad:
Z relace AUTA vybereme všechna identifikační čísla, názvy a rok výroby aut, jejichž cena je menší než 150 000 Kč. Klíčová slova budou podtržena.

restrict AUTA where cena < 150000 giving MEZIVYS1
project MEZIVYS1 over ident_a, nazev_a, rok_vyr, cena giving VYSL

Tuto operaci provedeme pomocí hnízdění (nesting):
project (restrict AUTA where cena < 150000) over ident_a, nazev_a,
rok_vyr, cena giving VYSL

Základní množinové operace aplikované na relace

Podmínky pro všechny množinové operace:

- obě relace jsou stejného stupně, tj. mají stejný počet sloupců
- každý i-tý atribut z obou relací je definován na stejné doméně

Sjednocení
R U S = {t|t ˛ R U t ˛ S}

Sjednocení vytvoří novou tabulku, která obsahuje řádky obou výchozích tabulek. Pokud mají tyto tabulky některé řádky shodné, ve výsledné tabulce se objeví pouze jednou.

Průnik
R ∩ S ={t|t ˛ R & t ˛ S}

Výsledná tabulka bude obsahovat pouze totožné řádky obou tabulek.

Množinový rozdíl
R - S = {t|T ˛ R & t not˛ S}
nová tabulka bude obsahovat všechny řádky první tabulky, ale pouze ty, které se nevyskytují v druhé tabulce.

Symetrický rozdíl
nová tabulka bude obsahovat všechny řádky obou tabulek, s výjimkou těch, které se vyskytují v obou tabulkách.

Kartézský součin
Již není typickou množinovou operací, protože nesplňuje její 2 podmínky
R x S relace R stupně m a relace S stupně n je relace stupně m+n, která je definována:
R x S = { rs|r ˛ R & s ˛ S}, kde rs představuje prvek relace (r1, r2, ...rm,s1,s2,....sn)
r,s označuje prvky relací.

Kartézský součin vytvoří novou tabulku tak, že spojuje řádky z obou tabulek systémem každý s každým. Počet řádků nové tabulky je součinem počtů řádků obou vstupních tabulek.

Rozdíl mezi spojením a kartézským součinem

Spojení je vytvoření podmnožiny kartézskémo součinu vstupních tabulek. Kartézský součin spojí každý řádek s každým, spojení se provede jen při splněné podmínce. Podmínkou je výraz, ve kterém se porovnávají dva
srovnatelné atributy ze dvou tabulek.

4 Modely organizace dat v relačních databázových systémech
Zásobníky dat v DFD slouží z dlouhodobému uchovávání dat, proto při počítačové realizaci jsou tyto prvky řešeny jako datové soubory.

Datové modely slouží pro návrh datových struktur souborů. Jejich tvorba nezávisí na fyzickém uložení v paměti počítače a na operačním prostředí. Nejčastěji se používají:

relační model

entitně relační model

Relační model dat
Stanoví pravidla pro uspořádání dat do relací (dvourozměrných tabulek). Každý záznam obsahuje údaje o zadaných vlastnostech objektu. Při uchovávání dat se setkáváme se dvěma základními problémy:

redundance dat - nadbytečnost - údaje se zbytečně opakují ve více záznamech - řeší se přesunutím do samostatných tabulek

opakující se skupiny dat - v jednom záznamu se vyskytují skupiny položek , které se v záznamu opakují.

Tyto problémy je nutno z navržené struktury odstranit. K tomu slouží postup nazvaný **normalizace logické struktury**.

Normalizace logické struktury

Provádí se postupně od *první normální formy* (1NF) až ke *čtvrté normální formě* (4NF). Jejich řešení vyžaduje důkladnou znalost souvislostí mezi jednotlivými položkami. Praktický význam pro tvorbu má 1NF, BCNF - Boyce-Coddova normální forma a 4NF.

Normalizace představuje rozklad výchozí struktury dat do několika nových struktur tak, aby byly vyloučeny redundance a opakující se skupiny.

První normální forma

Definice: *Struktura záznamu odpovídá 1NF tehdy, obsahují-li všechny možné druhy záznamů vždy stejný počet datových prvků*.

Logické struktura v 1NF obsahuje prostý výčet identifikátorů jednotlivých atributů záznamu. Záznamy v takovéto struktuře obsahují množství redundantních údajů. Proto tato forma slouží jako výchozí forma pro transformaci do vyšších normálních forem. Příkladem je modelová struktura dat pro sledování údržby lokomotiv:

```
LOKO.1NF=C_loko+C_loko+Rok_vyroby+Datum_zar+Stup_udržby+Datum_udržb
        y+Mistr+Mistr_Jméno+Mistr_Příjmení
```

Pokud použijeme tuto strukturu jako základ k popisu každého údržbového zásahu provedeného na lokomotivě, výsledný soubor bude obsahovat mnoho redundantních údajů. Struktura totiž obsahuje atributy, které se vztahují k různým objektům (Loko a Mistr). Proto je nutné rozdělit strukturu do samostatných struktur. Každá takováto struktura musí obsahovat takový atribut, aby jednoznačně určil příslušnost k objektu - klíčový prvek. Tím je `C_loko` a `Mistr`.

```
LOKO = @C_loko+Rok_vyroby+Datum_zar

UDRŽBA = @C_loko+@Datum_udržby+Stup_udržby +Mistr

PRACOVNICI=@Mistr+ Mistr_Jméno+Mistr_Příjmení
```

V těchto strukturách je pak každý záznam definován svým klíčem.

Pro další postup normalizace jsou nutné další pojmy:

Definiční obor atributu - interval jeho přípustných hodnot.

Jednoznačný prvek - prvek, kterému lze v záznamu přiřadit vždy pouze jednu jedinou hodnotu z jeho definičního oboru. Příklad: Hmotnost v kg - nemůže být současně hodnota 20 kg a 30 kg.

Mnohoznačný prvek - prvek, u kterého v rámci jednoho záznamu můžeme přidělit víc než jednu hodnotu. Příklad: Vzdělání - obsahuje výčet absolvovaných škol.

Mnohoznačný a nezávislý prvek - prvek jehož každá hodnota je určena pouze hodnotou jednoho jednoznačného prvku. Příklad: `Datum_udržby` - jeho výskyt závisí na hodnotě jednoznačného prvku `C_loko`.

Mnohoznačný a závislý prvek - jeho výskyt je závislý na hodnotě jiného mnohoznačného prvku.

Vložená entita - skupina prvků struktury, která v rámci struktury popisuje jiný objekt. Příklad: Dílčí struktura UDRŽBA ve struktuře LOKO.1NF.

Opakující se skupina - skupina prvků, jejichž hodnoty se ve struktuře opakují.

Nepovinná skupina - prvky, které v některých záznamech nemají smysl. Příklad: Datum_porodu u mužů.

Variantní skupina - skupina prvků, která má pro různé objekty různou skladbu prvku.

Primární klíč - minimální množina prvků, jejichž hodnoty jednoznačně identifikují každý záznam. Existence tohoto primárního klíče vylučuje možnost existence dvou stejných záznamů.

Determinant - minimální množina takových datových prvků jejichž hodnoty jednoznačně určují hodnoty zbývajících prvků záznamu. Dvěma různým determinantům může odpovídat jedna skupina ostatních prvků.

Boyce-Coddova normální forma

Definice BCNF: *Logická struktura záznamu odpovídá BCNF tehdy a jen tehdy, jestliže determinant každého záznamu je využitelný jako primární klíč zajišťující také vazbu na charakterizovaný objekt.*

Dalším krokem v analýze struktury je převedení na syntaxi zápisu obsahu zásobníku dat z diagramu datových toků. Tím uspořádáme položky do logicky souvisejících skupin. používáme následující syntaktické znaky, známé z DFD:

{ } - opakující se skupiny datových prvků a mnohoznačných prvků

( ) - nepovinné datové prvky

[|] - variantní skupina ve struktuře

Pozn:

U opakujících se skupin předpokládáme, že se mohou vyskytovat vícekrát ale i vůbec.

Mnohoznačný nezávislý prvek vyznačíme jako opakující se skupinu.

Proto je nutné z každé opakující se skupiny, variantní skupiny, vložené entity, a nepovinné skupiny vytvořit samostatné datové struktury. Ze zbylých prvků vzniká samostatná struktura - bázová struktura.

Po vytvoření struktur provádíme analýzu každé z nich na podmínky BCNF:

- stanovit determinant nové struktury
- kontrola determinantu na použití jako primárního klíče, popř. jej doplnit vhodným prvkem na primární klíč
- prověřit vazbu na objekt, který popisuje - determinant se musí doplnit takovými prvky, aby obsahoval primární klíč objektu který popisuje.

Definice 4NF: *Struktura je ve 4NF tehdy, jestliže je v BCNF a obsahuje nanejvýš jeden nezávislý mnohoznačný prvek.*

Praktické řešení normalizace

1. Pravidlo jedinečnosti polí

Každé pole v tabulce by mělo představovat jedinečný typ informace.

- sloučené pole rozdělit na samostatná jednoduchá pole a zařadit do samostatné tabulky;
- pro opakující se skupiny vytvoříme samostatné tabulky

2. Pravidlo primárního klíče

Každá tabulka musí mít jednoznačný identifikátor neboli primární klíč, který je vytvořen z jednoho nebo více polí v této babulce.

♦ pro klíč je nutno použít co nejjednodušší "přirozený" údaj

3. Pravidlo funkcionální závislosti

Pro každou jedinečnou hodnotu primárního klíče se musí hodnoty v datových sloupcích týkat předmětu tabulky a musí tento předmět úplně popisovat.

♦ v tabulce by neměly být žádná data, která se netýkají popisovaného předmětu tabulky (ten je definovaný primárním klíčem);

♦ předmět tabulky by data v tabulce měla úplně popisovat.

4. Pravidlo nezávislosti polí

Musíme být schopni provést změnu do dat v libovolném poli (které netvoří primární klíč) bez toho, aby byla ovlivněna data v jakémkoliv jiném poli.

Při návrhu nových tabulek v rámci normalizace musíme v tabulce použít pole, pomocí kterého můžeme spojit tabulku popisující původní objekt s nově vytvořenou tabulkou. tato pole se nazývají *cizí klíče*. Ty by měly přispívat k efektivnosti celé databáze. Tyto klíče můžeme připojit k primárním klíčům a tím vytvořit relační vazby.

Příklad normalizace struktur v relačním modelu

Informační subsystém údržby lokomotiv

Jedná se o data z provozu elektrické šestinápravové lokomotivy. Informace popisují základní údaje o této lokomotivě (C_loko, Rok_vyroby, Datum _zar). Číslo lokomotivy je jedinečné pro každou lokomotivu a nemůže se opakovat.

U každého vozidla se provádí pravidelná údržba. O každém jejím provedení se zaznamenávají údaje Datum_udržby, Mistr, Mistr_Jméno, Mistr_Příjmen, Pracovník. V jednom dni se může na vozidle provést pouze jeden údržbový zásah. Za jeho provedení odpovídá mistr. Ten má jméno a příjmení. Na zásahu se podílí jeden nebo více pracovníků.

Na některých lokomotivách se v rámci provozu mohou provádět úpravy. Při jejich realizaci se zaznamená kód úpravy Upravy a datum jejího provedení Datum_provedení. Každá úprava má svoji technickou dokumentaci, která je evidovaná pod evidenčním číslem Popis_upravy.

Lokomotiva je osazena 6 trakčními motory, které mají svá identifikační čísla TM_číslo. Informační systém eviduje jejich umístění na dané lokomotivě, jejich poslední závadu před montáží Datum_závady, Závada. Za odstranění této závady byl odpovědný pracovník Odstranil. Ve skladu se nacházení i nepoužité TM, které jsou v evidenci (nejsou namontované na lokomotivě).

Pracovníci, kteří se podílejí na údržbě lokomotiv a trakčních motorů jsou ze stejné skupiny.

Popis jednotlivých datových položek je v následující tabulce.

C_loko	evidenční číslo lokomotivy je pro každou lokomotivu jedinečné
Rok_vyroby	rok výroby
Datum _zar	datum zařazení do provozu
Stup_udržby	provedený stupeň údržby
Datum_udržby	datum dokončení údržbového zásahu, popsaný stupněm údržby
Mistr	kód mistra
Mistr_Jméno	křestní jméno mistra
Mistr_Příjmen	příjmení mistra
Pracovník	kódy pracovníků provádějících údržbu
Upravy	kód provedené rekonstrukce
Datum_provedení	datum realizace úpravy na vozidle
Popis_upravy	popis rekonstrukce - určení technické dokumentace
TM_typ	typové označení trakčního motoru
TM_číslo	výrobní číslo trakčního motoru
Datum_závady	datum výskytu závady
Závada	popis závady
Odstranil	kód pracovníka, který závadu odstranil

Z tohoto výčtu pak sestavíme strukturu LOKO v 1.NF a určíme primární klíč struktury. Položky zařazené do klíče označujeme symbolem @.

LOKO.1NF = @C_loko+C_loko+Rok_vyroby+Datum _zar+

Stup_udržby+@Datum_udržby+

@Mistr+Mistr_Jméno+Mistr_Příjmení+@Pracovník+

@Upravy+Datum_provedení+Popis_upravy+TM_typ+@TM_číslo+

@Datum_závady+Závada+Odstranil

Strukturu převedeme do tvaru, který je obdobný popisu struktur v datovém slovníku modelu DFD. Tím dosáhneme uspořádání položek do vzájemně souvisejících skupin. Používáme následující syntaktické znaky:

{ } - opakující se skupiny datových prvků a mnohoznačných prvků
( ) - nepovinné datové prvky
[|] - variantní skupina ve struktuře

LOKO.DFD = C_loko+C_loko+Rok_vyroby+Datum _zar+

{Stup_udržby+Datum_udržby+Mistr+Mistr_Jméno+

Mistr_Příjmení+{Pracovník}}+

({Upravy+Datum_provedení+Popis_upravy})+

{TM_typ+TM_číslo+Datum_závady+Závada+Odstranil}

Tímto postupem jsme docílili toho, že jsou zřejmé skupiny položek, které popisují objekt lokomotivu a pak další objekty jako údržba, úpravy a trakční motory. Proto strukturu v DFD tvaru můžeme rozdělit do samostatných struktur. U nich pak je potřeba stanovit determinant a primární klíč. V tomto případě jsou struktury následující:

LOKO.1 = @C_loko+C_loko+Rok_vyroby+Datum _zar

LOKO.2 = @C_loko+@Datum_udržby+Stup_udržby+Mistr+Mistr_Jméno+

Mistr_Příjmení+Pracovník

LOKO.3 = @C_loko+@Upravy+Datum_provedení+Popis_upravy

LOKO.4 = @C_loko+@TM_číslo+TM_typ+Datum_závady+Závada+Odstranil

Z pohledu BCNF je definitivní pouze struktura LOKO.1. Determinant C_loko je současně i primárním klíčem.

Ostatní struktury popisují objekty, jež jsou s LOKO.1 logicky svázány, proto k jejich determinantům musíme připojit jako *cizí klíč* primární klíč z této struktury. Tím vznikly primární klíče těchto struktur. Vznikla definitivní struktura u struktury LOKO.4.

Struktury č. 2 a 3 obsahují ještě další skupiny položek, které by se zbytečně opakovaly. Proto musíme ještě znovu provést rozklad.

LOKO.2 obsahuje položky, které se vztahují k dalším podřízeným objektům - mistr a pracovník. Proto musíme strukturu rozdělit:

LOKO.21 = @C_loko+@Datum_udržby+Stup_udržby

LOKO.22 = @C_loko+@Datum_udržby+Mistr

LOKO.23 = @Mistr+Mistr_Jméno+Mistr_Příjmení

LOKO.24 = @C_loko+@Datum_udržby+Pracovník

Ve struktuře LOKO.3 by se v každém záznamu o provedení dané úpravy objevoval stejný obsah položky Popis_upravy. Tyto údaje by byly redundantní. Proto je možné tuto položku oddělit do samostatné struktury a přidat jí cizí klíč. Nové struktury budou následující:

LOKO.31 = @C_loko+@Upravy+@Datum_provedení

LOKO.32 = @Upravy+Popis_upravy

Na závěr provedeme znovu kontrolu na správnost transformace všech struktur do BCNF.

Můžeme konstatovat že všechny navržené struktury odpovídají BCNF a můžeme je použít pro konstrukci datových tabulek pro relační databázi.

Entitně relační model dat
Základní pojmy entitně relačního diagramu (ERD)
Tento model představuje grafické znázornění **entit** a jejich vzájemných vazeb. Představuje statický model systému. Používá se pro návrh fyzické struktury souborů. Slouží pro:

♦pohled z hlediska vazeb mezi entitami o kterých se data uchovávají;

♦je vhodný pro sestavování množiny struktur přímo v 4NF;

♦poskytuje podklad pro návrh fyzické struktury souborů.

Obr. N.1: Základní komponenty entitně relačního modelu (a) entita, b) relační vazba, c) sub- a super-typ).

Model se sestavuje ze tří základních komponent:

a) *Datová entita* - reprezentuje určitý typ objektů. Objekt je nějaká existující realita o které jsou zaznamenávány charakteristické údaje (atributy). Stejné typy entit tvoří třídu datových entit. Název je tvořen podstatným jménem v jednotném čísle. Entita musí splňovat podmínky:

každá entita musí být popsaná pomocí jednoho nebo více datových prvků.

entita musí být obsažena v odpovídajícím informačním systému.

b) *Relační vazby* - představují logické vztahy mezi entitami, vyjadřuje se slovesem. Důležité i stanovení charakteru vazby (kardinalitu). Kardinalita popisuje vztah mezi výskyty záznamů svázaných entit. Může být následující:

N:M	**N** výskytů v první entitě je svázáno s **M** výskyty ve druhé entitě
1:M	**jeden** výskyt v první entitě je svázán s **M** výskyty ve druhé entitě
M:1	**M** výskytů v první entitě je svázáno s **jedním** výskytem ve druhé entitě
1:1	**jeden** výskyt v první entitě je svázán s **jedním** výskytem v entitě druhé

Pozn:

Počet M, N musíme chápat ve smyslu *žádný a více* (Student absolvuje žádný a více Předmětů)

c) *Subtypy a supertypy* - slouží k zobrazení variantní struktury záznamu.

Modelování pomocí entitně - relačního modelu (ERD) je velmi populární a znázorňuje logickou strukturu pro každou entitu a relační vazbu mezi entitami.

Algoritmus sestavení ERD modelu je sice jednoduchý, praktická tvorba s velkým počtem datových prvků je však náročná, Vyžaduje perfektní znalost obsahu a souvislostí všech datových prvků.

Výchozím bodem je sestavený DFD včetně sestavených slovníků dat.

Tvorba entitně-relačního modelu

Algoritmus tvorby ERD modelu můžeme rozdělit do sedmi základních kroků.

První krok

Ze seznamu všech datových prvků vytvoříme tří základní skupiny:

1. skupina - obsahuje pouze determinanty

2. skupina - obsahuje prvky, jejichž hodnoty jsou jednoznačně určeny determinanty z prvků 1. skupiny.

3. skupina - ostatní prvky, nezařazené do předchozích skupin.

Jména prvků se mohou ve všech skupinách vyskytovat pouze jednou. Determinant ověřujeme tak, že určíme konkrétní hodnotu vybraného prvku a určíme ty prvky, které jsou touto hodnoto určeny.

Druhý krok

V tomto kroku provádíme reorganizaci 3. skupiny datových prvků. Prověřujeme možnost determinovat prvek pomocí složeného determinantu z prvků 1. skupiny. Nemá-li prvek takovouto možnost. přeřadíme jej do 1. skupiny. a tvoří determinant sám sobě.

Třetí krok

Datové položky roztříděné v kroku 2 spojíme do spolusouvisejících entit a subtypů a přiřadíme jim vhodné názvy.

Čtvrtý krok

Nyní sestavíme předběžný entitně-relační diagram, která obsahuje pouze jednotlivé entity a určíme kardinalitu jednotlivých vztahů. Jako nejčastější otázkou při stanovení kardinality je:

"Kolik záznamů v cílové entitě může náležet k jednomu záznamu ve výchozí entitě."

V případě, že výchozí položce neodpovídá žádný nebo jeden záznam v cílové entitě, pak se jedná o kardinalitu 1:M. U supertypu je kardinalita typu 1:1, proto ji není nutné uvádět.

Pátý krok

Do předběžného ERD doplníme značky vazeb s výstižným slovesným pojmenováním.

Šestý krok

ERD představuje pouze grafické znázornění modelu. Pro další použití je nutné definovat logické struktury jednotlivých entit a navrhnou relační vazby. To představuje návrh struktur záznamů včetně stanovení primárních klíčů.

Sedmý krok

Tento poslední krok představuje normalizaci navržených struktur v závislosti na zajištění relačních vazeb. Při normalizaci používáme následující pravidla (podle [KON96]:

1. Je-li kardinalita 1:1, vloží se determinant jedné entity do struktury druhé entity. Toto rozšíření se provede u obou entit.

2. Je-li kardinalita 1:M vloží se primární klíč entity 1 do logické struktury M.

3. Je-li kardinalita M:N můžeme říci, že jsou struktury normalizované, prakticky to však znamená, že tuto vazbu musíme převést přes další strukturu, která bude obsahovat determinant složený s determinantů obou entit.

4. Zrušíme ty struktury, které obsahují pouze jeden prvek, protože zákonitě musí existovat i v jiné struktuře.

Mohou nastat i zvláštní případy vazeb:

Dvě nebo více vazeb mezi entitami

Tento jev se vyskytuje poměrně často. Tady je nutné udělat důkladný rozbor situace a důsledně rozlišit vlastnosti jednotlivých vazeb. Pokud je vazba nadbytečná, pak se zruší, pokud mají vazby podobnou strukturu ale odlišné vlastnosti (např.: Množství_nakupované a Množství_reklamované ve vazbě mezi ZÁKAZNÍKEM a PRODEJCEM), je nutné datové prvky odlišit již jejich pojmenováním.

Rekurzivní relace

Může se vyskytnout situace, kdy relační vazby je na tutéž entitu. Tuto vazbu nazýváme *cyklická* nebo *rekurzivní*. Tato vazby by vznikla v případě, že data o Mistrovi i Pracovníkovi jsou uloženy v jedné entitě. Pak vazby Mistr Řídí Pracovníka představuje rekurzivní vazbu. Tento problém se většinou řeší pomocí rozdělení entity na dvě samostatné entity s podobnou strukturou a doplněním primárních klíčů.

Ternární relační vazby

Představuje typ vazby, svazující více jak dvě entity. Tyto vazby se musí nahradit binárními vazbami (vazby mezi dvěmi entitami).

Další zvláštní případy a jejich řešení jsou uvedeny např v [KON96].

Algoritmus sestavení entitně relačního modelu je sice jednoduchý, tvorba je však náročná zvlášť pro větší počet datových prvků.. Výchozím bodem je DFD a jeho slovník dat.

Příklad návrhu entitně-relačního modelu

ERD model tvoříme pro IS popsaný v příkladu relačního modelu. Postup vychází z [KONEČNÝ,1996].

Prostý výčet datových položek je na Obr. N.4.

V prvém kroku provedeme roztřídění položek do tří skupin. Například `C_loko` je determinant pro prvky `Rok_vyroby`, `Datum_zar`, protože konkrétní číslo definuje i datum zařazení do provozu i rok výroby. Položky `Pracovníci` a `Datum_provedeni` nemají determinant, proto jsou ve skupině 3.

Rozdělení prvků je v tabulce Tab. N.1.

Tab. N.1: Rozdělení prvků po 1. kroku

Determinanty	Prvky definované determinantem	Zbývající prvky
1. skupina	2. skupina	3. skupina
C_loko	Rok_vyroby	Pracovník
	Datum _zar	Datum_provedení

Datum_udržby	Stup_udržby	
Mistr	Mistr_Jméno	
	Mistr_Příjmen	
TM_číslo	TM_typ	
	Datum_závady	
	Závada	
	Odstranil	
Upravy	Popis_upravy	

Ve druhém kroku posuzujeme datové prvky 3. skupiny. Prvek `Datum_provedeni` můžeme determinovat kombinací determinantů `C_loko` a `Upravy`. Prvek `Pracovníci` je determinantem sám sobě, proto jej zařadíme do 1. skupiny.. Výsledkem je pal tabulka Tab. N.2.

Krok druhý doplníme krokem třetím, kde přeskupíme datové prvky podle souvislostí do jednotlivých entit a výstižně pojmenujeme.

Ve čtvrtém kroku vytvoříme předběžný entitně-relační diagram, který obsahuje pouze entity a vazby vyznačující kardinalitu. Tu zjišťujeme např. otázkou pro vazbu LOKO a UDRZBA:

"Kolik lokomotiv může mít v jednom dnu údržbový zásah?". Odpověď zní "M"

"Kolik údržbových zásahů může mít lokomotiva v jednom dnu?" Odpověď je "1".

Pomocí podobných otázek stanovíme kardinality jednotlivých vazeb. Tento diagram je na obrázku Obr. N.5. (Pro ilustraci jsou použity jiné symboly pro vyznačení kardinality vztahů, význam je patrný.)

Tab. N.2:

Determinanty	Prvky definované determinantem	Název entity
C_loko	Rok_vyroby	LOKO

	Datum _zar	
Datum_udržby	Stup_udržby	UDRŽBA
Mistr	Mistr_Jméno	MISTR
	Mistr_Příjmen	
TM_číslo	TM_typ	TRAKČNÍ MOTOR
	Datum_závady	
	Závada	
	Odstranil	
Upravy	Popis_upravy	REKONSTRUKCE
C_loko+Upravy	Datum_provedení	REALIZACE
Pracovník		PRACOVNÍK

Pátý krok realizuje pojmenování jednotlivých navržených vazeb. Pojmenování může být následující:

Opravuje LOKO *Opravuje* UDRZBA

Ridi UDRZBA *Ridi* MISTR

Provadi UDRZBA *Provadi* PRACOVNIK

Pohani LOKO *Pohani* TRAKCNI MOTOR

Modernizuje LOKO *Modernizuje* REALIZACE

Popisuje REALIZACI *Popisuje* REKONSTRUKCE

Upravený ERD je na obrázku Obr. N.6.

V rámci šestého kroku navrhneme logické struktury jednotlivých tabulek. Tato struktura prozatím neodpovídá normálním formám. Ta je cílem následujícího kroku.

LOKO	=@C_loko+Rok_vyroby+Datum _zar
UDRZBA	=@Datum_udržby+Stup_udržby
MISTR	=@Mistr+Mistr_Jméno+Mistr_Příjmen
TRAKČNÍ MOTOR	=@TM_číslo+TM_typ+Datum_závady+Závada+Odstranil
REKONSTRUKCE	=@Upravy+Popis_upravy
REALIZACE	=@C_loko+@Upravy+Datum_provedení
PRACOVNÍK	=@Pracovník

Finále tvoří 7. krok, ve kterém upravíme struktury do normální formy. Rozebereme jednotlivé struktury:

Opravuje: Vazba má kardinalitu M:1, proto podle pravidla 2 přidáme primární klíč entity UDRZBA do entity LOKO.

Ridi: Vazba má opět kardinalitu 1:M, postup bude podobný jako v předchozím případě.

Provadi: Tato vazba je kardinality M:N, struktury entit jsou sice normalizovány, musíme však zavést novou strukturu, které bude obsahovat determinanty obou struktur. Zároveň však uplatníme pravidlo č. 4, o zrušení struktur s jedním datovým prvkem. Proto struktury PRACOVNIK můžeme zrušit, tento prvek obsahuje struktura PROVADI

Pohani: vazba je typu 1:M, proto postupujeme podobně jako v případě vazby *Ridi*.

Modernizuje: pro vazbu platí obdobný přístup jako u předchozích vazeb 1:M.

Popisuje: Vazba je sice kardinality M:1, struktura entity realizace má však složený determinant. Pro úpravu struktur platí pravidlo č.1, - vzájemné přidání determinantů do obou struktur.

Jednotlivé struktury budou vypadat následovně.

LOKO	=@C_loko+Rok_vyroby+Datum_zar
UDRZBA	=@Datum_udržby+@C_loko+Stup_udržby+Mistr

MISTR	=@Mistr+Mistr_Jméno+Mistr_Příjmen
PROVADI	=@Datum_udržby+@C_loko+@Pracovnik
TRAKČNÍ MOTOR	=@TM_číslo+@C_loko+TM_typ+Datum_závady+Závada+Odstranil
REKONSTRUKCE	=@Upravy+Popis_upravy
REALIZACE	=@C_loko+@Upravy+Datum_provedení

Takto stanovené struktury slouží jako podklad pro návrh databázových struktur informačního systému.

Relace

Relace jsou mocným nástrojem ve světě databází, které nám umožní udržovat konzistentní tabulky, které na sebe navzájem odkazují. MySQL podporu pro relace má jen pro některá úložiště a pokud používáme výchozí MyISAM, tak jsme o tuto možnost ochuzeni. Nicméně phpMyAdmin přesto umožní relace mezi tabulkami používat díky vlastnímu zpracování relací.
Pokud nepoužíváte úložiště podporující relace, je potřeba pro používání internách relací mít v phpMyAdminovi nastaveno používání rozšířených možností. Pokud relace vaše úložiště přímo podporuje (například InnoDB), můžete bez obav tuto část pro tuto chvíli přeskočit, ale přijdete o jiné zajímavé funkce, které jsou na relace navázány a o kterých budu psát v některém dalším článku.

Relace určuje vztah mezi dvěma sloupci v tabulkách. phpMyAdmin stejně tak jako většina relačních databází podporuje přímo jen relace 1:N, relace N:N musí být prováděny přes pomocnou mapovací tabulku.

Vytváření relací je velmi jednoduché – stačí na vlastnostech tabulky kliknout na odkaz Zobrazit relace a dostaneme se na jednoduchou stránku, která nám umožní relace upravit. V horní části můžeme definovat vztahy mezi sloupci, v dolní potom sloupec, který phpMyAdmin zobrazí, pokud budete procházet tabulku, která odkazuje na nějaký řádek z aktuální tabulky. Pro klasické tabulky obsahující identifikátor (na který bude druhá tabulka odkazovat) a například jméno tedy logicky vybereme jméno.

V čem nám zadání relací usnadní práci? Při vkládání nové položky máme na výběr z hodnot v odkazované tabulce, při procházení si můžeme rovnou otevřít odkazovaný řádek a v neposlední řadě nám na stránce s operacemi přibude odkaz na kontrolu integrity. To je bohužel také nevýhoda interních relací, phpMyAdmin nemůže vynucovat korektní editaci databáze a proto může dojít k vytvoření odkazů na neexistující položky.

Tento problém je možné vyřešit jen podporou na úrovni MySQL serveru, tedy použitím úložiště, které toto přímo podporuje, například InnoDB. Ta nám nabídne kromě možnosti výběru sloupce i operace, které se mají provést při vymazání a aktualizaci sloupce. Tím MySQL bude vynucovat integritu všech tabulek a budeme mít o starost méně.

Nativní relace v MySQL jsou podporovány na stejné úrovni jako ty interní phpMyAdmina, takže veškeré funkce můžete používat při obou možnostech. Kromě pomocí při editaci a procházení nám správně nadefinované relace můžou usnadnit i generování dokumentace

Relační algebra

Relační algebra je nástrojem pro manipulaci s relacemi, je to jazyk který pracuje s celými relacemi, operátory relační algebry se aplikují na relace a výsledkem jsou opět relace.

Teoretická východiska dotazovacích jazyků (relační algebra a relační kalkul)

- **RMD**
  - odděluje data, která jsou chápana jako relace, od jejich implementace
  - přístup k datům je symetrický, tj. při manipulaci s daty se nezajímáme o jejich přístupové mechanizmy
  - pro manipulaci využíváme dva silné prostředky - relační kalkul a relační algebru
  - pro omezení redundance dat v relační DB jsou navrženy pojmy umožňující normalizovat relace

- **Definice RDM**
  Mějme množiny $D_1,D_2...D_n$. Z každé vybereme 1 prvek. Tím vytvoříme uspořádanou n-tici. Kartézský součina $D_1xD_2.... x .... D_n$ je množina všech posloupností. Z hlediska databazového systému se množiny **D** označují jako množiny hodnot atributů, tvz. **domény.**

- **Od matematické relace se databázová liší**
  - relace je vybavena pomocnou strukturou, které se říká schéma relace, schéma relace se skládá ze jména relace a jmen atributů a domén
  - prvky domén, ze kterých se berou jednotlivé komponenty prvků relace jsou atomické (dále nedělitelné) hodnoty, tomuto omezení se říká 1.normální forma relací (1NF)

.

projekce
umožňuje potlačit atributy v relaci, umožňuje přejít z relace o **n** sloupcích na relaci o **p** sloucích, kde p<n, může obsahovat i méně řádků, protože duplicitní se nemí vyskytovat.

selekce(restrikce)
operací selekce vznikne vybrání nových řádků z původní relace pomocí logických spojek

spojení
spojení dvou relací vytvoří třetí relaci, výsledná relace obsahuje všechny kombinace které vyhovují zadané podmínce, podmínka vyjadřuje vztah mezi dvěma relacemi,
druhy spojení rovnost, nerovnost, inkluze (spojení s tím rozdílem že do výsledné relace se přidají i nespojené řádky z první (ev. z druhé relace, ev. z obou relací). Pak příslušné atributy nejsou vyplněny, nabývají hodnoty NULL.

příklad:
z relace AUTA vybereme všechna identifikační čísla, názvy a rok výroby aut, jejichž cena je menší než 150 000 kč.
project (**restrict** AUTA **where** cena<150000) **over** ident_a, nazev_a, rok_v, cena **giving** VYSL

Sjednocení - $R \cup S = \{t | t \in R \text{ v } t \in S\}$
pokud jsou n-tice shodné, objeví se pouze jednou
Průnik
Množinový rozdíl R-S
Symetrický rozdíl - nová relace bude obsahovat n-tice obou relací, s vyjímkou těch, které se vyskytují v obou relacích
Kartézský součin - spojuje každou n-tici s každou, kardinalita (počet n-tic) je součin kardinalin vstupních n-tic.

sjednocení : **UNION R and S**
rozdíl : **MINUS R and S**
součin : **CARTESIAN R and S**
projekce : **PROJECT R over Seznam_Atributů** definice: STUDENT[rč,jmeno]
selekce : **SELECTION** : definice STUDENT(jmeno = 'radovan')
průnik : **INTERSECT R and S**
spojeni : **JOIN R and S over Atribut** definice: STUDENT[rč=id]ZAMESTNANEC
podíl : **DEVIDE R by S**

Způsoby zachování referenční integrity
RESTRICT - při mazání a aktualizaci v nezávislém entitním typu se restriktivně vyžaduje, aby v podřízeném entitním typu byla nejdříve vymazána nebo aktualizována svázána instance. Jinak mazání resp. aktualizace v nadřazeném entitním typu je zakázáno.

CASCADE způsobí aktualizace resp. mazání v podřízeném ent. typu, resp. mazání v typu nadřazeném. Tento typ může způsobit kaskádovité mazání velkého počtu instancí.

SET NULL - nastavení cizího klíče na NULL (nejde pokud je to zároveň PK)

Data Query Language - dotazovací jazyky
máme dva typy:
navigační (procedurální, algebrácké), při formuli dotazu je třeba zadat algoritmus jako posloupnost operací prováděných nad relacemi, který zajistí výběr příslušných dat. Navigační jazyky jsou založeny na **relační algebře**

specifikační (neprocedurální, deskriptivní, deklarativní) - požadavky na výběr se zadávají jako prediká, charakterizující výslednou relaci. Výsledek výběru dat je relace, jejíž n-tice splnňují podmínky výběru uvedené ve formuli. Specifikační jazyky jsou založeny na **relačním kalkulu**

Relační kalkul
Relační kalkul vychází z predikátové logiky 1.řádu a v relčních databázích se vyskytuje ve dvou formách. Jedná se o n-maticový a doménový relační kalkul.

Přepisovycí jazyk n-ticového relčního kalkulu:
seznam_hodnot WHERE formule
Seznam hodnot představuje především seznam atributů, spojených n-ticovou proměnnou s danou relací, dále může obsahovat agregační funkce, resp. aritmetické výrazy. Formule se skládá z atomických formulí s unárním predikátem (jména relací s příslušnými n-ticovými proměnnými)

Kvantifikátory v relačním kalkulu : EXISTS,FORALL.

příklad
mějme relace
KNIHKUPEC(JMENO_K,ADRESA...)
KNIHOVNA(NAZEV_K,...)
DODAVA(JMENO_K,NAZEV_K,...)
Najdi jména a adresy knihkupců, kteří dodávají knihy do všech knihoven. Dotaz zapiště v n-ticovém relačním kalkulu.

x.JMENO_K, x.ADRESA where KHIHKUPEC(X) and forall y(KNIHOVNA(y) implies exists z(DODAVA(z) z.JMENO_K=x.JMENO_K and z.NAZEV_K=y.NAZEV_K))

zapiš predchozí příklad pomocí existenčního kvantifikátoru.

x.JMENO_K, x.ADRESA where KNIHKUPEC(x) and not exits y(KNIHOVNA(y) imlies not exists Z(DODAVA(z) z.JMENO_K=x.JMENO_K and z.NAZEV_K=y.NAZEV_K))

příklad
Proveďte spojení relací KNIHA(ISBN,AUTOR,TITUL) a REZERV(ISBN,Č_ČT,D_REZ) včetně projekce všech neduplicitních atributů v relační algebře a to jakz definice,tak využitím jednoduchého přepisovacího jazyka a relačním kalkulu:

KNIHA[ISBN=ISBN] REZERV [ISBN,AUTOR,Č_ČT,D_REZ]

project(join KNIHA and REZERV over ISBN) over ISBN,AUTOR,TITUL,Č_ČT,D_REZ

x.ISBN,x.AUTOR,x.TITUL,Y.Č_ČT, x.D_REZ where KNIHA(x) and REZERV(y) and x.ISBN=Y.ISBN

příklad:
Mějme relaci
STUDENT(ČIS_SKUPINY,JMENO,PŘÍJMENI,MA,BYDLIŠTĚ,...)
formulujte následující dotaz v relační algebře jak z definice, tak použitím přepisovacího jazyka.
Najděte jména a příjmení studentů, kteří chodí do stejné skupiny jako Novák a nemají zkošku z matematiky (v atributu MAT je prázdný řetězec).

STUDENT(ČIS_SKUPINY= (STUDENT(PŘÍJMENÍ='Novák')[ČÍS_SKUPINY] (STUDENT(MAT=''))) [JMÉNO,PŘÍJMENÍ]
project (restrict ČÍS_SKUPINY= project(STUDENT(JMENO='novak') over ČIS_SKUPINY) and MAT='') over jmeno,přijmeni

příklad:
KINO(NAZEV_K,ADRESA)
FILM(JMENO_F,HEREC,ROK)
PROGRAM(NAZEV_K,JMENO_F,DATUM)
zapište v relační algebře z definice i v přepisovacím jazyku dotaz: Najděte název kina a jeho adres, kde dávají film Kolja.

(PROGRAM(JMENO_F='Kolja')[NAZEV_K=NAZEV_K] KINO) [NAZEV_K,ADRESA]

project(join (restrict PROGRAM where JMENO_F='Kolja') and KINO over NAZEV_K) over NAZEV_K,ADRESA

x.NAZEV_K, x.ADRESA where KINO(x) and exists p(PROGRAM(p) and x.NAZEV_K=p.NAZEV_K and NAZEV_F='Kolja')

příklad:
ve kterých filmech hrají kromě jiného všechny filmy s Menšíkem? (relační algebra z definice)

PROGRAM[NAZEV_K,JMENO_F] / FILM(HEREC='Menšík')[JMENO_F]
nebo
PROGRAM[NAZEV_K] - ((PROGRAM[NAZEV_K] x FILM(HEREC='Menšík')[JMENO_F])- PROGRAM[NAZEV_K,JMENO_F])[NAZEV_K]

Definice pojmů
základní relace

- relace odvozená přímo na základě atributů
- v databázovém schématu je základní relace reprezentována tabulkou

odvozená relace

- relace definována na základě jiných relací (což mohou být základní i odvozené relace)
- v MS Access je odvozená relace tvořena dotazem (v MS SQL Serveru pohled)

dotazy (pohledy)
definovány pomocí relačních operátorů (MS Access nabízí pro definici dotazů i grafické rozhraní)

SQL
Structured Query Language (Strukturovaný dotazovací jazyk) sloužící k vyjadřovaní relačních operací

příkaz SELECT
syntaxe příkazu:

```
SELECT <SeznamPolí>
FROM <SeznamMnožinZáznamů>
    <TypSpojení> JOIN <SpojovacíPodmínka>
WHERE <VýběrováKritéria>
GROUP BY <SeznamPolíKSeskupení>
HAVING <VýběrováKritéria>
ORDER BY <SeznamPolíKSeřazení>
```

popis jednotlivých částí syntaxe:

SELECT <SeznamPolí>

seznam obsahuje jedno nebo více polí, která budou tvořit výslednou množinu záznamů, pole mohou být přímo v množinách záznamů, ze kterých jsou vybírána, nebo mohou být vypočtena
- použití v příkazu je povinné

FROM <SeznamMnožinZáznamů>

<TypSpojení> JOIN <SpojovacíPodmínka>

<SeznamMnožinZáznamů> obsahuje seznam tabulek a dotazů (pohledů),na kterých je příkaz SELECT založen,

JOIN definuje vztah mezi množinami záznamů

- použití v příkazu je povinné
WHERE <VýběrováKritéria>

<VýběrováKritéria> omezují data ve výsledné množině záznamů

- použití v příkazu není povinné
GROUP BY <SeznamPolíKSeskupení>

Seskupí záznamy do jediného v případě, že mají v polích ze zadaného seznamu stejné hodnoty

- použití v příkazu není povinné
HAVING <VýběrováKritéria>

Další omezení dat z vráceného pole seskupených záznamů v klausuli GROUP BY

- použití v příkazu není povinné
ORDER BY <SeznamPolíKSeřazení>

Umožňuje třídění množiny záznamů podle polí uvedených v <SeznamPolíKSeřazení>
- použití v příkazu není povinné

Relační operátory

Restrikce

omezení množiny záznamů
pracuje pouze nad jedinou množinou záznamů (tou může být i dotaz založený na mnoha množinách záznamů
k restrikci slouží klausule WHERE :

SELECT *
FROM Tab_Zakaznici
WHERE Zakaznik_Prijmeni = ´Vomáčka´;

(* značí, že mají být vybrány všechny atributy ze seznamu tabulek v klausuli FROM)
výběrová kritéria mohou být jakkoli složitá, kritéria se spojují pomocí operátorů AND a OR

Projekce

- z původní množiny záznamů vrátí pouze vybrané atributy
- za SELECT je uveden seznam polí (atributů), která se zahrnou do výsledné množiny záznamů:

SELECT Zakaznik_Prijmeni, Zakaznik_Jmeno

```
FROM Tab_Zakaznici
WHERE Zakaznik_Prijmeni = ´Vomáčka´
ORDER BY Zakaznik_Prijmeni, Zakaznik_Jmeno;
```

- klausule ORDER BY uspořádá data abecedně podle pole Zákazník_Příjmení a následně podle pole Zákazník_Jméno

Spojení

- nejběžnější případ relačních operací
- propojuje množiny záznamů na základě porovnání polí
- spojení na rovnost (vnitřní):
  - spojení na základě operátoru rovnosti
  - vrátí pouze ty záznamy ve kterých si vzájemně odpovídají hodnoty

```
SELECT Tab_Objednavky.IDObjednavka,
Tab_Vyrobky_Na_Objednavkach.IDVyrobek,
Tab_Vyrobky_Na_Objednavkach.Mnozstvi_Vyrobku,
Tab_Vyrobky_Na_Objednavkach.Jednotkova_Cena
FROM Tab_Objednavky
INNER JOIN Tab_Vyrobky_Na_Objednavkach
      ON Tab_Objednavky.IDObjednavka =
 Tab_Vyrobky_Na_Objednavkach.IDObjednavka
WHERE (((Tab_Objednavky.IDObjednavka)=4))
```

výsledek dotazu vypadá takto:

IDObjednavka	IDVyrobek	Mnozstvi_Vyrobku	Jednotkova_C
4	2	2	3,00 Kč
4	3	10	2,00 Kč

obr 04_01

Theta spojení (vnitřní):

- spojení na základě operátorů <>, >, >=, <, <=
- používají se většinou při porovnávání záznamů jejichž nějaká hodnota je např. větší než průměr hodnot všech záznamů

Vnější spojení (outer join)**:**

- vrátí všechny záznamy, které by vrátilo spojení vnitřní plus všechny záznamy z jedné nebo obou výchozích množin záznamů, přičemž místo chybějících hodnot (vzájemně neodpovídajících) se vypíší hodnoty Null
- existují vnější spojení levá, pravá a plná, směr spojení se rozlišuje podle pořadí v jakém jsou množiny uvedeny v příkazu SELECT

následující příklad tedy v obou případech vrací všechny záznamy z množiny A a z množiny B jen ty, které splňují podmínky uvedené v <podmínka>

```
SELECT * FROM A LEFT OUTER JOIN B ON <podmínka>
SELECT * FROM B RIGHT OUTER JOIN A ON <podmínka>
```

plné vnější spojení vrátí všechny záznamy z obou zadaných množin, a spojí vzájemně záznamy, které splňují podmínky v <podmínka>

```
SELECT * FROM A FULL OUTER JOIN B ON <podmínka>
```

Dělení:

- Z jedné množiny vrátí všechny záznamy, které mají shodné hodnoty se všemi odpovídajícími hodnotami ve druhé množině záznamů
- V příkladu s objednávkami by to znamenalo, že chceme najít jen ty objednávky, na kterých byli objednány všechny výrobky
- Příkaz SELECT jazyka SQL operaci relačního dělení přímo nepodporuje

Množinové operátory

Sjednocení

- spojení množin záznamů za sebou
- výsledek odpovídá situaci, kdy bychom záznamy z množiny B přidali na konec množiny A
- příklad: je nutné získat adresy a telefony všech zákazníků a zároveň všech výrobců

```
SELECT Zakaznik_Prijmeni & " " & Zakaznik_Jmeno AS Jmeno, Zakaznik_Tel
AS Telefon, Zakaznik_Adresa AS Adresa
FROM Tab_Zakaznici
  UNION SELECT Vyrobce_Nazev AS Jmeno, Vyrobce_Tel AS Telefon,
  Adresa_Ulice & " " & Adresa_Mesto & " " & Adresa_PSC AS Adresa
  FROM Tab_Vyrobci;
```

výsledek takového dotazu je na obrázku, pomocí & " " & je možné spojit více atributů do jednoho pole a pomocí „AS“ přejmenovat (takto je upraveno Zakaznik_Jmeno a Zakaznik_Prijmeni do společného pole Jmeno, stejně je provedeno spojení pole Adresa z atributů Adresa_Ulice, Adresa_Mesto, Adresa_PSC)

Průnik
vrací záznamy, které mají společné hodnoty pro obě původní množiny záznamů
využívá se při hledání duplicitních záznamů (při spojení dvou systémů se v různých tabulkách mohou nacházet shodní zákazníci)

Rozdíl
vrací záznamy, které nejsou průnikem pro obě původní množiny
využívá se při hledání „sirotků“, vytváří záznamy, které náleží pouze do jedné množiny záznamů

Kartézský součin
kombinuje každý záznam z první množiny se všemi záznamy z druhé množiny
vytváří se velmi snadno, někdy může vzniknout opomenutím klausule JOIN
následující příkaz vrátí všechny kombinace Výrobců a Zákazníků

```
SELECT Zakaznik_Prijmeni, Vyrobce_Nazev
FROM Tab_Zakaznici, Tab_Vyrobci;
```

Speciální relační operátory

Souhrnné operace

- klausule GROUP BY vytvoří souhrnné údaje
- následující příkaz vrátí množinu záznamů s údajem kolik bylo prodáno kterého výrobku

```
SELECT Tab_Vyrobky.Vyrobek_Nazev,
        Sum(Tab_Vyrobky_Na_Objednavkach.Mnozstvi_Vyrobku)
FROM Tab_Vyrobky
INNER JOIN Tab_Vyrobky_Na_Objednavkach
ON Tab_Vyrobky.IDVyrobek=Tab_Vyrobky_Na_Objednavkach.IDVyrobek
GROUP BY Tab_Vyrobky.Vyrobek_Nazev;
```

- pole uvedená za SELECT musí být uvedena v <SeznamPoliKSeskupeni> v klausuli GROUP BY nebo musí být uvedena jako argument některé agregační funkce (v předchozím příkladě je Mnozstvi_Vyrobku jako argument funkce SUM tedy součet hodnot, Vyrobky_Nazev je uveden v klausuli GROUP BY)

Agregační funkce jazyka SQL

AVERAGE – průměr
COUNT – počet
SUM – součet
MAXIMUM – nejvyšší hodnota
MINIMUM – nejmenší hodnota
Po úpravě předchozího příkazu záměnou funkce SUM za funkci COUNT získáme počet objednávek, ve kterých byl objednán daný výrobek

Záměnou SUM za funkci MAXIMUM zjistíme jaké největší množství daného výrobku bylo objednáno na jedné objednávce (obdobně lze použít i ostatní agregační funkce)

Přejmenování

- slouží k přejmenování celé množiny záznamů (tabulky) nebo jednoho pole (atributu)
- virtuální pole se vypočítávají z hodnot uložených v databázi, fyzicky se nikam neukládají
- definují se přímo v <SeznamPoli> příkazu SELECT

```
SELECT Jednotkova_Cena AS Cena
FROM Tab_Vyrobky_Na_Objednavkach AS Objednavky;
```

Rozšíření

- slouží k definici virtuálních polí
- virtuální pole se vypočítávají z hodnot uložených v databázi, fyzicky se nikam neukládají
- definují se přímo v <SeznamPoli> příkazu SELECT

```
SELECT [Mnozstvi_Vyrobku] * [Jednotkova_Cena] AS Cena_Celkem
FROM Tab_Vyrobky_Na_Objednavkach;
```

Normální formy

Normalizace je odstranění redundantních(opakujících) se dat, omezení složitosti (rozloženít složité relace na dvojrozměrné tabulky) a zabránění tzv. aktualizačním anomáliím (např. abychom smazáním všech knih autora nepřišli o data o autorovi). Což by mělo vést k databázi přehlednější, rozšiřitelnější a výkonnější.
Normalizace by měla vést k vzniku tabulek, které lze snadno udržovat a efektivně se na ně dotazovat. Normalizované schéma musí zachovat všechny závislosti původního schémat a relace musí zachovat původní data, což znamená, že se musíme pomocí přirozeného spojení dostat k původním datům.
Normální formy:

- 1.NF – První normální forma
- 2.NF – Druhá normální forma
- 3.NF – Třetí normální forma
- BCNF – Boyce Coddova normální forma
- 4.NF – Čtvrtá normální forma
- 5.NF – Pátá normální forma

1. normální forma (1.NF)
Relace je v první normální formě, pokud každý její atribut obsahuje jen atomické hodnoty. Tedy hodnoty z pohledu databáze již dále nedělitelné. Například v relaci obsahující data o nějaké osobě budeme chtít mít více telefonních čísel:

Osoba

Jméno	Přijmení	Adresa	Telefony
Jan	Novák	Havlíčkova 2 Praha 3	125789654;601258987;789456123
Petr	Kovář	Svatoplukova 15 Brno	369852147;357951456;963852741
Pavel	Pavel	Papalášova 25 Kocourkov	546789123;123456789;987456123

S takovouto tabulkou by byla spousta problémů, například by se dost špatně prováděly změny čísel, případně vyhledávání podle telefonního čísla.

Aby tabulka byla v 1NF musíme buďto rozdělit atribut telefon do více atributů (pouze za předpokladu, že jsme si jisti, že se množství telefonních čísel nezvýší), nebo oddělit telefoní čísla do samostatné tabulky, což já osobně preferuji, protože je to podstatně flexibilnější řešení:

Osoba

ID	Jméno	Příjmení	Adresa
1	Jan	Novák	Havlíčkova 2 Praha 3
2	Petr	Kovář	Svatoplukova 15 Brno
3	Pavel	Pavel	Papalášova 25 Kocourkov

Telefon

ID_osoby	Cislo
1	125789654
1	601258987
1	789456123
2	369852147

ID_osoby	Cislo
2	357951456
2	963852741
3	546789123
3	123456789
3	987456123

2.normální forma (2.NF)

Relace se nachází v druhé normální formě, jestliže je v první normální formě a každý neklíčový atribut je plně závislý na primárním klíči, a to na celém klíči a nejen na nějaké jeho podmnožině. Z čehož vyplívá, že druhou normální formu musíme řešit pouze v případě, že máme vícehodnotový primární klíč. Zní to poněkud složitě, ale nic na tom není, opět pomůže příklad:

V tabulce zboží v obchodě bude název zboží, výrobce, telefon na výrobce, cena zboží a množství na skladě.

Sklad

Název	Výrobce	Telefon Výrobce	Cena	Množství
Mléčná čokoláda	Milka	+420123456789	30Kč	2500
Oříšková čokoláda	Milka	+420123456789	30Kč	2800
Tyčinka milkyway	Milka	+420123456789	10Kč	7000
Mléčná čokoláda	Orion	+420987654321	25Kč	5800
Oříšková horalka	Horalka	+420897654321	7Kč	4560

Klíčem této relace je kombinace atributů Název a Výrobce. Telefon výrobce ovšem není závislí na celém klíči, ale pouze na atributu výrobce. To by vedlo k aktualizační anomálii a to k té, že pokud by se vymazaly veškeré výrobky od výrobce Milka, ztratilo by se telefoní číslo na výrobce Milka, což není zrovna žádané. Řešením je opět rozpad na dvě tabulky:

Výrobek

Název	Výrobce_ID	Cena	Množství
Mléčná čokoláda	1	30Kč	2500
Oříšková čokoláda	1	30Kč	2800
Tyčinka milkyway	1	10Kč	7000
Mléčná čokoláda	2	25Kč	5800
Oříšková horalka	3	7Kč	4560

Výrobce

Vyrobce_ID	Vyrobce	Telefon
1	Milka	+420123456789
2	Orion	+420987654321
3	Horalka	+420897654321

3.normální forma (3.NF)

V této formě se nachází tabulka, splňuje-li předchází dvě formy a žádný z jejich atributů není tranzitivně závislý na klíči. Jiné vyjádření téhož říká, že relace je v 3.NF, pokud je ve 2.NF a všechny neklíčové atributy jsou navzájem nezávislé.

Opět definice, která zní nesrozumitelně, ale její použití je vlastně jednoduché. Tranzitivní závislost je taková závislost, mezi minimálně dvěma atributy a klíčem, kde jeden atribut je funkčně závislý na klíči a druhý atribut je funkčně závislý na prvním.

Koukám, že jsem tomu opět moc nepomohl, takže nejlepší bude příklad:

Řekněme, že firma chce uchovávat informace o zaměstnancích, takže vytvoříme relaci Zaměstnanec s atributy r.č. (primární klíč), Jméno, Příjmení, Město, PSČ, Funkce a Plat, zbytek adresy vynecháme, protože pro příklad není důležitý.

Zaměstnanec

r.č	Jméno	Příjmení	Město	PSČ	Funkce	Plat
1	Jack	Smith	Jihlava	58601	CEO	150000
2	Franta	Vomáčka	Praha10	10000	Senior Software Architect	80000
3	Pepa	František	Plzeň	10000	Senior Software Architect	80000
4	Pavel	Novák	Kocourkov	99999	Junior Developer	30000
5	Petr	Koukal	Praha10	12345	Database Designer	75000
6	Honza	Novák	Plzeň	12345	Junior Developer	30000

Z této tabulky je vidět kromě závislosti všech atributů na klíči ještě závislost PSČ a Města a závislost Platu na Funkci. Aby jsme si to ukázali pomocí obou vyjádření definic. Závislost r.č -> Město -> PSČ je tranzitivní závislost PSČ na klíči, stejně tak závislost r.č. -> Funkce ->Plat. Pochopitelnější je asi druhé vyjádření, podle něj jsou závislosti Město -> PSČ a Funkce ->Plat přesně ty, které porušují sousloví: "všechny neklíčové atributy jsou navzájem nezávislé". Řešením problému je opět rozpad na více relací, v tomto případě dokonce na 3, protože jsme 3.NF porušily rovnou dvakrát.

Zaměstnanec

r.č	Jméno	Příjmení	Město_ID	Funkce_ID
1	Jack	Smith	1	1
2	Franta	Vomáčka	2	2
3	Pepa	František	4	2
4	Pavel	Novák	3	4
5	Petr	Koukal	2	3
6	Honza	Novák	4	4

Město

Město_ID	Město	PSČ
1	Jihlava	58601
2	Praha10	10000
3	Kocourkov	99999
4	Plzeň	12345

Funkce

Funkce_ID	Funkce	Plat
1	CEO	150000
2	Senior Software Architect	80000
3	Database Designer	75000

Funkce_ID	Funkce	Plat
4	Junior Developer	30000

Boyce Coddova normální forma (BCNF)
Boyce/Coddova normální forma se pokládá za variaci třetí normální formy a dokonce je původní definicí 3.NF tak jak byla publikována v 70 letech. Je vymezena stejnými pravidli jako 3.NF forma, říká, že musí platit i mezi hodnotami uvnitř složeného primárního klíče.

Relace se nachází v BCNF, jestliže pro každou netriviální závislost X -> Y platí, že X je nadmnožinou nějakého klíče schématu R.

Zní to poněkud šíleně, ale ničeho se nebojte, k tomu, aby byla porušena BCNF musí být splněno několik podmínek a to poměrně specifických:

- Relace musí mít více kandidátních klíčů
- Minimálně 2 kandidátní klíče musí být složené z více atributů
- Některé složené kandidátní klíče musí mít společný atribut.

Nejsnáze Boyce/Coddovu normální formu pochopíme s pomocí funkčních závislostí. Boyce/Coddova normální forma v podstatě říká, že mezi kandidátními klíči nesmí být žádná funkční závislost. Jak známo, nejlépe se definice chápou na příkladech, takže mějme relaci adresář:

Původní příklad byl odstraněn, byl chybný, tento jsem si vypůjčil ze script Databázové systémy, Prof. RNDr. Jaroslav Pokorný CSc., Ing Ivan Halška

Adresa

Město	Ulice	PSČ
Praha 10	Černokostelecká	100 00
Jihlava	Žižkova	58601
Praha 10	Vrátkovská	100 00
Brno	Dvořákova	589 74
Praha 6	Chaloupeckého	160 00

V této relaci platí dvě netriviální funkční závislosti:
{Město,Ulice} -> PSČ a PSČ->Město
Protože neplatí Ulice -> PSČ ani Město -> PSČ, tvoří dvojice {Město, Ulice} klíč schématu. Klíčem je ale i {Ulice, PSČ} platí totiž PSČ -> Město, nikoliv však PSČ -> Ulice. Tudíž je {PSČ, Ulice} kandidátním klíčem schématu. Schéma má všechny atributy atomické a nemá žádný neklíčový atribut a tudíž je v 3.NF, ale není v BCNF. Tento fakt vede k tomu, že nelze evidovat města s PSČ bez znalosti Ulice a krom toho jsou v relaci redundantní data, pokud by se evidovalo velké množství ulic v jednom městě, začal by to být problém.

Klasické řešení, rozpad na dvě tabulky. Vzhledem k tomu, že neplatí PSČ -> Ulice, musíme spojit PSČ a Ulice. Výsledkem tudíž budou relace Města(PSČ, Město) a Ulice(PSČ, Ulice)

Město

PSČ	Město
100 00	Praha 10
160 00	Praha 6

PSČ	Město
586 01	Jihlava
Brno	589 74

Adresa

Ulice	PSČ
Černokostelecká	100 00
Vrátkovská	100 00
Dvořákova	586 01
Chaloupeckého	160 00
Dvořákova	589 74

Čtvrtá normální forma (4.NF)

Tabulka je ve čtvrté normální formě, je-li v BCNF a popisuje pouze příčinnou souvislost (jeden fakt). Sice jednoduché vyjádření bez složitých definic, ale poněkud nicneříkající, takže zkusíme jinou definici: " Relace je ve čtvrté normální formě, pokud je v Boyce/Coddově normální formě, a navíc všechny vícehodnotové závislosti jsou zároveň funkčními závislostmi z kandidátních klíčů. " Mno koukám, že jsem tomu moc nepomohl, tak zkusíme definici a příklad ze skript Tvorba datového modelu v prostředí strategických informačních systému, Prof. Ing. Jindřich Kaluža, CSc. : "ve čtvrté normální formě je relace tehdy, je-li v BCNF a všechny vícehodnotové závislosti obsažené v relaci jsou zároveň funkčními závislostmi. Vícehodnotovou závislost atributů lze definovat následovně: V relaci R, která je v BCNF, s atributy A, B, C nastává vícehodnotová závislost atributu B na atributu A právě tehdy, jestliže množina hodnot B přiřazená dvojici hodnot A, C závisí jen na hodnotě atributu A a je nezávislá na hodnotě atributu C."

Tak teď už je to definice přesná a všeříkající, ale bez perfektní znalosti všech použitých pojmů je opět špatně pochopitelná, tudíž příklad si vypůjčím vysvětlení a příklad ze skript Databázové systémy, Vostrovský, Merunka:

Čtvrtá normální forma se zabývá vztahy uvnitř složeného primárního klíč. Pokud je v tabulce složený primární klíč, může se stát, že některé hodnoty tohoto klíče jsou na sobě nezávislé, ale tím, že spolu tvoří klíč, vzniká falešná souvislost mezi těmito hodnotami a nemohou existovat nezávisle na sobě, což není v souladu s modelovanou realitou. 4.NF proto vyžaduje, aby klíč tvořily jen ty hodnoty, které mají skutečnou vzájemnou souvislost.

Mějme relaci zachycující vztah zaměstnance, kvalifikace a úkolu: Pracovní zařazení(Zaměstnanec, Úkol, Kvalifikace)

Pracovní zařazení

Zaměstnanec	Úkol	Kvalifikace
Ing Petr Pastyňák	Tvorba webu	Webdeveloper
Ing PetrPastyňák	Návrh databáze podnikového IS	Database Specialist
Eva Petrželová	Asistentka Ing Pastyňáka	Psaní na stroji
Eva Petrželová	Asistentka Pastyňáka	ECDL
Pavel Mrkvička	Analytik podnikového IS	Aanalyst
Pavel Mrkvička	Analytik podnikového IS	UML

Všechny atributy dohromady tvoří klíč schématu a neexistuje mezi nimi žádná funkční závislost, tudíž je v BCNF a všechno vypadá ideálně, ale není tomu tak. I když se dá předpokládat, že atributy

Kvalifikace a Úkol jsou na sobě nezávislé, tak tabulka neumožňuje zachytit kvalifikaci zaměstnance, který nemá přiřazen žádný úkol (a úkolujte někoho o kom netušíte co umí) a nelze ani úkolovat zaměstnance bez kvalifikace. Krom ztráty informací se rozkladem vyvarujeme i redundance dat. Tudíž je opět nutno tabulku rozdělit a to na dvojici: Kvalifikace (Zaměstnanec, Kvalifikace), Úkol (Zaměstnanec, Úkol).

Kvalifikace

Zaměstnanec	Kvalifikace
Ing Petr Pastyňák	Webdeveloper
Ing Petr Pastyňák	Database Specialist
Eva Petrželová	Psaní na stroji
Eva Petrželová	ECDL
Pavel Mrkvička	Aanalyst
Pavel Mrkvička	UML
Ing Petr Cibula	Project manager
Ing Petr Cibula	RUP Specialist

Úkol

Zaměstnanec	Úkol
Ing Petr Pastyňák	Tvorba webu
Ing Petr Pastyňák	Návrh databáze podnikového IS
Eva Petrželová	Asistentka Ing Pastyňáka
Pavel Mrkvička	Analytik podnikového IS
Jan Celer	Kopání odvodňovacího kanálu

Do rozložených relací jsem záměrně přidal data, která v původní relaci nebyla, ale měla by být. Krásně se tím ukazuje, jak snadné je teď najít project m,anagera na tvorbu podnikového IS, ale zkuste si to v nenormalizované tabulce, když pan Cibula zrovna nemá přidělen žádný úkol.

Pátá normální forma (5.NF)
Relace je v páté normální formě, pokud je ve čtvrté a není možné do ní přidat další atribut (skupinu atributů) tak, aby se vlivem skrytých závislostí rozpadla na několik dílčích relací.

A už je to tu zase, poměrně normálně znějící definice, ale opět docela naprd. Takže zkusíme jinou:

Relace je v páté normální formě jestliže je ve 4NF a nemůže-li být dále bezeztrátově rozložena. Jinými slovy relace, která má n klíčových atributů (n >= 3) a která se rozloží na relace o n-1 klíčových atributech, nemůže být opětovně spojena operací přirozeného spojení do jedné relace, aniž by došlo ke ztrátě informace.

To už začíná být trošku lepší, ale zkusme to ještě jednou trošku jinak:

Pátá normální forma se týká primárních klíčů, které jsou tvořeny nejméně třemi atributy. V případě, že mezi těmito hodnotami v klíči existují párové cyklické závislosti, tak je třeba tyto závislosti extrahovat do samostatných tabulek, ale původní tabulku je v některých případech třeba zachovat!

To byli definice, teď zkusím trošku jiný popis. K porušení 5NF musí opět být splněno několik podmínek a to dost specifických. Relace musí být ve 4NF a musí mít klíč složený z třech nebo více

atributů a mezi nimi musí být párové cyklické závislosti, ale nikoliv funkční, ani multizávislosti, to by nebyla ve 4NF. Typicky se jedná o vztah třech a více tabulek, kde platí vztahy M:N:O:M a tento vztah je vytvořen jednou relací. 5NF řeší redundanci dat a možnou ztrátu závislostí.

Myslím, že příklad opět pomůže. Mějme firmu, která provozuje síť obchodních zástupců strojírenských firem pro celou Evropu. Ta potřebuje vědět, který zástupce zastupuje kterou firmu a v jakých státech a ve kterých státech působí firmy. Předpokládejme, že o Zástupcích, Firmách i Státech máme vytvořeny informační relace a použité hodnoty jsou pouze cizí klíče, kterými řešíme vztahy mezi těmito relacemi. Zdánlivě jednoduché:

Obchodní zastoupení

Zástupce	**Firma**	**Stát**
Antonín Bahel	Siemens	Německo
Antonín Bahel	Siemens	Rakousko
Ctirad Drba	Siemens	Francie
Ctirad Drba	Škoda Plzeň	Rakousko
Antonín Bahel	Škoda Plzeň	Norsko

Problém vypadá na první pohled vyřešeně, ale dle naší definice páté normální formy tomu tak není, neboť zde existují závislosti Zástupce-> Firma -> Sát -> Zástupce a to jsou párové cyklické závislosti. Mohlo by se stát, že s vymazáním obchodního zástupce, by se mohlo ztratit informace o tom, že firma prodává v zemi, kde jí zastupoval pouze ten smazaný zástupce a to je pochopitelně nežádoucí. Stejně tak odebrání firmy může způsobit ztrátu informace o působení obchodního zástupce v některé zemi a to je taktéž nežádoucí. Takže musíme provést rozpad tři relace, které nám pokryjí všechny vztahy.

Pusobi

Zástupce	**Stát**
Antonín Bahel	Německo
Antonín Bahel	Rakousko
Ctirad Drba	Francie
Ctirad Drba	Rakousko
Antonín Bahel	Norsko

Zastupuje

Zástupce	**Firma**
Antonín Bahel	Siemens
Ctirad Drba	Siemens
Ctirad Drba	Škoda Plzeň
Antonín Bahel	Škoda Plzeň

Zastoupeni

Firma	**Stát**
Siemens	Německo
Siemens	Rakousko
Siemens	Francie
Škoda Plzeň	Rakousko

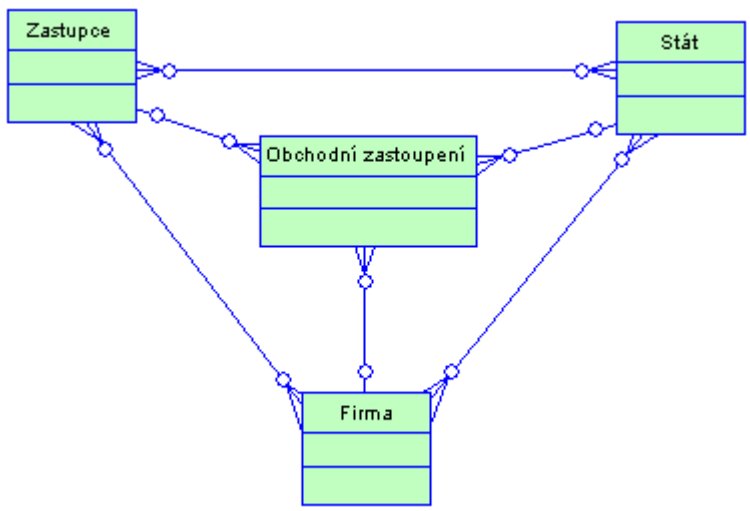
Firma	Stát
Škoda Plzeň	Norsko

Zdá se, že problém je vyřešen, nicméně není. Jedna z definic říká, že relace je v páté normální formě pokud již nelze bezeztrátově rozdělit a menší relace. Důležité je slovíčko bezeztrátově. Protože pokud si spojíme výsledné tabulky pomocí přirozeného spojení, nedostaneme původní výsledek. Dostaneme úplně jiné informace.

Takže jak z toho, tříatributová relace není dobře, tři dvouatributové jsou taky špatně. A co takhle nechat oboje? Ve své podstatě udržuje každá relace jinou informaci. Zastupuje nám říká, které firmy kdo zastupuje, Pusobi říká, kde nám pracuji zástupci a Zastoupeni říká, kam prodávají firmy a ObchodniZastoupeni, říká kdo koho kde.

Pátá normální forma v tomto příkladu nebyla ani tak o špatném převodu konceptu do fyzického modelu databáze, jako spíš o neuvědomění si skutečných vztahů. Ve své podstatě jsem se snažili vymodelovat tuto situaci:

Schéma databázového modelu



Normalizovat je určitě potřeba a čím složitější databáze a čím více dat, tím více je potřeba normalizovat. Ale i tady platí všeho s mírou. Například u příkladu u 3.NF by firma s několika desítkami zaměstnanců asi neměla potřebu dávat PSČ do další tabulky a bylo by to zbytečné. Ale v tabulce zákazníků některého z mobilních operátorů s milióny zákazníků to už význam určitě má.