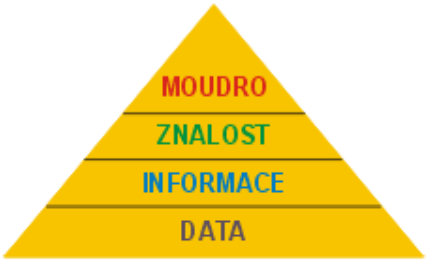
Počítačová podpora ve strojírenství

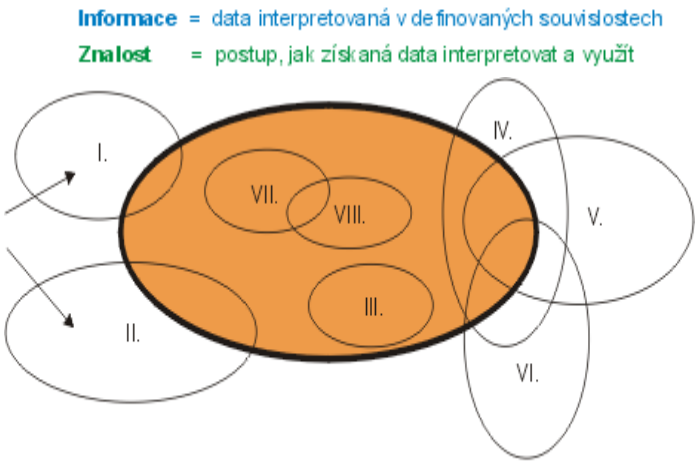
doc. Ing. Pavel Kopeček, CSc.

- Funkční a datové modelování
- Normalizace dat
- Jazyk SQL
- Základy databázového zpracování
- Víceuživatelský přístup a bezpečnost dat
- Lineární datové struktury
- Nelineální datové struktury
- Strojírenské datové struktury
- Příklady datové analýzy

Data a informace

- ***data*** **-** jakékoliv fyzicky (materiálně) zaznamenané údaje, vědomosti, poznatky nebo výsledky pozorovaných procesů reálného světa
- ***informace*** **-** smysluplná interpretace dat a vztahů mezi nimi
  - jsou podkladem pro rozhodování
  - snižují míru neurčitosti
  - jsou časově závisle
  - mají užitnou hodnotu
  - jsou vypovídající schopností dat
- ***znalosti*** **-** schopností abstrakce a generalizace dat a informací
- **datová doména** – množina dat jednoho dílčího uživatele

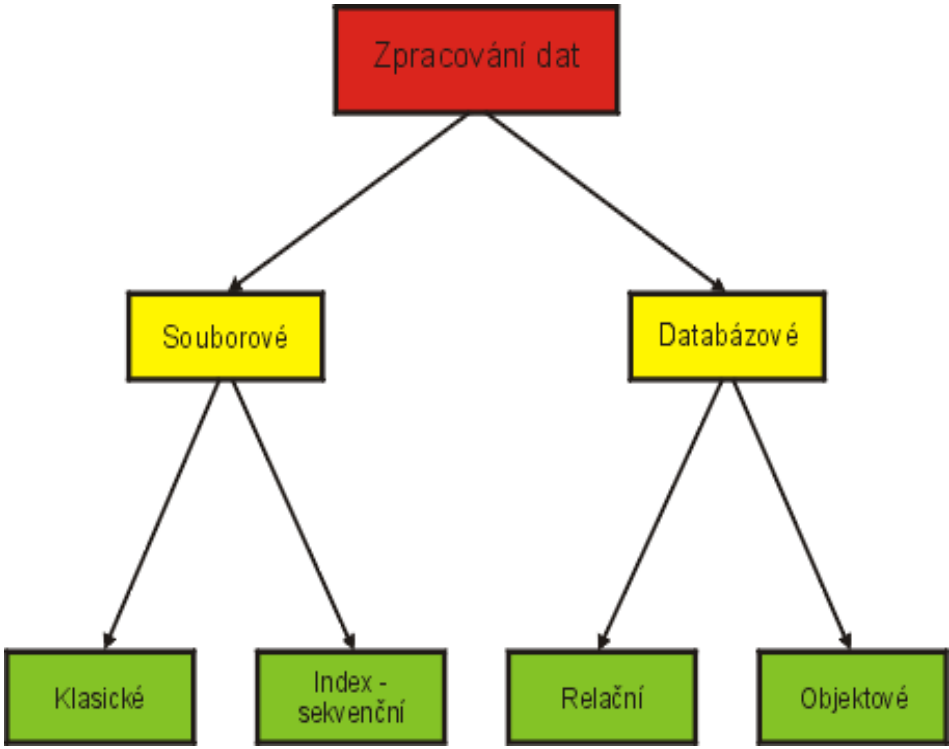




I. - VIII. Datové domény jednotlivých uživatelů

Soubory a databáze

- Uložení dat
  - v operační paměti – při uzavření aplikace nebo vypnutí počítače zanikají
  - persistentní – v souborech a databázích na externích trvalých pamětech
    - Paměti s přímým přístupem – disky, flash
    - Paměti se sekvenčním přístupem – magnetické pásky
- Zpracování dat
  - Souborové (klasické, indexsekvenční)
  - Databázové (navigační, relační, objektové)



Zpracování dat

- Procedurální
  - *„Jakým způsobem získám výsledek ze vstupních dat?“*
  - Zpracování seřazených souborů
  - Zpracování indexsekvenčních souborů
- Databázové
  - *„Jaké vlastnosti mají mít data, která chci vybrat?“*
  - *„Jakou množinu dat chci vložit?“*
  - *„Jakou množinu dat chci změnit?“*
  - *„Za jakých podmínek chci data odstranit?“*
  - Práce se daty s využitím množinových operací – relační databáze
  - Obvykle uživatelsky příjemné grafické rozhraní

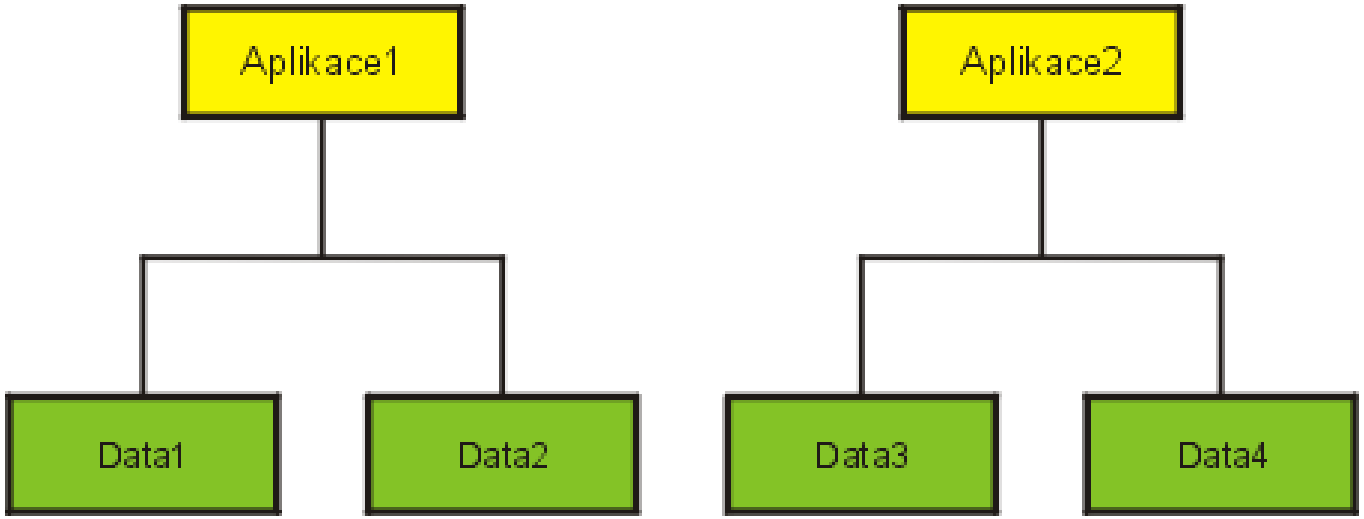
Porovnání práce se soubory a databázemi

Databáze mohou být v jednom souboru (ACCESS – soubor obsahuje tabulky, dotazy, formuláře, sestavy i programové kódy) nebo v několika souborech (FOXPRO – každá tabulka je jeden až tři soubory, kódy jsou zvlášť)

	Výhody	Nevýhody
Soubory	• Programátor může naprogramovat složitější věci šikovně • Aplikace může být rychlejší, programátor může odhadnout dobu zpracování • Uživatel si zakoupí jen spustitelný program, nemusí mít vývojové prostředí	• Programátor musí vše naprogramovat • Vývoj aplikace je pracný a trvá dlouho • Ovládání aplikace je bez obvyklého komfortu
Databáze	• Databázový stroj řadu věcí řeší sám • Je zajištěn víceuživatelský přístup • Data jsou lépe zabezpečena • Vývoj aplikace je rychlejší • Uživatelský vzhled aplikace je přívětivý a moderní	• V řadě případů si uživatel musí koupit databázový program • Zpracování mohou být pomalejší, zrychlení aplikace provádí specialista

Vývoj technologie zpracování dat

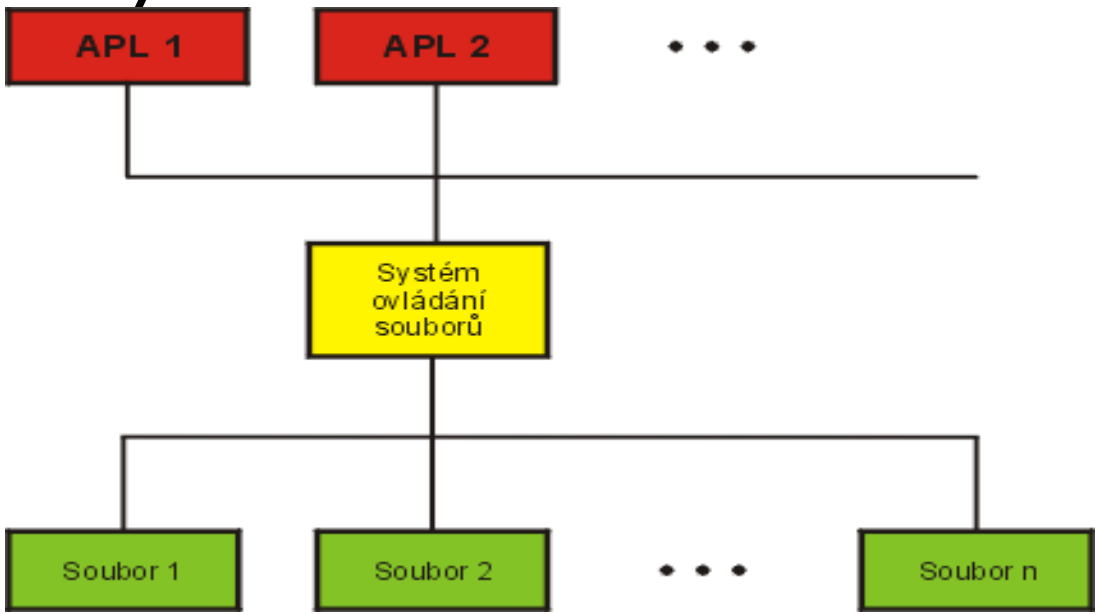
- 1. Aplikační program obsahuje popis dat i vlastní algoritmus jejich zpracování.



Programy mohou přitom pracovat nad více množinami dat.
Data jsou většinou ve formě souborů, s nimiž se přímo jednotlivě pracuje pomocí speciálních příkazů jazyka.
Programy jsou psány v jazycích 3. generace (Cobol, Pascal, C, Ada, VB …).

Vývoj technologie zpracování dat

- II. Systém ovládání souborů.

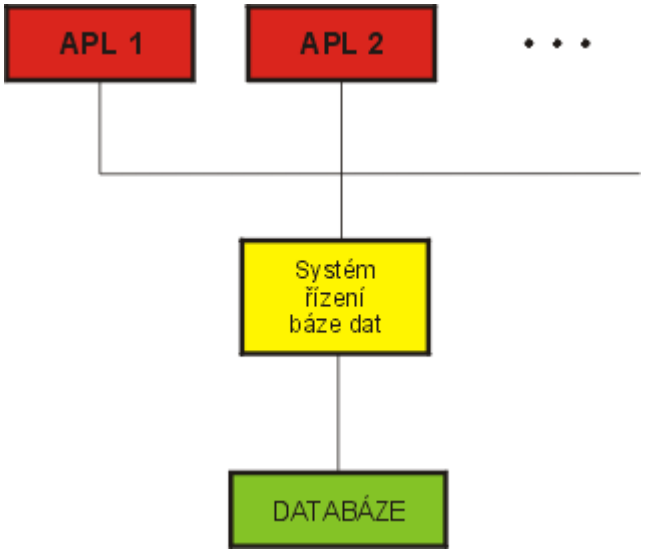


Aplikace nepřistupují k datům přímo, ale prostřednictvím tzv. systému pro ovládání souborů (SOS).
SOS je běžnou součástí operačního systému (OS) a umožňuje programovacím jazykům poměrně komfortní přístup k datům.

- SOS umožňují i současný přístup několika uživatelů k týmž souborům dat.
- Popis dat je stále v aplikačních programech.
- Stále se objevují stejná data v různých souborech pro různé aplikace (redundance).
- Aktualizace dat není centrálně řízena (nebezpečí porušení konzistence dat).
- Nejmodernější SOS mají již více či méně znaků databázového zpracování.

Vývoj technologie zpracování dat

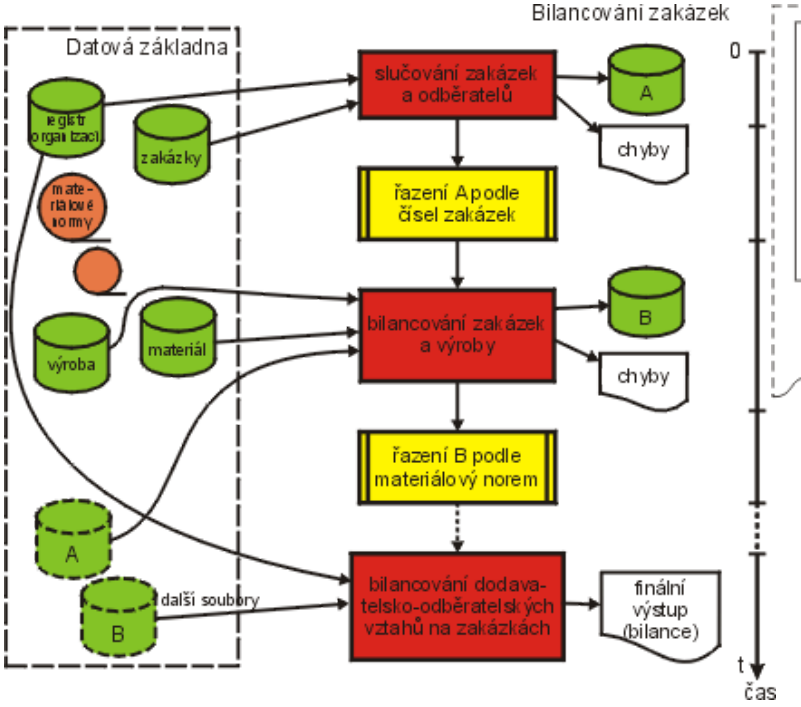
- III. Databázová technologie zpracování dat.



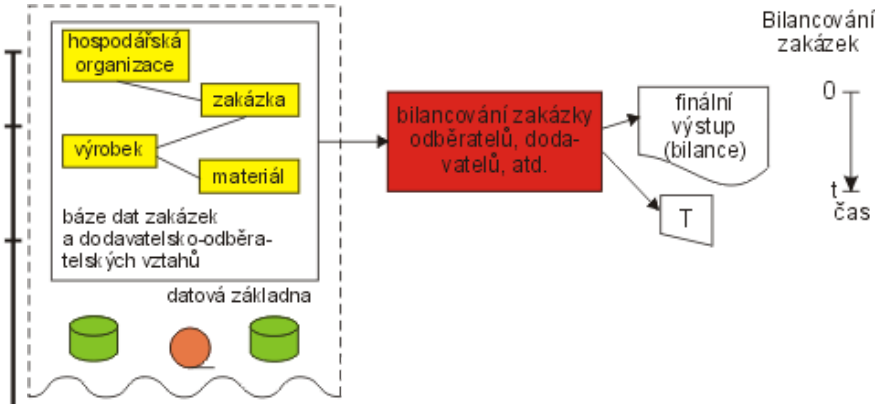
- Data jsou uložena v databázi, kde jsou jednotně popsána.
- Popis dat není součástí programů. Bývá uložen v tzv. slovníku dat, který může, ale nemusí být součástí databáze.
- Běžný uživatel má přístup k datům pouze přes zvláštní software, kterému se říká Systém Řízení Báze Dat (SŘBD).
- SŘBD data logicky centralizuje a sjednocuje práci s nimi.
- Aplikační programy jsou nezávislé na fyzickém uložení dat (ta mohou být uložena i na různých počítačích).
- Jednotný přístup k práci s daty přináší minimalizaci redundance, případně další výhody.

Vývoj technologie zpracování dat

Souborové zpracování

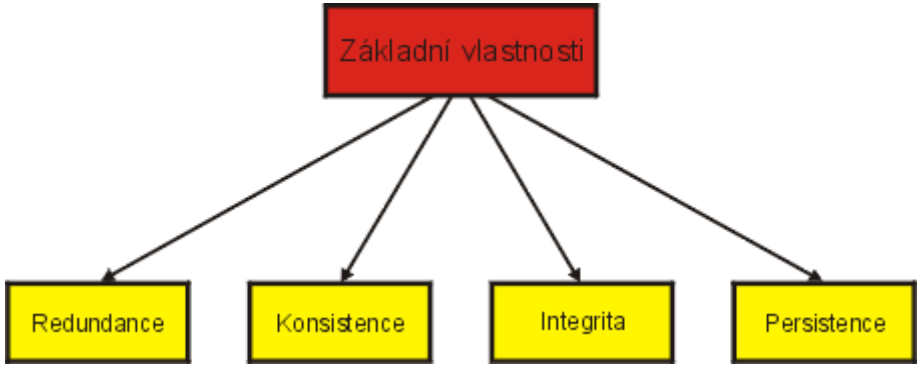


Databázové zpracování



Databanková technologie automatizovaného zpracování dat
T - terminálové zařízení

Základní vlastnosti z oblasti zpracování dat.



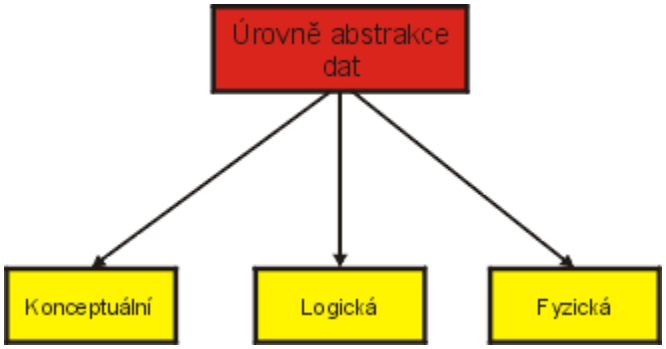
Redundance dat.
Redundancí je vícenásobné uložení dat. Existuje snaha ukládat data neduplicitně nejlépe s minimální redundancí. Jistá redundance je nutná pro zajištění vazeb (např. mezi jednotlivými tabulkami).
V některých případech může vhodná redundance podstatně urychlit zpracování.

Konzistence dat.
Konsistence dat znamená, že údaje v systému uložení dat si neodporují.

Integrita dat.
Integrita dat vyjadřuje soulad dat s popisovaným úsekem reálného světa v čase.

Perzistence dat.
Přetrvávání dat bez ohledu na to, zda se s nimi pracuje.

Úrovně abstrakce při pohledu na data



Konceptuální úroveň

• Na data se pohlíží jako na něco, co stojí mimo výpočetní techniku, tj. není momentálně podstatné, jakými konkrétními prostředky budou data zpracována.
• Pro formální popis budoucí uživatelské aplikace se vytváří konceptuální schéma.
• V souladu se zásadami Softwarového inženýrství se návrh nejčastěji provádí postupem shora dolů, méně často zdola nahoru.

Logická úroveň

• Na data se pohlíží z hlediska prostředků, které mají k dispozici aplikační programátoři v komerčních systémech řízení souborů nebo bází dat.
• Data zpracovávají konkrétní jazyky nebo programovací systémy (Cobol, SQL, …), které pracují s daty ve formě více či méně sofistikovaných datových struktur.
• Konceptuální schéma je na logické úrovni zobrazeno jako logické schéma.

Fyzická úroveň

Na data se pohlíží s ohledem na jejich optimální uložení na konkrétním nosiči se snahou
• minimalizovat potřebnou kapacitu vnitřní i vnější paměti
• minimalizovat čas zpracování I/O operací a vyhledávání dat
• dosáhnout co největší přenositelnost (portabilitu) z jednoho výpočetního systému na jiný.

- Základní pojmy

• **Datová základna** …všechna data, týkající se objektu našeho zájmu, např. společný datový fond podniku. Jsou zde i data, která jsou mimo počítačové techniky.

• **Databanka** … logicky propojená množina dat, zpravidla udržovaných na rychlých nosičích.

• **Databáze** … část datové základny, obsahující i některé informace, potřebné ke snadné manipulaci s daty (relace mezi tabulkami, informace o indexování, obslužné procedury, dotazy SQL, aj.).

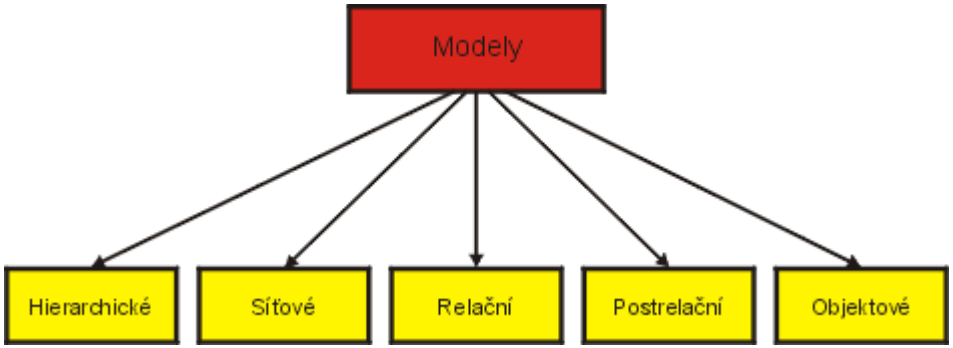
• **Systém řízení báze dat** (SŘBD) … softwarový balík, obsahující prostředky pro tvorbu a manipulaci s daty v databázi. Někdy jsou součástí SŘBD i prostředky pro tvorbu rozhraní s uživatelem.

Databázový systém je spojení databáze se systémem řízení báze dat, což lze schematicky znázornit jako

Databázový systém = Databáze + SŘBD

Databázové modely

- Hierarchické - stromové
- Síťové
- Relační
- Specializované - post relační
- Objektové



Hlavní výhody databázové technologie

- data jsou uložena pouze **jednou** (s výhradou),
- vysoký stupeň **provozního zabezpečení,**
- pružné a rychlé řešení uživatelských **změn,**
- snížení **náročnosti udržování** datové základny,
- možnost **automatického generování** programových kódů využitím vzájemných vazeb.(např. objednávky - dodávky do skladu),
- sémantické a formátové **sjednocení obsahu** datové základny,
- celkové **snížení počtu** aplikačních programů,
- přenos **nároků z aplikačního** programování na **technické a standardní** programové vybavení,
- **úspora práce a času** centralizovanou správou a údržbou datového fondu.

Další požadavky na databázové zpracování

- **poskytovat data:**
  - ve vhodné **formě,**
  - ve vhodném **čase,**
  - na vhodném **místě,**
- **transakční zpracování** (zpracování, které požadovanou změnu provede buď ve všech souvislostech nebo ji neprovede vůbec),
- **archivace** dat,
- kontrola **integrity** (nedojde k rozporu v uložených datech),
- **obnovení** stavu databáze po havárii systému,
- kvalitní **uživatelské rozhraní,**
- silné **dotazovací** možnosti,
- práci v síti s **víceuživatelským** přístupem (více uživatelů pracuje nad stejnými daty, např. rezervace místenek, materiálu pro zakázky),
- **distribuované** zpracování dat (uložení a zpracování dat na různých počítačích).

Účelový a zdrojový přístup

- Účelový přístup
  - Účelový přístup vychází z požadavku na výstupní data a hledají se algoritmy, jak tato data vyrobit a jaká vstupní data tyto algoritmy potřebují.
- Zdrojový přístup
  - Zdrojový přístup směřuje k hlavnímu cíli (uspokojení informačních požadavků řízení) ne přímo, ale přes dílčí cíl, čímž je vytvoření společného integrovaného zdroje dat. Ideálem je vytvořit datový model reality.

Vlastnosti účelového přístupu

- Účelový přístup vychází z požadavku na výstupní data a hledají se algoritmy, jak tato data vyrobit a jaká vstupní data tyto algoritmy potřebují.

- Nevýhody účelového přístupu:
  - Protože se účelově vytvářejí jednotlivé aplikace (části informačního systému organizace), vznikají v organizaci jakési ostrůvky automatizace, které se velmi obtížně propojují.

- Výhody:
  - Jelikož se vyžadují pouze data, potřebná pro jeden účel, je jich poměrně málo.
  - Jejich pořízení je relativně levné.
  - Realizační doba projektu je kratší.

Vlastnosti zdrojového přístupu

- Je reakcí na nevýhody účelového přístupu .
  - Zdrojový přístup směřuje k hlavnímu cíli (uspokojení informačních požadavků řízení) ne přímo, ale přes dílčí cíl, čímž je vytvoření společného integrovaného zdroje dat. Ideálem je vytvořit datový model reality.
- Nevýhody zdrojového přístupu:
  - Pořídit zdrojovou databázi bývá dražší (hardware i software).
  - Kvůli nižší informační gramotnosti managementů se tvorba zdrojové databáze někde těžko prosazuje.
  - Výsledky se projevují později než v účelovém přístupu.
- Výhody:
  - Nevznikají ostrůvky automatizace.
  - Moderní prostředky umožňují uspokojovat i měnící se požadavky na funkce databázového systému.
  - Tento přístup je zvláště vhodný pro budování IS v prostředí, které je tzv. analyticky nestabilní (existuje předpoklad, že požadavky na funkce IS se budou často, i během jeho realizace měnit).
  - Zdrojovou databázi je možno využít i pro moderní principy vytěžování informací z dat (OLAP, data warehousing, datamining, aj.).

Uživatelé databázových systémů

- **Systémoví pracovníci**
  - **Správce** (administrátor) databáze nese odpovědnost za autorizovaný přístup k databázi. Je to podle velikosti systému buď jednotlivec, nebo specializovaný útvar (u nejmenších systémů může chybět). Je schopen zasahovat do databáze na všech (i nejnižších) možných úrovních. Plní konkrétní požadavky zpravidla střední úrovně řízení.

  - **Aplikační programátoři** jsou pracovníci, kteří vytvářejí uživatelské aplikace. Během implementace IS jich může být více (i externích pracovníků zavádějící firmy), během rutinního provozu IS mohou i chybět. Jsou schopni upravovat uživatelský interface a vytvářet např. parametrizované formuláře pro běžné uživatele, kteří většinou nedokáží aktivně používat jazyk SQL, ale i plnit požadavky nejvyšší úrovně řízení, což bývají nejčastěji funkce OLAP.

Uživatelé databázových systémů

- **Manažéři střední úrovně řízení**
  - Tato skupina bývá většinou vedle svého odborného zaměření natolik "informaticky gramotná", že rozumí filozofii IS a je schopna jej aktivně využívat, eventuálně formulovat požadavky na jeho změny či úpravy pro aplikační programátory, a to jak směrem na nižší úroveň (běžní pracovníci), tak na vyšší (vrcholové řízení). Měli by dokázat formulovat a využít i jednodušší dotazy SQL i používat funkce systému OLAP.

Uživatelé databázových systémů

- **Manažéři vyšších a taktických úrovní řízení**
  - Tato poměrně úzká ale velice důležitá skupina pracovníků se zajímá většinou jen o strategické a taktické informace, které ji mohou poskytnout různé typy MIS (manažerský informační systém). Různé typy funkcí OLAP, které využívají technologie datawarehouse a datamining pro ně parametrizují specializovaní pracovníci. Ideální technikou pro tuto úroveň bude přímá hlasová komunikace s počítačem, která je zatím ve stádiu vývoje.

Uživatelé databázových systémů

- **Běžní výkonní pracovníci - naivní, parametričtí uživatelé.**
  - Jsou to uživatelé, jejichž hlavní pracovní náplní je jiná činnost než tvůrčí práce s DBS. Ten používají jako prostředek ke své činnosti a vyžadují jednoduchou, účelnou, spolehlivou a snadno pochopitelnou činnost DBS. Zkušený implementátor IS tyto lidi "hýčká" a snaží se co nejlépe splnit jejich (často kuriózní) požadavky nebo jim je diplomaticky rozmluvit. Tato skupina totiž v rozhodující míře rozhoduje o úspěšnosti nasazení celé aplikace. Nepřijme-li běžný výkonný pracovník informační systém, je mnohdy úsilí celého implementačního týmu marné. Je to jeden z důvodů, proč (jak uvádí literatura) je skoro celá polovina nasazení IS v podnicích buď částečně nebo zcela neúspěšná.

Projekční technologie

- Každá technologie musí být podložená metodicky a obsahovat tyto prvky:
  - **Algoritmy** řešení
  - Mechanismy **kontrol**
  - Prostředky tvorby **dokumentace**
  - Vhodné automatizované vývojové **nástroje**

- Projekční technologie musí splňovat tyto požadavky:
  - **Obecnost**
  - **Kompletnost** (zahrnuje celý řešený problém)
  - **Ucelenost** (např. společný datový model)
  - **Řešitelnost** (algoritmizovatelnost)
  - **Ověřitelnost** (opravné a kontrolní kroky
  - **Jednoduchost** a mentální zvládnutelnost
  - **Interaktivnost** (požadavek poslední doby)

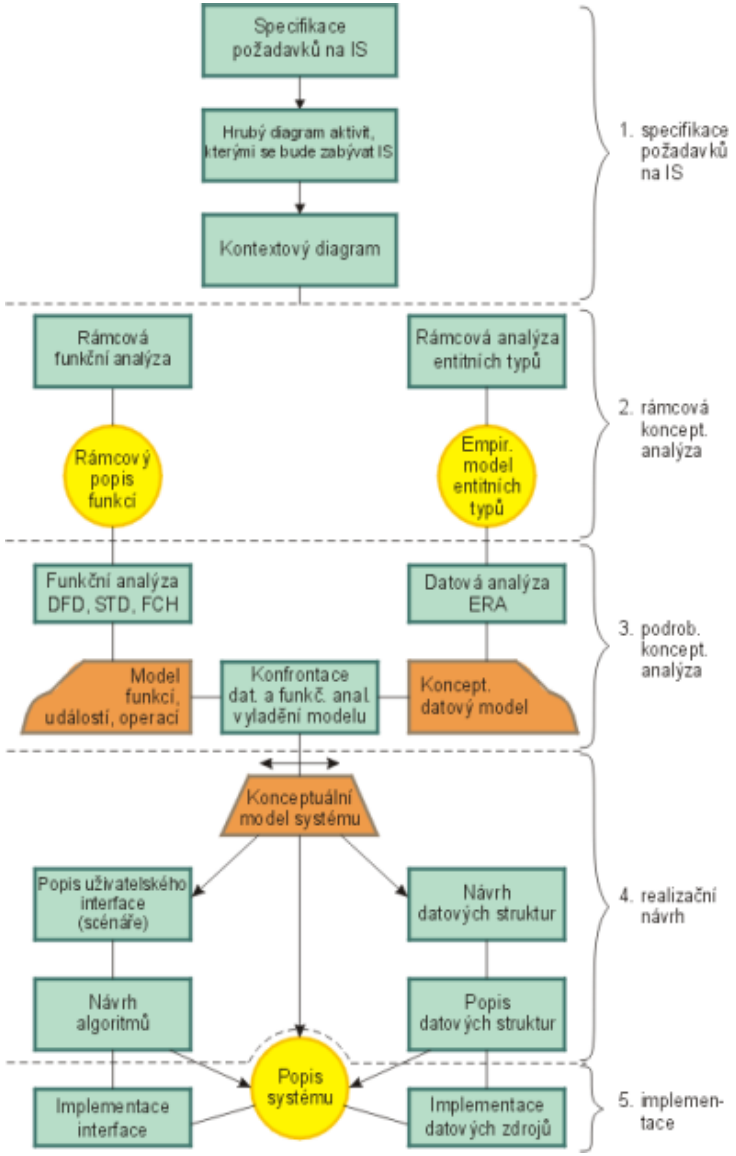
Datová analýza

- Metody datové analýzy (DA) se zaměřují na:
  - vymezení obsahu datové základny
  - návrh její struktury
  - návrh její realizace.
- Nástrojem DA jsou různé formy datových modelů (diagram entit a relací, výskytový diagram, stavový diagram).
- Metody DA vycházejí z analýzy reálných objektů a procesů a hledání možností jejich optimálního zobrazení v použitém datovém modelu a analýzy informačních potřeb a hledání možností jejich souhrnného uspokojení.

Funkční analýza

- Metody ***funkční analýzy*** *(FA)* se zaměřují na použití dat v algoritmech k uspokojování informačních potřeb.
- Cílem FA jsou algoritmy zpracování dat. Podmínkou realizace těchto algoritmů je existence vstupních dat.
- Popisují se **datové toky**.
- Východiskem FA je globální specifikace požadavků na automatizovaný informační systém a pomocí funkční dekompozice směřovat k algoritmům. Často se používají různé softwarové prostředky typu CASE (SDW, Power Designer, atd.).

Kombinace funkční a datové analýzy



Etapy tvorby informačního systému

- *ANALÝZA*
  - Analýza popisuje reálný svět a pomocí abstrakcí jej zjednoduší. Je nezávislá na technických a konkrétních programových prostředcích.
- *DESIGN (návrh)*
  - Při návrhu se zvolí určité ***paradigma*** pro následující implementaci (např. relační, strukturovanou, objektovou).
- *IMPLEMENTACE*
  - Na začátku implementace se zvolí konkrétní implementační nástroj, např. Pascal, C++, Java, Visual basic, Access, Oracle, FoxPro,…

Modely

- Model popisuje systém z určité stránky.
- Je tvořen tzv. ***konstrukty***, čili základními prvky.
- Často mluvíme o modelu, ale máme na mysli diagram (schéma).
- Každá metoda může zahrnovat různé typy modelů podle hlediska a potřeb účelu, ke kterému model slouží, např.:
  - **konceptuální** model (abstrakce reálného světa),
  - **databázový** model,
  - **uživatelský** model (z hlediska uživatele - interface),
  - **fyzický** model (přizpůsobený konkrétní situaci).
- Modely mají různé vlastnosti, které mohou zlepšovat nebo zhoršovat jejich funkčnost nebo srozumitelnost:
  - **Grafika**, množina konstruktů (jak se kreslí - notace).
  - **Modularita** (jestli lze moduly skládat, hierarchicky řadit, …).
  - **Redundance** (nadbytečnost informací) by měla být minimální.
  - **Srozumitelnost** (Syntaxe, čitelnost, přehlednost). Někdy se nevyhneme pro zlepšení srozumitelnosti jistému zvětšení redundance.

Kontextový diagram

- **Kontextový** diagram (KD) popisuje na nejvyšší úrovni vztah systému k jeho okolí. Je to základní diagram, ze kterého vychází další analýza funkcionality systému!
- Je to nesmírně důležitý diagram - odpovídá tomu, co chce od systému uživatel.
- Získá se studiem událostí (events), které mohou nastat mezi systémem a subjekty v jeho okolí. Každá událost mezi systémem a jeho okolím vyvolá konkrétní větší nebo menší přenos informace, tedy dat (někdy i materiální přenos).

- Událost může být různého typu a může být vyvolána:
- **vstupem** do systému typ **F (Flow oriented)**
- uplynutím nějakého **času** typ **T (Temporar)**
- nějakým řídícím nebo informačním **signálem** typ **C (Control) -(**např. stisknutí tlačítka na formuláři, příjem signálu od nějakého čidla nebo ochrany, požární signál, atp.

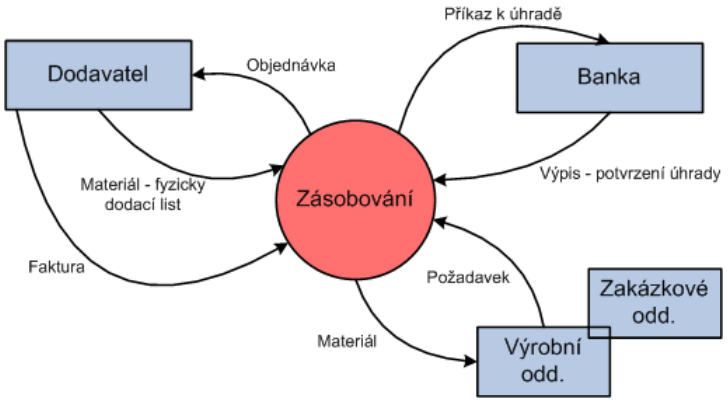


Diagram datových toků

- **Data Flow Diagram** (DFD, Diagram datových toků) je jeden z nástrojů pro modelování funkcí systémů (zejména informačních systémů). Pomocí DFD lze modelovat celé organizace, slouží tedy i jako nástroj podnikatelského a strategického plánování. DFD je součástí strukturované analýzy a návrhu systémů.

Základní vlastnosti DFD

- Navazuje na **kontextový** diagram.
- Popisuje **vnitřní funkcionalitu** systému na základě analýzy toků dat mezi interními funkce systému navzájem a s okolím systému.
- Popisuje **datová úložiště** (datastory) systému.
- Datová úložiště by měla být výsledkem souběžně prováděné analýzy metodou ER diagramů.
- Zachycují **tok informací** v systému mezi jednotlivými procesy (funkcemi), které je zpracovávají.
- Tokem informací se zde myslí, jak vydávání nebo přijímání nějakých signálů, tak výměna dokumentů (neříká se v jaké formě) mezi terminátory.
- DFD diagramy jsou velice **vhodné pro dávkové zpracování dat**, dnešní interaktivní aplikace je poněkud problematické popsat pouze DFD diagramem.
- DFD diagram **je hierarchický**, nejvyšší **úroveň 0 vychází z kontextového diagr**amu.
- DFD vznikl původně v ekonomické oblasti pro analýzu **toků dokumentů** v organizaci (špatnou analýzou toků může dojít ke vzniku zbytečných dokumentů). Jako příklad je možno uvést následující fiktivní scénku, jejímž důsledkům DFD pomůže zabránit:

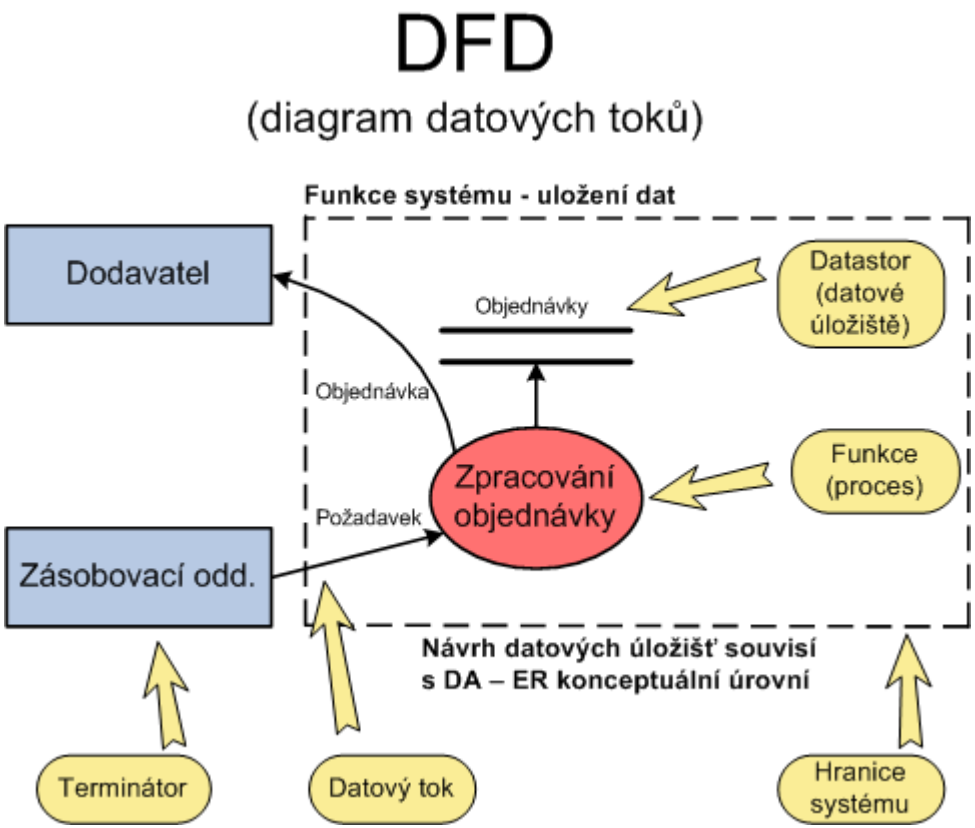
Notace diagramu datových toků

Notace DFD

Existují různé typy notací, lišící se v detailech:
DeMarco, Yourdon, Gane-Sarson, ...

Terminátor	Dodavatel	Představuje subjekt v okolí systému (z KD).
Tok informací	Požadavek →	Představuje tok dat → informací mezi terminátory, procesy a datastory.
Proces (funkce)	Zpracování objednávky	Představuje funkci (proces) systému, který zpracovává informace.
Datastor (datové úložiště)	Objednávky	Místo pro ukládání dat (databáze).

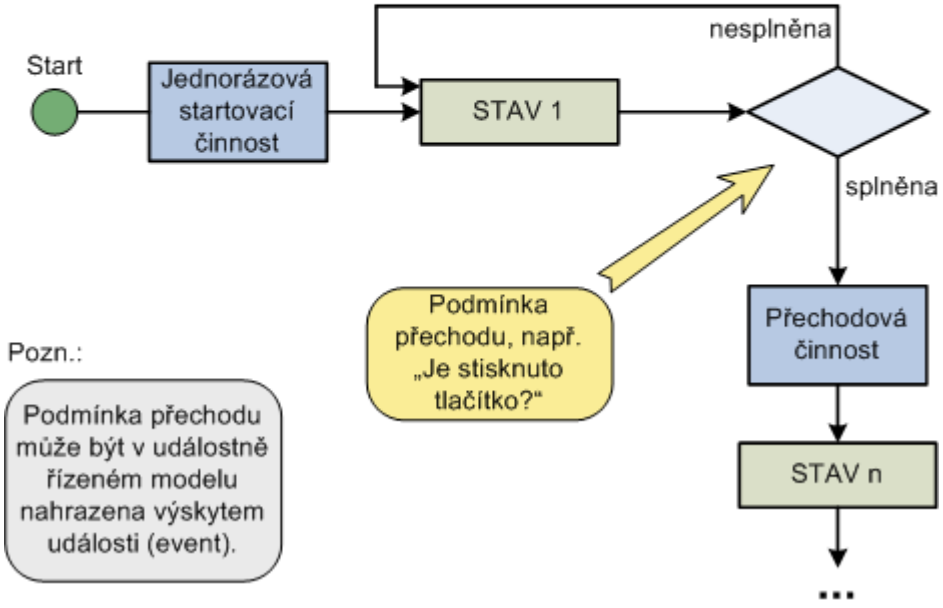
Příklad diagramu datových toků



Stavový diagram

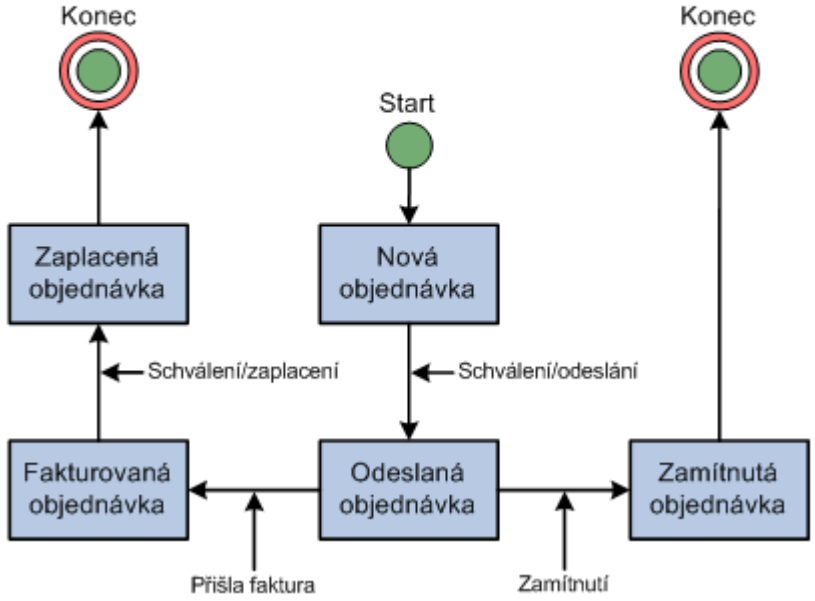
Způsob záznamu (notace) STD

Vyjádření filozofie STD pomocí klasického vývojového diagramu by mohlo vypadat následovně:



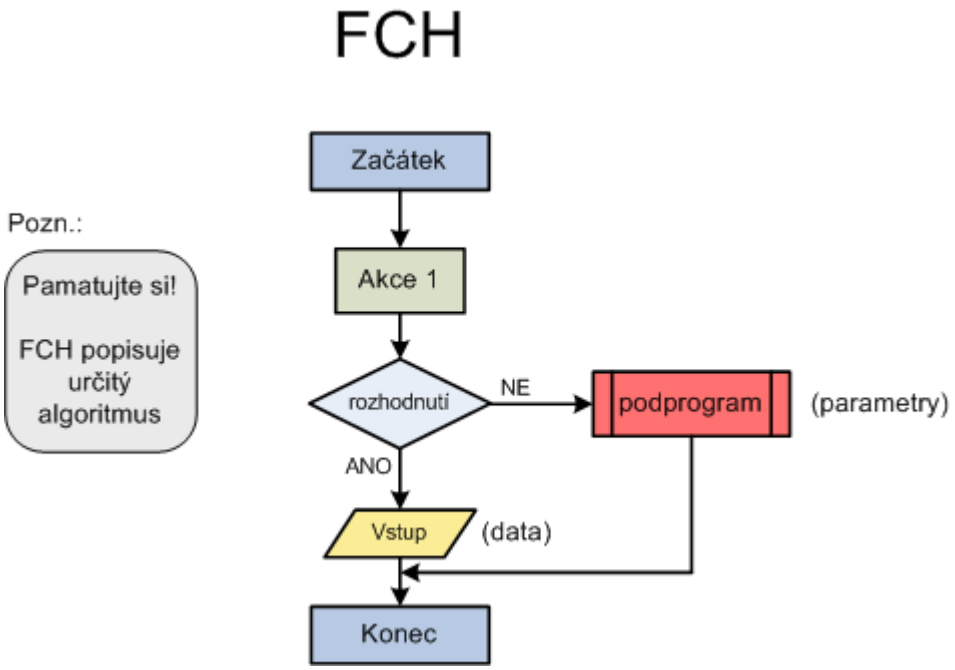
Jiný příklad stavového diagramu

Příklad zápisu STD pro různé stavy objednávky



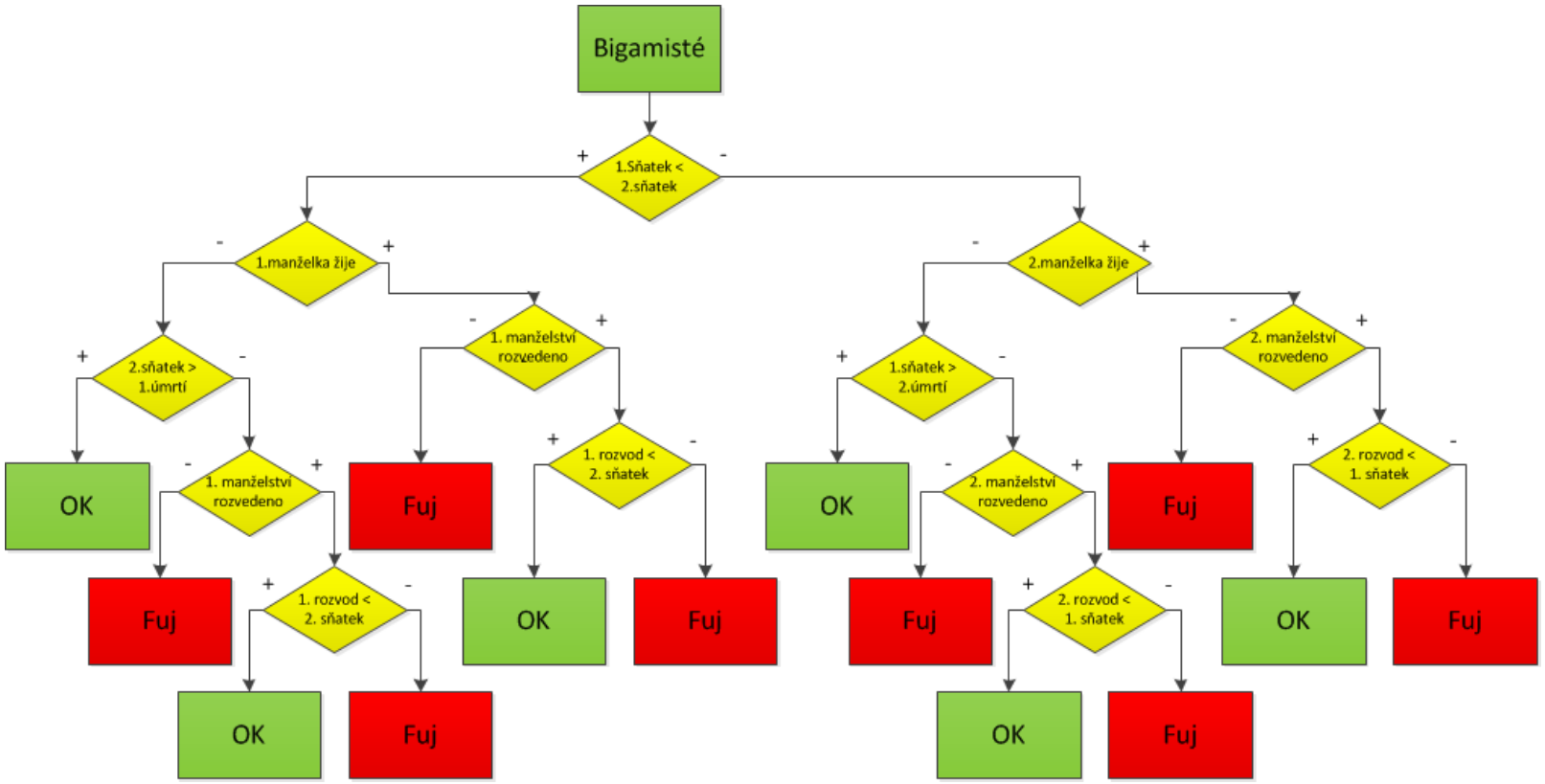
Vývojový diagram

- Hlavní vlastnosti FCH lze shrnout do těchto bodů:
- zachycuje určitý **algoritmus**,
- lze ho použít pro **popis určité funkce** - například metody v nějakém objektu nebo akce v STD,
- je dobře známý -stále se ještě učí v základech programování, protože je velmi dobře **srozumitelný** a je blízký inženýrskému myšlení,
- ve **strukturovaném** programování je nahrazován **struktogramy**, které jsou však pro popis drobnějších záležitostí méně přehledné.



Vývojový diagram pro rozhodování

- Pro příklad v předmětu Databáze v CIM byl použit diagram pro rozhodování, kdo je bigamista. Z databáze byly vybrány dvojice, které měli mezi sebou uzavřené manželství.



Konceptuální modelování

- **Konceptuální modely** jsou pokusem umožnit vytvoření popisu strukturu dat v databázi, tj. ***konceptuálního schématu***, nezávisle na fyzickém uložení databáze. Tento popis by měl co nejvěrněji vystihovat konceptuální pohled člověka na danou část reálného světa. Někdy se na sémantické úrovni uplatní relační model dat, jindy je realita tak složitá, že i nejkonceptuálnější modely jsou pro ni příliš hrubé. Limitním případem konceptuálních modelů je přirozený jazyk.
- ***Sémantické modely*** slouží obvykle k vytvoření schémat s následnou transformací na ***databázové schéma***.
- Informace, která se přitom ztrácí, by měla být "doprogramována" v rámci daného SŘBD na databázové úrovni. Ideální by bylo spojit sémantickou a databázovou úroveň v jednu. Takové *SŘBD* jsou dnes již k dispozici v objektové technologii. Říká se jim **objektově orientované** *SŘBD (OOSŘBD)*.

E-R konceptuální model

- Konceptuální model zahrnuje **entitní typy** (pracovník, faktura, vyráběný díl) a jejich **vztahové typy** (vyrábí se na, obsahuje, skládá se)
- Vztahový typ je obvykle binární (vyjadřuje vztah mezi dvěma entitními typy (díl – techn. operace) nebo stejným entitním typem ve dvou rolích (kusovník)

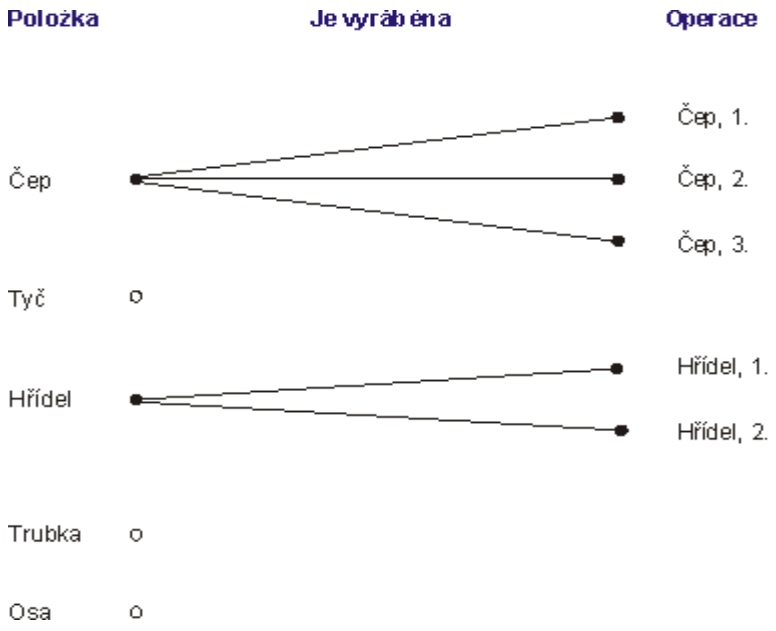
- Příklad na vztah studijního předmětu a studenta:

Kardinalita, parcialita

- **Kardinalitou** (násobností ve vztahu) rozumíme, kolik entit vstupuje do vztahu (př. díl má v postupu jednu nebo více operací, manžel má jednu manželku)
- **Parcialitou** (povinným/nepovinným členstvím ve vztahu) rozumíme, zda entitní typ vždy vstupuje do vztahu (př. předmět nemusí mít zapsán žádný student, každá technologická operace musí patřit k jednomu dílu a je předepsána na jedno pracoviště)

Výskytový diagram

- Výskytový diagram názorným příkladem ukazuje kardinalitu i parcialitu (je vhodnou formou komunikace analytika s uživatelem)
- Příklad na výskytový diagram:
  - Některé položky jsou nakupované (trubka, osa, tyč) a tudíž nemají žádnou operaci (nepovinné členství)
  - Jiné položky (čep, hřídel) mají více operací (kardinalita `1:N`)
  - Každá operace musí patřit nějakému dílu (povinné členství)

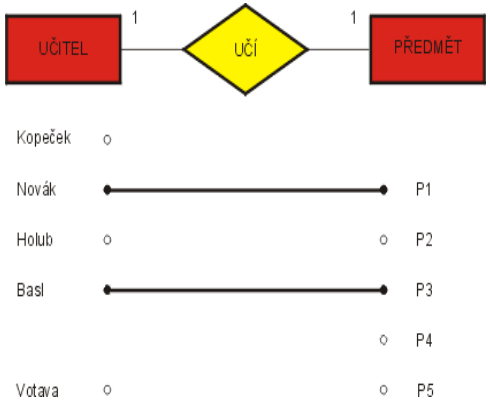


Základní typy kardinalit

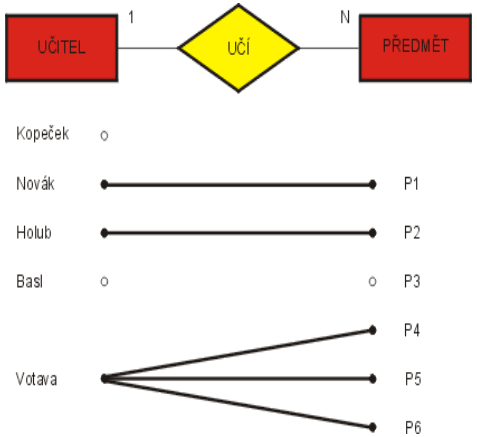
- Vztah `1:1`
  - učitel učí maximálně jeden předmět
  - předmět je vyučován maximálně jedním učitelem
- Vztah `1:N`
  - Učitel může učit více než jeden předmět
  - Předmět je učen maximálně jedním učitelem
- Vztah `M:N`
  - Učitel může učit více než jeden předmět
  - Předmět může být učen více než jedním učitelem

Příklady ER a výskytových diagramů

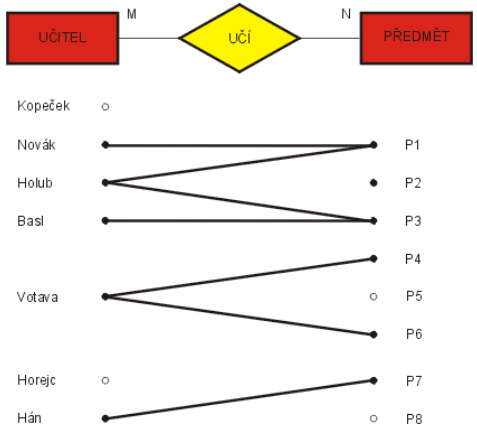
- Vztah 1 : 1



Vztah 1 : N



Vztah M : N

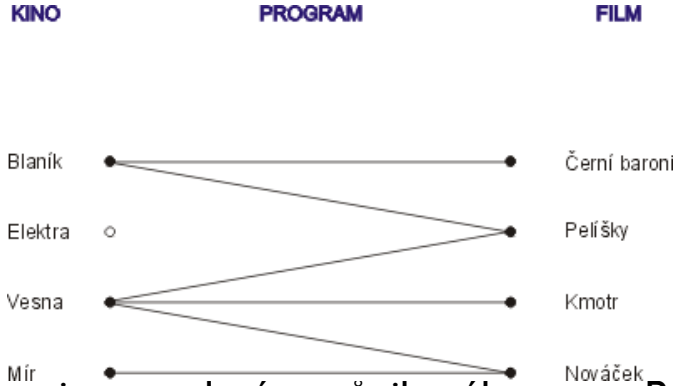


Dekompozice vztahu M:N

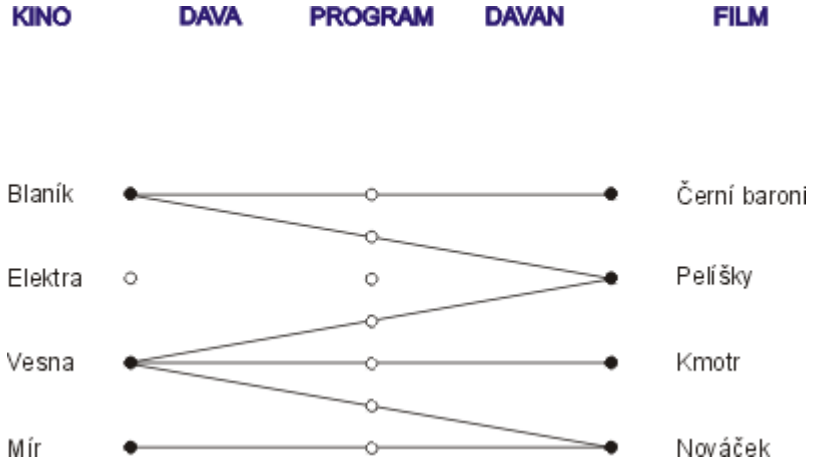
- Vztah `M:N` nelze obvykle v databázích implementovat, snažíme se o dekompozici na dva vztahy `1:N` nalezením tzv. **průnikového** typu
- Průnikový typ je buď **přirozený** (např. operace pro vztah díl – pracoviště), nebo je třeba jej uměle vytvořit zavedením nového pojmu (např. materiál v paletě)

Příklad na dekompozici vztahu M:N – Kino - Film

- Původní výskytový diagram



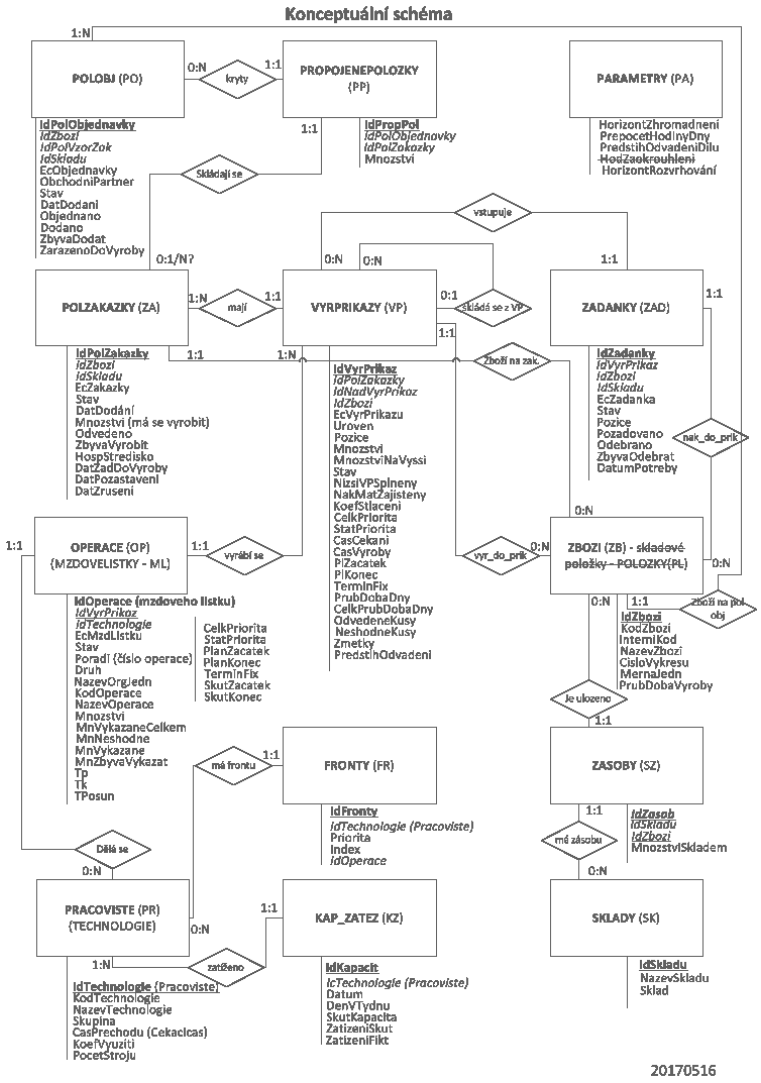
- Dekompozice zavedením průnikového typu „Program“



Primární klíče pro průnikový typ

- Obvykle se tvrdí (v učebnicích pro databáze), že průnikový typ obdrží složený klíč z obou entitních typů, které odděluje. Téměř nikdy to není pravda, např.:
  - V příkladu Kino – Film by se film nesměl krát v jednou kině vícekrát (ani ve dvou sálech najednou, ani jindy v jednom sále -> správný klíč je **Kino, Sál, Datum, Čas**)
  - V Technologickém postupu nestačí operaci identifikovat jako Číslo dílu a Číslo pracoviště, neboť to by znamenalo:
    - a) že na pořadí operací v postupu nezáleží,
    - b) že se v postupu nesmí pracoviště vyskytnout vícekrát
    - Správné řešení je **Číslo dílu, Pořadové číslo operace v postupu**

Příklad ERA diagramu



Normalizace dat

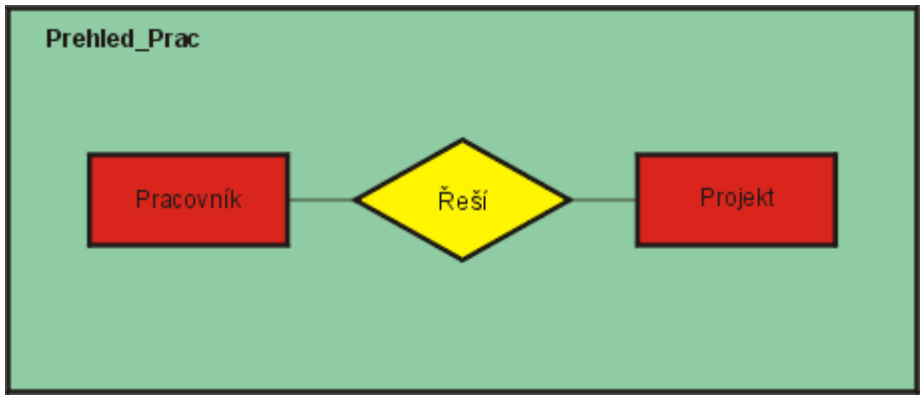
- Cílem normalizace je snížení redundance, zajištění konsistence dat, úspora paměti a jednoznačná manipulace s daty.
- **Normalizace dat** je metodou *datové analýzy:*
  - při návrhu struktur databáze, zejména tabulek relačního modelu .
  - při kontrole konceptuálního návrhu.
- Je třeba kromě jiného, aby návrh splňoval tyto podmínky:
  - Co nejmenší nároky na **paměť**.
  - Jednoznačná **manipulace** s daty.
  - Zajištění **konzistence** dat.
- Uvedené podmínky umožňuje splnit ***normalizace dat***.
- Základní myšlenkou je provést rozklad nenormalizovaného relačního schéma (tj. popisu tabulek) do dílčích relačních schémat bez ztráty informace (bezeztrátová dekompozice).
- Proces normalizace dat je prováděn tzv. postupnými normalizačními kroky, přičemž vznikají jednotlivé **normální formy**:
  - **Nenormalizované schéma -> 1.normální forma -> 2.normální forma -> 3.normální forma …**
- *Pozn.: Je-li relační schéma v N. normální formě , je automaticky i v (N-1). Obráceně to neplatí.*

Nenormalizovaný tvar

- Záznam považujeme za nenormalizovaný, obsahuje-li jednu nebo více skupin **opakujících** se neklíčových položek. Ilustrujme nenormalizovaný záznam na příkladu.
  - Př.: **Prehled_Pracovníků(Cis_Pracovníka,** Jmen_Prac, Tarif, Plat, … , Cis_Projektu, Dat_Zahaj, Termin, Inv_Naklad, …)
- Důsledky:
  - Protože projektant může pracovat na více projektech najednou, vznikají záznamy různé délky.
  - Délka se může měnit i v čase (projekt je ukončen nebo se zahájí nový).
  - V případě prodloužení nastávají problémy s přepisem záznamů, případně s jejich seřazením, atd.
  - Nenormalizovaný tvar se velmi často vyskytuje v tabulkách Excelu

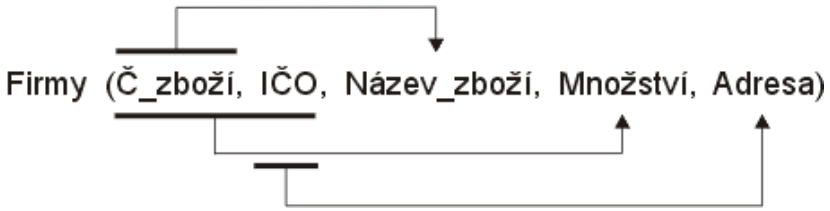
1. normální forma

- V předchozím příkladu provedeme rozdělení původního jednoho entitního typu (tabulky) do dvou.
- Tím je vyřešen problém, že pracovník může pracovat na jednom i více projektech.
- **Pracovnik**(**Cis_Pracovníka**, Jmen_Prac, Tarif, Plat, … ,)
- **Projekt**(**Cis_Projektu**, Dat_Zahaj, Termin, Inv_Naklad,…,*Cis_Pracovníka*)
- Vazbu mezi oběma záznamy zajišťuje položka *Cis_Pracovníka*.
- Záznamům, kde se atributy (sloupce) neopakují, říkáme, že jsou v 1. normální formě.



2. normální forma

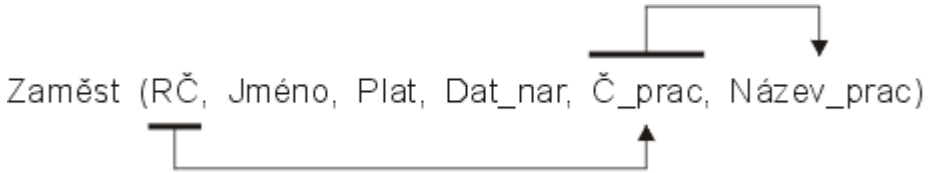
- Záznam je ve 2. normální formě, je-li již v1.NF a všechny neklíčové položky jsou **funkčně závislé na celém klíči** (nestačí částečná závislost).
  - Př.: Údaje o zboží a firmách, kde je vyráběno



  - Název zboží nezávisí na IČO
  - Adresa nezávisí na číslo zboží
  - Vztah M:N byl nesprávně sloučen do jediné tabulky – Množství
- Důsledky:
  - Velká redundance (Adresa a název zboží se mnohokrát opakují)
  - Ztrácíme údaje o firmách, které momentálně nevyrábějí sledované zboží
  - A naopak, ztrácíme přehled o zboží, které momentálně žádná firma nevyrábí
- Řešení spočívá v rozložení na tři tabulky:
  - Firma(**IČO**, Adresa,…)
  - Zboží(**Č_Zboží**, Název_Zboží, …)
  - Produkce(**IČO, Č_Zboží**, Množství, …)

3. normální forma.

- Třetí stupeň normalizace vyžaduje, aby záznam neobsahoval tzv. **tranzitivní** závislosti.
- Každý neklíčový atribut musí být funkčně závislý na (celém) klíči (požadavek 2. NF). Je-li tato závislost zprostředkována přes jiný neklíčový atribut, říkáme, že tento atribut je na klíči tranzitivně (zprostředkovaně) závislý.
- Př. Data přehledu zaměstnanců



- Porušení 3.NF nastalo spojením následujícího vztahu 1 : N do jednoho záznamu.
- Řešení spočívá v rozdělení na dva záznamy, propojené přes atribut Č_Prac:
  - **Pracovník** (**RČ**, Jméno, Plat, Dat_Nar, ***Č_Prac***)
  - **Pracoviště** (**Č_Prac**, Název_Prac)

Příklad pro technologický postup – bez normalizace

- Nenormalizovaný tvar (Excel)

Postupy

Postup	Položka	Popis	Materiál	Vyráběná	Operace1				Operace2				Operace3				Operace4			
					Číslo	Pracoviště	TP	TK	Číslo	Pracoviště	TP	TK	Číslo	Pracoviště	TP	TK	Číslo	Pracoviště	TP	TK
1	Bezcenna	Bezcenná položka	bláto	ne																
2	BezPostup	Bez postupu	ocel	ne																
3	Drát	Drát do kola	ocel	ne																
4	Kolo	Jízdní kolo	sestava	ano	10	Zámečník	25	15												
5	Konus	Konus přední/zadní	ocel	ne																
6	Kulička	Kulička do ložisek	ocel	ne																
7	Niple	Matice na dráty	ocel	ne																
8	Pr.kolo	Přední kolo	sestava	ano	10	Zámečník	10	15												
9	Pr.náboj	Přední náboj	sestava	ano	10	Zámečník	15	10												
10	Pr.osa	Přední osa	sestava	ano	10	Soustruh	50	5	15	Zámečník	25	5	20	Kontrola	0	5				
11	Ráfek	Ráfek 622	ocel	ne																
12	Rám	Pánský rám do kola	ocel	ne																
13	Tyčka	Materiál na osu	ocel	ne																
14	Zad.kolo	Zadní kolo	sestava	ano	10	Zámečník	20	12												
15	Zad.náboj	Zadní náboj	sestava	ano	10	Zámečník	15	10												
16	Zad.osa	Zadní osa	sestava	ano	5	Pila	10	5	10	Soustruh	50	5	15	Zámečník	25	10	20	Kontrola	0	5

Příklad pro technologický postup – 1. norm. forma

- Rozdělíme postup do dvou tabulek
  - Díly (Číslo dílu, Název dílu)
  - Operace (Číslo dílu, Číslo operace, Číslo materiálu, Přípravný čas, Kusový čas, Číslo pracoviště, Tarif pracoviště)
- Sloupce se v tabulkách již neopakují, ale *Číslo materiálu* nezávisí na čísle operace – porušení 2. norm. formy a *Tarif pracoviště* nezávisí na čísle dílu a číslu operace, nýbrž na *Čísle pracoviště* – porušení 3. normální formy

Příklad pro technologický postup – 1. norm. forma

1. Normální forma

TblPolozky Klíč: Položka

Postup	Položka	Popis	Vyráběná
1	Bezcenna	Bezcenná položka	ne
2	BezPostup	Bez postupu	ne
3	Drát	Drát do kola	ne
4	Kolo	Jízdní kolo	ano
5	Konus	Konus přední/zadní	ne
6	Kulička	Kulička do ložisek	ne
7	Niple	Matice na dráty	ne
8	Pr.kolo	Přední kolo	ano
9	Pr.náboj	Přední náboj	ano
10	Pr.osa	Přední osa	ano
11	Ráfek	Ráfek 622	ne
12	Rám	Pánský rám do kola	ne
13	Tyčka	Materiál na osu	ne
14	Zad.kolo	Zadní kolo	ano
15	Zad.náboj	Zadní náboj	ano
16	Zad.osa	Zadní osa	ano

TblOperace Klíč: Položka, Číslo

Položka	Číslo	Pracoviště	TP	TK	Materiál
Kolo	10	Zámečník	25	15	sestava
Pr.kolo	10	Zámečník	10	15	sestava
Pr.náboj	10	Zámečník	15	10	sestava
Pr.osa	10	Soustruh	50	5	ocel
Pr.osa	15	Zámečník	25	5	ocel
Pr.osa	20	Kontrola	0	5	ocel
Zad.kolo	10	Zámečník	20	12	sestava
Zad.náboj	10	Zámečník	15	10	sestava
Zad.osa	5	Pila	10	5	ocel
Zad.osa	10	Soustruh	50	5	ocel
Zad.osa	15	Zámečník	25	10	ocel
Zad.osa	20	Kontrola	0	5	ocel

Příklad pro technologický postup – 2. norm. forma

- Přesuneme číslo materiálu do tabulky dílů
  - Díly (Číslo dílu, Název dílu, Číslo materiálu)
  - Operace (Číslo dílu, Číslo operace, Přípravný čas, Kusový čas, Číslo pracoviště, Tarif pracoviště)
- Technologický postup je ve 2. normální formě, ale stále je problém s tarifem – porušení 3. norm. formy

Příklad pro technologický postup – 2. norm. forma

- 2. normální forma

TblPolozky Klíč: Položka

Postup	Položka	Popis	Vyráběná	Materiál
1	Bezcenna	Bezcenná položka	ne	bláto
2	BezPostup	Bez postupu	ne	ocel
3	Drát	Drát do kola	ne	ocel
4	Kolo	Jízdní kolo	ano	sestava
5	Konus	Konus přední/zadní	ne	ocel
6	Kulička	Kulička do ložisek	ne	ocel
7	Niple	Matice na dráty	ne	ocel
8	Pr.kolo	Přední kolo	ano	sestava
9	Pr.náboj	Přední náboj	ano	sestava
10	Pr.osa	Přední osa	ano	sestava
11	Ráfek	Ráfek 622	ne	ocel
12	Rám	Pánský rám do kola	ne	ocel
13	Tyčka	Materiál na osu	ne	ocel
14	Zad.kolo	Zadní kolo	ano	sestava
15	Zad.náboj	Zadní náboj	ano	sestava
16	Zad.osa	Zadní osa	ano	sestava

TblOperace Klíč: Položka, Číslo

Položka	Číslo	Pracoviště	TP	TK	Tarif pracoviště
Kolo	10	Zámečník	25	15	150
Pr.kolo	10	Zámečník	10	15	150
Pr.náboj	10	Zámečník	15	10	150
Pr.osa	10	Soustruh	50	5	300
Pr.osa	15	Zámečník	25	5	150
Pr.osa	20	Kontrola	0	5	200
Zad.kolo	10	Zámečník	20	12	150
Zad.náboj	10	Zámečník	15	10	150
Zad.osa	5	Pila	10	5	180
Zad.osa	10	Soustruh	50	5	300
Zad.osa	15	Zámečník	25	10	150
Zad.osa	20	Kontrola	0	5	200

Příklad pro technologický postup – 3. norm. forma

- Zavedeme další tabulku pracovišť
  - **Díly** (Číslo dílu, Název dílu, Číslo materiálu)
  - **Operace** (Číslo dílu, Číslo operace, Přípravný čas, Kusový čas, Pracoviště)
  - **Pracoviště** (Číslo pracoviště, Tarif pracoviště)
- Technologický postup je nyní znormalizován v e 3. normální formě, obvykle se další normalizace již nedělá

Příklad pro technologický postup – 3. norm. forma

- 3. normální forma

TblPolozky Klíč: Položka

Postup	Položka	Popis	Vyráběná	Materiál
1	Bezcenna	Bezcenná položka	ne	bláto
2	BezPostup	Bez postupu	ne	ocel
3	Drát	Drát do kola	ne	ocel
4	Kolo	Jízdní kolo	ano	sestava
5	Konus	Konus přední/zadní	ne	ocel
6	Kulička	Kulička do ložisek	ne	ocel
7	Niple	Matice na dráty	ne	ocel
8	Pr.kolo	Přední kolo	ano	sestava
9	Pr.náboj	Přední náboj	ano	sestava
10	Pr.osa	Přední osa	ano	sestava
11	Ráfek	Ráfek 622	ne	ocel
12	Rám	Pánský rám do kola	ne	ocel
13	Tyčka	Materiál na osu	ne	ocel
14	Zad.kolo	Zadní kolo	ano	sestava
15	Zad.náboj	Zadní náboj	ano	sestava
16	Zad.osa	Zadní osa	ano	sestava

TblOperace Klíč: Položka, Číslo

Položka	Číslo	Pracoviště	TP	TK
Kolo	10	Zámečník	25	15
Pr.kolo	10	Zámečník	10	15
Pr.náboj	10	Zámečník	15	10
Pr.osa	10	Soustruh	50	5
Pr.osa	15	Zámečník	25	5
Pr.osa	20	Kontrola	0	5
Zad.kolo	10	Zámečník	20	12
Zad.náboj	10	Zámečník	15	10
Zad.osa	5	Pila	10	5
Zad.osa	10	Soustruh	50	5
Zad.osa	15	Zámečník	25	10
Zad.osa	20	Kontrola	0	5

TblPracoviště Klíč: Pracoviště

Pracoviště	Tarif
Kontrola	200
Pila	180
Soustruh	300
Zámečník	150

Další příklady na normalizaci

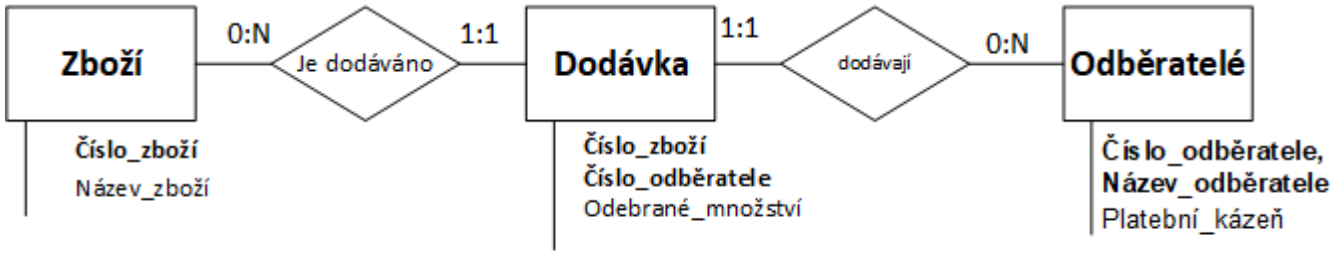
- **1. Dodávka**(Číslo_zboží, Název_zboží, Název_odběratele, Platební_kázeň, Odebrané_množství)

- **2. Cena**(Číslo_výrobku, Název_výrobku, Číslo_operace, Název_operace, Cena_operace, Číslo_materiálu, Název_materiálu, Skladová jednotka, Počet_jednotek, Cena_jednotky, Cena_materiálu, Celková cena)

- **3. Film**(Jméno_kina, Datum, Jméno_filmu,Jméno_hlavního_hrdiny, Národnost_hrdiny, Adresa_kina)

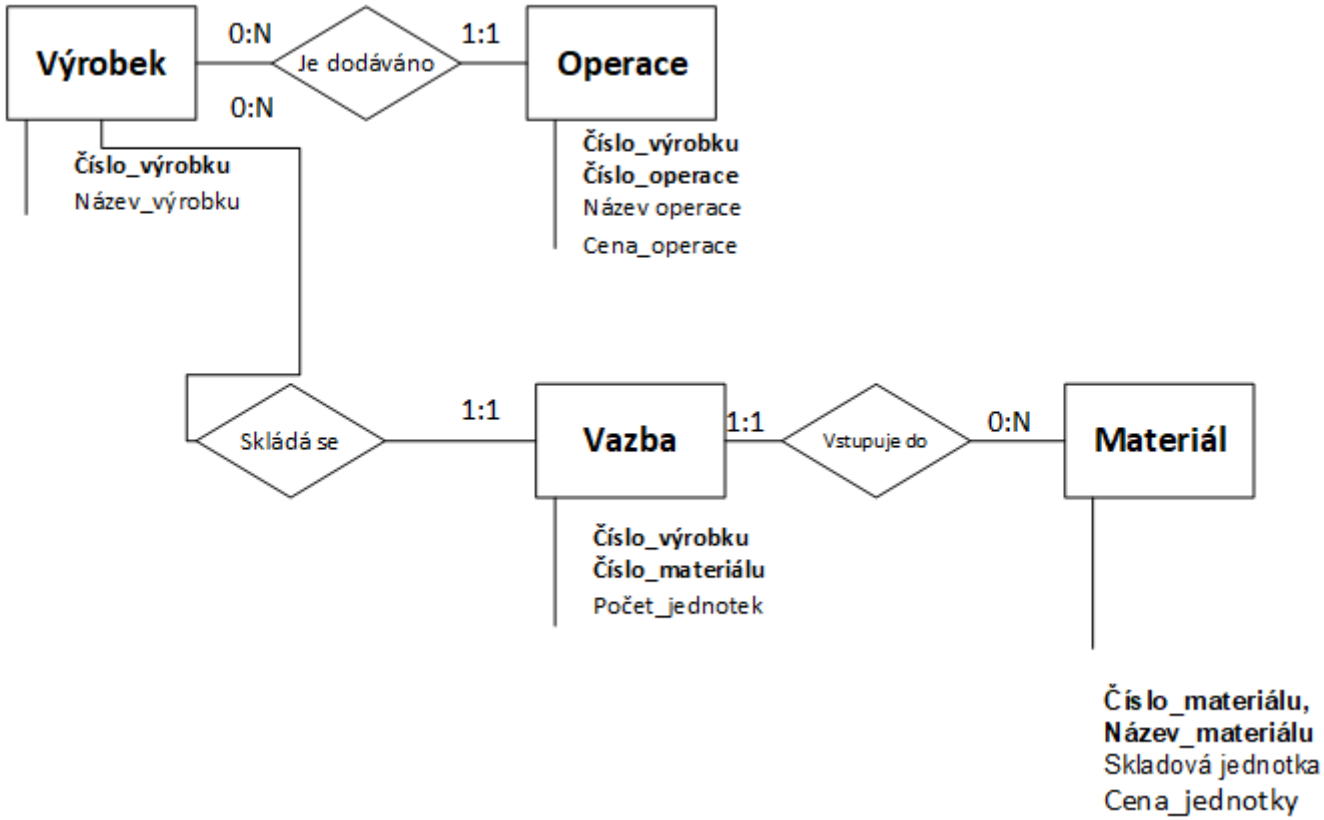
Řešení příkladů

- Zboží - Odběratelé



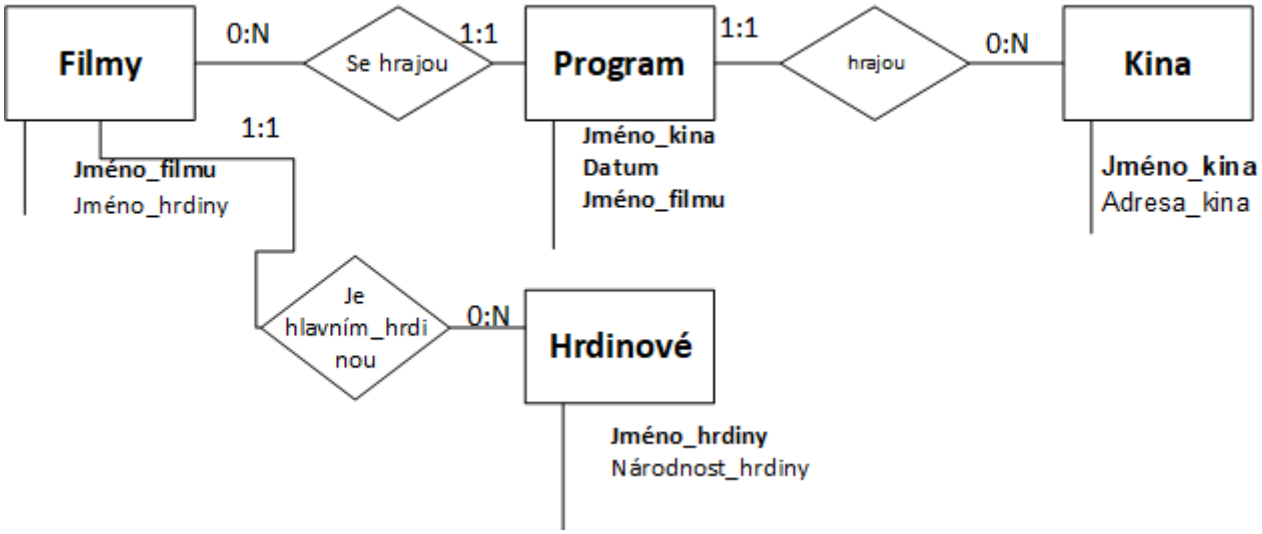
Řešení příkladů

- Výrobek – Operace - Materiál



Řešení příkladů

- Kina - Filmy



Postup při datové analýze

- Podle zadání se sepíší spolu se zadavatelem všechny nejdůležitější informační **datové položky** (budoucí atributy tabulek) a udělá se jejich seznam, na který se lze dívat jako na jednu velkou tabulku, která není ani v první normální formě.
- Z těchto dat se intuitivně vytipují všechny **entitní typy** ET (eventuálně hned jejich primární klíče, ostatní atributy nás zatím nezajímají). Tím se získá seznam entitních typů v první normální formě.
- Nakreslí se první **hrubý ER diagram**, kde ET seřadí tak, aby byly blízko u sebe ty, které spolu vstupují do vztahů a vztahy se zakreslí.
- Entitní typy se zapíší do **seznamu ET**.
- Určí se **kardinality** vztahů. V souladu se zadáním se pro každý vztah formuluje množina **integritních omezení**.
- Pro každý vztah se nakreslí výskytový diagram a určí se jeho **parcialita**. Tím se získá výsledný ER diagram, případně ERA diagram (konceptuální schéma).
- Transformuje se konceptuální schéma (ER diagramu) do **databázového schématu** (většinou relačního). Transformace se dělá podle pravidel pro jednotlivé typy kardinality.
- Tím se získá databázové schéma, do kterého se doplní další atributy. Přitom se kontroluje, aby všechny atributy byly závislé pouze na příslušném **primárním klíči** (třetí normální forma).

Relační databázový model

- Relační databázový systém obsahuje (jako každý databázový systém) databázi a systém řízení báze dat (SŘBD).
- Databáze se skládá z jednotlivých relací, které jsou realizovány pomocí tabulek. Každá tabulka odpovídá jednotlivým **entitním** nebo **vztahovým** typům. Každá tabulka má své jméno a tvoří ji libovolný počet řádek. Každá řádka odpovídá výskytu jedné entity. Kromě řádek má tabulka pojmenované sloupce, kterým se říká **atributy**.
- Na uspořádání řádků ani sloupců nezáleží.

Klíče tabulky

- Alespoň jeden z atributů nebo skupina atributů každé tabulky jsou **klíčové**. Hodnota takového atributu nebo skupiny je jedinečná v celé tabulce. Jeden z takových atributů (klíčů) je prohlášen za **primární,** ostatní, které jsou také jedinečné, za **kandidátní** (např. Rodné_číslo, Číslo_OP, Pracovní_číslo,…).
- Dále se v tabulce mohou vyskytovat klíče, které nejsou jednoznačné, tj. tatáž hodnota atributu se může vyskytnout ve více řádkách. Takové klíče se nazývají **sekundární** nebo také **pravidelné**, v závislosti na použitém databázovém systému.
- Sekundární a kandidátní klíče se velmi často vážou na **primární klíče z jiné tabulky**. V tomto případě se také hovoří o **cizích** klíčích. Tím vznikají relace mezi tabulkami.
- Moderní databáze dávají přednost **autoinkrementálním** primárním klíčům před významovými klíči

Transformace konceptuálního schématu do relačního databázového modelu

- **Podmínky, ze kterých transformace vychází**
  - Máme navržené konceptuální schéma v E-R-A diagramu.
  - Zápisy entitních a vztahových typů jsou v požadované normální formě, většinou je požadována **třetí normální forma**.
  - Máme určeny kardinality všech vztahů
  - Máme provedenu analýzu povinného výskytu klíčových entit ve všech vztazích (parciality).
  - Žádný atribut žádné relace nesmí obsahovat hodnotu NULL. (Někdy se to připouští jako „černý trik“).
  - Za těchto podmínek můžeme provést transformaci, která bude z mnoha hledisek optimální. V případě nutnosti je možné tento návrh upravovat tak, aby lépe vyhovoval některým požadavkům (např. snadnější formulaci nejčastějších dotazů apod.).
- **Transformace** vychází z **kardinality** a **parciality** vztahů mezi entitními typy

Reprezentace vztahu 1 : 1 – zadání příkladu

- NC programy jsou v podniku v následujícím vztahu s výkresy (tvoří množinu integritních omezení) :
  - Žádný výkres detailu nevyužívá více než jeden NC program.
  - Žádný NC program se nevyskytne na více než jednom výkrese.
  - Výkresy jsou identifikovány číslem výkresu.
  - NC programy jsou identifikovány číslem NC programu.
- Konceptuální schéma v E-R modelu bude vypadat takto:

  - E: VÝKRES(Č_Výkresu, DETAIL, MATERIAL, …)
  - E: NC_PROGRAM(Č_NC, AUTOR, …)
  - R:VYUŽÍVÁ(Č_Výkresu, Č_NC)

Reprezentace vztahu 1 : 1 - *Povinné členství pro oba entitní typy*

- Doplňkové integritní omezení (v praxi nerealizovatelné):
  - každý výkres obsahuje právě jeden NC program
  - každý NC program je právě na jednom výkrese

- V tomto případě můžeme atributy obou entitních typů (ET) zařadit do **jedné relace**. Atributy, vážící se k atributu NC_PROGRAM, mohou být považovány za další atributy ET VÝKRES a můžeme je k tomuto schématu "přilepit". Tomuto kroku se říká *slévání* schémat entitní a vztahové relace.
- **Primárním klíčem** nové jedné relace může být libovolný z obou původních primárních klíčů, druhý se stane klíčem kandidátním

Reprezentace vztahu 1 : 1 - *Povinné členství pro NC_PROGRAM, nepovinné pro VÝKRES*

- Doplňkové integritní omezení:
  - každý NC program musí být identifikován některým výkresem
  - výkres nemusí obsahovat žádný NC program (příslušný díl se nevyrábí

- Každý NC program má přidělen výkres, nabízí se tedy přidat do schématu NC_PROGRAM další atribut, a to klíč Č_Výkresu (primární klíč entitního typu VÝKRES).
  - VÝKRES(**Č_Výkresu**, DÍL, …)
  - NC_PROGRAM(**Č_NC**, AUTOR, …, *Č_Výkresu*)
- Klíč Č_Výkresu, přidaný do schématu NC_PROGRAM, vyjadřuje logické spojení konkrétního NC programu na jeho výkres.
- ***Č_Výkresu je kandidátním klíčem !***

Reprezentace vztahu 1 : 1 - *Nepovinné členství ve vztahu pro oba entitní typy*

- Doplňkové integritní omezení:
  - výkresy mohou, ale nemusí odkazovat na některý NC program
  - NC program může, ale nemusí patřit k některému výkresu
- Vztah *Využívá* nemůžeme zobrazit ani přidáním klíče schématu VÝKRES k schématu NC_PROGRAM a naopak. V obou případech bychom museli připustit hodnoty NULL *(pozn.: V SQL je toto řešení možné)*.

Řešením je vytvoření vztahové relace, do níž vložíme atributy, odpovídající identifikačním klíčům těch entitních typů, které se vztahu zúčastní. Oba jsou kandidáty na primární klíč. Primárním klíčem se stane ten atribut, z jehož hlediska převážně informace sledujeme (evidence výkresů x evidence NC programů).

- VÝKRES(**Č_Výkresu**, DETAIL, …)
- NC_PROGRAM(**Č_NC**, AUTOR, …)
- VYUŽÍVÁ(**Č_Výkresu, Č_NC**, …)

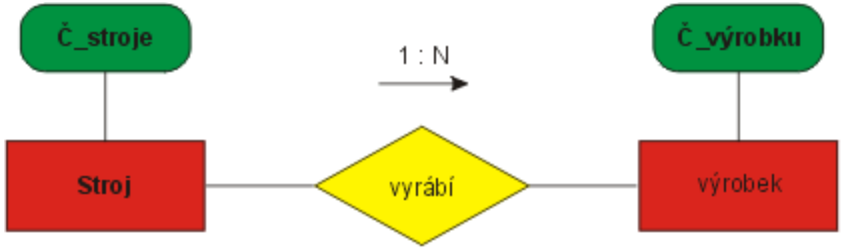
- Poznámka: I vztah může mít atributy (např. identifikaci pracovníka, který provedl přiřazení). Tyto atributy umístí tam, kde jsou oba klíče Č_Výkresu a Č_NC.

Shrnutí a pravidla transformace pro vztah 1 : 1.

- Mají-li oba ET ve vztahu *nepovinné členství,* definujeme tři tabulky. Pro každý ET jednu, třetí pro typ vztahu. Ta bude obsahovat klíče obou předchozích, jako *cizí klíče*. Primárním klíčem bude libovolný z nich. Do tabulky vztahu přidáme vhodné atributy.
- Má-li jeden člen vztahu *členství nepovinné* (druhý je existenčně závislý na prvním), definujeme dvě schémata relací (tabulky). Existenční závislost druhého typu na prvním vyjádříme tak, že k jeho schématu přidáme atributy, odpovídající identifikačnímu klíči nezávislého entitního typu. Libovolný z těchto klíčů může být pak primární. Také atributy vztahu budou v závislém entitním typu.
- Mají-li oba členy členství ve vztahu povinné, můžeme vytvořit schéma jediné, které vznikne spojením z původních dvou. Primárním klíčem bude libovolný klíč spojených entitních typů. Atributy vztahu budou také v tomto jediném schématu.

Reprezentace vztahu 1 : N – zadání příkladu

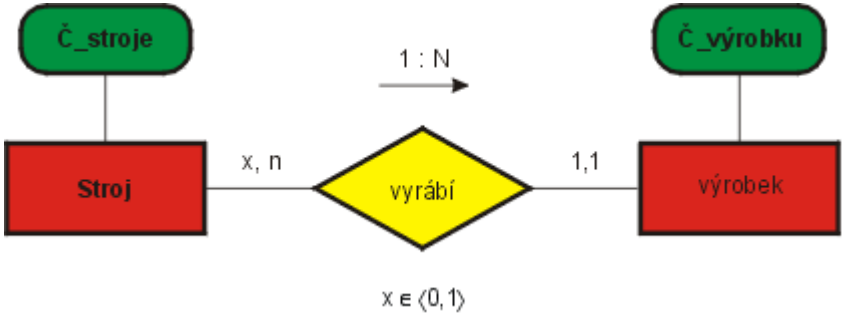
- Uvažujme relaci Stroj – Výrobek



- Základní množina integritních omezení :
  - na jednom stroji je možno vyrábět více výrobků.
  - konkrétní výrobek je možné vyrábět jen na jednom konkrétním stroji.
- Entitní typ VÝROBEK je **determinantem** entitního typu STROJ. Opačně to neplatí.
- Typ členství ve vztahu u ET, který není determinantem druhého typu (STROJ) není podstatný.

Representace vztahu 1 : N - *Povinné členství determinantu ve vztahu*

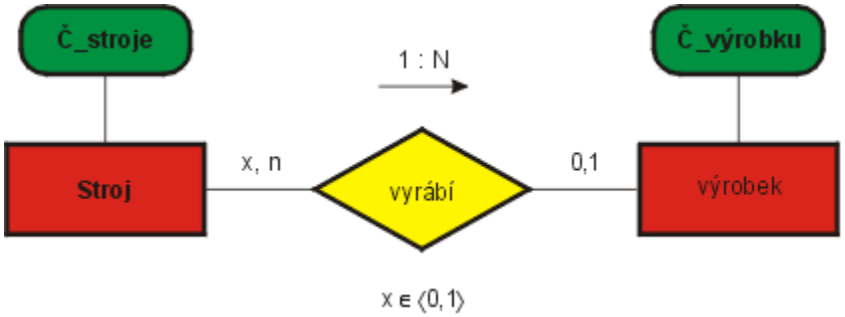
- Jestliže je předepsáno povinné členství ve vztahu pro entitní typ VÝROBEK (databáze se týká pouze výrobků momentálně vyráběných), nejjednodušší reprezentace vztahu VYRÁBÍ bude taková, že se přidá atribut Č_STROJE ke schématu VÝROBEK. Každá n-tice odpovídající relace potom bude popisovat jeden výrobek včetně informace, na kterém stroji je vyráběn.



- Konceptuální schéma:
  - E: VÝROBEK(**Č_SOUČÁSTI**, …)
  - STROJ(**Č_STROJE**, TYP_STROJE, …)
  - R: VYRÁBÍ(**Č_SOUČÁSTI, Č_STROJE**)
- Odpovídající část relačního schématu databáze:
- VÝROBEK(**Č_SOUČÁSTI**, …, *Č_STROJE*)
- STROJ(**Č_STROJE**, TYP_STROJE, …)

Representace vztahu 1:N - *Nepovinné členství determinantu ve vztahu*

- Výrobku nemusí být přiřazen výrobní stroj (výrobky vyráběné v kooperaci, nakupované). V tomto případě již není možné jen tak přidat ČÍSLO_STROJE ve schématu VÝROBEK, aniž bychom připustili hodnotu NULL. Řešením je trojice schémat relací.



- Odpovídající část relačního schématu databáze:
  - VÝROBEK(**Č_SOUČÁSTI**, …)
  - STROJ(**Č_STROJE**, TYP_STROJE, …)
  - SE_VYRÁBÍ_NA(**Č_SOUČÁSTI, Č_STROJE**, …)

Pozn.: Oblíbený „černý" trik programátorů je v tom, že se použije hodnota NULL, a tím se vystačí se dvěma tabulkami jako u nepovinného členství.

Shrnutí a pravidla transformace pro vztah 1 : N.

- Má-li **determinant** vztahu **povinnou** účast ve vztahu, definujeme **dvě schémata** (pro každý entitní typ). Ke schématu determinantu přidáme atributy, odpovídající identifikačnímu klíči druhého entitního typu. Primárním klíčem je klíč determinantu.
- Má-li **determinant** vztahu **nepovinnou účast** ve vztahu, definujeme tři schémata (pro každý ET jedno, třetí bude vztahové a bude obsahovat atributy, mezi nimiž budou identifikační klíče obou ET.

Reprezentace vztahu M : N.

- Obvykle se provádí transformace vztahu N : M v předchozích etapách analýzy. Nalezne se **přirozený** nebo **umělý průnikový typ**.
- Bez ohledu na typ členství ve vztahu definujeme **tři schémata**, jedno pro každý entitní typ, třetí vztahové, bude obsahovat primární klíče z obou entitních typů a další (vztahové) atributy.
- Primárním klíčem schématu může být někdy dvojice příslušných primárních klíčů, často to nejde (např. ve vztahu *Díl – Pracoviště* je průnikovým typem *Operace*, a protože se pracoviště v postupu může opakovat a také záleží na pořadí operací, pak je primárním klíčem pro *Operace* - *Číslo dílu* + *Číslo operace*).

Příklad datové a funkční analýzy - Formulace úlohy

- Název informačního systému: ***Skladování*** materiálu, polotovarů a hotových výrobků.
- Analyzujte a realizujte část informačního systému, týkající se skladování materiálu, polotovarů a výrobků v regálovém skladu. Uvažovaný systém je součástí širšího systému ***ZÁSOBOVÁNÍ*** a ***ODBYT***. Neřeší vlastní dopravu palet, tj. řízení zakladačů a evidenci skladových míst. Obě tyto úlohy řeší podřízený systém ***Sklady a doprava***. Systém ***Skladování*** bude používán obsluhou skladu. Nikdo jiný se systémem pracovat nebude, přístup do dat jinými uživateli (Materiál) musí být však možný. Kromě evidence uložení palet a jejich obsahu bude poskytovat služby ZASKLADŇOVÁNÍ a VYSKLADŇOVÁNÍ s příslušnými kontrolami. Vyskladňovat / zaskladňovat se bude na základě dokumentu *Výdejka/Příjemka*. Detailní vazby na nadřazený (Zásobování) ani podřízený (Doprava) systém se zde neřeší, návrh však musí být v tomto směru otevřený.

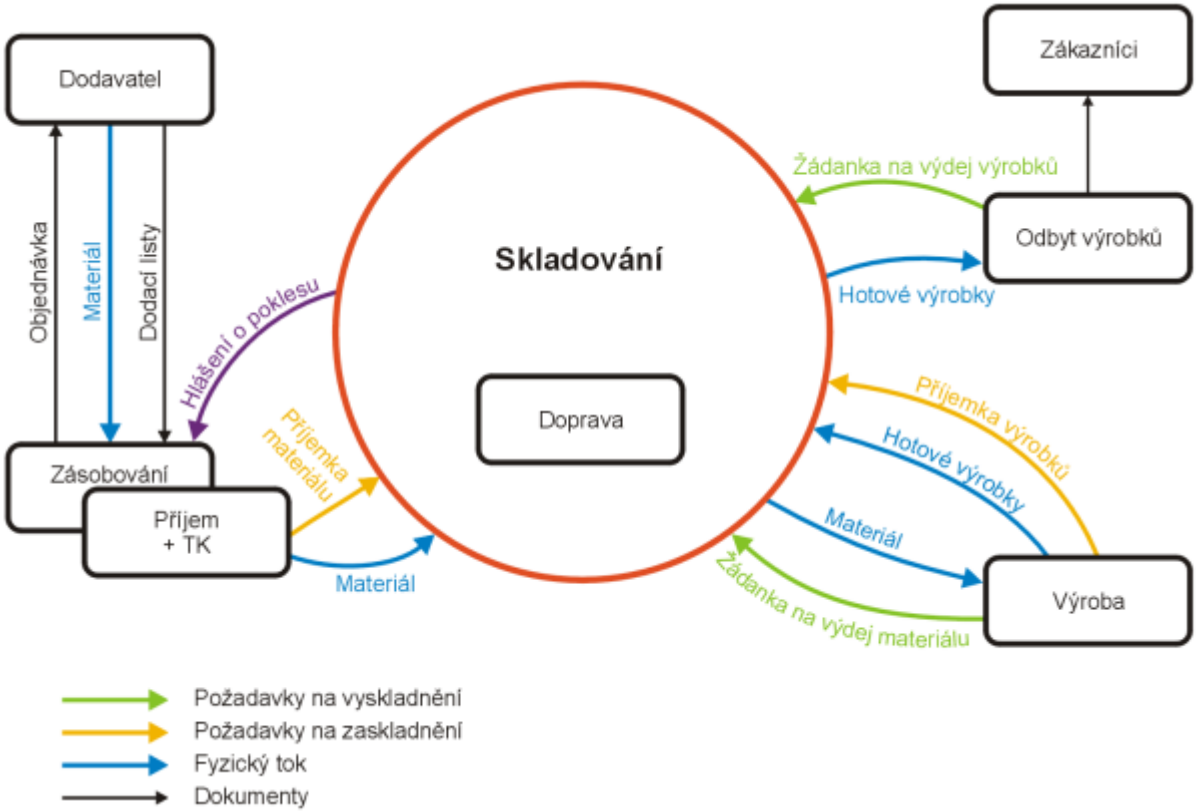
Příklad datové a funkční analýzy - Popis systému

- **Sklad** je rozdělen na dvojice regálů, mezi kterými pojíždí zakladač. Každá ulička je určena pro jeden druh palet (viz dále). Regály jsou tvořeny buňkami příslušné velikosti, jejichž počet se pohybuje v řádech tisíců. Buňky skladu jsou označeny adresou, která se skládá z čísla skladu, uličky, čísla sloupce, výšky a směru (vpravo, vlevo), např. 2.3.56.5.P.
- **Palety** jsou očíslovány v zadaném rozsahu. Existují 3 typy palet:
  - malé (Europalety),
  - velké (dvojnásobné velikosti),
  - ploché (pro ukládání plechů).
- Počet palet v systému je větší, než je počet míst v regálech.
- **Materiál** je jednoznačně označen interním číslováním. Do každé palety je ukládán materiál jednoho druhu, každý materiál může být uložen ve více paletách. Materiál z různých dodávek se zde pokládá za jeden druh. V praxi bychom museli u materiálu (zvláště u takového, jako jsou plechy), rozlišovat ještě další atributy (např. *číslo tavby*). Takový atribut by se potom stal členem (složeného) primárního klíče.

Příklad datové a funkční analýzy - Funkcionalita - požadované funkce informačního systému

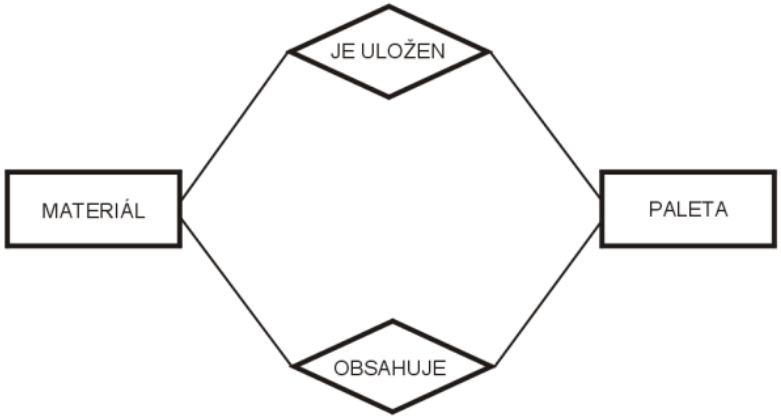
- Navržený IS musí vykonávat následující činnosti:
  - Sledovat **přehled materiálu** (identifikovaného číslem materiálu), jeho aktuální množství na skladě a upozornit uživatele na eventuální pokles zásoby pod minimální množství, charakteristické pro každý materiál.
  - Poskytnout seznam **prázdných palet**.
  - Dovolit změnu **rezervace** prázdné palety pro určitý materiál (paleta má přiděleno číslo materiálu, ale množství je nulové).
  - **Vyskladnit** nebo **zaskladnit** vybranou paletu.
  - **Změnit obsah** u vyskladněné palety s příslušnými kontrolami.

Příklad datové a funkční analýzy - Kontextový diagram



Příklad datové a funkční analýzy – *Informační položky*

- ***Nenormalizovaný zápis.***
- Sklad(Číslo_materiálu, Název_materiálu, Měrná jednotka, Množství_celkové, Množství_minimální, Max_Množství_v_paletě, Číslo_palety, Typ_palety, Stav_palety, Adresa_uložení, Množství_v_paletě,…).
- V tomto nenormalizovaném zápisu jasně rozeznáme dva entitní typy a jejich vztahy : **Materiál** a **Paleta**. Jejich vztah můžeme formulovat dvojím způsobem:
  - paleta obsahuje materiál,
  - materiál je uložen v paletě.

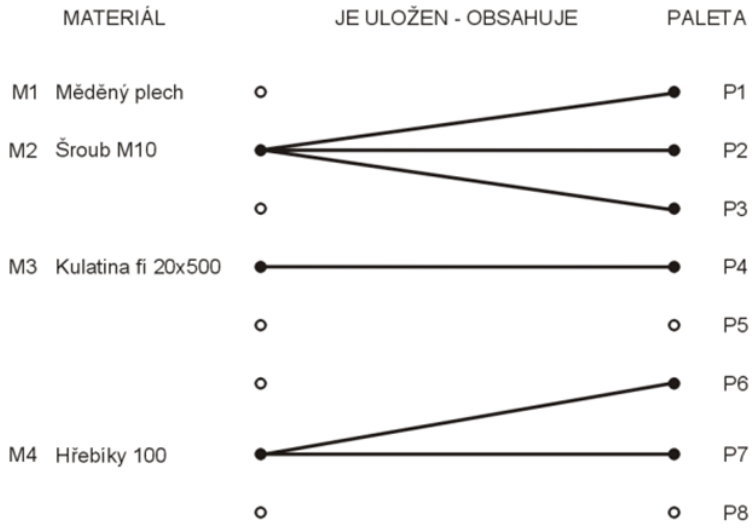


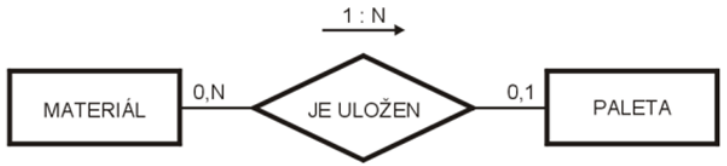
- ***Zápis ve třetí normální formě***
-
- **E: Materiál**(Číslo_materiálu, Název_materiálu, Měrná jednotka, Množství_minimální, Max_Množství_v_paletě,…)
- **Palety(**Číslo_palety, Typ_palety, Stav_palety, Adresa_uložení,…)
- **R: Obsahuje(**Číslo_materiálu, Číslo_palety, Množství_v_paletě,…)

Příklad datové a funkční analýzy – Integritní omezení

- Paleta obsahuje pouze materiál jednoho druhu.
- Některá paleta nemusí být určena pro žádný materiál, potom je volná.
- Prázdná paleta ještě nemusí být volná (je rezervovaná).
- Materiál jednoho druhu může být uložen ve více paletách. Některý materiál nemusí být na skladě.
- Materiál z různých dodávek může být směšován (může se přidat do palety, obsahující materiál stejného druhu).

Příklad datové a funkční analýzy – Výskytový a ER diagram





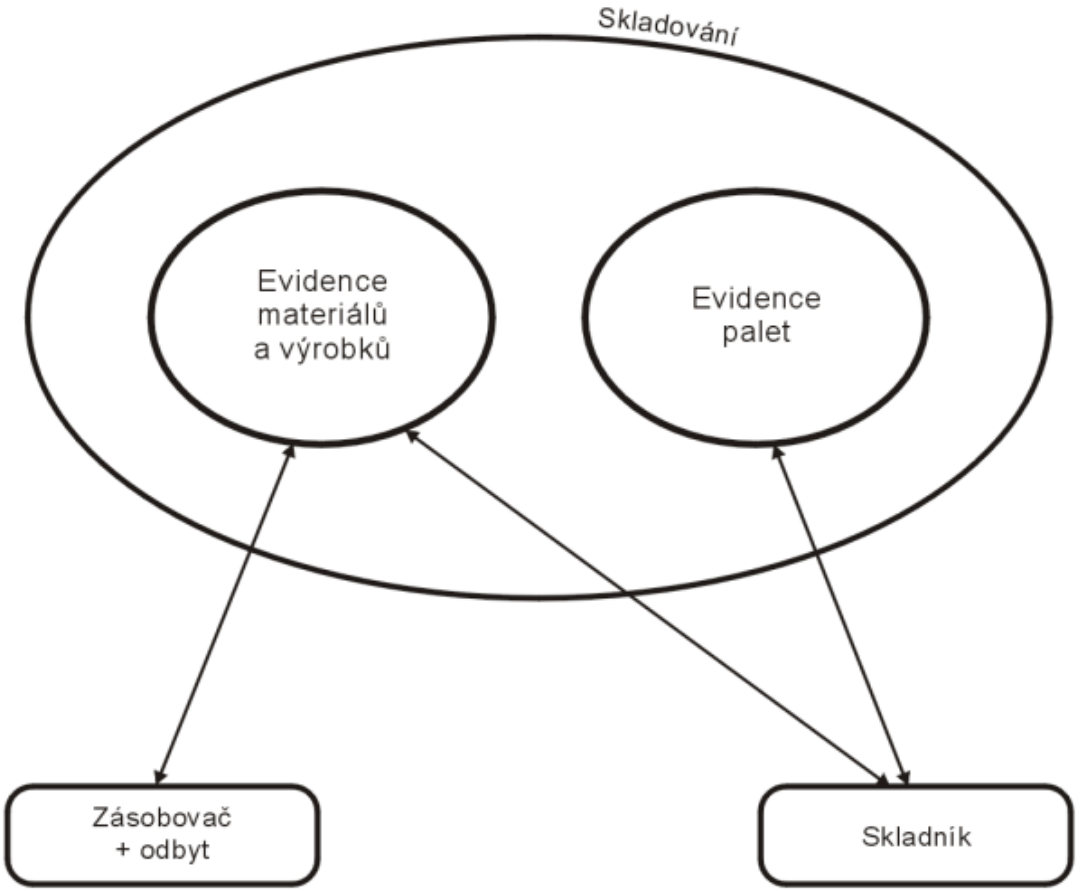
- Existují dva Entitní typy: **Materiál** a ***Paleta***.
- Vztah JE_ULOŽEN (zleva doprava) má kardinalitu 1 : N, vztah OBSAHUJE (zprava doleva) 1 : 1.
- Oba členy jsou ve vztahu **nepovinné**.

Příklad datové a funkční analýzy- Transformace konceptuálního schématu do relačního databázového modelu

- Podle transformačních pravidel pro vztah $1:\mathbb{N}$ budou oba entitní typy reprezentovány samostatnou tabulkou. Pro reprezentaci vztahu platí, že je-li na straně **N** ve vztahu **povinné** členství, stačí do tabulky, reprezentující tento entitní typ (Paleta) přidat jako cizí klíč primární klíč entitního typu druhého (Materiál).
- V našem vztahu je však členství na straně N **nepovinné** (paleta nemusí být rezervována pro žádný materiál). V takovém případě se buď musí použít samostatná vztahová tabulka, nebo v entitním typu s nepovinným členstvím dovolit použití hodnoty NULL. Vzhledem k charakteru úlohy je výhodné v tabulce Palety použití hodnoty NULL dovolit.
- Výsledné databázové schéma se bude skládat ze dvou relací (tabulek), pro každý entitní typ jedna. Vztah se v tomto případě vyřeší tak, že se do tabulky Palety přidá atribut *Číslo materiálu*. V případě, že paleta nebude rezervována pro žádný materiál, bude hodnota tohoto atributu NULL. (Rovněž atribut *Adresa uložení* bude mít hodnotu NULL v případě, že paleta nebude mít přidělenou adresu).

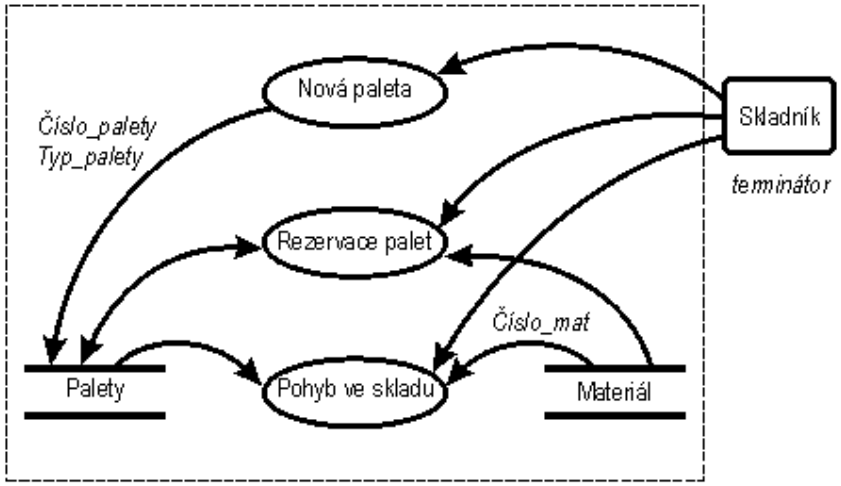
Příklad datové a funkční analýzy- Diagram datových toků 0. úrovně

- Na nejvyšší úrovni můžeme znázornit diagram datových toků

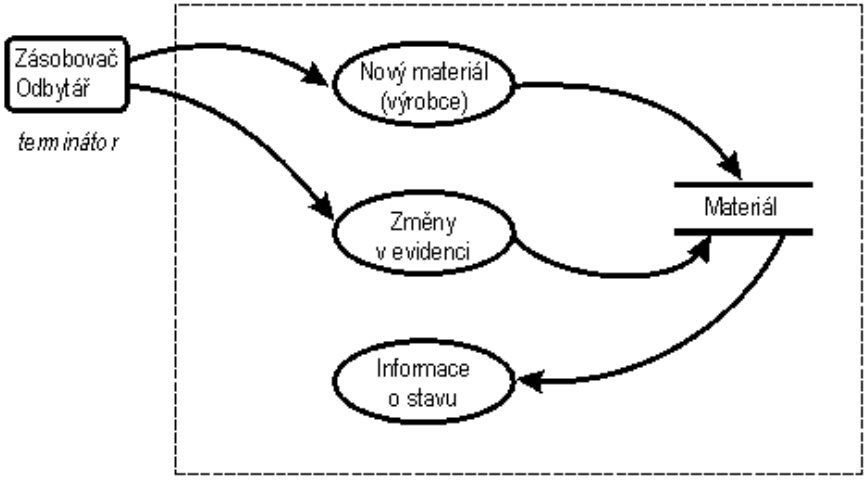


Příklad datové a funkční analýzy- Diagram datových toků I. úrovně

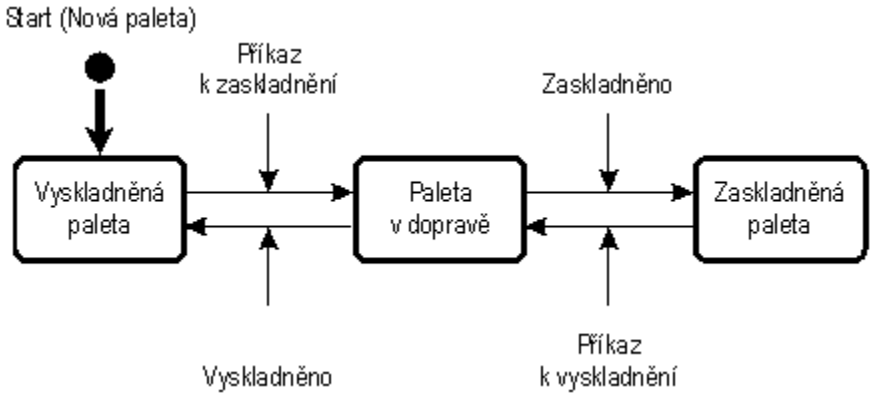
Evidence palet - DFD - úroveň 1



Evidence materiálu a výrobků - DFD - úroveň 1

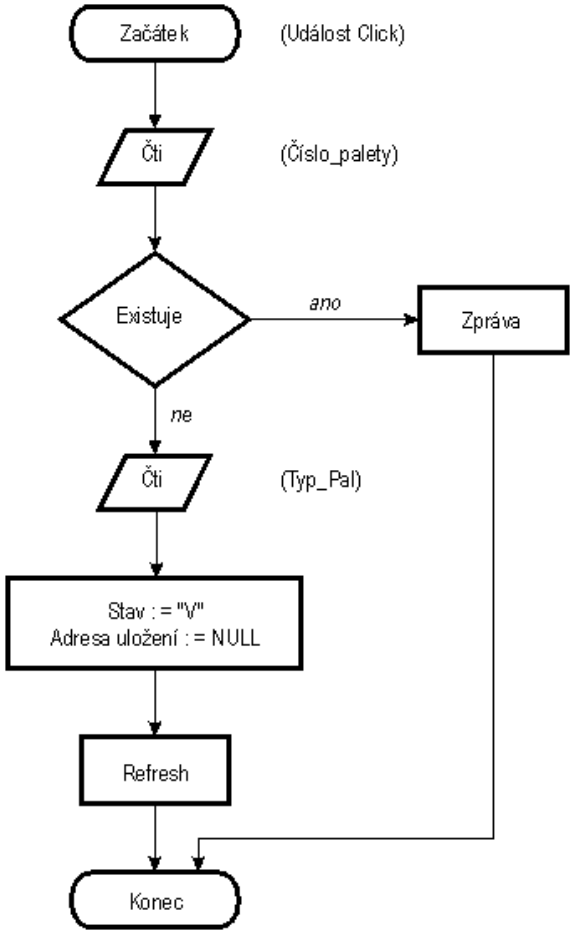


Příklad datové a funkční analýzy- Stavový diagram

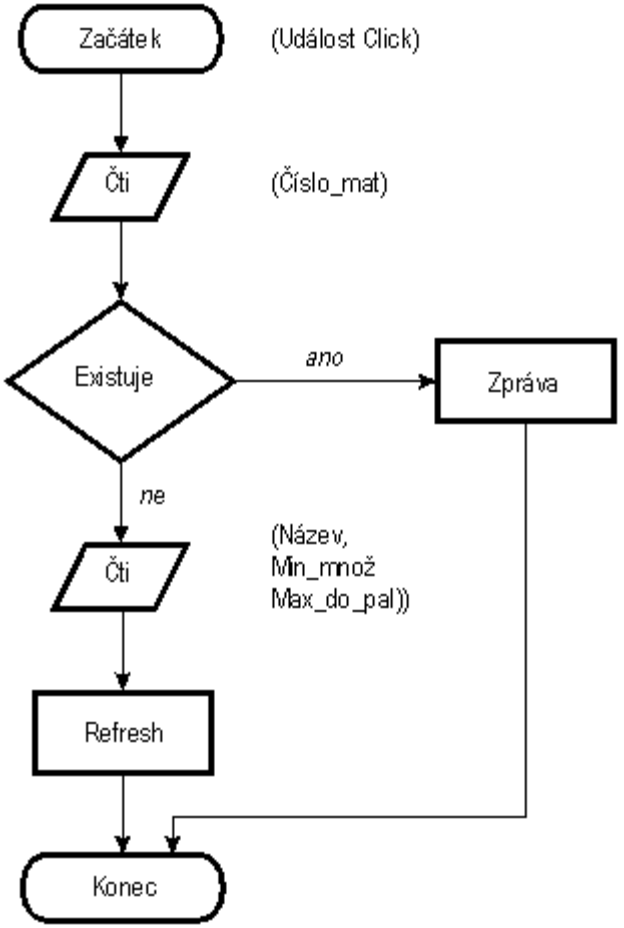


Příklad datové a funkční analýzy- Ukázka vývojového diagramu I

Nová Paleta

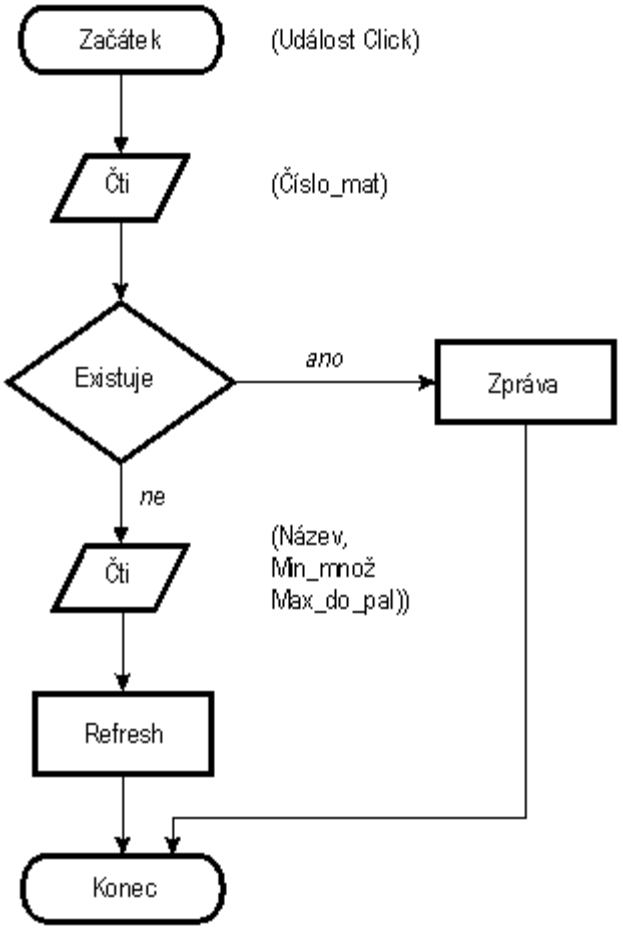


Nový materiál

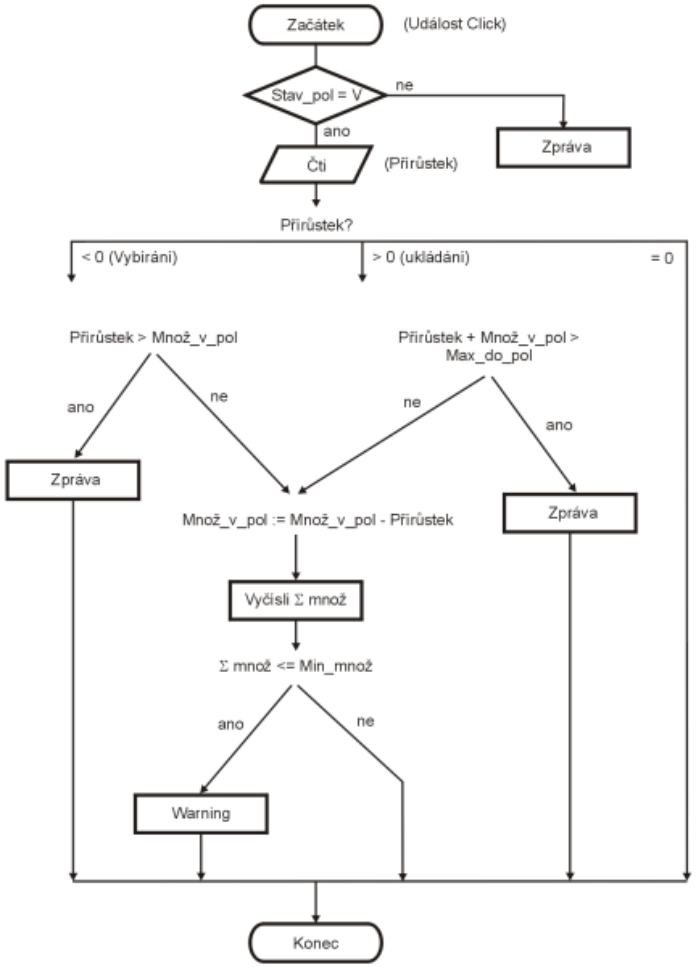


Příklad datové a funkční analýzy- Ukázka vývojového diagramu 2

Rezervace palet



Pohyby ve skladu



Základní pojmy SQL - I

- **SQL (Structured Query Language** je neprocedurální jazyk definice dat a pro manipulaci s nimi.
- **Procedurální jazyky** (např. Pascal, C, Simula) jsou založeny na postupech, jak získat určitou informaci, jak ji zpracovat, eventuálně zobrazit. Ať už se jedná o neobjektově orientované jazyky, kde prvotní je algoritmus pracování dat, nebo o objektově orientované jazyky, kde jsou pro data definovány operace jejich zpracování, základní otázkou, kterou si klademe, je „JAK ZPRACOVAT“.
- **Neprocedurální jazyky** jsou založeny spíše na představě, co chceme udělat. Základní otázkou je „JAKÁ DATA CHCEME ZÍSKAT“ nebo „JAK SE MAJÍ DATA ZMĚNIT“.
- **Definice dat** umožňuje jejich popis, určuje, zda jsou numerická nebo znaková, určuje jejich obor hodnot, počet platných míst (znaků nebo číslic). Dále určuje, jaká data souvisí s jinými, např. že datový typ zaměstnanec se skládá z položek *jméno, osobní číslo, plat, oddělení, telefon*, atd.
- Pod pojmem **manipulace s daty** chápeme vytvoření, změnu a zrušení určité datové struktury.

Základní pojmy SQL - 2

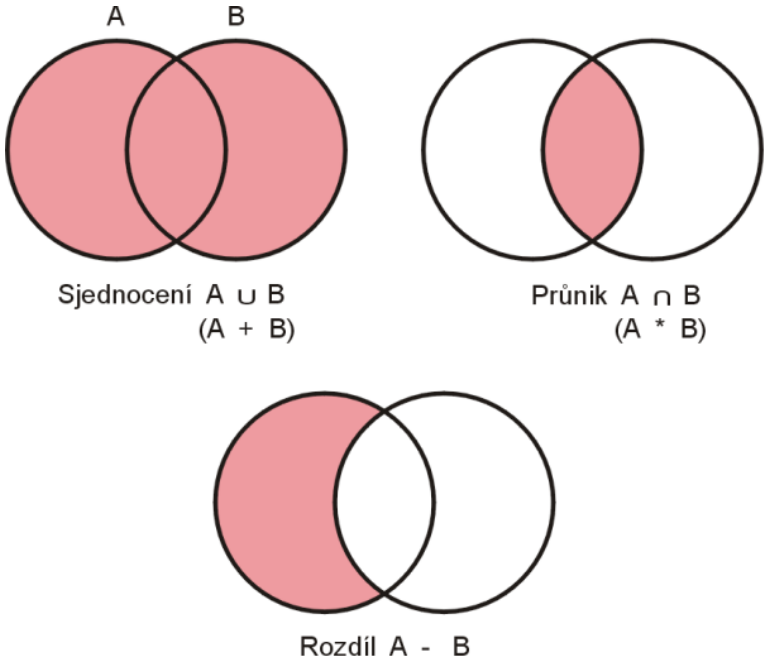
- **Doména**
- Doména je množina hodnot **stejného významového typu**. Doménou může být číslo materiálu nebo jeho množství na skladu. Hodnoty v doméně jsou stejného datového typu - číslo, logická hodnota, řetězec znaků, datum atd.
- **Kartézský součin množin A, B**
- Kartézský součin množin dvojic [x,y], pro které platí že (x € A) a zároveň (y € B). Počet prvků v kartézském součinu je dán součinem počtu prvků v množině A a počtu prvků v množině B.
- Př. Mějme množinu A osob, např. {Franta, Pepík} a množinu B denní doby {ráno, poledne, večer}. Pak kartézský součin C = A x B obsahuje tyto dvojice {[Franta, ráno],[Franta, poledne],[Franta, večer],[Pepík, ráno],[Pepík, poledne],[Pepík, večer]}.
- *Pozor: Nepleťte si kartézský součin dvou množin (obecně různého typu) s průnikem dvou množin stejného typu C = A*B, kde prvky množiny C jsou současně obsaženy jak v A, tak v B.*
- **Relace**
- Relace je libovolná **podmnožina** kartézského součinu.
- Př. Tabulka zaměstnanců je relace mezi kartézským součinem domén jejích položek (jméno, osobní číslo, plat …).
- **Atribut**
- Atribut je **název domény** pro použití v relaci. Pokud uvažujeme, že data jsou v tabulkách, pak atribut je **název sloupce**.
- **Tabulka**
- Tabulka je praktický, upravený a zjednodušený **pohled na relaci**. Pořadí řádek v tabulce je podstatné, pořadí sloupců je nepodstatné. Sloupec odpovídá atributu, resp. doméně, řádek instanci entitního typu.

Základní pojmy SQL - 3

- **Klíč nebo identifikátor**
  - Klíč nebo identifikátor je **sloupec nebo skupina sloupců** v tabulce identifikující řádek tabulky.
- **Primární klíč**
  - Klíč, který **jednoznačně** identifikuje řádek v tabulce. Pokud je takových klíčů více, pak je primárním klíčem obvykle ten, který má nejmenší délku.
- **Kandidátní klíč**
  - Kandidátní klíč také **jednoznačně** identifikuje řádek v tabulce. Mohl by stát primárním klíčem, pokud by primární klíč z nějakého důvodu nevyhovoval.
- **Pravidelný klíč**
  - Pravidelný klíč **není jednoznačný**, obvykle identifikuje více řádek.
- **Cizí klíč**
  - Cizí klíč nebo také **sekundární** je sloupec nebo skupina sloupců použitá jako odkaz v jiné tabulce. Cizím klíčem je obvykle pravidelný klíč nebo kandidátní klíč, někdy i primární klíč nebo jeho část.

Relační algebra - I

- Množinové operace sjednocení, průnik a rozdíl můžeme aplikovat na relace, pokud mají stejnou strukturu, tj. stejný počet sloupců v tabulce.

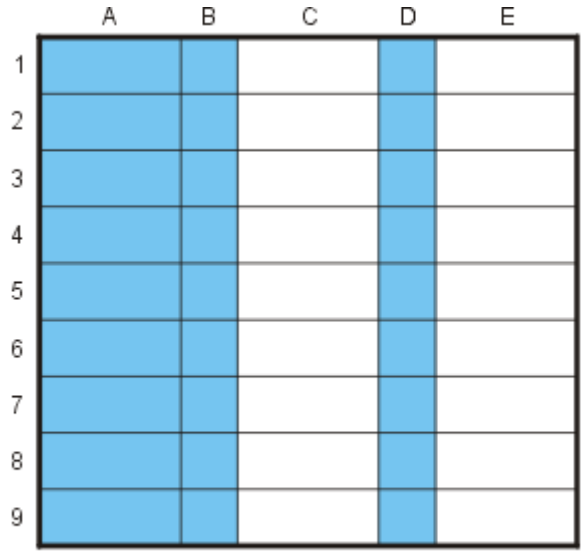


Příklady základních množinových operací

- *Příklad sjednocení*
  - Máme tabulku zaměstnanců v jednom výrobním oddělení a tabulku zaměstnanců v druhém výrobním oddělení. Pro každého zaměstnance bez ohledu na oddělení uvažujeme stejné atributy.
  - Sjednocením vytvoříme tabulku zaměstnanců v obou odděleních.
  - *Pozn. Průnikem získáme tabulku zaměstnanců, kteří pracují na částečný úvazek v jednom oddělení a na částečný v druhém oddělení. To ovšem znamená jiné integritní omezení i jiný datový model.*
- *Příklad průniku*
  - Máme tabulku nerez materiálů a tabulku válcovaných materiálů. Obě tabulky mají stejné atributy (číslo materiálu, cenu za měrnou jednotku, množství na skladu, ...)
  - Průnikem získáme tabulku válcovaných nerez materiálů.
  - *Pozn. Sjednocením získáme tabulku materiálů, které jsou buď nerez, nebo jsou válcované.*
- *Příklad rozdílu*
  - Máme tabulku všech materiálů a tabulku materiálů dodávaných firmou Chroust & Cvrček. Rozdílem obou tabulek je tabulka materiálů dodávaných jinými firmami.

Speciální množinové operace nad relacemi - *Projekce*

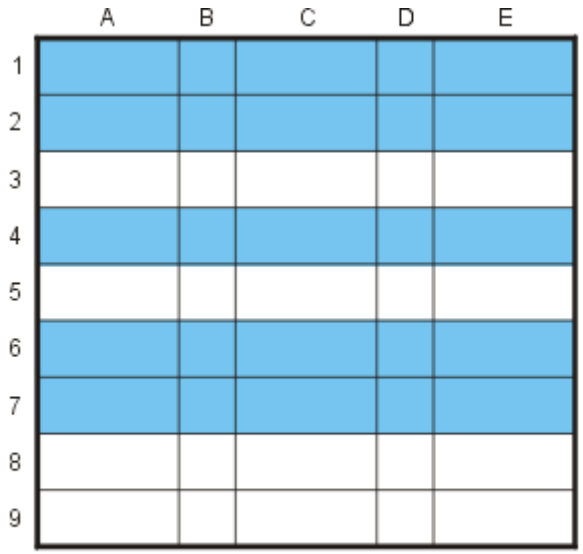
- Projekce vybírá **sloupce** z tabulky A do sloupců z tabulky B na základě seznamu vybíraných sloupců. Počet řádek zůstává, počet sloupců se snižuje.



Výběr sloupců A, B, D

Speciální množinové operace nad relacemi - *Restrikce*

- Restrikce vybírá **řádky** z tabulky A do tabulky B řádky na základě definované podmínky. Počet sloupců zůstává a počet řádek se snižuje.



Výběr řádek 1, 2, 4, 6, 7

Speciální množinové operace nad relacemi - *Spojení tabulek*

- Spojení tabulek představuje **kartézský součin** obou tabulek. Výsledná tabulka má počet řádek rovný součinu počtu řádek obou tabulek a počet sloupců rovný součtu počtu sloupců obou tabulek.

T1:

1	A	B	C	D	E
2	F	G	H	I	J

T2:

1	a	b	c	d
2	e	f	g	h
3	i	j	k	l

T3:

1	A	B	C	D	E	a	b	c	d
2	A	B	C	D	E	e	f	g	h
3	A	B	C	D	E	i	j	k	l
4	F	G	H	I	J	a	b	c	d
5	F	G	H	I	J	e	f	g	h
6	F	G	H	I	J	i	j	k	l

Spojení T3 = T1 × T2

SQL Příkaz SELECT – obecný tvar

- Příkaz **SELECT** je základním a nejvíce využívaným příkazem jazyka SQL.

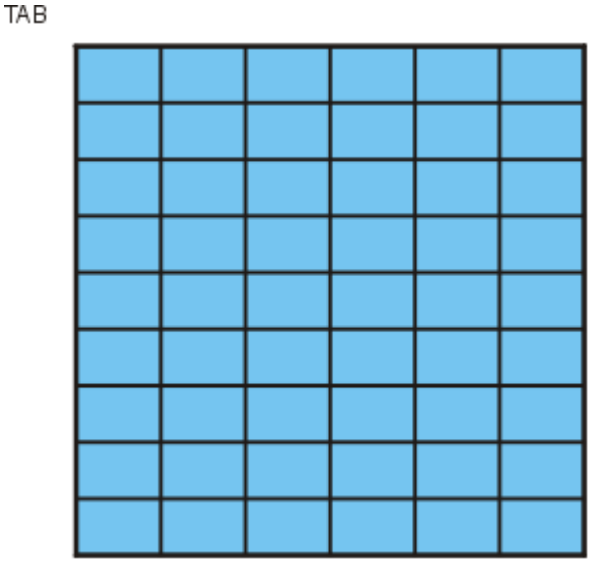
```
SELECT <seznam_výstupních_sloupců>
    FROM <seznam_tabulek>
    [WHERE<podmínka_řádku>]
    [GROUP BY<seznam_výrazů_seskupení>]
    [HAVING<podmínka_skupiny>]
    [{UNION|UNION ALL|
      INTERSECT|
      MINUS}<příkaz_select>]
    [ORDER BY<seznam_kriterií_řazení>]
```

- Na první pohled se tento příkaz zdá být velmi složitý. Většina klausulí je ovšem nepovinná.
- Výstupními sloupci mohou být:
  - Konstanty
  - Sloupce z tabulek
  - Výrazy ze sloupců (aritmetické operace, výběr na základě podmínky)
  - Funkce (ze sloupců, výrazů, konstant) z (propojených) řádek tabulek nebo agregační funkce ze skupin řádek

SQL Příkaz SELECT – nejjednodušší tvar

SELECT * FROM *tab*

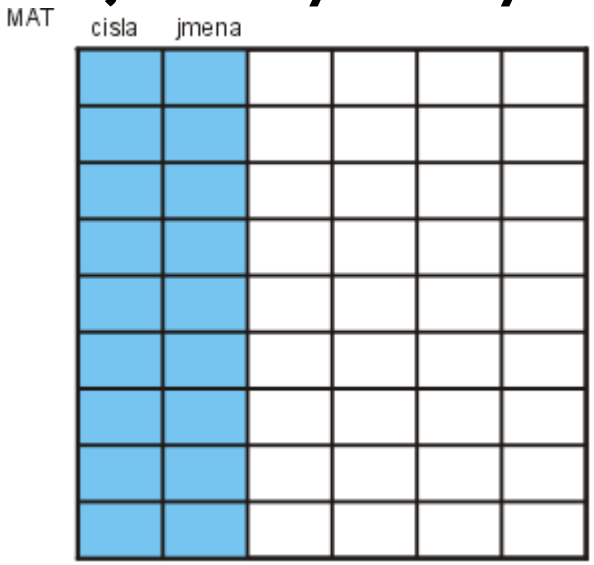
- Výsledkem takového příkazu je **celá** tabulka, tj. veškeré řádky a sloupce bez projekce a restrikce.



SQL Příkaz SELECT – Jednoduchá projekce

SELECT *číslo*, *jméno* FROM *mat*

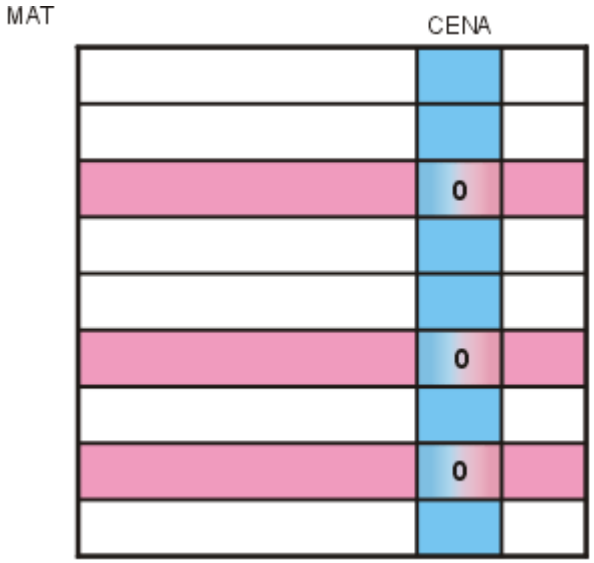
- Výsledkem výběru z tabulky *mat* jsou pouze **sloupce** *číslo a jméno*, ostatní sloupce nejsou vybrány



SQL Příkaz SELECT – Jednoduchá restrikce

SELECT * FROM *mat* WHERE *cena* = 0

- Výsledkem výběru z tabulky jsou pouze ty **úplné řádky**, kde je zadána nulová cena položky.



SQL Příkaz SELECT – Kombinace jednoduché projekce a restrikce

SELECT *číslo* FROM *mat* WHERE *cena* < 1000

- Výsledkem výběru z tabulky jsou pouze čísla materiálu (**buňky tabulky**), jehož cena je menší než 1000.

MAT

cislo	cena
	1 500
	3 500
	800
	2 000
	2 500
	0
	4 050
	200
	1 100

SQL Příkaz SELECT – Řazení řádek na výstupu

SELECT * FROM *mat* ORDER BY *cena*

- Výsledkem příkazu bude celá původní tabulka **seřazena** podle ceny

MAT

cislo	cena
A	1 500
B	500
C	800
D	2 000
E	2 500
F	0
G	50
H	1 200
I	100

cislo	cena
F	0
G	50
I	100
B	500
C	800
H	1 200
A	1 500
D	2 000
E	2 500

SQL Příkaz SELECT – Agregační funkce

- **Jednoduchá agregační informace za celou tabulku**

  SELECT COUNT(*), AVG(cena), MAX(dodací_množství), MIN(dodací_množství) FROM *mat*

  - Výsledkem dotazu je počet řádek tabulky (COUNT), průměrné (AVG), maximální (MAX) a minimální (MIN) dodací množství.
- **Agregační informace za skupinu**

  SELECT *číslo_dílu*, COUNT(*), SUM(přípravný_čas) FROM *operace* GROUP BY *číslo_dílu*

  - Výsledkem dotazu je výběr z tabulky operací, zjištění, kolik operací má každý díl v postupu a jaký je součet (SUM) přípravných časů pro každý díl.
- *Pozn.: Pokud se použije klausule GROUP BY, potom mohou vybraná data obsahovat jen konstanty, sloupce tabulky použité v GROUP BY a agregační funkce*

SQL Příkaz SELECT – Jednoduché testování skupin

SELECT *číslo_dílu*, COUNT(*) FROM *operace*

GROUP BY *číslo_dílu*

HAVING COUNT(*) > 5

- Výsledkem dotazu je seznam dílů a počtů operací v postupu, které mají více než 5 operací v postupu.
- *Pozn.: HAVING může obsahovat tytéž podmínky jako WHERE, ale je vhodné ji používat jen pro podmínky s agregačními funkcemi a WHERE používat jen pro výběr řádek*

SQL Příkaz SELECT – Jednoduché spojení tabulek s restrikcí s WHERE

SELECT *díly.číslo_dílu*, *číslo_operace*, *počet_kusů*,
přípravný_čas, *kusový čas*,
*počet_kusů*kusový čas+přípravný_čas* AS
Výrobní_čas_operace
FROM díly, operace
WHERE díly.číslo_dílu = operace.číslo dílu

- Nyní se vytvoří **kartézský součin** tabulek *díly* a *operace*, **restrikční** podmínkou s WHERE je, že se číslo dílu v tabulce díly rovná číslu dílu v tabulce operací. **Projekce** omezuje vybrané sloupce. Navíc je v příkladu vytvořen nový sloupec jako výraz a pomocí klíčového slova AS je vytvořen název tohoto nového sloupce.

SQL Příkaz SELECT – Jednoduché spojení tabulek s restrikcí s JOIN

SELECT *díly.číslo_dílu, číslo_operace, počet_kusů,*
*přípravný_čas, kusový čas, počet_kusů*kusový*
čas+přípravný_čas AS *Výrobní_čas_operace*
FROM díly INNER JOIN operace
ON díly.číslo_dílu = operace.číslo dílu

- Nyní se vytvoří **kartézský součin** tabulek *díly* a *operace*, **restrikční** podmínkou s INNER JOIN je, že se číslo dílu v tabulce díly rovná číslu dílu v tabulce operací. **Projekce** omezuje vybrané sloupce. Navíc je v příkladu vytvořen nový sloupec jako výraz a pomocí klíčového slova AS je vytvořen název tohoto nového sloupce.
- *Pozn.: INNER JOIN ignoruje ty řádky jedné tabulky, které nemají odpovídajícího partnera v druhé tabulce*

SQL Příkaz SELECT – Vnořený příkaz SELECT

```
SELECT Polozky.JmenoPolozky
    FROM Polozky
    WHERE ((Polozky.JmenoPolozky NOT IN
        (SELECT DISTINCT Operace.JmenoPolozky
            FROM Operace))
     AND (Polozky.Vyrabena=True))
```

- Výsledkem příkazu jsou jména vyráběných položek, které nemají v tabulce operací technologický postup. Klausule NOT IN znamená, že se prvek nenachází ve vybrané množině, klausule DISTINCT znamená, že se ve výběru mají vyloučit duplicity.

SQL Příkaz SELECT – Nalezení chybějícího partnera při spojení dvou tabulek


```
SELECT Polozky.JmenoPolozky
    FROM Polozky LEFT JOIN Operace
        ON Polozky.JmenoPolozky = Operace.JmenoPolozky
    WHERE (Operace.Číslo IS NULL)
     AND (Polozky.Vyrabena=True)
```

- Výsledkem příkazu jsou jména vyráběných položek, které nemají technologický postup. Klausule LEFT JOIN oproti INNER JOIN vybírá i ty řádky z tabulky Polozky, které nemají partnera v tabulce Operace. V tomto případě jsou všechny sloupce tabulky Operace naplněny hodnotou NULL (nedefinovaná hodnota). Restrikce tuto hodnotu testuje.
- *Pozn.: Obdobně funguje i RIGHT JOIN.*

SQL Příkaz SELECT – Výběr hodnoty s maximálním výskytem

- Vybrat čísla tavby, seskupit a seřadit sestupně podle počtu vyválcovaných balíků z jedné tavby a z výsledku zobrazit jen první řádek:

```
SELECT TOP 1 cis_tavby, count(*) AS pocet
  FROM baliky
  GROUP BY  cis_tavby
  ORDER BY pocet desc
```

- Klausule *TOP n* omezí výběr na prvních n řádek, klausule *desc* řadí sestupně (implicitně je vzestupně, pro vzestupné řazení lze použít i *asc*)

SQL Příkaz SELECT – Výběr hodnoty s maximálním výskytem – jiný způsob 1

- Použijeme příkazy SELECT vnořené v sobě.

```
SELECT cis_tavby, count(*) AS pocet
        FROM baliky
        GROUP BY cis_tavby)
```

- vypočte pro každou tavbu, kolik balíků se z ní vyválcuje.

SELECT Max(pocet) FROM *předchozí příkaz* AS bbbb

- vypočte maximání počet odválcovaných balíků.

SELECT *něco1*, *něco2* FROM *nějaká tabulka*
WHERE něco2 IN (*nějaká množina*)

- vybere jen řádky v tabulce, kde sloupec něco2 je obsažen v této množině.
- Spojíme do jednoho příkazu:

SQL Příkaz SELECT – Výběr hodnoty s maximálním výskytem – jiný způsob 2

- Spojením do jednoho příkazu obdržíme:

```
SELECT cis_tavby, pocet
  FROM
     (SELECT cis_tavby, count(*) AS pocet
          FROM baliky
          GROUP BY cis_tavby) AS aaaa
  WHERE pocet IN
       (SELECT max(pocet) FROM
          (SELECT cis_tavby, count(*) AS pocet
               FROM baliky
               GROUP BY cis_tavby) AS bbbb)
```

SQL Příkaz SELECT – Současníci 1

- V databázi osob: **Osoba**(<u>číslo osoby</u>, příjmení, jméno, datum narození, datum úmrtí)
- Určete osoby, které žijí nebo žily současně
- Budeme porovnávat dvojice osob, první s druhou, první s třetí, první se čtvrtou, ..., druhou se třetí, druhou se čtvrtou, ..., předposlední s poslední.
- Kdybychom tak nepostupovali, objevily by se nán dvojice (x, y) a (y, x).
- Současníci mohou nastat za těchto podmínek
- první osoba je narozena později než druhá a (druhá vůbec nezemřela nebo zemřela až po narození první osoby)
  - (tblOsobaPrvni.Narozeni > tblOsobaDruha.Narozeni) AND (tblOsobaPrvni.Narozeni< tblOsobaDruha.Umrti OR tblOsobaDruha.Umrti IS Null)
- první osoba zemřela a úmrtí první osoby je až po narození druhé osoby a (první osoba zemřela dříve než druhá nebo druhá osoba žije)
  - (tblOsobaPrvni.Umrti IS NOT NULL) AND (tblOsobaPrvni.Umrti > tblOsobaDruha.Narozeni) AND (tblOsobaPrvni.Umrti < tblOsobaDruha.Umrti OR tblOsobaDruha.Umrti IS Null)
- druhá osoba je narozena později než první a (první vůbec nezemřela nebo zemřela až po narození první osoby)
  - (tblOsobaDruha.Narozeni > tblOsobaPrvni.Narozeni) AND (tblOsobaDruha.Narozeni< tblOsobaPrvni.Umrti OR tblOsobaPrvni.Umrti IS Null)
- druhá osoba zemřela a úmrtí druhé osoby je až po narození první osoby a (druhá osoba zemřela dříve než první nebo první osoba žije)
  - (tblOsobaDruha.Umrti IS NOT NULL) AND (tblOsobaDruha.Umrti > tblOsobaPrvni.Narozeni) AND (tblOsobaDruha.Umrti < tblOsobaPrvni.Umrti OR tblOsobaPrvni.Umrti IS Null)

SQL Příkaz SELECT – Současníci 2

- Na tomto základu můžeme přímo napsat SQL příkaz:

```
SELECT
   tblOsobaPrvni.ID, tblOsobaPrvni.Prijmeni, tblOsobaPrvni.Jmeno,  tblOsobaDruha.ID, tblOsobaDruha.Prijmeni, tblOsobaDruha.Jmeno,
   tblOsobaPrvni.Narozeni, tblOsobaPrvni.Umrti,  tblOsobaDruha.Narozeni, tblOsobaDruha.Umrti
FROM
   tblOsoba AS tblOsobaPrvni, tblOsoba AS tblOsobaDruha
WHERE
   (tblOsobaPrvni.ID < tblOsobaDruha.ID) And (
   (tblOsobaPrvni.Narozeni > tblOsobaDruha.Narozeni) AND
         (tblOsobaPrvni.Narozeni< tblOsobaDruha.Umrti OR  tblOsobaDruha.Umrti IS Null)
                              OR
   (tblOsobaPrvni.Umrti IS NOT NULL) AND
        (tblOsobaPrvni.Umrti > tblOsobaDruha.Narozeni) AND
        (tblOsobaPrvni.Umrti < tblOsobaDruha.Umrti OR  tblOsobaDruha.Umrti IS Null)
                              OR
   (tblOsobaDruha.Narozeni > tblOsobaPrvni.Narozeni) AND
         (tblOsobaDruha.Narozeni< tblOsobaPrvni.Umrti OR  tblOsobaPrvni.Umrti IS Null)
                              OR
   (tblOsobaDruha.Umrti IS NOT NULL) AND
        (tblOsobaDruha.Umrti > tblOsobaPrvni.Narozeni) AND
        (tblOsobaDruha.Umrti < tblOsobaPrvni.Umrti OR tblOsobaPrvni.Umrti IS Null)
 )
ORDER BY  tblOsobaPrvni.ID, tblOsobaDruha.ID
```

SQL příkaz INSERT – obecný tvar pro 1 řádku

- Příkaz **INSERT** přidává nové **řádky** do tabulky.

- Obecný tvar příkazu je

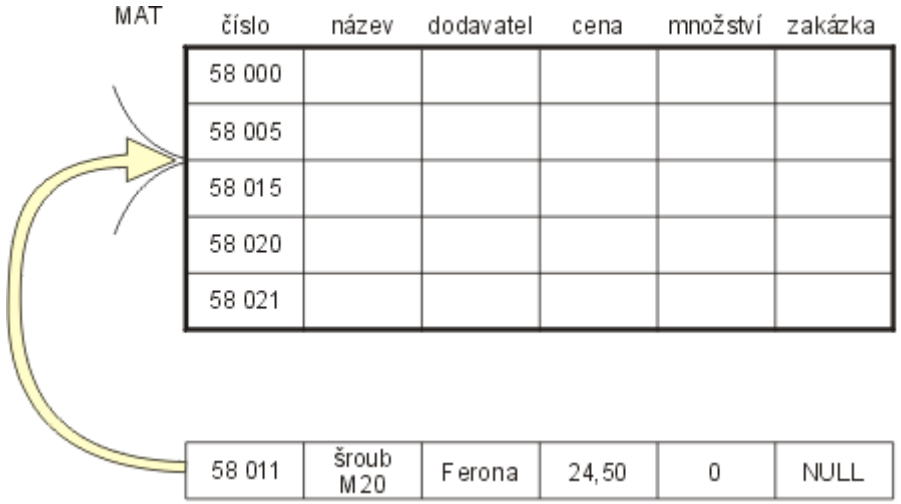
```
INSERT INTO tab [(seznam_výstupních_sloupců)]
   VALUES (seznam_hodnot)
```

- Pokud se neuvede seznam vstupních sloupců, pak se musí zadat hodnota všech sloupců. Pokud má sloupec dovoleno zadat hodnotu NULL (neznámá hodnoty), lze NULL zadat.
- Kam bude nový řádek fyzicky zařazen, nás nemusí zajímat. Při prohlížení tabulky lze totiž nastavit, podle kterého sloupce se mají záznamy řadit.

SQL příkaz INSERT – příklad

INSERT INTO *mat* VALUES (*58011*, '*šroub M20*', '*Ferona*', *24.50, 0, NULL*)

- Do tabulky *Mat* se vloží nový záznam o materiálu číslo *58011*, s názvem '*šroub M20*', dodavatelem *Ferona*', cenou *24.50* a množstvím na skladu 0, bez rezervace pro zakázku (NULL).



SQL příkaz INSERT – více řádek
Kombinace INSERT a SELECT

- *(příklad je z databáze ACCESS, nepoužívá se klausule VALUES).*

```
INSERT INTO XPolozky ( JmenoPolozky, PrubeznaDoba,
                                     CenaZPostupu )
   SELECT DobaACenaOperaceSuma.JmenoPolozky,
          DobaACenaOperaceSuma.SumOfCelkDobaOperace,
          DobaACenaOperaceSuma![Soucet CenaOperace]
        AS Výraz
    FROM DobaACenaOperaceSuma
```

- Z dotazu *DobaACenaOperaceSuma* se vyberou tři sloupce. Ty jsou vloženy do tabulky *XPolozky* jako nový záznam.

SQL příkaz UPDATE– obecný tvar

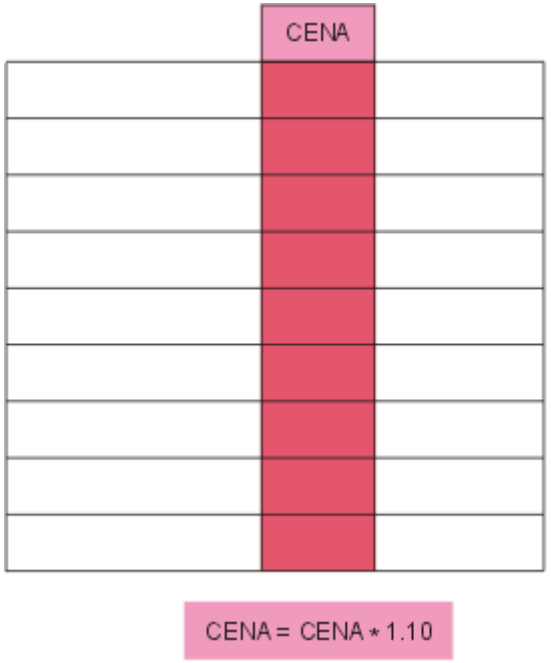
```
UPDATE tab
    SET NázevSloupce1     = hodnota1
          [, NázevSloupce2 = hodnota2 ...]
    [WHERE <podmínka_řádku>]
```

- Nahradí jmenované sloupce v řádcích daných podmínkou novými hodnotami.

SQL příkaz UPDATE– Jednoduchá změna sloupce pro všechny řádky

UPDATE *mat* SET *cena* = *cena**1.10

- Příkaz zvyšuje cenu všech materiálů v ceníku o 10% (ve všech **řádkách tabulky**).



SQL příkaz UPDATE– Změna sloupce pro vybrané řádky

UPDATE *duchody* SET *vyplatit* = *vyplatit* + 500

WHERE *vyplatit* < 10000

- Příkaz zvýší důchod o 500 Kč všem důchodcům, kteří pobírali méně než 10000 Kč.

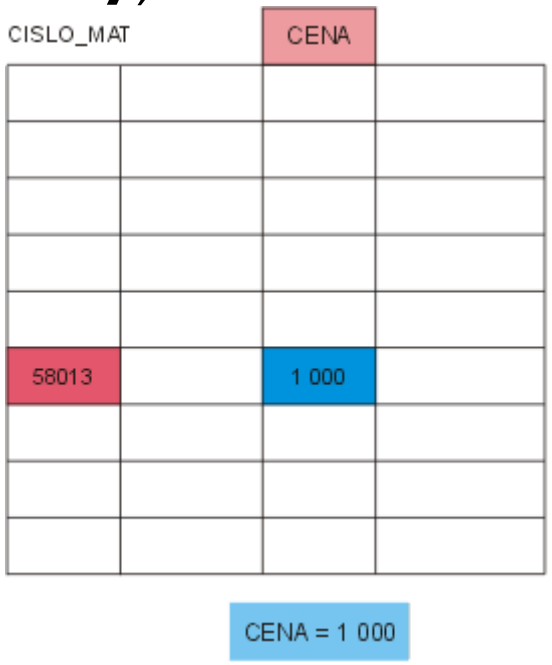
VYPLATIT
10 000
5 000
8 000
15 000
5 000
18 000
7 500
12 000
8 000

VYPLATIT = VYPLATIT + 500

SQL příkaz UPDATE– Změna sloupce pro vybraný řádek


```
UPDATE mat SET cena = 1000
        WHERE číslo_mat = 58013
```

- Příkaz nastaví cenu materiálu 58013 na 1000 Kč (**jednu buňku tabulky**).



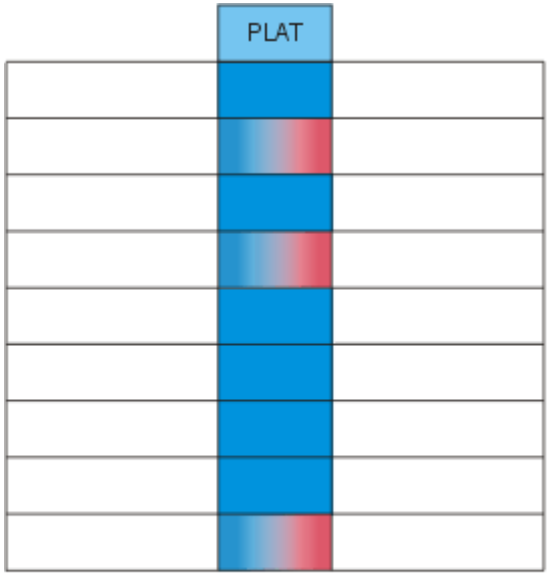
SQL příkaz UPDATE– *Kombinace příkazu UPDATE a SELECT*

UPDATE *zam*

SET *plat* = 1.1**plat*

WHERE *plat* < 0.5 * (SELECT AVG(*plat*) FROM *zam*)

- Příkaz zvýší o 10% plat všem zaměstnancům, kteří mají nižší plat než je polovina průměru platu všech zaměstnanců.



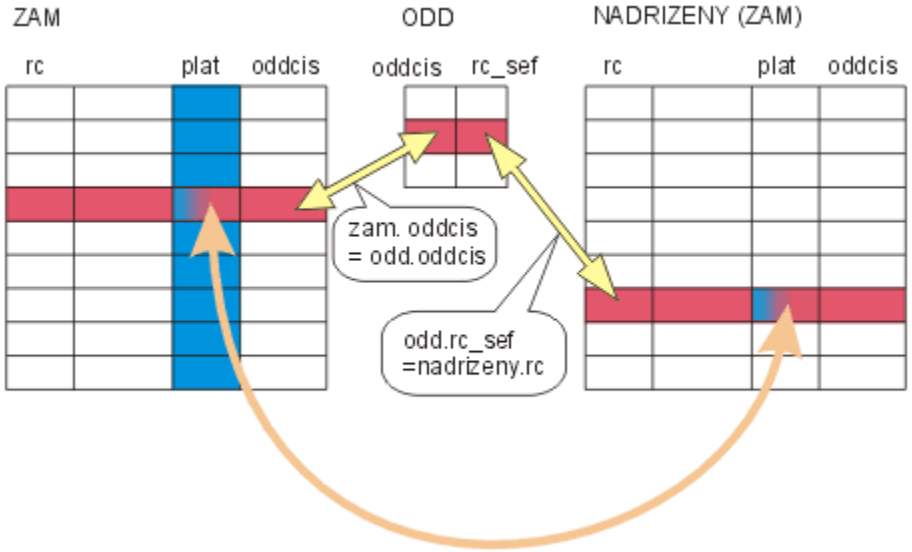
$p = \Sigma\ plat_i / n$

plat = plat * 1.1 (pokud plat < p)

SQL příkaz UPDATE– *Složitější kombinace UPDATE a SELECT*


```
UPDATE zam SET plat = plat + 1000
    WHERE plat < 0.5*
     (SELECT plat FROM zam nadrizeny, odd
          WHERE (zam.oddcis = odd.oddcis)      AND (odd.rc_sef = nadrizeny.rc))
```

- Příkaz zvyšuje plat všem zaměstnancům, jejichž plat nedosahuje polovinu platu svého šéfa. V tomto okamžiku jsou ve hře dvě tabulky, zaměstnanci *zam* a oddělení *odd*. Tabulka *zam* se používá dvakrát, neboť jednou se jedná o plat zaměstnance, podruhé plat šéfa. To se rozliší tak, že v příkazu SELECT se v klausuli FROM zavede pro tabulku *zam* tzv. **alias** nadrizeny. První část podmínky v příkazu SELECT vybírá pro zaměstnance řádku jeho oddělení a druhá část podmínky propojuje oddělení s řádkou jeho šéfa v tabulce zaměstnanci.



SQL příkaz DELETE – obecný tvar

- Příkaz **DELETE** umožňuje rušit řádky v tabulce.

- **Obecný tvar příkazu je:**

DELETE FROM tab
    [WHERE <podmínka_řádku>]

SQL příkaz DELETE – příklady

- **Výmaz všech řádků tabulky**

DELETE FROM *tab*

- Vymaže **všechny řádky** tabulky *tab*, tabulka zůstane zachována, ale bude prázdná.

- **Výmaz řádků se zadanou vlastností**

DELETE FROM *mat*
WHERE *cena* IS NULL

- Vymaže všechny materiály s nezadanou cenou (**vybrané řádky**).

- **Výmaz řádků se spojením tabulek**

DELETE FROM prac
WHERE cisprac NOT IN (SELECT DISTINCT cisprac FROM operace)

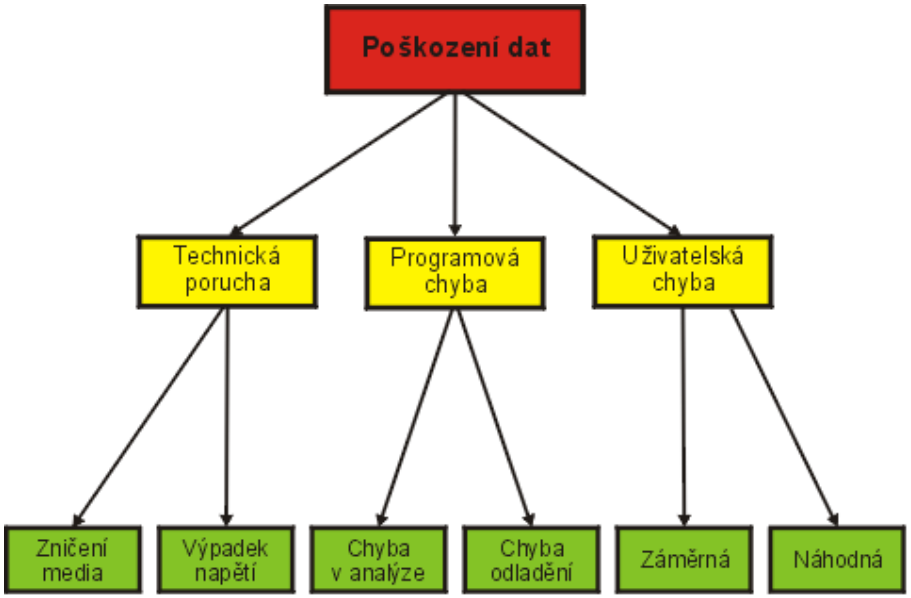
- nebo

DELETE FROM prac
WHERE NOT (cisprac = ANY (SELECT DISTINCT cisprac FROM operace))

- Vymaže řádky z tabulky pracovišť, které nejsou použita v postupech (**vybrané řádky**).

Příčiny poškození databáze

- **Žádný technický systém není zcela bezpečný**. Různými opatřeními můžeme **bezpečnost zvýšit**, **snížit možnost poruchy nebo havárie.**
- V zásadě můžeme rozdělit možné zdroje poškození dat na:
  - **technické** (výpadky proudu, poškození diskového media)
  - **programové** (nedostatečná analýza nebo odladění programu)
  - **uživatelské** (nedostatečná kvalifikace, nesvědomitost nebo zlá vůle)



Technická porucha - Zničení media

- **Spolehlivost** paměťových medií je dnes velmi **vysoká**. Přesto každý z nás zná situaci (vlastní zkušenost nebo někdo známý), kdy dojde k trvalé poruše disků a data jsou bez náhrady zničena. Jak se tomu dá předejít?
- Nejjednodušší metodou je časté **zálohování** paměťových medií. V některých případech to zcela postačí, v nejhorším případě přijdeme o práci od **poslední zálohy**. Ale jistě si dovedeme představit situace, kdy periodické zálohování nestačí. Jedná se např. o bankovní systémy, systémy řízení dopravy, automatické sklady a systémy péče o pacienta. V tomto případě je třeba, aby i v případě zničení paměťového media, informační, resp. řídící, systém pokračoval ve své činnosti bez přerušení. Takové systémy provádění zálohování v reálném čase. V principu existují dvě metody:
- **zrcadlení** dat, kdy operační systém zajišťuje paralelní zápis dat na dvě nebo více paralelních medií a je okamžitě schopen pokračovat i při výpadku jednoho media, případně v reálném čase vytvořit další kopii datové základny
- **přírůstková**, kdy se kromě pravidelného zálohování datové základny vytváří seznam všech změn v souborech na jiném mediu a v případě výpadku se buď ze zálohy a změn automaticky vytvoří nový aktuální systém (to trvá určitý čas) nebo se určitý čas pracuje současně se zálohou a změnami (což může zpomalit zpracování)

Technická porucha - Výpadek napětí

- Výpadek napětí a násilné vypnutí počítače může způsobit, že **změna** dat není provedena **důsledně** na všech místech. Změna buď musí proběhnout důsledně ve všech souvislostech, nebo nesmí proběhnout vůbec.
- *Příklad: převody peněz mezi účty*.
  - Existuje požadavek na převod mezi účty. Správný výsledek je, že jeden účet je snížen o určitou částku, druhý zvýšen a požadavek na změnu je označen jako vyřízený. Takové změny se řeší pomocí tzv. transakčního zpracování, které je předmětem dalšího článku.
- *Příklad: valorizace mezd*
  - Existuje požadavek na valorizaci mezd o dané procento pro všechny zaměstnance v podniku. Pokud by aktualizace neproběhla pro všechny pracovníky a došlo k přerušení programu, pak by mohlo dojít ke dvěma situacím:
  - někdo by nedostal přidáno
  - pokud by operátor po opětném zapnutí počítačem znovu spustil valorizační program, někdo by dostal přidáno dvakrát.
- Také taková úloha je řešitelná transakčním zpracováním, ale to je náročné na množství zapamatovaných změněných dat. Úlohu lze programovat také tak, že se vytvoří kopie dat, tam (ne)proběhne valorizace. Pokud vše dopadne dobře, valorizovaná kopie se zpětně zkopíruje do původních dat.
- Obecně lze říci, že klasické zpracování souborů je odolnější vůči výpadku napětí než databázové zpracování v reálném čase, neboť se postupně tvoří různé verze souborů a přerušené zpracování lze spustit z definovaného bodu.

Programová chyba

- Platí věta o úplné indukci v informatice.
  - *Každý dostatečně velký informační systém obsahuje chybu.*
  - Matematicky je tato věta nedokazatelná (dokonce nesmyslná), ale statisticky lze pravdivost tvrzení ověřit na konkrétních programových produktech.
- Opatření, jak pravděpodobnost chyby snížit lze rozdělit na:
  - **konstruktivní** opatření (užití moderních softwarových nástrojů pro analýzu, návrh a implementaci programových produktů, použití metodik a směrnic při vývoji software, vyloučení nebezpečných a nepřehledných programových konstrukcí)
  - metody **ověřování** spolehlivosti (matematické, metody bílé a černé schránky)
  - častým zdrojem chyb jsou dodatečné požadavky na změnu funkčnosti. Bohužel tyto změny ze strany uživatele jsou velmi časté, neboť obvykle až při užívání programů si uživatel uvědomí, co vlastně všechno chce nebo může chtít.

Uživatelská chyba - *Náhodné poškození dat*

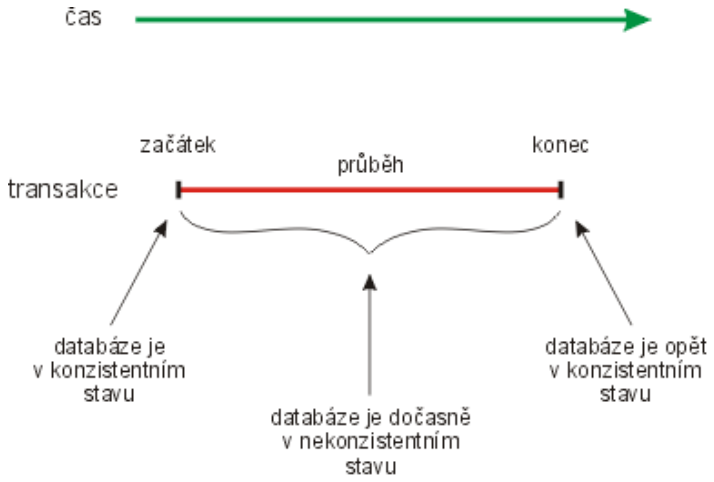
- Obvyklou chybou je **nedostatečná kvalifikace**. Zejména moderní programové produkty vycházejí z intuitivního ovládání. Žádný informační systém nemůže zabránit, aby uživatel omylem nepoškodil určitá data. Jedinou ochranou je systém přístupových práv. Ten např. znemožňuje, aby personalista omylem upravil časy v technologických postupech nebo postupář snížil plat ředitele.
- Nepozornost a nesvědomitost při obsluze informačního systému lze omezit vhodnou **personální politikou, školením, pozitivní nebo penalizační motivací**.

Uživatelská chyba - Záměrné poškození dat

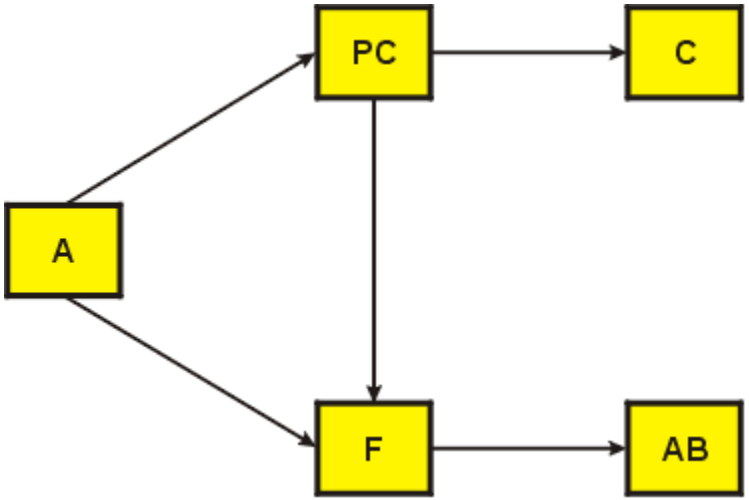
- Zlou vůli lze výrazně omezit rovněž systémem přístupových práv, výběrem a prověřením osob, omezením přístupu nekompetentním osobám a dalšími programovými a technickými prostředky.
- Důležité je rovněž pravidelná aktualizace operačního systému, antivirových programů a vhodné nastavení firewalů.

Transakční zpracování - princip

- Základní myšlenkou **transakčního** zpracování je rozdělit úlohu na **malé úseky**, které by měly buď proběhnout **celé**, nebo by měla být data ve stavu, jako by zpracování úseku vůbec **nebylo zahájeno**.
- Prakticky to probíhá tak, že při práci s daty se automaticky v dočasných souborech **zapamatovává původní obsah** měněných záznamů. Pokud vše proběhne dobře, původní obsah se v dočasných souborech smaže. Pokud dojde k přerušení programu, je po restartu **automaticky** z těchto změn **obnoven** stav **před zahájením** zpracování úseku.
- V databázích je obvyklé označit zahájení transakce označit vhodným příkazem (např. **BEGIN TRANSACTION**). Dokončení transakce se oznámí databázovému systému jiným příkazem (téměř standardně **COMMIT**). Navíc dostává programátor do rukou nástroj, jak transakci programově přerušit a vrátit data do původního stavu (**ROLLBACK**).



Transakční zpracování – stavy transakce



ku provádění transakce)
tav po provedení poslední operace transakce)
rmálním průběhu transakce nelze pokračovat)
LBACK, databáze je ve stavu před transakcí)
končení, tj. po provedení operace COMMIT)

v. **dvoufázovým protokolem**.

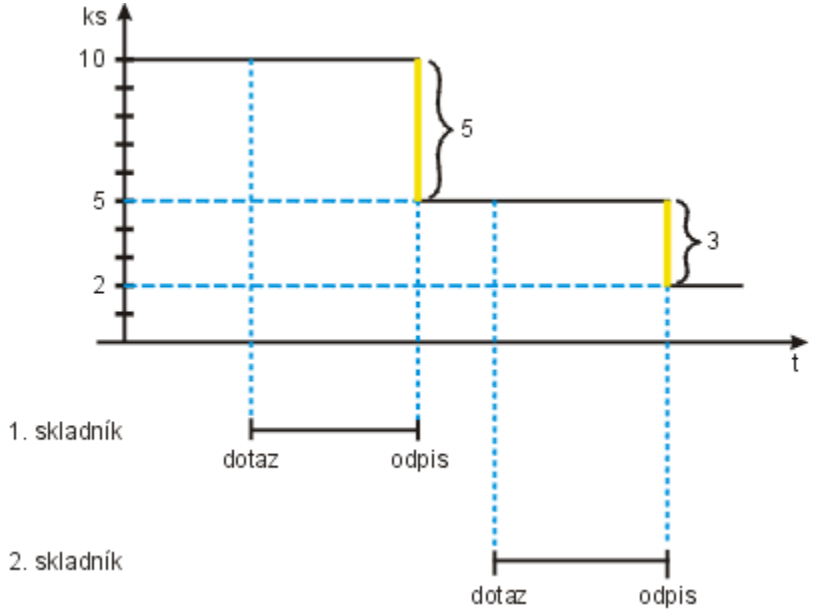
ěz z jedné banky do druhé. Každá má svůj informační systém a svůj
by převod skutečně proběhl na obou počítačích. Řešení této úlohy
su.

Víceuživatelský přístup – příklad

- Představme si následující situaci. Ve skladu pracují dva skladníci, každý má svůj terminál. Na skladě je 10 šroubů. Prvý skladník vydá 5 šroubů, druhý skladník 3 šrouby. Oba svůj výdej zapíší řádně do počítače.
  - Kolik šroubů je na skladu?
  - Kolik šroubů je evidováno v databázi?
- Zatímco odpověď na první otázku je jednoduchá - 2 šrouby, odpověď na druhou otázku je poněkud složitější.

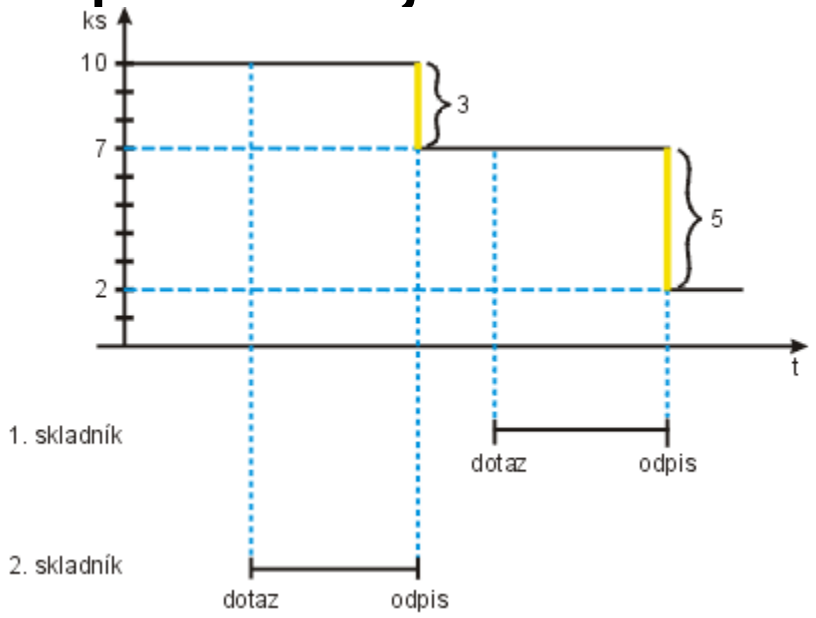
Víceuživatelský přístup – varianta A

- A) předpokládejme, že 1. skladník vydal 5 šroubů, výdej řádně zapsal, zůstatek je 5 šroubů, pak 2. skladník vydá 3 šrouby, v evidenci počítače jsou 2 šrouby (správně).



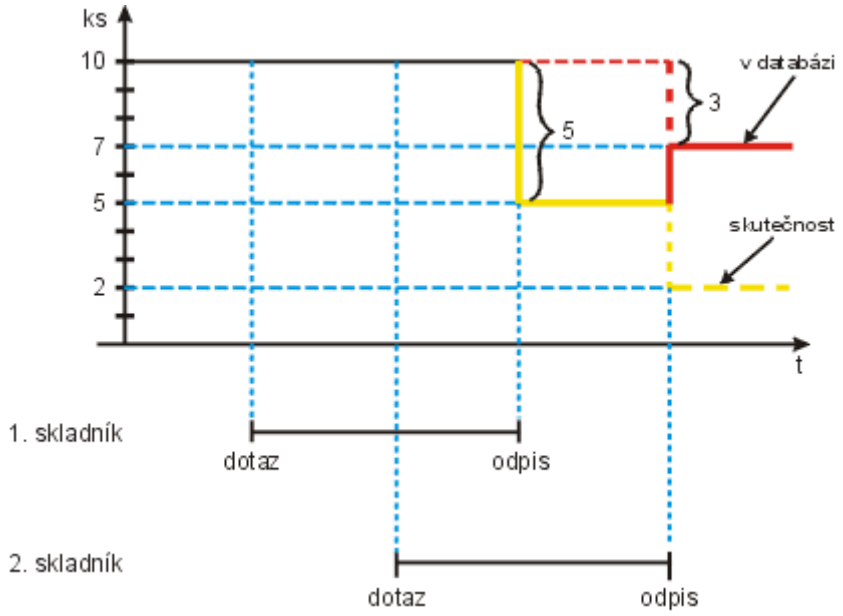
Víceuživatelský přístup – varianta B

B) předpokládejme, že 2. skladník vydal 3 šrouby, výdej řádně zapsal, zůstatek je 7 šroubů, pak 1. skladník vydá 5 šroubů, v evidenci počítače jsou 2 šrouby (správně).



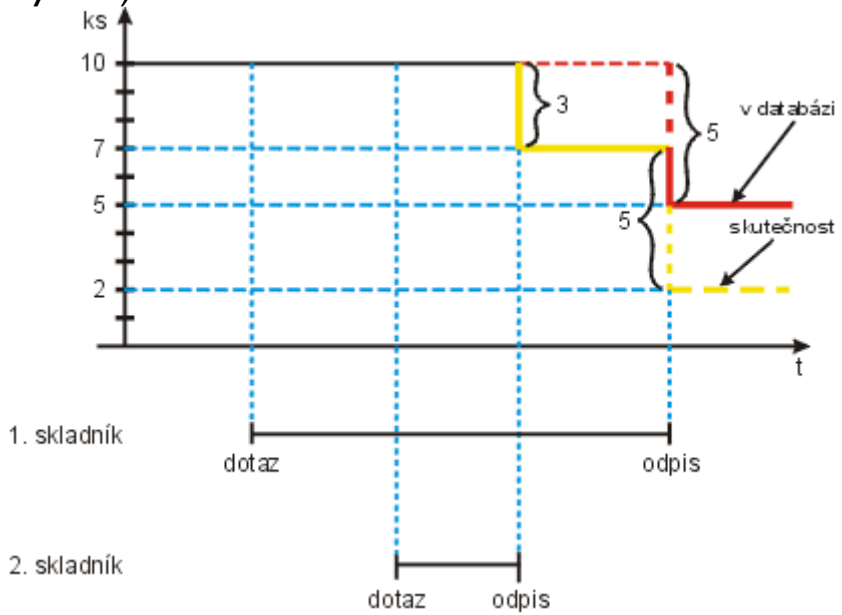
Víceuživatelský přístup – varianta C

C) předpokládejme, že 1. skladník se podíval do databáze, zjistil, že na skladu je 10 šroubů, druhý udělal totéž, pak první skladník odepsal 5 šroubů, v databázi je 5 šroubů, ale na obrazovce 2. skladníka je stále ještě 10 šroubů, on odepíše 3, a zapíše do databáze 7 šroubů (chybně).



Víceuživatelský přístup – varianta D

- D) předpokládejme, že 1. skladník se podíval do databáze, zjistil, že na skladu je 10 šroubů, druhý udělal totéž, pak druhý skladník odepsal 3 šrouby, v databázi je 7 šroubů, ale na obrazovce 2. skladníka je stále ještě 10 šroubů, on odepíše 5, a zapíše do databáze 5 šroubů (chybně).



- Jedná se o programátorskou chybu. Jaké je řešení?

Víceuživatelský přístup – Organizační opatření

- Většinou lze zajistit, aby si při víceuživatelském přístupu uživatelé **rozdělili kompetence** (jeden skladník vydává šrouby a druhý hřebíky). Toto řešení však sníží možnost kolizí, ale nemůže je zcela vyloučit (nemožnost rozdělení, chyba uživatele).

Víceuživatelský přístup – Zamykání záznamů

- V okamžiku, kdy je proveden dotaz a očekává se změna dat, tak se všechny záznamy, které vstupují do dotazu, **zamknou**. Další pokusy jiných uživatelů pracovat nad stejnými záznamy se buď **odmítnou** a těmto uživatelům se **zobrazí zpráva**, že data jsou uzamčená, nebo další uživatelé musí **čekat na odemčení** bez zprávy. Až je **dokončeno zpracování** pro uživatele, který data požadoval jako první, jsou příslušné datové záznamy **odemčeny**.
- Zamykání záznamů může mít dva nepříjemné důsledky:
  - Uživatel si data zamkne, odejde od terminálu a tím zablokuje práci ostatních.
  - V horším případě uživatel **X** uzamkne část dat A později se pokusí o uzamčení části dat B. Uživatel **Y** nejdříve zamkne část dat B a později se pokusí zamknout část dat A. Ta je již uzamčena uživatelem **X**. Došlo tzv. **zablokování** (**deadlock**, deadly embrace). Moderní databáze umí tento problém řešit.
- Obecně by se mělo provádět uzamčení jen na nezbytně dlouhou dobu. Ideální je pouze na provádění změny počítačem. V případě rezervace letenek např. po dobu, kdy zákazník hovoří s prodejcem.

Víceuživatelský přístup – Detekce kolize

- V tomto případě se záznamy nezamykají. Každý záznam je interně označen tzv. časovým razítkem (může to být i čítač, kolikrát byl daný záznam změněn). V okamžiku, kdy se má zapisovat do databáze, program se pokusí **celou databázi** zamknout.
- Pokud se to nepodaří, **čeká** (uzamčení trvá jen po dobu zápisu dat, typicky zlomky sekundy - není v něm zahrnuta doba, kdy uživatel si prohlíží databázi). Jakmile databázi zamkne, provede se **opětné interní čtení** a porovnává se, zda se **nezměnilo časové razítko**.
- Pokud ne, data se zapíší, jinak se vydá uživateli zpráva, že má **kolizi** s někým jiným a že musí svůj dialog **zopakovat**.
- Protože je velmi málo pravděpodobné, že dva uživatelé pracují nad stejnými záznamy, dochází ke kolizi jen velice zřídka, kolize se nepromítne do dat a mechanismus zabezpečení zaručuje integritu dat.

Datová struktura

- Datová struktura je účelné uspořádání prvků, částí nebo složek datového celku podle určitého jednotícího principu a soustava vnitřních vztahů, souhrn pravidel a omezení, účelně organizujících složky tohoto celku.
- Rozlišujeme **logickou** a **fyzickou** úroveň práce s datovou strukturou. Logická úroveň je abstraktní představou, fyzická představuje její implementaci. Struktura může být implementována několika způsoby. Např. komplexní číslo může být implementováno v kartézských nebo polárních souřadnicích, seznam lze implementovat polem nebo zřetězením dynamických proměnných.

Sémantika datové struktury

- **Sémantika** datové struktury (DS) určuje:
  - jak se struktura vytvoří, resp. kdy vzniká
  - jakými operacemi se s existující strukturou manipuluje a jak se struktura těmito operacemi mění
  - jak se struktura ruší, resp. kdy zaniká
  - jak se DS identifikuje
  - jaké má DS vlastnosti

Základní pojmy z datových struktur

- **Schéma** datové struktury, někdy se mluví o **typu** datové struktury. Schéma datové struktury se vyskytuje v systému **pouze jednou**.
- Způsob zavedení schémat se liší podle nástroje (programovacího jazyka), který se používá (např. definice/deklarace typu v strukturálních jazycích, zavedení třídy v objektové technologii, apod.).
- Konkrétně zavedeným a používaným datovým strukturám určitého typu, které mají každá své jméno, se říká **instance**. Počet instancí určitého typu není omezen.
- Existuje-li pro danou datovou strukturu aparát, který umožní jednorázové její naplnění DS konkrétními daty, říkáme, že pro datovou strukturu existuje **konstruktor.**
  - *Např. datová struktura pole (array) ve většině programovacích systémů nemá konstruktor.*
- Je-li možno přistupovat a pracovat s jednotlivými prvky datové struktury, říkáme, že pro datovou strukturu existuje **selektor.**
  - *Např. datová struktura množina nemá selektor. Selektorem datové struktury pole je index. Selektorem datové struktury záznam je jméno položky.*

Klasifikace datových struktur

- *Podle **strukturovanosti** na:*
  - **Jednoduché** (J) - nelze je rozložit na jednodušší
  - **Strukturované** (Str) - skládají se z jednodušších struktur

- *Podle **typu** položek na:*
  - **Homogenní** (H) - všechny položky struktury jsou stejného typu
  - **Nehomogenní** (Nh) - položky struktury mohou být různých typů

- *Podle **polohy** sousedních položek na:*
  - **Lineární** (L) - sousední položky ve struktuře lze uspořádat do posloupnosti s uspořádáním předchůdce/následník
  - **Nelineární** (NL) - sousední položky nelze uspořádat do lineární posloupnosti

- *Podle **změn počtu** položek na:*
  - **Statické** (Sta) - existují po celou dobu zpracování
  - **Dynamické** (D) - vznikají a zanikají během zpracování

Vybrané základní struktury

Struktura	Strukturovanost	Homogenita	Linearita	Změny počtu
Konstanta	J	H		Sta
Jednoduchá proměnná	J	H		Sta
Numerická proměnná	Str	H	L	Sta
Textový řetězec , vektor	Str	H	L	Sta
Matice	Str	H	NL	Sta
Věta (záznam)	Str	Nh	L	Sta
Textový soubor	Str	Nh	L	D
Soubor s typem	Str	H	L	D
Seznam	Str	H	L	D
Tabulka	Str	H	L	D
Fronta	Str	H	L	D
Zásobník	Str	H	L	D
Obecný strom	Str	H	NL	D
Hromada	Str	H	NL	D
Síť, graf	Str	Nh	NL	D

Konstanty

- **Sémantický význam**
  - Instance určitého typu, která nemění během své existence hodnotu.
- **Povolené operace**
  - Aritmetické a jiné operace dle typu.
- **Vlastnosti**
  - Typ, Hodnota

- *Příklady:*
  - **logické** konstanty: true, false
  - **znaková** konstanta: 'A', '$', '1'
  - **celočíselná** konstanta: 9,-3
  - **reálná** konstanta: 3.14159, -1.23E-23
  - **textová** konstanta: 'Franta je lump'

Jednoduché proměnné - Celočíselná proměnná

- **Abstraktní představa**

S .... znaménko
H .... hodnota

- **Sémantický význam**
  - Místo v paměti pro uložení hodnoty celého čísla n, pro které platí MIN <= n <= MAX, kde MIN,MAX jsou jistá celá čísla.
- **Vlastnosti**
  - Název, hodnota, zobrazení, rozsah (podle druhu jazyka)
- **Povolené operace:**
  - aritmetické operace (sčítaní, odečítání, násobení, celočíselné dělení, zbytek)
  - relační operace (<. <=, =, >=. >, <>)
  - přiřazení hodnoty
  - testy (sudé, liché)
  - celočíselné funkce

Jednoduché proměnné - Reálná proměnná

- **Abstraktní představa**

S .... znaménko mantisy
M .... mantisa
Z .... znaménko exponentu
E .... exponent

- **Sémantický význam**
  - Místo v paměti pro uložení modelu reálného čísla v semilogaritmickém tvaru, pro něž platí MIN <= r <= MAX, s určitým počtem platných desetinných míst a nejmenším kladným číslem, které lze odlišit od nuly.
- **Vlastnosti**
  - Název, hodnota, zobrazení, rozsah (podle druhu jazyka)
- **Povolené operace:**
  - aritmetické operace
  - přiřazení hodnoty
  - relační operace

Jednoduché proměnné - Komplexní proměnná

- **Abstraktní představa**

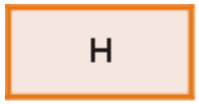
RS	RM	RZ	RE
IS	IM	IZ	IE

RS .... znaménko reálné části
RM .... mantisa reálné části
RZ .... znaménko exponentu reálné části
RE .... exponent reálné části
IS .... znaménko imaginární části
IM .... mantisa imaginární části
IZ .... znaménko exponentu imaginární části
IE .... exponent imaginární části

- **Sémantický význam**
  - Místo v paměti pro uložení komplexního čísla v semilogaritmickém tvaru.
- **Vlastnosti**
  - Název, hodnota, zobrazení
- **Povolené operace:**
  - aritmetické operace
  - přiřazení hodnoty
  - získání reálné a imaginární části
  - relační operace

Jednoduché proměnné - Logická (Booleovská) proměnná

- **Abstraktní představa**

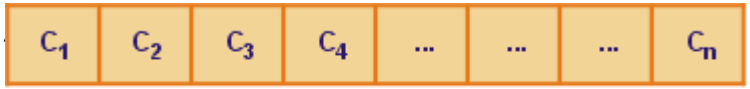


H . . . . logická hodnota (true/false)

- **Sémantický význam**
  - Místo v paměti pro uložení dvouhodnotové proměnné (pravda/nepravda).
- **Vlastnosti**
  - Název, hodnota, zobrazení
- **Povolené operace:**
  - logické operace
  - přiřazení hodnoty
  - logické funkce

Jednoduché proměnné - Numerická proměnná

Speciální typ v některých databázových systémech nebo v jazyku Cobol.

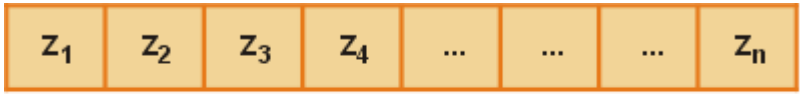
- 

C_i ... číslice, speciální znaky ('+', '-', '.')

- **Sémantický význam**
  - Místo v paměti pro uložení posloupnosti po sobě jdoucích numerických a speciálních znaků ('+','-') povolených pro zápis čísla. Desetinná tečka pro racionální čísla je většinou myšlená.
- **Vlastnosti**
  - Název, hodnota, délka
- **Povolené operace:**
  - Všechny operace jako u jednoduchých číselných proměnných.

Indexované proměnné
Textový řetězec

- **Abstraktní představa**



Z_i ... alfanumerické znaky

- **Sémantický význam**
  - Místo v paměti pro uložení posloupnosti po sobě jdoucích alfanumerických a speciálních znaků, které jsou v textu povoleny .
- **Vlastnosti**
  - Název, hodnota, délka
- **Povolené operace:**
  - přesun celého řetězce nebo jeho části
  - srovnání dvou řetězců
  - prohlížení řetězce znak po znaku
  - spojení dvou řetězců
  - rozdělení dvou řetězců
  - změna znaku v řetězci
  - vyhledání podřetězce
  - vypuštění podřetězce
  - vložení řetězce do jiného řetězce
- *Pozn.: Řetězce mohou být v paměti implementovány různě:*
  - zakončení binární nulou (C, Delphi)
  - uvedené binární délkou (Pascal standardní i v Delphi).
  - uvedené délkou a počtem užití (Pascal v Delphi)

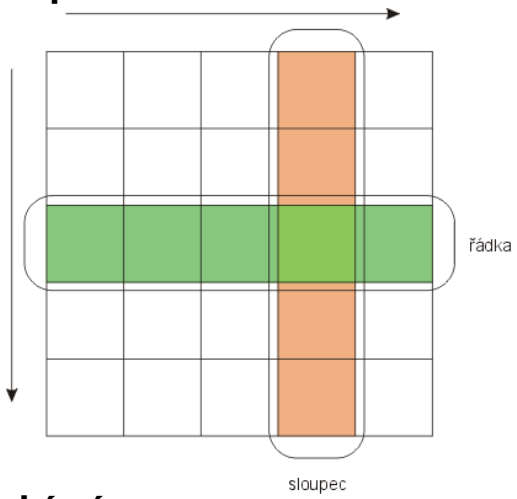
Indexované proměnné - Jednorozměrné pole - vektor

- **Abstraktní představa**

- **Sémantický význam**
  - n prvků stejného typu opakovaně uloženo za sebou. Na ně lze odkazovat pořadovým indexem i <1,n> nebo/někdy i <MIN,MAX> (MIN,MAX může být i záporné). Prvek pole může mít vnitřní strukturu.
- **Vlastnosti**
  - Název, hodnota jednotlivých prvků, max. počet opakovaných prvků, typ prvků
- **Povolené operace:**
- **přiřazení** hodnoty určitému prvku
- **přiřazení** stejné hodnoty všem prvkům
- **vyhledání** prvku s určitou hodnotou.
- **seřazení** prvků podle velikosti určité hodnoty prvku
- pro číselné hodnoty prvků vektorů i **vektorové operace**

Indexované proměnné - Vícerozměrné pole - dvou a vícerozměrná matice

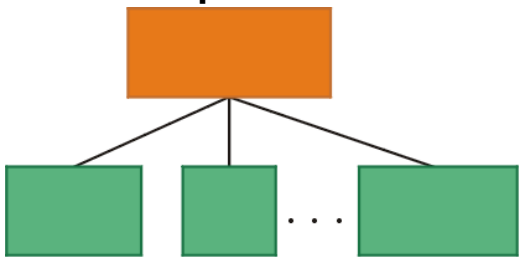
- **Abstraktní představa**



- **Sémantický význam**
  - Jednorozměrné pole, jehož prvkem je jednorozměrné nebo vícerozměrné pole
- **Vlastnosti**
  - Název, hodnota jednotlivých prvků, max. počet opakovaných prvků v řádku, max. počet řádků,(max. počet v dalších dimenzích,) typ prvků.
- **Povolené operace:**
- **přiřazení** hodnoty určitému prvku
- **přiřazení** stejné hodnoty všem prvkům
- **vyhledání** prvku s určitou hodnotou.
- **seřazení** prvků podle velikosti určité hodnoty prvku
- pro číselné hodnoty prvků vektorů i **vektorové operace**

Skupinová proměnná (skupina struktur)

- **Abstraktní představa:**



Sémantický význam:

- Několik datových struktur tvoří skupinu
- Na rozdíl od pole jde o skupinu struktur různého typu
- Sounáležitost ke skupině je obvykle odvozena z hlediska použití dat
- Skupina má buď svoje jméno, nebo číslo úrovně
- Datové struktury, ze kterých se struktura skládá, mohou být opět skupinové proměnné

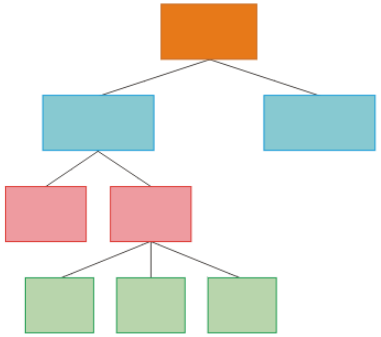
- **Vlastnosti**
  - Název
  - Výčet struktur, které skupinu tvoří
- **Povolené operace:**
  - přesun skupiny nebo její části
  - nulování všech prvků skupiny
  - porovnání dvou skupin
- **Pozn.:**
  - V některých jazycích nemusí být implementovány všechny operace (např. nulování)
  - Hloubka vnoření bývá omezena (např. Cobol - 49).

Záznam (struktura, věta, record)

- **Abstraktní představa:**

  Rozlišujeme lineární větu, kterou tvoří jednotlivé, dále nedělitelné položky a hierarchickou větu (se skupinovými proměnnými).

a) lineární věta



b) hierarchická věta

Sémantický význam:

- Záznam je lineární skupina datová struktura (položek) na logické úrovni, se kterými se pracuje buď najednou (I/O operace, přesuny) nebo s každou zvlášť. Nejčastěji popisuje nějaký entitní typ (zaměstnanec, materiál, objednávka) nebo vztah (materiál pro výrobu dílu).
- **Vlastnosti**
  - Pevná nebo proměnná délka záznamu
  - Počet položek záznamu
- **Povolené operace:**
  - **přiřazení** celého záznamu jinému záznamu
  - **nulování** záznamu (naplnit každou položku její hodnotou NULL).
  - **vstupní/výstupní operace** - čtení ze souboru a zápis do něj.
  - získání **hodnoty** jednotlivých položek
  - **změny** jednotlivých položek

Soubor (File)

- Soubor je posloupnost záznamů, zpravidla uložená na externí paměti.
- **Abstraktní představa**
- **Sémantický význam**
  - Lineární množina výskytu datové struktury záznam
  - Počet prvků souboru lze měnit programem (přidávat a ubírat na konci)
  - Prvky souboru jsou na sobě nezávislé a je možno je samostatně zpracovat

a) s pevnou délkou záznamu

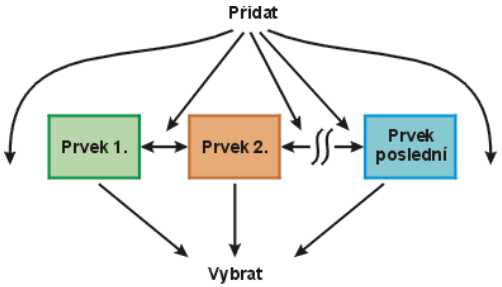
- **Vlastnosti**
  - Název (fyzické jméno - jméno, pod kterým zná soubor operační systém)
  - Identifikace (logické jméno - pod kterým je soubor znám v programu)
  - Organizace souboru
  - Způsob přístupu k informacím (sériový, přímý)
  - Typ a počet záznamů v souboru

b) s proměnnou délkou záznamu

- **Povolené operace:**
  - **přidělení** logického **jména** k fyzickému - identifikace souboru - např. Assign (F1,'Seznam_pracovníků')
  - **otevření a uzavření** souboru - zpřístupnění, test existence a ukončení práce se souborem
  - Čtení/zápis/změna **obsahu** existujícího záznamu
  - **zavedení** nového/**zrušení** existujícího záznamu
  - **vytvoření** prázdného souboru, **zrušení** souboru, **rozšíření** souboru (pokračování v zápisu), **kopírování** souboru
  - **řazení** souboru (podle nějakého klíče)

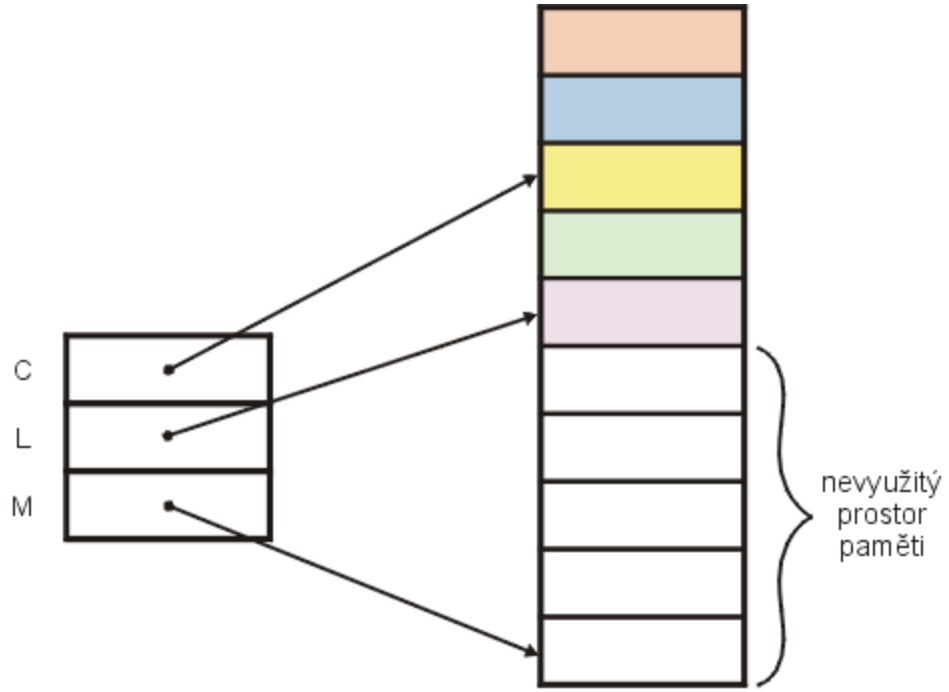
Seznam – lineární struktura základní vlastnosti

- **Abstraktní představa seznamu**



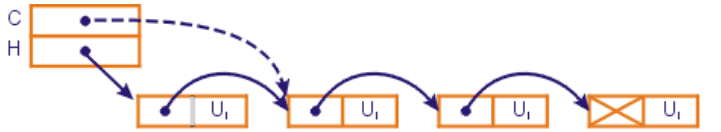
- **Sémantický význam**
  - Seznam je **dynamická lineární homogenní** datová struktura s proměnným počtem prvků, představovaná posloupností jednotlivých prvků. Prvky seznamu jsou **uspořádané**. Seznam může být prázdný nebo může obsahovat prvky stejného typu. Každému prvku neprázdného seznamu, který není na konci seznamu, jednoznačně přísluší jeden další prvek a každému prvku neprázdného seznamu, který není na začátku seznamu, jednoznačně přísluší jeden předchozí prvek. Prvek seznamu, který se právě zpracovává, se nazývá **běžný**.
- **Povolené operace** se seznamem:
  - **vytvoření** prázdného seznamu (inicializace seznamu)
  - **vložení** nového prvku do seznamu za běžný prvek
  - **vložení** nového prvku do seznamu před běžný prvek
  - **vyjmutí** určitého prvku do seznamu
  - **zpřístupnění** běžného prvku seznamu
  - **nastavení běžného** prvku na začátek nebo konec
  - **posunutí běžného** prvku na **předchozí** prvek
  - **posunutí běžného** prvku na **následující** prvek
  - test, zda je seznam **prázdný**
  - test zda běžný prvek je **první**
  - test, zda běžný prvek je **poslední**

Implementace seznamu polem

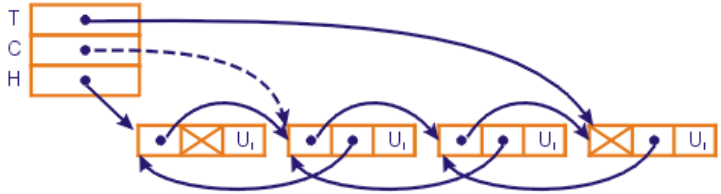


- **Implementace polem** požaduje předem definovat dostatečně dlouhé pole. Tomuto poli přísluší **řídící proměnné**, které obsahuje index **posledně obsazeného** prvku pole (L), **délku** rezervovaného pole (M) a index **běžného** prvku (C). Realizace seznamu polem má řadu nevýhod. Kromě nutnosti statické rezervace zbytečně velkého pole a nutnosti přesunů prvků pole při vkládání a rušení není možné další zobecnění.

Implementace seznamu spojovým seznamem



a) jednoduché zřetězení

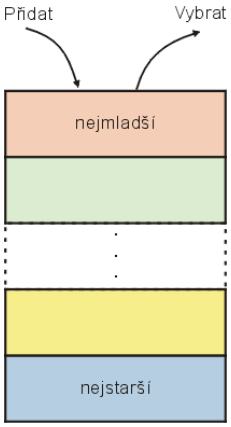


b) dvojité zřetězení

- **Spojový seznam** má tyto vlastnosti:
- řídící proměnné seznamu obsahují ukazatel na **první** prvek seznamu (H -hlavička, head)
- řídící proměnné seznamu mohou dále obsahovat ukazatel na **poslední** prvek seznamu (T - patička, anglicky se používá tail, doslovně ocas) a ukazatel na **právě** zpracovávaný prvek seznamu (C - běžný, current)
- **logicky sousední** prvky nemusí být **fyzicky sousední**, při realizaci v paměti se používá dynamicky přidělovaná paměť, při realizaci na disku správa volných míst nebo záznamů
- každý prvek seznamu obsahuje kromě uživatelské informace ještě **ukazatel na další** prvek seznamu
- pokud prvek neobsahuje další ukazatel na jiný prvek seznamu, hovoříme o jednosměrném seznamu
- **poslední** prvek jednosměrného seznamu může obsahovat **prázdný ukazatel** (nil), pokud ukazuje na první prvek seznamu, hovoříme o jednosměrném **cyklickém** seznamu
- každý prvek seznamu může dále obsahovat ukazatel na **předchozí** prvek seznamu, pak hovoříme o **obousměrném** seznamu
- pokud poslední prvek obousměrného seznamu ukazuje na první prvek a první prvek na poslední, hovoříme o **cyklickém obousměrném** seznamu

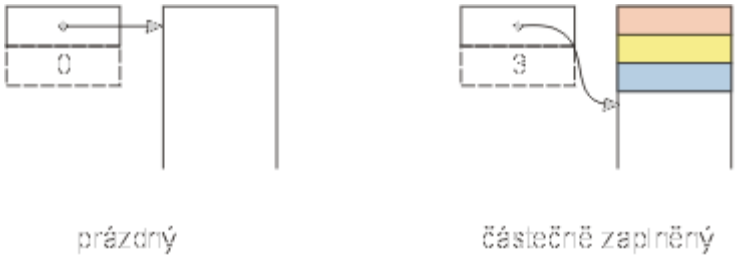
Zásobník – lineární struktura základní vlastnosti

- **Abstraktní představa zásobníku**

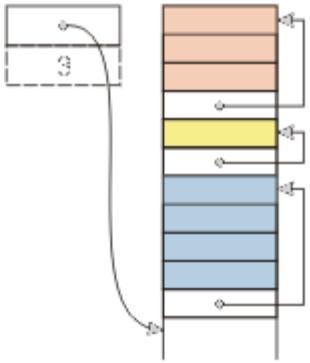


- **Sémantický význam**
  - **Dynamická lineární homogenní** datová struktura, představovaná posloupností jednotlivých prvků. Prvky se do zásobníku ukládají v **pořadí, jak přicházejí**. Prvek, který byl do zásobníku uložen jako poslední, tvoří tzv. vrchol zásobníku (Top of Stack) a pouze ten je přístupný, případně je možno jej ze zásobníku vyjmout. Slovně lze zásobník charakterizovat výrokem „**Poslední dovnitř, první ven**“, anglicky „Last In First Out“, zkráceně LIFO, lidově pravidlo šíleného mlynáře ("Poslední přišel, první mlel"). Zásobník se použije tehdy, chceme-li dočasně uložit data tak, aby při jejich výběru byly zpracovány jako první ty, které přišly jako poslední.
- **Povolené operace se zásobníkem:**
  - **Inicializace** zásobníku.
  - **Přidání** prvku do zásobníku (PUSH)
  - **Odstranění** prvku ze zásobníku
  - **Přístup** k hodnotě prvku na vrcholu zásobníku. Někdy bývají operace zpřístupnění a odstranění sloučeny do jedné (POP)
  - **Test**, zda je zásobník prázdný

Implementace zásobníku polem



a) s pevnou délkou záznamu

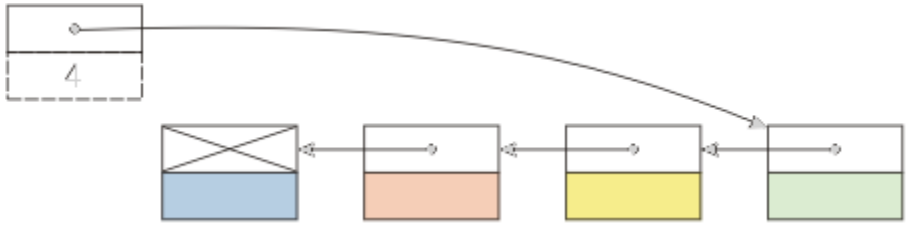


b) s proměnnou délkou záznamu

- Použije se tehdy, když víme, že velikost zásobníku bude menší než určitá hodnota nebo dokonce známe maximální velikost přesně.

.

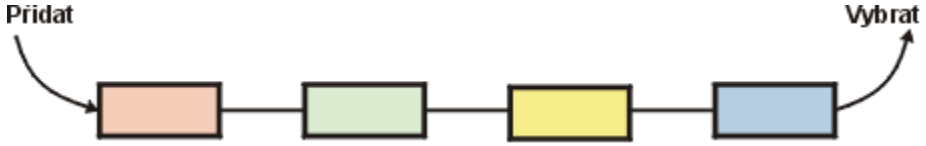
Implementace zásobníku zřetězenými dynamicky alokovanými proměnnými



- Použije se tehdy, neznáme-li předem počet prvků, které se mohou do zásobníku uložit. Protože se pracuje jen s vrcholem zásobníku, potřebujeme jen jeden ukazatel. Prvky však musí být rozšířeny o zpětný ukazatel na minulý prvek.

Fronta – lineární struktura základní vlastnosti

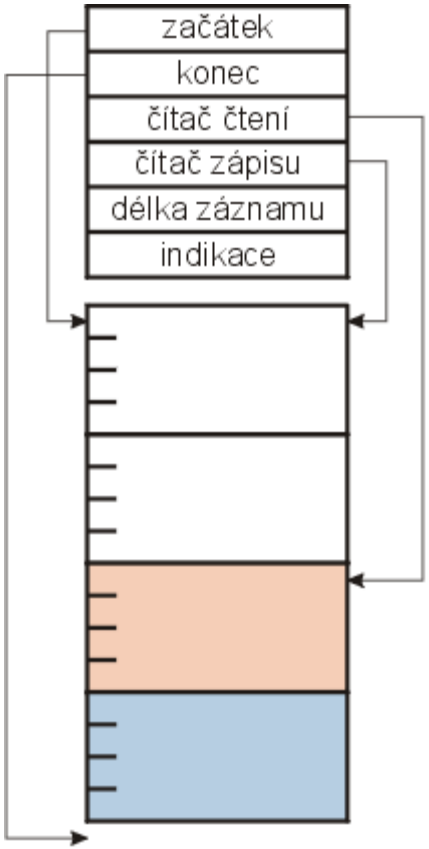
- **Abstraktní představa fronty**



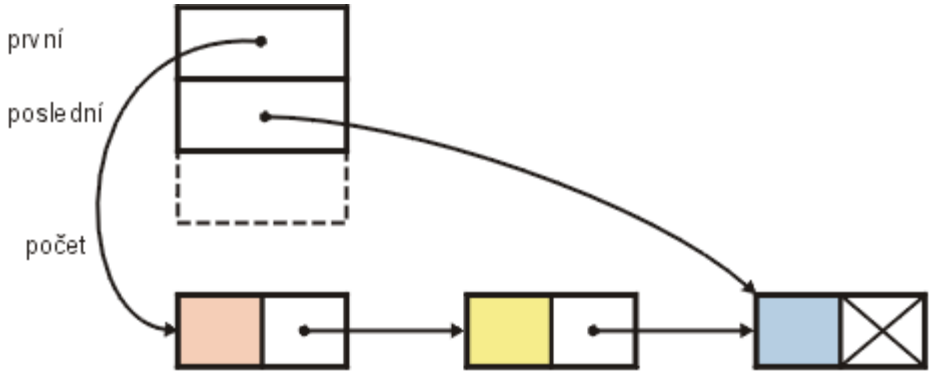
- **Sémantický význam**
  - **Dynamická lineární homogenní** datová struktura, která se používá tehdy, chceme-li dočasně uložit informační položky, které budeme později zpracovávat ve stejném pořadí, v jakém byly uloženy. Lze ji charakterizovat výrokem „První dovnitř, první ven“, anglicky „First In First Out“, odtud FIFO, lidově pravidlo mlynáře ("První přijde, první mele").
- **Povolené operace s frontou:**
  - **inicializace** fronty
  - **přidání** do fronty (zápis)
  - **výběr** z fronty (čtení)

Fronta– implementace

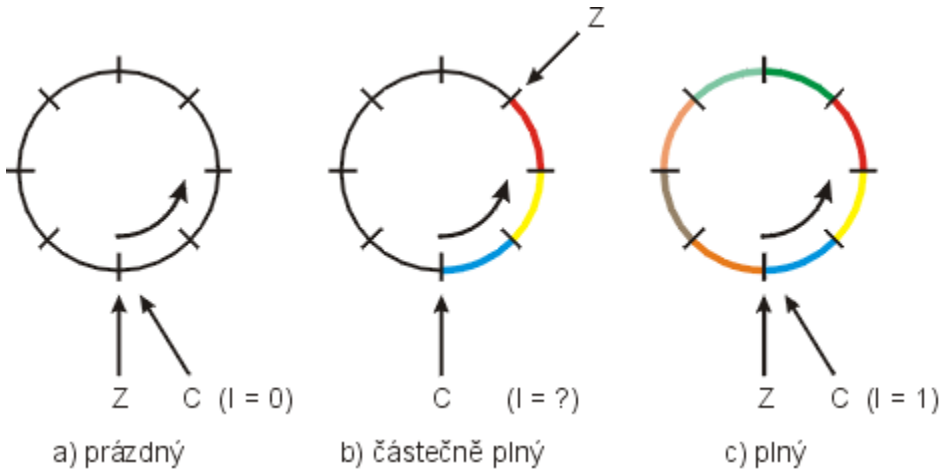
- **Implementace fronty**
  - **polem**



zřetězenými dynamickými proměnnými



Fronta– implementace (cyklický buffer)



- Frontu lze implementovat **polem** i **zřetězeným seznamem**. Při implementaci polem musíme velikost pole staticky omezit (ukazatel na/za poslední prvek) a je jasné, že během práce s frontou se aktivní část fronty neustále posouvá směrem k vyšším hodnotám indexů, takže skoro určitě jednou dojde k vyčerpání pole. Tento konflikt se řeší např. cyklickým bufferem. Představa cyklického bufferu je na obr. , C ... ukazatel čela fronty, Z ... ukazatel konce fronty, I ... rozlišuje plný a prázdný buffer.
- Operace s frontou se navrhnou tak, že pole se svými indexy funguje cyklicky, tzn., že po dosažení maximální hodnoty se začne znovu od začátku. Zvolí-li se velikost pole dostatečná vzhledem k maximálnímu počtu prvků, které se mohou do fronty dostat, je toto opatření vyhovující.
- Není-li známa délka fronty, je lépe implementovat frontu pomocí dynamicky alokované paměti. Potíže, vznikající při implementaci fronty polem, odpadají. V podstatě jde o spojový seznam, nad kterým jsou definovány příslušné operace tak, aby se choval jako požadovaná fronta.

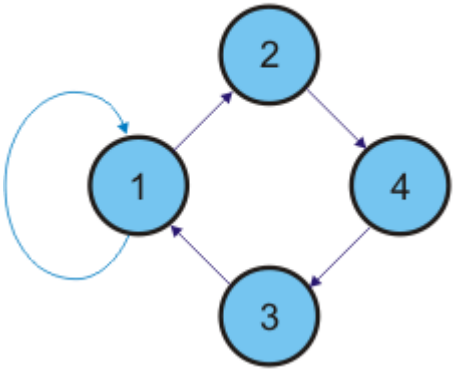
Graf – nelineární struktura -definice

- Graf je **dynamická nelineární datová struktura**, která dovoluje vyjadřovat vazby mezi prvky. Prvky se nazývají **vrcholy** nebo **uzly** grafu (nodes, vertex - vertices), vazby mezi objekty jsou **hrany** grafu (edges).
- Příklady:
  - Vrcholy grafu jsou názvy měst, hrany silnice.
  - Transformační stanice jsou vrcholy grafu, elektrická vedení vysokého napětí jsou hrany.
- Každé **hraně** v grafu přísluší **dvojice vrcholů**. Hrana tyto vrcholy spojuje. Pokud je dvojice vrcholů, příslušejících jednotlivým hranám, orientovaná (můžeme si představit, že silnice mezi městy jsou jednosměrné), jde o graf **orientovaný**, jinak o graf **neorientovaný**.
- S hranami i vrcholy mohou být spojeny další informace (např. u měst počet obyvatel, u hran vzdálenost měst). Pak hovoříme o grafu hranově nebo vrcholově **ohodnoceném**.
- Graf můžeme vyjádřit buď pomocí dvourozměrné **matice,** nebo dynamickými **seznamy** vrcholů a hran.

Maticový popis grafu

- Známe-li maximální počet vrcholů v grafu (nejčastější případ), dají se někdy operace nad grafem převést na maticové operace. Graf je reprezentován tzv. **incidenční maticí** (dvojrozměrným polem) . Ta vyjadřuje incidenci (sousedních) vrcholů. Snadno se realizují operace **přidání** hrany, **odebrání** hrany**, test existence** hrany. Obtížněji se pracuje s vrcholy. Typickým příkladem je např. návrh (analýza) elektrických sítí.

Graf



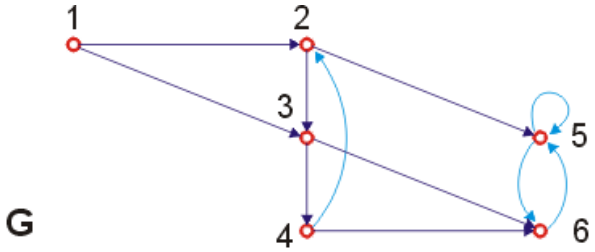
Incidenční matice

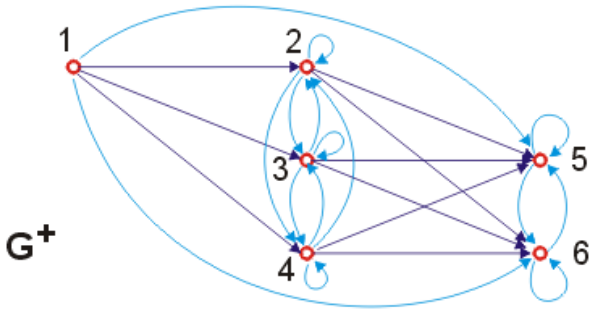
	1	2	3	4
1	1	1	0	0
2	0	0	0	1
3	1	0	0	0
4	0	0	1	0

- Při reprezentaci grafu hranově-vrcholovou incidenční maticí jsou vrcholy identifikovány čísly řádků a sloupců, hrany pozicemi matice.
- Sledujeme-li **řádek** v matici zleva doprava, vyjadřují prvky matice booleovskými hodnotami, **do** kterých uzlů se z daného uzlu orientovanou hranou lze dostat. Sledujeme-li v matici **sloupec** shora dolů, zjistíme, **ze** kterých uzlů se lze dostat **do** uzlu sledovaného.

Graf a jeho tranzitivní uzávěr

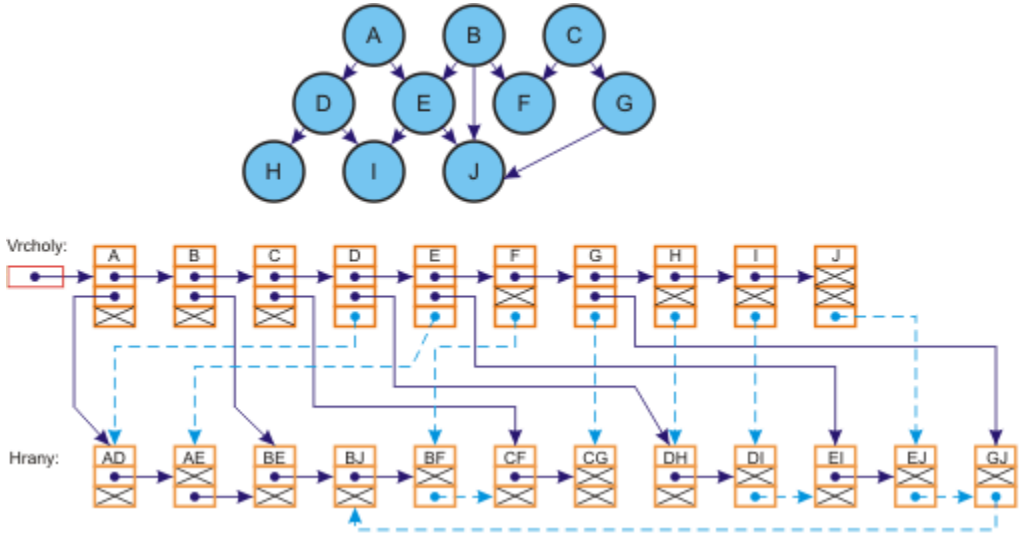
- Pokud popisujeme graf maticí, pak druhá mocnina této matice ukazuje, které vrcholy grafu jsou dosažitelné přes další vrchol, třetí mocnina pak přes dva vrcholy.
- Důležitý je pojem tzv. **tranzitivní uzávěr grafu** . Je to incidenční matice, která popisuje vzájemnou dosažitelnost všech vrcholů bez ohledu na to, jak složitou cestou se toho dosáhne.





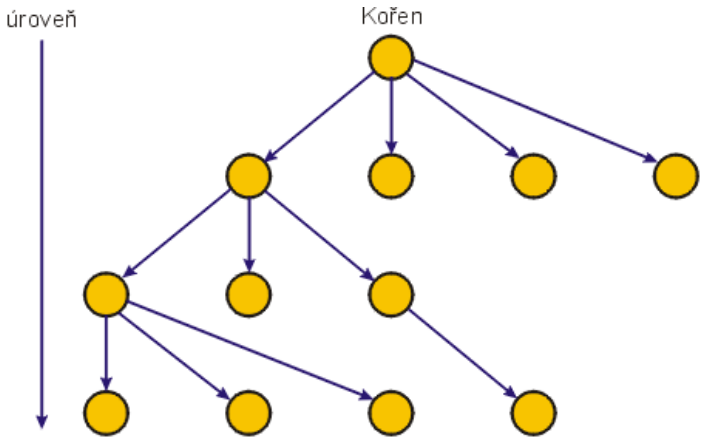
Implementace grafu spojovými seznamy

- Pro každý vrchol bude existovat jeden seznam a ten bude obsahovat tolik prvků, kolik z něj vychází nebo do něj vstupuje hran. U orientovaného grafu bude celkový počet prvků v seznamech rovný počtu hran, u neorientovaného grafu bude každá hrana v seznamech dvakrát.
- V takové reprezentaci se dobře přidávají i ubírají hrany, dobře se přidávají vrcholy, více práce je s rušením vrcholu, protože se musí vyjmout ze všech seznamů prvky, odpovídající hranám vycházejícím nebo vcházejícím do rušeného vrcholu.



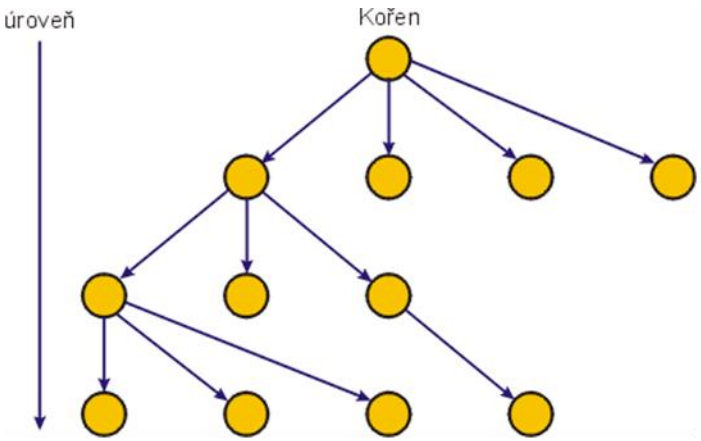
Obecný strom – nelinární struktura - definice

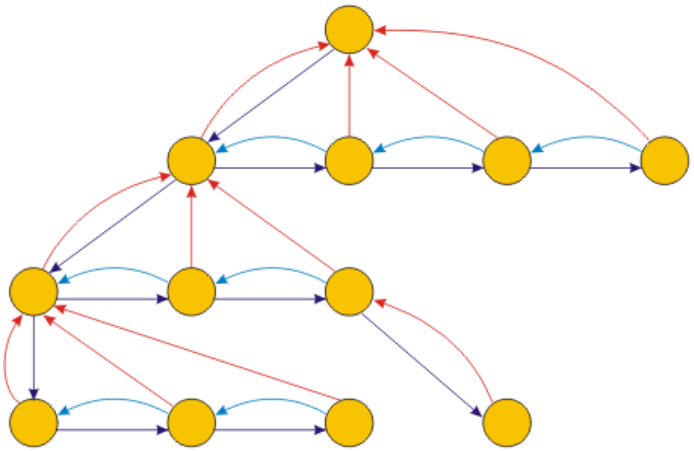
- Strom je **orientovaný** graf, který **neobsahuje cykly**, přičemž do každého vrcholu vstupuje **maximálně jedna hrana**. Z analogie z rodinné generační hierarchie je zvykem označovat vrchol na vyšší úrovni rodičem a vrcholy, které z něj bezprostředně vycházejí jako potomky (následníky)
- **Kořen** ... vrchol, do kterého žádná hrana nevstupuje.
- **List** ... vrchol, ze kterého žádná hrana nevystupuje.
- Je zvykem zobrazovat stromy kořenem vzhůru s orientací dolů směrem k listům.
- Důležitou vlastností stromů je, že **každý** jeho **vrchol** může být pokládán za **kořen dalšího stromu**.Vrcholy mohou být **ohodnoceny**. Protože do každého vrcholu vstupuje maximálně jedna hrana (žádná pouze do kořene stromu), není třeba hrany hodnotit zvlášť, nýbrž hodnocení hran může být součástí hodnocení vrcholu.



Obecný strom - implementace

- Při implementaci stačí popisovat vrcholy. Obecný strom si lze představit jako **rekurzivní** řadu vzájemně vázaných **jednoúrovňových stromů**. Protože každý kořen jednoúrovňového stromu má maximálně jednoho rodiče a řadu potomků, lze uvažovat pouze jediného potomka a všechny další potomky prohlásit za mladší bratry tohoto potomka. Řada rodič - nejstarší syn -mladší bratr - ještě mladší bratr - atd. je jednoduchým lineárním seznamem s hlavou, kterou představuje rodič a členy, které jsou potomky daného rodiče.





Obecný strom - použití

- V praxi může být datová struktura obecný strom použita v **genealogii**:
  - strom života (potomci dané osoby)
  - rodokmen (předkové dané osoby)
- v **organizaci**
  - organigram (struktura organizace)
- v **biologii**:
  - klasifikace organismů (říše, kmen, třída, řád, čeleď, rod, druh)
- nebo ve **strojírenství**:
  - **strukturní kusovník** daného výrobku (rozpad výrobku na montážní skupiny, podskupiny, díly a nakupované položky)
  - **inverzní kusovník** (použití daného dílu v různých podskupinách, skupinách a výrobcích)

Binární strom - definice

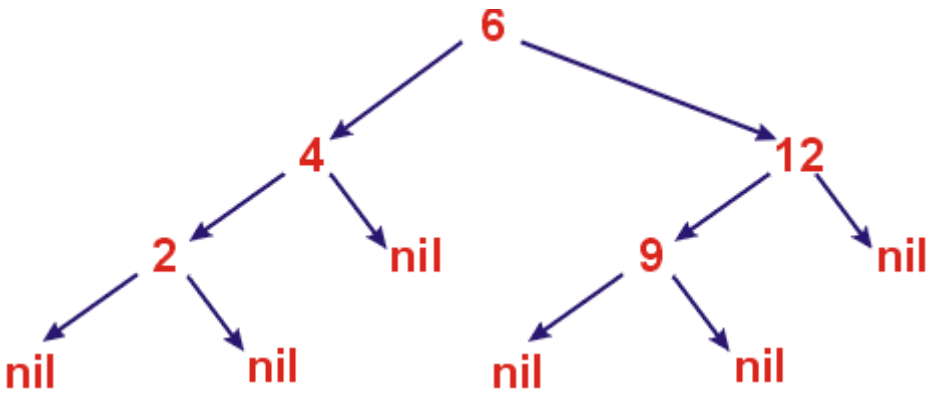
- Má-li každý vrchol nejvýše dva potomky, nazývá se takový strom **Binární**. Každý vrchol binárního stromu má tedy **levého**, případně **pravého** následníka (potomka).
- **Uspořádaný binární strom** je takový, jehož leví následníci každého vrcholu mají ohodnocení nižší než sám vrchol, praví následníci mají ohodnocení vyšší.
- Takové stromy jsou velice důležitým pojmem a často se využívají pro implementaci nejrůznějších rozhodovacích procesů.

Binární strom - implementace

- Každý vrchol stromu je reprezentován trojicí:
  - hodnota
  - ukazatel na levého potomka
  - ukazatel na pravého potomka
- Poznámky:
  - Hodnotou může být i složitě strukturovaný objekt, např. záznam.
  - Celý strom je reprezentován ukazatelem na kořen.
  - Nejvýhodnější implementace binárního uspořádaného stromu je pomocí dynamického spojového seznamu.

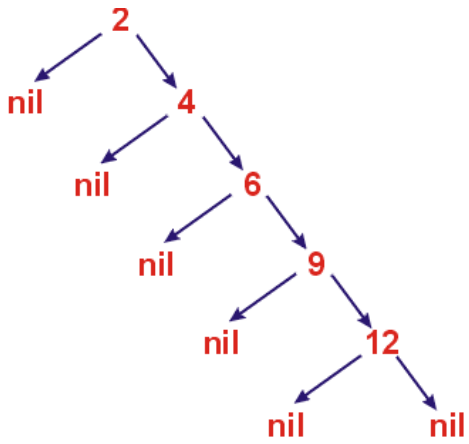
Binární vyhledávací strom

- Binární vyhledávací strom má ohodnocení levého potomka nižší než rodiče a pravého potomka větší než rodiče.



Vyvážený a degenerovaný binární strom

- Binárních stromů se často používá pro ukládání, vyhledávání a třídění informací. Aby byl proces efektivní, nestačí, aby binární strom byl **uspořádaný**, ale aby byl tzv. **vyvážený**, tj. aby počet **úrovní** jeho uzlů byl **minimální**. Náš strom je vyvážený, ale kdybychom jinak uspořádali vstupní hodnoty, mohla by vzniknout v krajním případě situace, kdy prohledávání takového stromu je vlastně sekvenční. Jde o **degenerovaný binární strom** .



Výška ideálně vyváženého stromu je menší než log2 n. V nejhorším případě může být výška degenerovaného stromu až n, kde n je počet vrcholů.

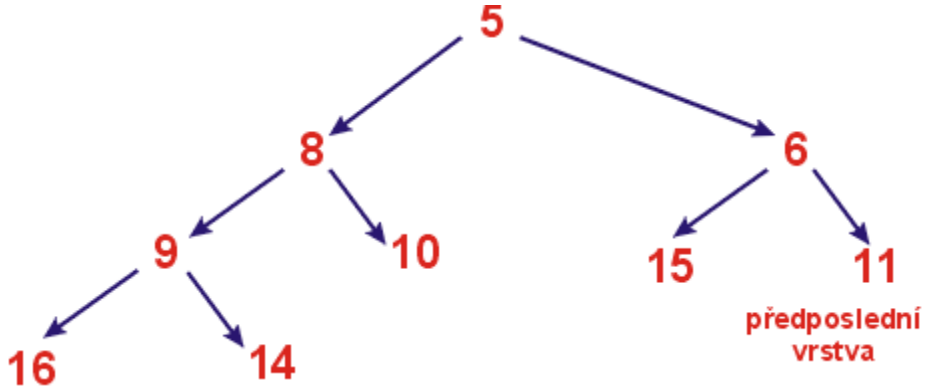
- **Doba** vyhledání informace ve vyváženém stromu je

  T » log2 n

- **Vyvažování** binárních (i jiných) stromů tvoří samostatnou problematiku a o této algoritmizaci je možno se dočíst v literatuře.

Hromada - definice

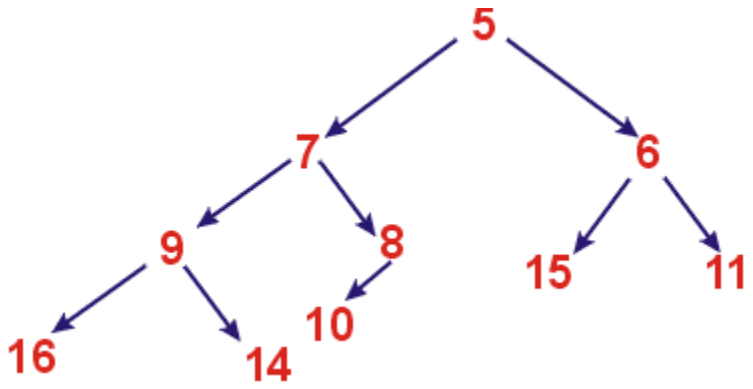
- **Hromada** je **binární strom**, jehož vrcholy jsou ohodnoceny celými čísly a který má následující vlastnosti:
  - Každý uzel, který neleží v poslední nebo předposlední vrstvě, má dva následníky. V předposlední vrstvě jsou zleva doprava uzly se dvěma následníky, pak max. jeden uzel s jedním následníkem, pak vrcholy bez následníků.
  - Ohodnocení je provedeno tak, že hodnota libovolného uzlu není větší než hodnota libovolného následníka.



- Nad hromadou se provádějí dvě operace:
  - **Insert** (Hodnota) - Vložení nového uzlu
  - **Extract_min** - Výběr minimálního prvku.

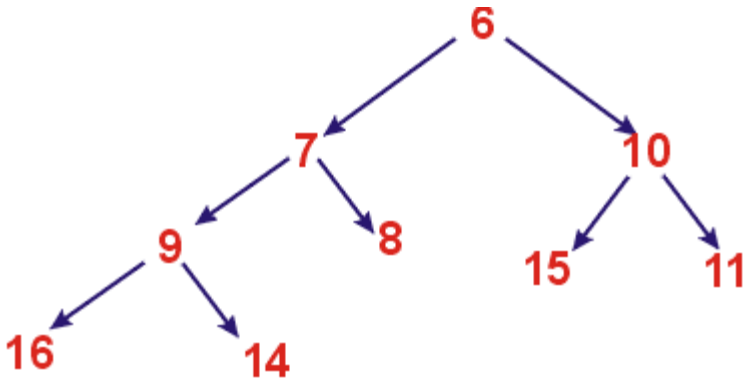
Hromada – operace vložení

- **Insert** (Hodnota) - Vložení nového uzlu
- Umístění nového uzlu je **jednoznačně** dáno z definice hromady. **Ohodnocení** uzlů se musí překontrolovat, případně **zaměnit prvky** tak, aby ohodnocení odpovídalo definici.
- Provedeme-li na předcházející hromadu operaci Insert (7), bude hromada vypadat tak, jak je to na obrázku.



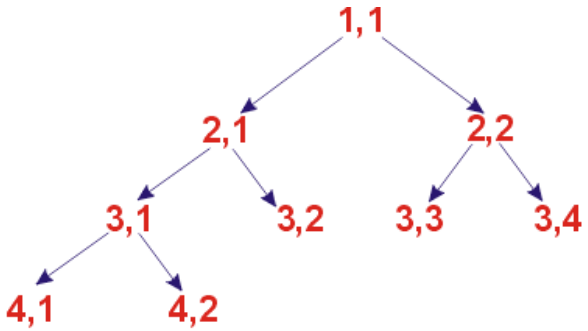
Hromada – operace výběr minimálního prvku

- **Minimální** prvek je na **vrcholu** hromady. Na místo odebraného vrcholového prvku se uloží posledně vkládaný vrchol (je jednoznačně dáno, který to je) a provede se záměna prvků hromady tak, aby odpovídalo pravidlům hromady.
- Provedeme-li na předcházející hromadu operaci Extract_min, bude hromada vypadat tak, jak je to ukázáno na obrázku.



Hromada – implementace a použití

- Výhodou uspořádání stromu do hromady je mezi jiným i to, že lze její prvky uložit do **vektoru** (jednorozměrné pole).

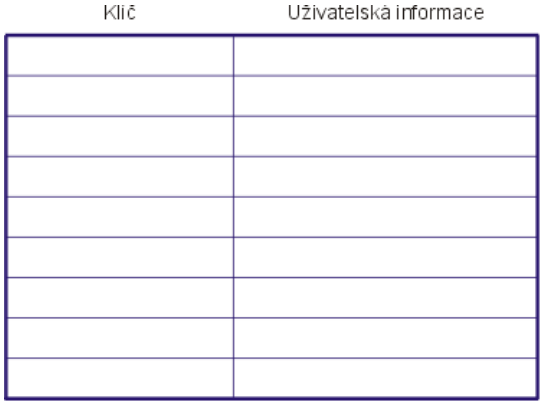


hromada	1,1	2,1	2,2	3,1	3,2	3,3	3,4	4,1	4,2
i	1	2	3	4	5	6	7	8	9

- Levý následník prvku v_i je na pozici 2*i
- Pravý následník prvku v_i je na pozici 2*i+1.
- Hromada se využívá pro **řazení**. Metoda **není stabilní** ani **přirozená**.
- Doba řazení je:
  - $T \gg n \cdot \log_2 n$
- *Pozn.: Řadící algoritmus je **stabilní** tehdy, jestliže po seřazení zachovává vzájemné pořadí prvků se stejným klíčem. Řadící algoritmus je **přirozený**, je-li čas, potřebný pro seřazení již částečně uspořádané posloupnosti menší než doba seřazení posloupnosti neuspořádané.*

Tabulky a jejich klíčové položky

Tabulka je **dynamická lineární homogenní datová struktura** spojená s ukládáním dat v operační paměti počítače. Je-li hlavním místem uložení disk, hovoříme o souboru s udaným typem. Řadu algoritmů lze použít jak na tabulky, tak na soubory s udaným typem. Přitom je třeba si uvědomit podstatný rozdíl v rychlosti zpracování dat v operační paměti a na disku.

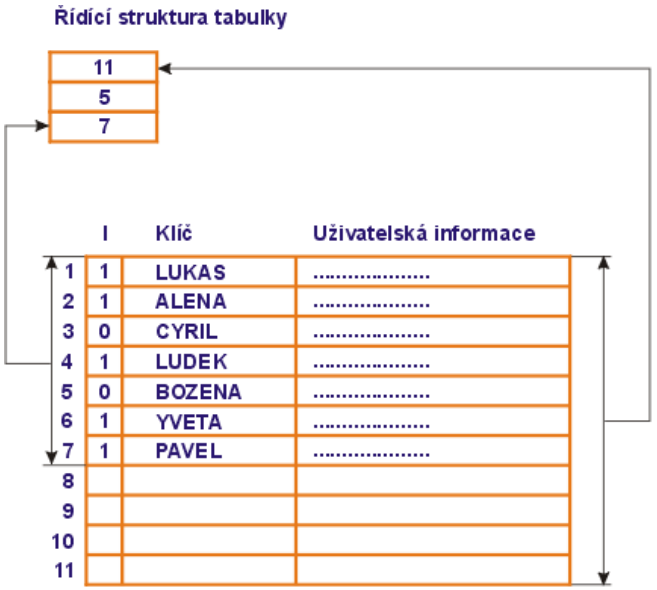


- Tabulku si lze představit jako jednorozměrné pole proměnných typu záznam (jinými slovy jde o homogenní lineární strukturu prvků, které mohou samy mít nehomogenní strukturu). Prvek tohoto pole nazýváme položkou tabulky. Tabulka se zpracovává podle určitého klíče (někdy se nazývá identifikační klíč), který je částí položky tabulky.
- Každá položka tabulky obsahuje dvě části:
  - **identifikační klíč**
  - **uživatelská** informace
- Klíčů v tabulce může být více. Potom hovoříme o primárním a sekundárních klíčích. Primární klíč by měl být v tabulce jedinečný a měl by jednoznačně určovat položku tabulky. Naproti tomu sekundární klíče nejsou obvykle jedinečné, hodnota jednoho sekundárního klíče se může vyskytovat ve více položkách.

Tabulky – typy a operace

- Základní operace pro tabulku jsou:
  - **Inicializace** (vytvoření prázdné tabulky)
  - **Vložení** nové položky
  - **Vyhledání** položky podle primárního klíče
  - **Vyhledání první** položky podle sekundárního klíče
  - **Vyhledání další** položky podle sekundárního klíče
  - **Změna** (přepsání) vyhledané položky
  - **Zrušení** položky
- Tabulky můžeme dále rozdělit na **jednoduché**, kde klíče jsou přímo součástí tabulky a **indexované**, kde jsou klíče vedeny v samostatných tabulkách indexů, přičemž některé klíče jsou vedeny jen v základní tabulce a jiné v samostatných tabulkách.
- Z hlediska vkládání, vyhledávání a rušení lze definovat následující základní typy tabulek:
  - vstupněsekvenční
  - seřazená
  - rozptýlená
- Indexní tabulky mohou být stejných typů, nebo mohou být tvořeny speciálními metodami (např. stromovými strukturami).

Vstupněsekvenční tabulka - definice



- **Vstupněsekvenční** tabulka je nejjednodušším typem. Položky v tabulce nejsou organizovány podle **žádného** klíče, ale jsou zaznamenávány v pořadí jejich příchodu.
- Výhodou vstupněsekvenčních tabulek je jejich **jednoduchost**. Používají se zejména tam, kde se vkládá velké množství položek a zpracovává se většina položek sekvenčním způsobem.

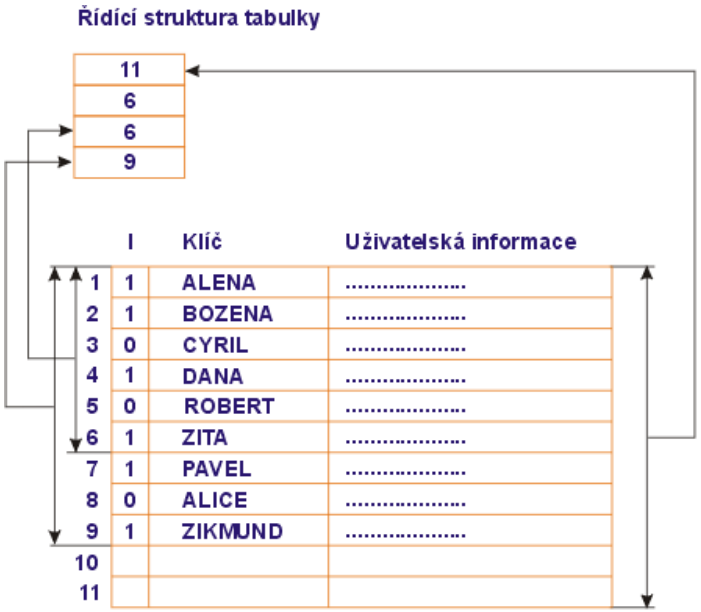
Vstupněsekvenční tabulka - implementace

- Tabulku lze implementovat dostatečně dlouhých polem. Prvkem tohoto pole je datový typ záznam, který obsahuje:
  - **indikaci**, zda je položka tabulky platná nebo zrušená
  - **klíč**
  - **uživatelskou** informaci
- Indikace platnosti položky se používá pro urychlení rušení. Při rušení položky se nemusí posouvat následující položky. Stačí pouze označit příslušnou položku za zrušenou. Fyzické vypuštění se provede operací, která se nazývá **reorganizace** tabulky. Ta se provádí v době nízkého zatížení počítače nebo při určitém stavu zaplnění tabulky.
- Dále je vhodné definovat **řídící strukturu** tabulky, která obsahuje tři prvky:
  - **maximální** počet položek v tabulce (počet prvků pole)
  - **skutečný** počet položek v tabulce
  - **index** poslední položky v tabulce

Vstupněsekvenční tabulka - algoritmy

- **Iniciace tabulky**
- Při inicializaci tabulky je nastaven maximální počet položek v tabulce a ostatní dva prvky řídící struktury jsou vynulovány.
- **Vložení do tabulky**
- Při vložení nové položky se kontroluje, zda index poslední položky v tabulce není roven maximálnímu počtu položek. Pokud ano a skutečný počet položek je menší, než maximální, pak se musí provést tzv. reorganizace tabulky, kdy jsou vypuštěna volná místa. Pokud je potom index poslední položky v tabulce menší, než maximální počet položek, vloží se nová položka za poslední, její indikace se nastaví na platnou a skutečný počet a index poslední položky se zvýší o jedna.
- **Vyhledání v tabulce**
- Vyhledání položky podle zadaného klíče se provádí sekvenčním způsobem. Postupně se zadaný klíč porovnává s klíči v tabulce počínaje prvním záznam, přičemž se tabulce vynechávají položky s indikací zrušení. Pokud se nalezne shoda, položka je nalezena, pokud se dosáhne indexu poslední položky a klíč se neshoduje, položka není v tabulce. Pokud označíme index poslední položky v tabulce N, pak doba prohledávání vstupněsekvenční tabulky je průměrně úměrná N/2, pokud položka existuje a je úměrná N, pokud položka neexistuje.
- **Změna položka v tabulce**
- Změna položky spočívá ve vyhledání podle klíče a změně uživatelské informace. Řídící struktura tabulky zůstává nezměněna.
- **Rušení položky z tabulky**
- Rušení položky spočívá ve vyhledání podle klíče, nastavení indikace na zrušení a snížení skutečného počtu položek v tabulce o jednu.
- **Reorganizace tabulky**
- Po reorganizaci je index poslední položky v tabulce roven skutečnému počtu položek v tabulce.

Seřazená tabulka - definice



- Seřazená tabulka je dalším typem. Položky jsou uspořádány v tabulce **vzestupně** podle jejich klíče. **Rušení** se provádí opět označením **neplatnosti**. Problém vzniká při vkládání. Při každém vložení bychom potřebovali nalézt vhodné místo pro vložení a, pokud není náhodou volné (zrušená položka), udělat místo pro vložení přesunem zbývající části tabulky o jedno místo. To by podstatně snižovalo rychlost při vkládání. Proto se tabulka rozdělí na **seřazenou** část a **neseřazenou** část. Při **reorganizaci** se neseřazená část seřadí a její položky se zařadí na příslušné místo v seřazené části.

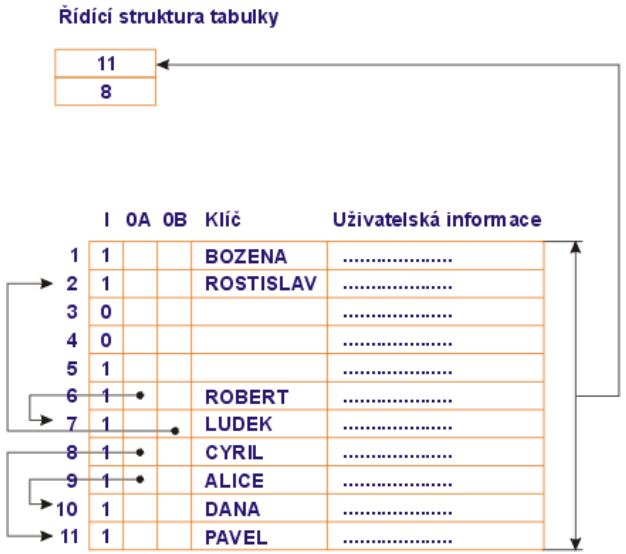
Seřazená tabulka - implementace

- Tabulku lze implementovat dostatečně dlouhých polem. Prvkem tohoto pole je datový typ záznam, který obsahuje:
  - **indikaci**, zda je položka tabulky platná nebo zrušená
  - **klíč**
  - **uživatelskou** informaci
- **Řídící struktura** tabulky obsahuje čtyři prvky:
  - **maximální** počet položek v tabulce (počet prvků pole)
  - **skutečný** počet položek v tabulce
  - **index** poslední položky v **seřazené** části tabulky
  - **index** poslední položky v **neseřazené** části

Seřazená tabulka - algoritmy

- **Inicializace tabulky**
  - Při inicializaci tabulky je nastaven maximální počet položek v tabulce a ostatní tři prvky řídící struktury jsou vynulovány.
- **Vložení do tabulky**
  - Při vložení nové položky se kontroluje, zda index poslední položky v neseřazené části tabulky není roven maximálnímu počtu položek. Pokud ano a skutečný počet položek je menší, než maximální, pak se musí provést alespoň částečná reorganizace tabulky, kdy jsou vypuštěna volná místa. Pokud je potom index poslední položky v neseřazené části tabulky menší, než maximální počet položek, vloží se nová položka za poslední, její indikace se nastaví na platnou a skutečný počet a index poslední položky se zvýší o jedna.
- **Vyhledání položky**
  - Vyhledání položky podle zadaného klíče se provádí v seřazené části tzv. půlením intervalu a v neseřazené části sekvenčním způsobem. Vyhledání půlením intervalu spočívá v tom, že se nastaví meze - první a poslední položka seřazené části. Pak se vyhledá prvek v polovině nastavených mezí a jeho klíč se porovnává s hledaným klíčem. Pokud dojde ke shodě, vyhledání končí, jinak se pokračuje buď v prohledání tak, že pokud je hledaný klíč menší než klíč prvku v polovině nastavených mezí, nastaví se horní mez na index prvku v polovině nastavených mezí snížený o jedničku, jinak se nastaví dolní mez na index prvku v polovině nastavených mezí zvýšený o jedničku. Opět se vybere prvek v polovině zadaných mezí. To se provádí tak dlouho dokud se prvek buď nenalezne, nebo je horní mez je menší než dolní mez. Neseřazená část se prohledá sekvenčně.
  - Pokud označíme index poslední položky v seřazené části tabulce N, pak doba prohledávání seřazené tabulky je průměrně úměrná $\log_2(N)$. Z toho vyplývá, že jde o velmi rychlý algoritmus a že je třeba se snažit omezit délku neseřazené části tabulky, neboť zde je doba vyhledání lineárně závislá na počtu prvků.
- **Změna položky**
  - Změna položky spočívá ve vyhledání podle klíče a změně uživatelské informace. Řídící struktura tabulky zůstává nezměněna.
- **Zrušení položky**
  - Rušení položky spočívá ve vyhledání podle klíče, nastavení indikace na zrušení a snížení skutečného počtu položek v tabulce o jednu.
- **Reorganizace tabulky**
  - Po reorganizaci je index poslední položky v seřazené části tabulky roven skutečnému počtu položek v tabulce.
  - Neseřazená část buď může být zařazena hned za seřazenou, nebo může být na konci.
  - Seřazené tabulky se používají, pokud je třeba rychle vyhledávat danou položku a také zpracovávat metodou zpracuj nejbližší další položku vzestupně.

Rozptýlená tabulka - definice



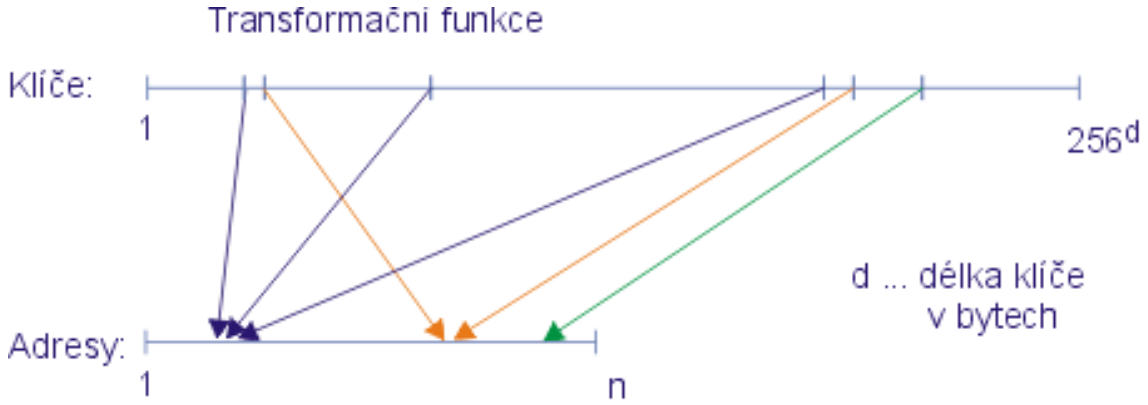
- Další možností, jak organizovat tabulku, je provádět výpočet adresy v tabulce z hodnoty klíče podle vzorce:
- **A = F(K,N)**
  - A - adresa v tabulce
  - K - hodnota klíče
  - N - počet položek v tabulce
- Funkce F se nazývá transformační funkce . Problémem je nalézt tuto funkci F. Přitom lze často využít vlastností klíče.

Rozptýlená tabulka – jednoduchá transformační funkce

- **Jednoduchá transformační funkce**
- Nejjednodušší transformační funkcí je **lineární** funkce, kdy klíčem je přirozené číslo v rozsahu B až B+N, kde B je libovolné přirozené číslo. Příkladem jsou např. čísla příjemek ve skladu, které jsou číslovány souvisle vzestupně.
- Jiným jednoduchým algoritmem je výpočet adresy pro úložné místo ve skladu. Řady regálů (R) jsou značeny ´A´-´F´, souřadnice dálky (D) 1-50, výška (V) 1-10. Funkce F pak může mít tvar:
- A = (ord(R) - ord(´A´)) * 500 + (V-1)*50 + D

Rozptýlená tabulka – obecná transformační funkce

- **Obecná transformační funkce – rozptýlení**



- Obecně však využití vlastností klíče **není možné.** Klíč si lze přestavit jako hodnotu v rozsahu 1 - 256^L, kde L je délka klíče v bytech. Funkce F má zobrazit tuto hodnotu do rozsahu velikosti tabulky. Dalším požadavkem je **zrovnoměrnění** využití tabulky nezávisle na jistém shlukování původních hodnot klíčů. Těmto požadavkům vyhovuje např. funkce, která vypočítává z hodnoty klíče zbytkem po dělení délkou tabulky adresu v tabulce. Nejlepšího zrovnoměrnění hodnot dosáhneme, jestliže je počet položek tabulky prvočíslo, jinak dochází ke shlukování s periodou prvočinitelů počtů položek tabulky. Funkce transformující klíč na adresu se pak nazývá rozptylovací (hash) funkce.
- **Synonyma**
- Problémem je, že více hodnotám klíče může být přiřazena **stejná** adresa. Hodnoty klíče, kterým je přiřazena stejná adresa, se nazývají **synonyma**.

Implementace rozptýlené tabulky

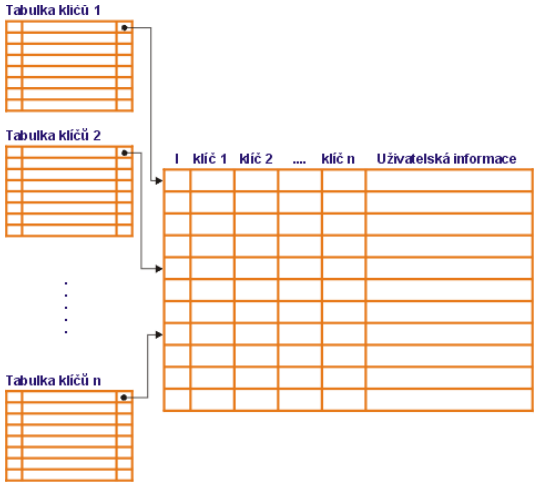
- Tabulku lze implementovat dostatečně dlouhých polem. Vzhledem k možnosti výskytu synonym se doporučuje maximální zaplnění tabulky 60-70%. Prvkem pole implementujícího tabulku je datový typ záznam, který obsahuje:
  - **indikaci**, zda je položka tabulky platná nebo zrušená
  - dva **indexy** pro řešení **synonym** (OA, OB)
  - **klíč**
  - **uživatelskou** informaci
- **Řídící struktura** tabulky obsahuje čtyři prvky:
  - maximální počet položek v tabulce (počet prvků pole)
  - skutečný počet položek v tabulce
- **Řešení synonym**
- Řešení synonym se provádí následujícím způsobem:
  - pokud je vkládána položka a je vypočtena adresa, která je již obsazena a index OA je nulový, pak se položka vloží na nejbližší volné místo v tabulce a index OA pro vypočtenou adresu se nastaví na skutečnou adresu.
  - pokud je vkládána položka a je vypočtena adresa, která je již obsazena a index OA není nulový, pak se položka vloží na nejbližší volné místo v tabulce, je testována položka kam ukazuje index OA, pokud je zde index OB nulový, pak se nastaví na skutečnou adresu, jinak se postupuje podle indexu OB tak dlouho, až se nalezne nulový, který se nastaví na skutečnou adresu.
- Je statisticky prokázáno, že pokud počet položek tabulky nepřekročí 60-70% délky tabulky, průměrný počet přístupů s využitím indexů OA a OB bude menší než 1.3, tedy o 30% horší než vždy na první pokus..

Rozptýlená tabulka - algoritmy

- **Inicializace tabulky**
  - Při inicializaci tabulky je nastaven maximální počet položek v tabulce, nulován skutečný počet položek v tabulce a v celé tabulce jsou všechny položky označeny jako zrušené a současně nulovány indexy OA a OB.
- **Vložení položky**
  - Při vložení nové položky se kontroluje, zda je v tabulce ještě místo. Pokud ano, provede se výpočet adresy, ošetří se synonyma a skutečný počet položek se zvýší o jedna.
- **Vyhledání položky**
  - Vyhledání položky podle zadaného klíče se provádí výpočtem adresy. Pokud na vypočtené adrese není platná položka nebo požadovaný klíč se neshoduje s nalezeným, pak se pokračuje podle indexů OA a OB.
  - Průměrná četnost přístupu je menší než 1.3, **bez ohledu na délku** tabulky a **počet obsazených položek**.
- **Změna položky**
  - Změna položky spočívá ve vyhledání podle klíče a změně uživatelské informace. Řídící struktura tabulky zůstává nezměněna.
- **Rušení položky**
  - Rušení položky spočívá ve vyhledání podle klíče, nastavení indikace na zrušení úpravu indexů OA a OB, přičemž položky zůstávají na místě, a snížení skutečného počtu položek v tabulce o jednu.
- **Reorganizace**
  - Reorganizace není nutná. Nevýhodou je, že nelze určit následující ani předcházející položku v tabulce ve smyslu seřazení podle abecedy.
- **Užití rozptýlené tabulky**
  - Rozptýlené tabulky se používají, pokud je třeba rychle vyhledávat danou položku a nepotřebujeme zpracovávat metodou zpracuj nejbližší další položku vzestupně.

Tabulky s více klíči

- V případě, že uživatelská informace má **velký rozsah** nebo tabulka obsahuje **více klíčů**, je vhodné pro každou tabulku vytvořit **nadřazenou** tabulku obsahující jen **jeden klíč a číslo položky** v základní tabulce. Těmto nadřazeným tabulkám se říká **indexní tabulky**.
- Indexní tabulky mohou být organizovány různými způsoby (vstupněsekvenčně, seřazeně, rozptýleně). Mohou obsahovat i **vícenásobné výskyty** stejných klíčů. Při vkládání, rušení a změně klíčů v základní tabulce se musí změnit položky indexních tabulek. Při vyhledávání se určí, podle kterého klíče se provádí hledání. Podle toho je vybrána indexní tabulka, v ní je vyhledán klíč a jemu odpovídající položka v základní tabulce.



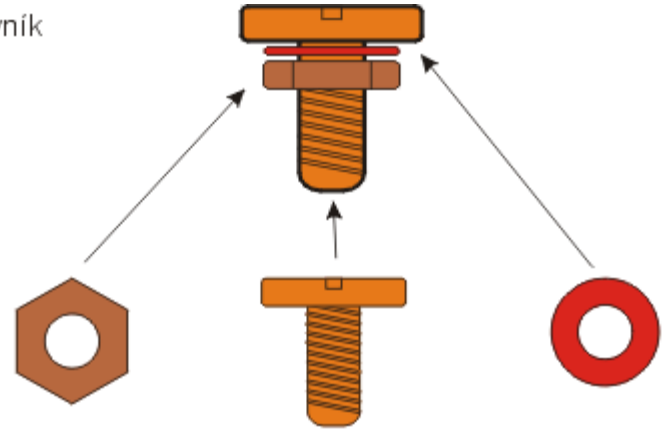
- Speciálně se prací s tabulkami zabývají **tabulkové procesory** a **databáze**. Ty poskytují velmi silné nástroje nejenom pro zpracování jednotlivých položek tabulek, ale celých skupin položek a sloupců tabulek, umožňují automatické plnění jednotlivých polí tabulek v závislosti na jiných polích, mezisoučty a řadu statistických funkcí.

Vybrané datové struktury v průmyslovém inženýrství

- Pro popis **výrobku**, jeho rozplánování a sledování ve výrobě potřebujeme data, která vznikají při zpracování **technické přípravy výroby** v **konstrukci** a **technologii**.
- V tomto kursu jsou probrána jen nejzákladnější data a funkce, které jsou potřebné při zajištění IS/IT podniku strojírenského typu. Jen pro představu, počet řádek demonstračního programu (cca 1 000) pro funkce kusovníku a postupu přestavuje zlomek promile průměrného skutečného programu pro řízení výroby (cca 4 000 000).
- Základními datovými strukturami a operacemi s těmito strukturami ve strojírenství jsou:
  - **Položky** (nakupované díly a materiál, vyráběné díly, montážní podskupiny, skupiny, výrobky)
  - **Kusovník**, jeho položky a vazby
  - **Technologický postup**, **operace**, **pracoviště**
  - Tisky **rozpadu** kusovníku, jeho **termínování** a **určení množství**
  - **Termínování** technologického postupu
  - Výpočet **nákladové ceny** výrobku
  - **Zakázka**, **pozice zakázky**, **plánování zakázky** v čase a množství

Kusovník a postup

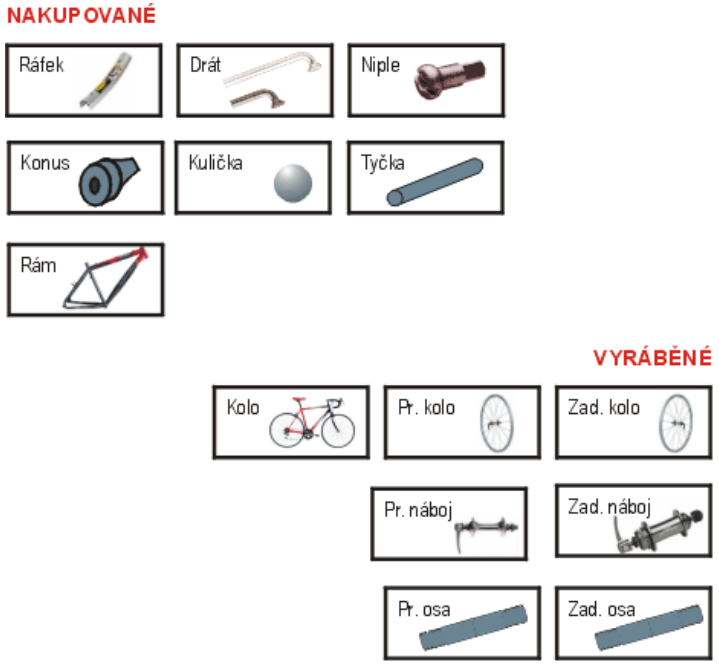
Kusovník



Technologický postup

41211 Díl XXX
1. Řezat 2. Soustružit 3. Frézovat 4. Kalit 5. Brousit

Vyráběné a nakupované položky



- **Položkami** se rozumí nakupované díly a materiál, vyráběné díly, montážní podskupiny, skupiny, výrobky.
- Každá **vyráběná položka** má svůj kusovník, který představuje seznam položek nejbližší nižší úrovně, které jsou potřebné pro jeho výrobu. Pro vyráběný díl je to buď materiál (např. ocelová tyč nebo plech) nebo nakupovaný polotovar (např. odlitek nebo výkovek). Montážní skupina má přiřazen seznam např. vyráběných dílů, jednodušších montážních skupin, spojovací díly, barvu.

Datová pole položky

- Protože by mohlo v této kapitole docházet k záměnám pojmů položka ve smyslu vyráběný a nakupovaný díl a položka ve smyslu datový prvek struktury záznam, budeme důsledně používat pojem **datové pole** místo **datová položka**. Pojem datové pole je navíc tradičně užívaným pojmem zpracování hromadných dat.
- **Minimální datová pole**, která jsou potřebná pro výklad základních funkcí, jsou:
  - **jméno položky** (zadávané) - jednoznačná identifikace, obvykle pořadové číslo ze seznamu nakupovaných položek a číslo výkresu pro vyráběnou položku
  - **požadované množství** (zadávané)
  - **plánované množství** (vypočtené)
  - **kalkulační množství** - (zadávané) množství, které se používá pro výpočet nákladů a průběžných dob v případě, že požadované a plánované množství nejsou ještě zadány ani vypočítány a je třeba odhadnout výrobní cenu a doby výroby
  - **průběžná** doba (obstarání nebo zhotovení - zadávaná nebo vypočtená, pokud jsou pro vyráběnou položku zadány operace)
  - **celková průběžná doba** (vypočtená) - součet max. dob jednotlivých montážních stupňů
  - **kód nízké úrovně** (vypočtený) - maximální montážní úroveň, kde se daná součást vyskytuje, tento pojem, který je podstatný pro algoritmizaci bude podrobněji vysvětlen později
  - **cena položky** (pro nakupované zadaná, pro vyráběné zadaná nebo vypočtená, v závislosti na existenci výrobního postupu)
  - **cena vypočtená z postupu** (vypočtená)
- Dále by mezi vlastnosti položky patřily měrné jednotky (zásob, nákupu, prodeje, ceny), normy materiálu, třídící čísla, názvy a textové popisy položky, dodavatel, datum vytvoření a změny, jména referentů, množství na skladě a ve výrobě. Prakticky položka má přiřazeny desítky až stovky datových polí.

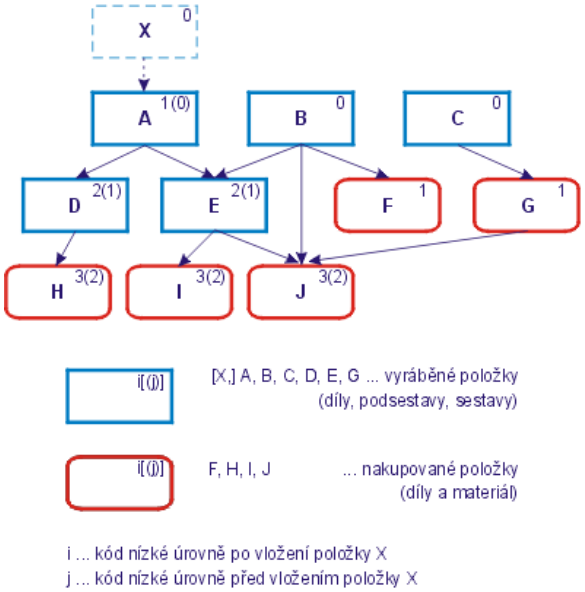
Kusovník – definice a vazby

- **Definice kusovníku**
- Kusovník je vyjádřením struktury výrobku. Z hlediska zpracování dat je kusovník acyklickou orientovanou síťovou strukturou. Uzly tvoří nakupované díly a materiál, vyráběné díly, podskupiny a skupiny výrobku. Hrany jsou vazby mezi vyšší a nižší položkou s uvedením množství nižší položky, které je potřebné pro výrobu vyšší položky. Nakupované položky jsou uzly, kde končí orientovaná hrana a žádná hrana z nich nevychází. Konečné výrobky tvoří vrcholové položky kusovníku, což představuje uzly, odkud vychází orientované hrany a kam nevstupuje žádná hrana.
- Hrany jsou vyjádřeny tzv. kusovníkovými vazbami.
- **Vazby kusovníku**
- Minimální datová pole, která jsou potřebná pro výklad základních funkcí, jsou:
  - **jméno vyšší položky** (zadávané)
  - **jméno nižší položky** (zadávané)
  - **množství nižší položky pro jednotku vyšší položky** (zadávané) - pro vyráběnou položku je obvykle v kusech, jinak může být v závislosti na měrné jednotce nižší položky toto množství i desetinné.
- Dále by bylo možné uvažovat např. číslo montážní operace, pro kterou bude nižší položka potřebná, čísla skladů pro výdej nižších položek a zaskladnění vyšší položky. Pro dělený materiál lze zadávat počty kusů, rozměry i hmotnosti. Lze zadávat i tzv. segmenty, kdy z jednoho kusu polotovaru vzniká několik dílů. Jedna nižší položka může být uvedena vícekrát, což umožňuje optimalizovat např. termíny výdeje do montáže v případě dlouhých dob montáže. Prakticky je počet datových polí pro jednu vazbu větší než deset.

Rozpad kusovníku

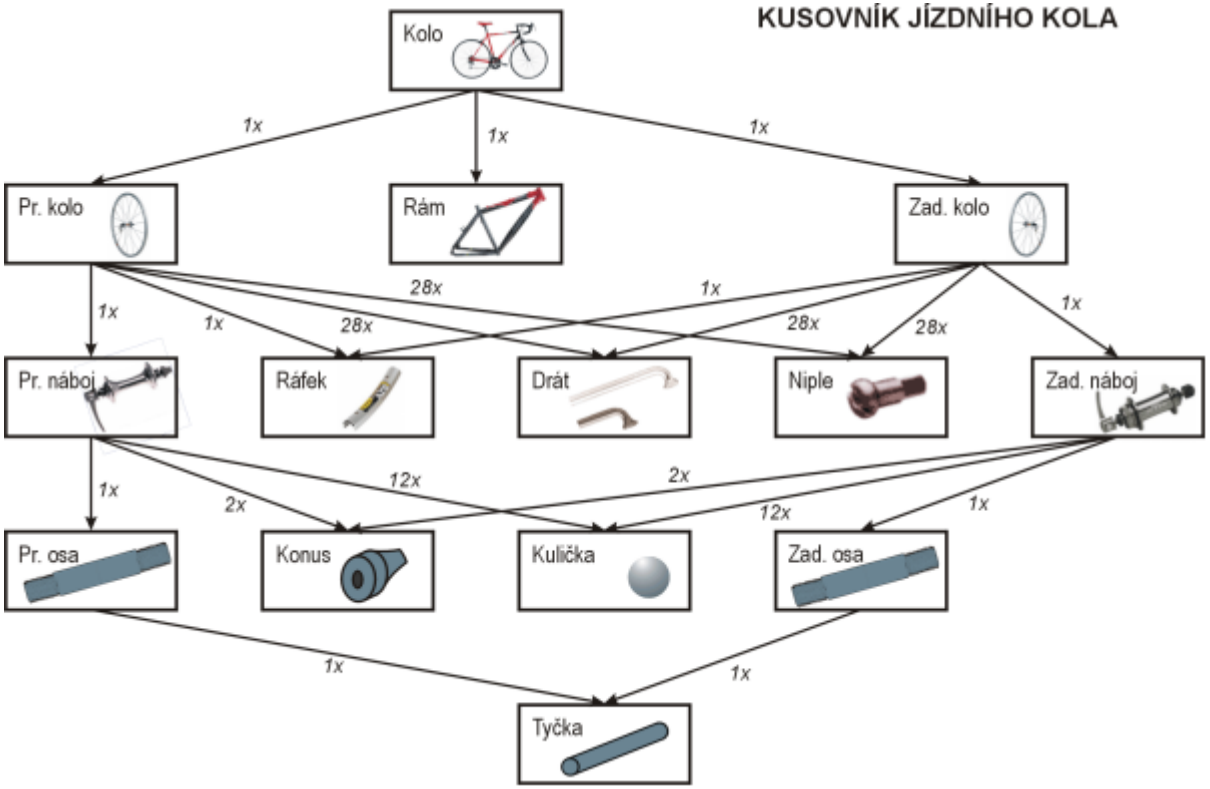
- Rozpad kusovníku představuje výčet položek, které jsou zapotřebí pro výrobu vyšší položky:
  - **jednoúrovňový rozpad** obsahuje jen položky, které přímo vstupují do vyšší položky
  - **víceúrovňový rozpad** obsahuje položky, které vstupují přímo do vyšší položky, dále položky, které vstupují do nižších položek a rozpad těchto položek. až do nakupovaných položek
  - **inversní rozpad** udává, kam lze součást nebo skupinu zamontovat
  - **víceúrovňový inversní rozpad** obsahuje položky, a vyšší úrovně podskupin, skupin a výrobků, kde je užita daná položka
  - **souhrnný kusovník** udává celkové množství jednotlivých položek (nejčastěji jen nakupovaných) bez ohledu na jejich pozici, které jsou zapotřebí pro výrobu jedné zadané vyráběné (nejčastěji vrcholové) položky
- Rozpad kusovníku není trvalou datovou strukturou, nýbrž výsledkem práce programu, který zpracovává kusovníkové vazby.
- V rozpadu je provedeno **vynásobení** potřebných množství jednotlivých stupňů (množství na provedení), ale **neprovádí** se jejích **sečtení**, takže se díly v rozpadu vyskytují opakovaně.

Obecný příklad kusovníku



- Na obrázku je vidět příklad kusovníkových struktur pro tři výrobky A, B a C. Nakupované položky jsou H, I, J a F. Šipky představují vazby.
- Na obrázku je dále vysvětlen pojem kódu nízké úrovně. Pokud jsou A, B a C vrcholové výrobky na nulté úrovni, pak D, E, F a G jsou položky na první úrovni a H, I, J jsou na druhé úrovni. Položka J vstupuje do 1 i 0 úrovně.
- Kódy nízké úrovně se změní např. zařazením výrobku A do nějakého vyššího celku X (např. turbíny do celé elektrárny).

Kusovník jízdního kola



Pracoviště

- Pro definici technologického postupu je třeba definovat **pracoviště**, kde probíhají vlastní **operace technologického postupu**.
- **Pracoviště** je technologicky a kapacitně určený, prostorově ohraničený a relativně samostatný soubor pracovních prostředků, který tvoří z hlediska řízení výrobního procesu dále již nedělitelný prvek výrobního systému.
- Z hlediska modelu základních dat jsou potřebná tato datová pole:
  - **číslo pracoviště** (zadávané)
  - **sazba** - Kč/hod (zadávaná)
  - **přechodový čas** (zadávaný)
- **Číslo pracoviště** může vyjadřovat konkrétní stroj, podskupinu nebo skupinu strojů, kterou je třeba rozlišovat za účelem rozvrhování a řízení výroby.
- **Sazba pracoviště** je údaj, kolik stojí hodina práce (mzda + režie)
- **Přechodový čas** je doba, kterou je třeba připočítat k technologické délce operace pro účely termínování, jedná se o součet transportního času a plánované časové rezervy (rozpracovanosti). Obecně lze uvažovat dvourozměrný model přechodových časů mezi každými dvěma možnými kombinacemi pracovišť, které po sobě následují v postupu. Ve zjednodušení lze přiřadit přechodový čas jen jednomu pracovišti, přičemž se předpokládá, že část tohoto času bude využita před zahájením operace a část po jejím ukončení. Konkrétní rozdělení je provedeno na úrovni dílenského řízení.

Technologický postup

- **Technologický postup** je posloupnost **operací**, které jsou nutné ke zhotovení součásti. Pro každou vyráběnou součást lze uvažovat více výrobních postupů, které se liší např. minimálním a maximálním ekonomicky vyráběným množstvím (univerzálním, kusovým, sériovým). Každý postup potom obsahuje tato datová pole:
  - **číslo vyráběné položky** (zadávané)
  - **index postupu** (zadávaný)
  - **minimální ekonomické množství** (zadávané)
  - **maximální ekonomické množství** (zadávané)
- Číslo položky a index postupu jednoznačně identifikují každý postup.

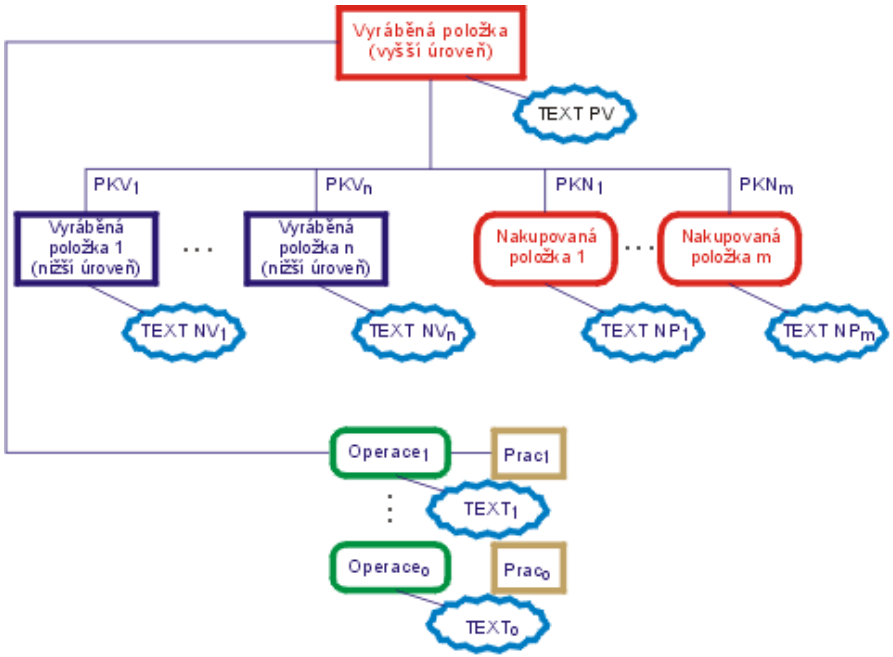
Technologická operace

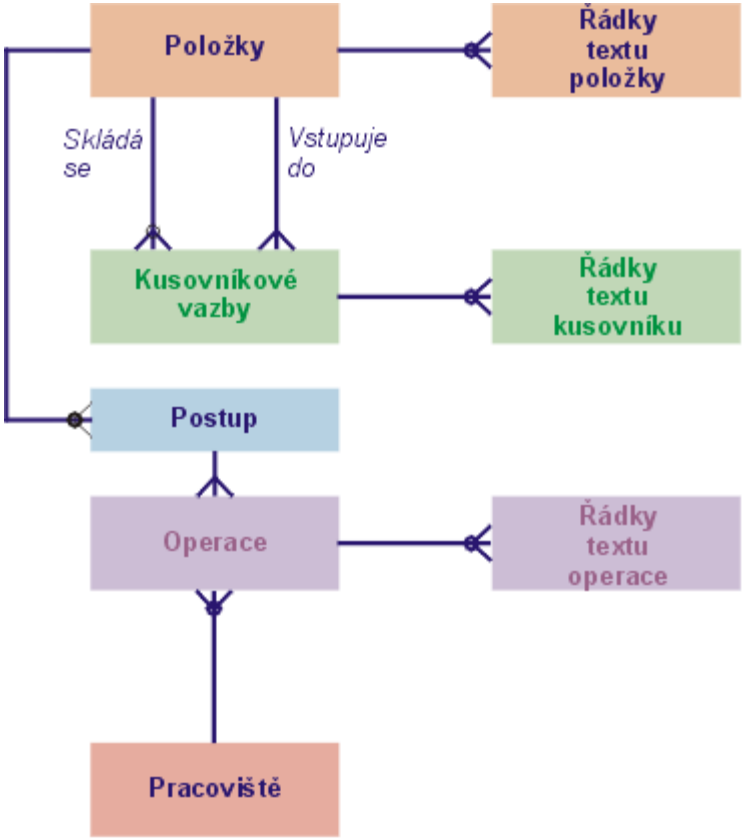
- Operace jsou číslované (tzv. pozice postupu), pořadí je pro výrobu závazné. Jedno pracoviště se může v postupu vyskytnout vícekrát, pro identifikaci operace jsou podstatné číslo položky, index postupu a číslo operace. Každá operace má přiřazena tato datová pole:
  - **jméno položky** (zadávané)
  - **index postupu** (zadávaný)
  - **pozice operace** (zadávaná)
  - **číslo pracoviště** (zadávané)
  - **přípravný čas** (zadávaný)
  - **kusový čas** (zadávaný)

Příklad technologického postupu

```
 #EM260396        S Y S K L A S S      (c) GTSystems2  DETVA      Tisk: 04.11.2005

-------------------------------------------------------------------------------
|    ELITEX    |  Cislo vykresu :3-002-08-7232        Alt : 1      | list :    1 |
|    KOYNE     |  Nazev : HALTER                                   | listu:    2 |
|              |---------------------------------------------------------------|
|    T L G     |  Tvarove cislo |    Vyrobek     |  Kmen.dilna  |Platnost TLGP od |
|    POSTUP    |   53311001     |  4511-H+B      |     334      |    04.04.2000   |
-------------------------------------------------------------------------------
Hlavicka: Frantisek SCHODELBAUER  21.02.1997       Vydano dne :

C.casti  Nomenklatura          N-kus      PED     Nm str.     Nm prac.    Mzdy [Kc]

       0  413413831               1
                                          99999      14.040       14.040       9.73

Jakost mat.       Cista hmot    Dr.mor.    Norma polotovaru    Tech.dod.podm.

ALSI10CU-231UA        0.0000                 4-060-15-0208
                                    CM

Spotrebni rozmer          ROZAP  ROZBP  ROZCP  ROZDP KSSR KSPO MJM Kj

ODLITEK      3548-EJAB      0.0    0.0    0.0    0.0    1    1 KS  0

Spothmot       Spotrmnoz      Spotcena   KDRM              KDRM-Pozn

     0.0000          1.0000        0.01

-------------------------------------------------------------------------------
Polozka vstupuje do :
Cislo vykresu        MJ      Mnozstvi      Cislo vykresu        MJ      Mnozstvi
2-008-10-1530        KS         1.0000
-------------------------------------------------------------------------------
COP  ZO STRED  PRAC     NAZEV OPERACE                      PDP      TP      TKS      TTK
     CENAKOOP           TYP STROJE                         POS              TK
-------------------------------------------------------------------------------
010  01 334     0941201 rysovani                            1       0     0.000   0R
                        RYSOVACSKE PRAC                     1             0.000

Prorysovat 10 kusu v davce pro navrtani stredicich
dulku
-------------------------------------------------------------------------------
020  00 334     0468201 vrtani                              1      14     1.277   5J
                        V20/4                               1             1.277

Navrtat 2x na dorazy (dorazet na jednu rovinnou plochu)
-------------------------------------------------------------------------------
030  00 335     0412503 soustruzeni                         1      23     2.759   6J
                        SVR18RA/1000                        1             2.759

Upnout mezi hroty, zarovnat celo ku L=34 k hrotu,
soustr. D20h8 L=20-0,5;sr.hr.0,5x45st.
```

Datový model výrobních dat





Integritní omezení výrobních dat

- Platí tato integritní omezení:
  - **vyšší** položka se může skládat z různých **nižších** položek.
  - **nižší** položka může vstupovat do různých **vyšších** položek
  - technologický **postup** se skládá z řady **operací**
  - každé **operaci** je přiřazeno jedno **pracoviště**
  - na jednom **pracovišti** může být zpracována řada **operací**
  - v **postupu** se totéž **pracoviště** může opakovat **vícekrát**
  - v postupu záleží na **pořadí** operací
- Položka a operace může mít přiřazen text proměnné délky, řeší se např. proměnným počtem řádek pevné délky.

Práce s kusovníkem – kontrola zacyklení

- Kusovník je definován jako **acyklická síťová struktura**. Nelze připustit, aby nadřazená položka byla přímým nebo nepřímým následníkem své přímé nebo nepřímé podřazené položky. Jako příklad takového zacyklení lze uvést, že do správné kusovníkové struktury lokomotivy by bylo omylem zadáno, že celá lokomotiva je další částí podvozku.
- V reálných systémech řízení výroby je třeba zajistit:
  - aby k zacyklení nemohlo dojít
  - v případě, že k němu např. poruchou dat nebo chybou nějaké části programu došlo, aby nedošlo k nekonečnému cyklu nebo nekonečné rekursi při zpracování, což může být velmi obtížný problém s velkou odpovědností programátorů.
- *Algoritmus:*

1. Předpokládá se, že se kusovník může postupně vytvářet a upravovat zadáváním dalších vazeb.
2. Prázdný kusovník (bez jakékoliv vazby) není zacyklen.
3. Před každým vložením nové vazby je třeba prohledat pro vkládanou podřazenou položku v jejím úplném kusovníkovém rozpadu, zda se tam nevyskytuje vkládaná nadřazená položka.
4. Dále je před každým vložením nové vazby třeba prohledat pro vkládanou nadřazenou položku v jejím úplném inversním kusovníkovém rozpadu, zda se tam nevyskytuje vkládaná podřazená položka.
5. Body 3 a 4 se řeší rekursí nebo vhodným převodem rekurse na cyklus.

Práce s kusovníkem – tisky rozpadu kusovníku

- Tisk rozpadu kusovníku, resp. inversního rozpadu je typická rekurzivní úloha.

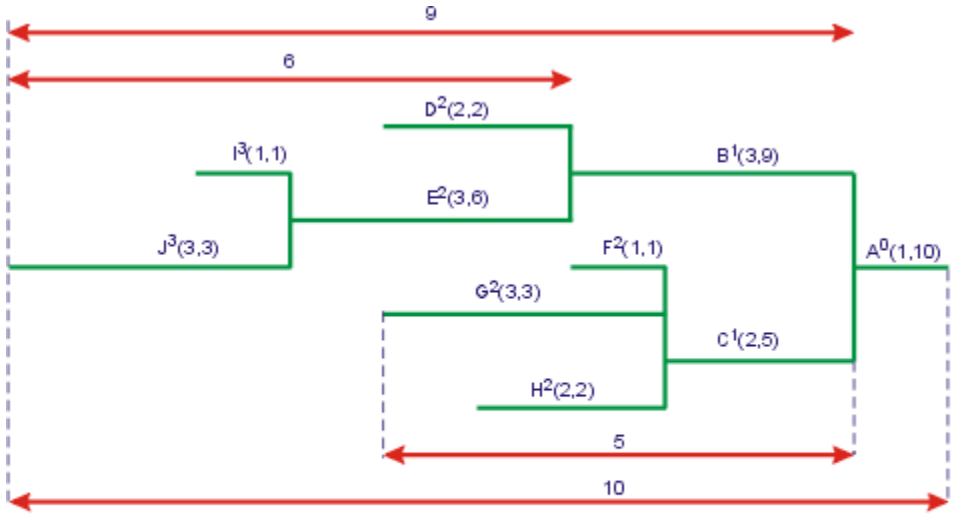
- *Algoritmus:*

1. Pro zadaný uzel (např. vrcholovou položku) se provede volání tisku kusovníku s parametrem celkového množství = 1.
2. V rámci tisku jedné úrovně se vypíší postupně jednotlivé položky nižší úrovně včetně množství nižší položky vstupující do vyšší položky a celkového množství na provedení jakožto součinu množství nižší položky vstupující do vyšší položky a parametru volání tisku kusovníku.
3. Po tisku každé vazby se volá rekurzivně úloha tisku, tentokrát je výchozím uzlem nižší položka a zadaným množstvím množství na provedení příslušné nižší položky.
4. Rekurze končí, pokud již neexistují nižší položky.
5. Při tisku se pro každou úroveň rekurse (rovnající se úrovni kusovníku, nikoliv kódu nízké úrovně) provede odsazení. Tím se vytiskne kusovník orientovaný zleva doprava.

Práce s kusovníkem – výpočet úrovně

1. nulování všech úrovní
2. pro všechny položky, které nemají vyšší položku, ponechání úrovně 0
3. pro nižší úroveň, nastavení úrovně na 1
4. pro úroveň 1 a další, nastavení všem rozpadlým položkám úroveň na maximum z již nastavené úrovně a běžné úrovně + 1

Práce s kusovníkem – výpočet průběžných dob

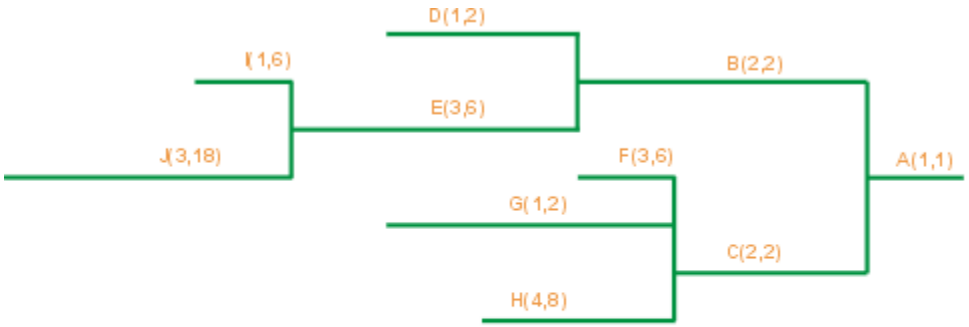


$X^k(i, j)$: ... i - doba zhotovení/nákupu součásti
j - celková průběžná doba součásti
k - úroveň rozpadu kusovníku

- nastavení celkové průběžné doby na průběžnou dobu (dobu nákupu nebo dobu z technologického postupu)

1. začne se nejnižší úrovní
2. pro vyšší úroveň, nastavení celkové průběžné doby na maximum z již nastavené průběžné doby a součtu průběžné doby vyšší položky a celkové průběžné doby nižší položky
3. pokud není dosaženo úrovně 1, pokračuje se bodem 2

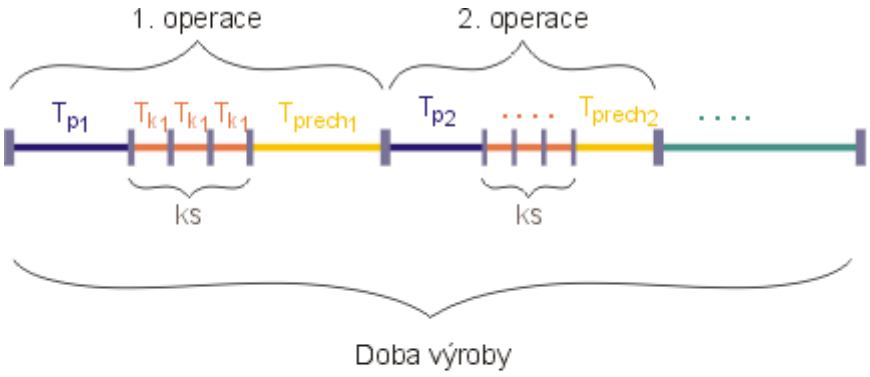
Práce s kusovníkem – určení množství



X(i, j): ... i - počet kusů nižší položky vstupující do vyšší
j - celkový počet kusů na zhotovení vrcholové položky

- je nastaveno požadované množství pro vrcholový výrobek, případně další položky
- nulování plánovaných kusů
- nastavení úrovně na 0
- pro zadanou úroveň, připočtení požadovaného množství do plánovaného množství, pro podřazenou úroveň připočtení potřebného rozpadu do plánovaného množství
- nastavení další úrovně a opakování předchozího kroku, pokud existuje nižší úroveň

Práce s postupem - termínování operací



T_{pi} ... přípravný čas i-té operace

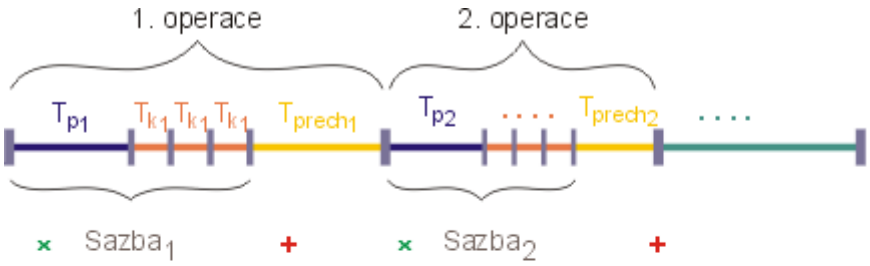
T_{ki} ... kusový čas i-té operace

T_{prechj} ... čas přechodu pracoviště pro i-tou operaci

ks ... množství

- Pro všechny položky, které mají postup, se provede výpočet takto:
  - pokud je již vypočteno plánované množství, počítá se pro trvání operace čas přípravný + čas přechodu (z pracoviště) + čas kusový * plánované množství
  - pokud plánované množství ještě není vypočtené, použije se kalkulační množství
  - průběžná doby výroby dílu nebo montáže je součtem dob jednotlivých operací.

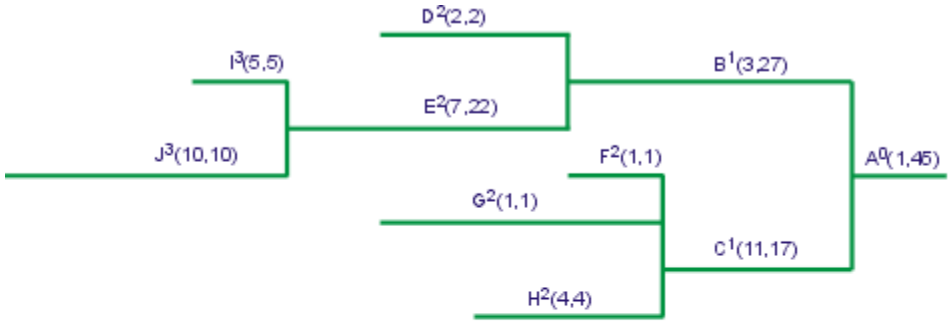
Práce s postupem´ - určení nákladů na výrobu z postupu



T_{pi} ··· přípravný čas i-té operace
T_{ki} ··· kusový čas i-té operace
T_{prechi} ··· čas přechodu pracoviště pro i-tou operaci
$Sazba_i$ ··· množství

- Pro všechny položky, které mají postup, se provede výpočet pro každou operaci takto:
  - pokud je již vypočteno plánované množství, počítá se cena = (čas přípravný + čas kusový * plánované množství) * sazba (z pracoviště)
  - pokud plánované množství ještě není vypočtené, použije se kalkulační množství

Výpočet nákladů (ceny) výrobku



$X^k(i, j)$: ... i - cena zhotovení/nákupu součásti
j - celková cena součásti
k - úroveň rozpadu kusovníku

- Cílem je stanovit cenu jednotlivých položek. Počítá se z kusovníku po stanovení cen jednotlivých dílů a montážních skupin z postupu.
- *Algoritmus:*

1. nastavení úrovně na maximální (nejnižší)
2. pro všechny položky této úrovně: pokud je nenulová cena vypočtená z postupu, sečte se tato cena s cenami všech položek nižší úrovně, které do dané položky vstupují, jinak se cena ponechá
3. nastavení vyšší úrovně a pokračovat bodem 2, pokud existuje vyšší úroveň (končí se na úrovni 0)

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Zadání příkladu na plánování množství

FAKULTA STROJNÍ KPV

Jméno	Požadovaný počet	Plánovaný počet	Nízká/Dispoziční úroveň
A	1		
B	2		
C	0		
D	2		
E	0		

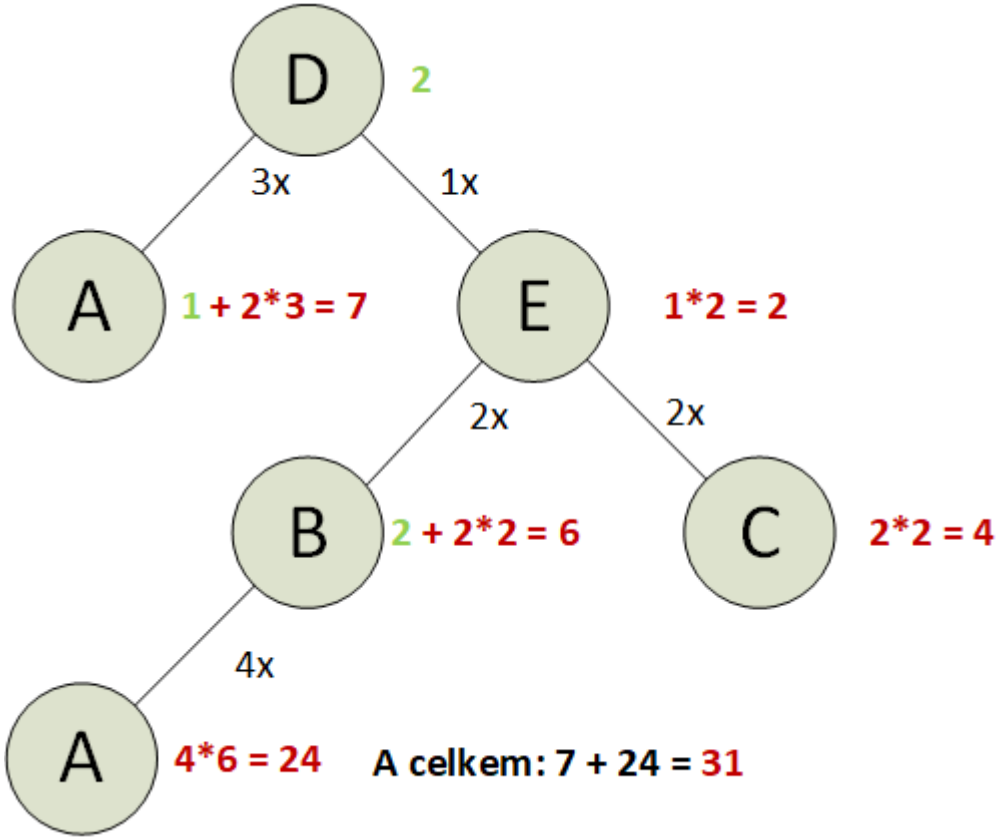
Vyšší položka	Nižší položka	Počet
E	C	2
E	B	2
B	A	4
D	A	3
D	E	1

Řešení příkladu na výpočet množství

Jméno	Požadovaný počet	Plánovaný počet	Nízká/ Dispoziční úroveň
A	1	1+6+24=31	~~0~~1~~2~~3
B	2	2+4=6	~~0~~1~~2~~
C	0	4	~~0~~1~~2~~
D	2	2	0
E	0	2	~~0~~1

Vyšší položka	Nižší položka	Počet
E	C	2
E	B	2
B	A	4
D	A	3
D	E	1

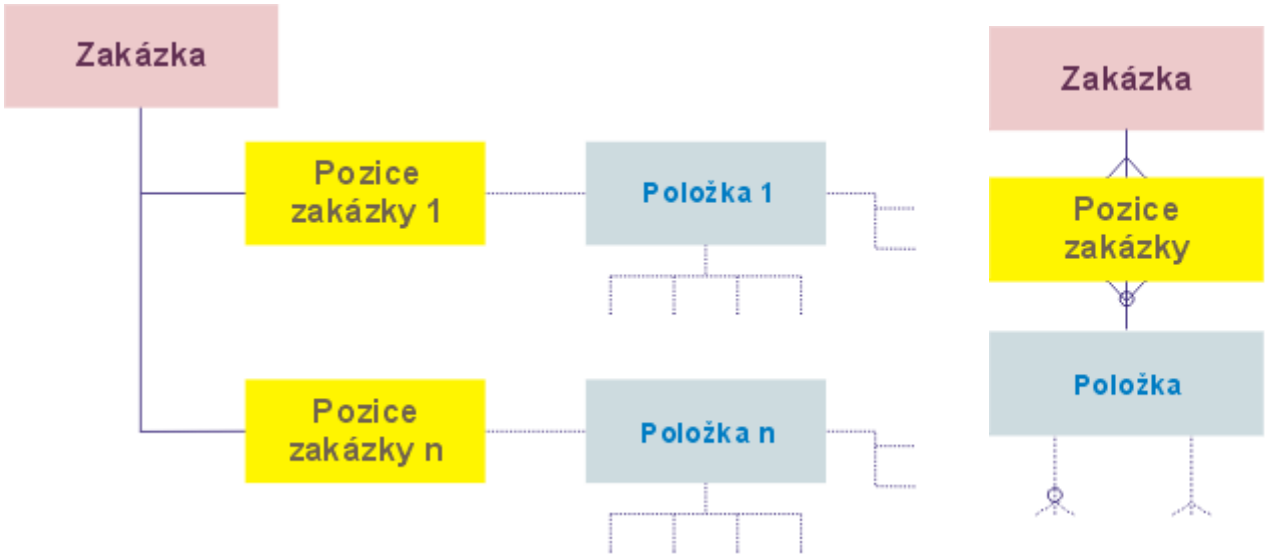
Grafické řešení výpočtu množství



Zakázka, pozice zakázky

- Zakázka se skládá z **hlavičky** a **pozic** zakázky.
- **Hlavička zakázky**
- Hlavička zakázky obsahuje např. tato datová pole:
  - **číslo zakázky** (zadávané) - jednoznačná identifikace
  - **jméno zákazníka** (zadávaný)
  - **termíny splatnosti** (zadávané)
- **Pozice zakázky**
  - Pozice zakázky obsahují např. tato datová pole:
  - **číslo zakázky** (zadávané)
  - **číslo pozice** (zadávané)
  - **číslo položky** (zadávané) - požadovaný výrobek nebo náhradní díl
  - **množství** (zadávané)
  - **termín dodání** (zadávaný)

Datový model zakázky



Plánování zakázky

- **Plánování zakázky**

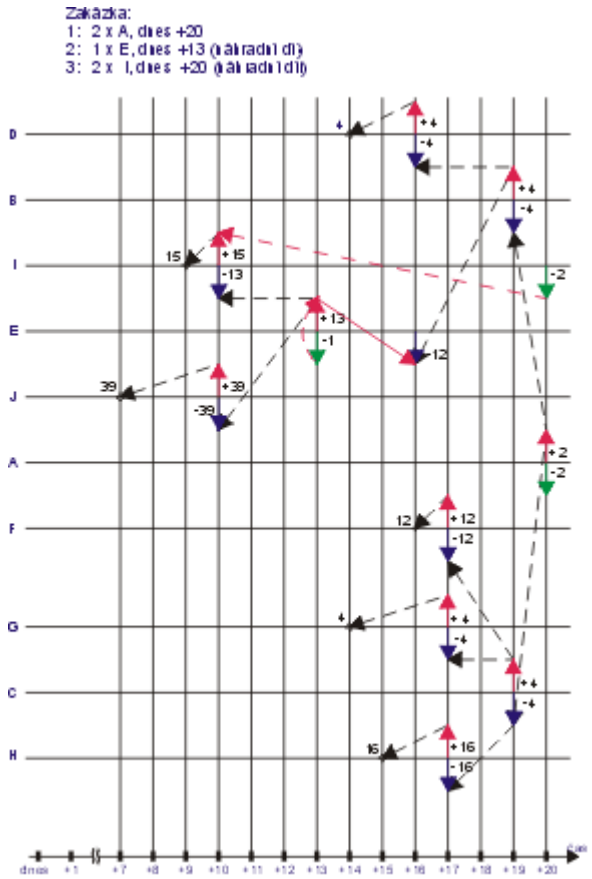
- Plánování zakázky je obdobné stanovení množství v kusovníku, navíc se zohledňují termíny pozic zakázky, jedna položka se může vyskytnout vícekrát, rozdíl je v termínu požadavku na výrobu nebo nákup jednotlivé položky.

- *Algoritmus plánování*
- Postupuje se od pozic zakázky, ty vygenerují požadavky v čase, z nich se vybere požadavek s nejnižším kódem úrovně, pro tento požadavek se zaplánuje výrobní příkaz nebo objednací návrh, výrobní příkazy vygenerují jednoúrovňovým rozpadem další požadavky na výrobu součástí s termínovým předstihem odpovídajícím době vypočtené postupu.

Vztah požadavku na výrobu a výrobního příkazu

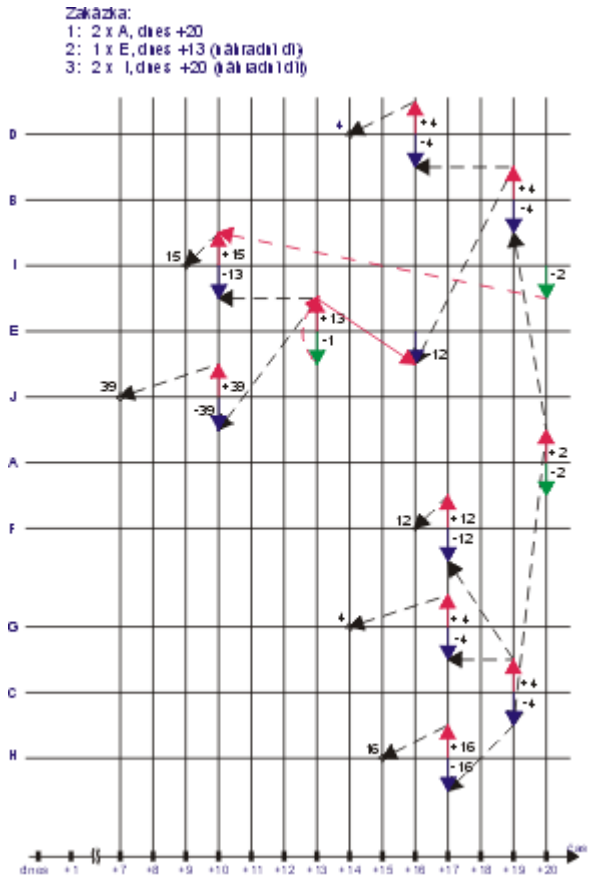
- Požadavek lze pokrýt různým způsobem, z nichž jsou dále uvedeny jen vybrané:
  - jeden požadavek = jeden výrobní příkaz (obvyklý způsob dávkování)
  - více požadavků = jeden výrobní příkaz (zhromadnění)
  - jeden požadavek = více výrobních příkazů (rozdávkování)
  - jeden požadavek = jeden nebo více výrobních příkazů o stejném množství (pevná velikost dávky)
- **Rozdávkování** se provádí tehdy, pokud by požadované množství přesahovalo maximální množství vhodné pro výrobu. Tento algoritmus zabraňuje dlouhodobému obsazení kritického pracoviště.
- **Zhromadnění** se provádí tak, že v případě, že se jedna položka vyskytne vícekrát v intervalu zhromadnění, vytvoří se výrobní příkaz nebo objednací návrh, který pokryje více požadavků, v tomto případě je termín dán nejdříve požadovaným termínem ze všech zhromadněných požadavků.
- **Pevná velikost dávky** se využívá například tehdy, je-li třeba použít pracoviště s pevnou kapacitou (pece, povrchové úpravy).

Příklad rozplánování I



- Na obrázku je vidět rozplánování zakázky v čase s použitím dříve uvedeného kusovníku. Požadavky na jednotlivé položky jsou zobrazovány na časové ose. Požadavky jsou na každé ose zobrazeny plnými šipkami orientovanými dolů, výrobní příkazy a objednání materiálu plnými šipkami orientovanými nahoru. Kusovníkové vazby jsou zobrazeny šikmými čárkovanými šipkami mezi jednotlivými časovými osami jednotlivých položek. Tyto čárkované šipky představují také průběžné doby výroby dílu a objednání materiálu.

Příklad rozplánování 2



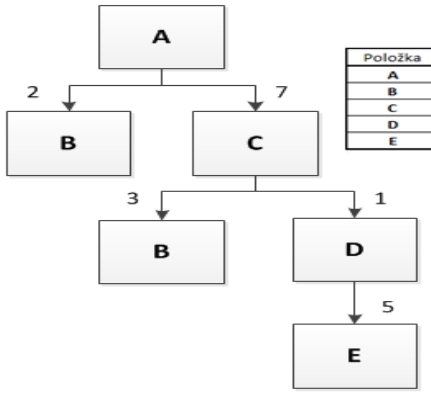
- Algoritmus výpočtu probíhá takto:
- Nejprve se do časových os zaznamenají požadavky zakázky - 2 výrobky A v čase dnes + 20 dní, 1 náhradní díl v čase dnes + 13 dní a 2 náhradní díly v čase dnes + 20 dní.
- Proběhne rozpad výrobku A (má nejnižší kód nízké úrovně, požadují se 2*2 = 4 kusy skupiny C v čase dnes + 19 dní a 4 kusy B v čase dnes + 19 dní
- Takto se postupuje pro další kódy nízké úrovně.
- Ke zhromadnění dochází u skupiny E - požadován 1 kus v čase dnes + 13 dní a 12 kusů v čase dnes + 16 dní, výrobní příkaz v čase dnes + 13 dní kryje oba požadavky, další zhromadnění nastává u materiálu I.
- Nákupy materiálu je třeba objednat: 4 kusy D v čase dnes + 14 dní, 15 I v čase dnes + 9, 39 J v čase dnes + 7, 12 F v čase dnes + 16, 4 G v čase dnes + 14 a 16 H v čase dnes + 15
- Rozplánování zakázky v čase je spolu s účetnictvím základní funkcí podnikových informačních systémů.

Animace plánování zakázky

mrpII.exe

Příklad na plánování zakázky - zadání

Plánování zakázky

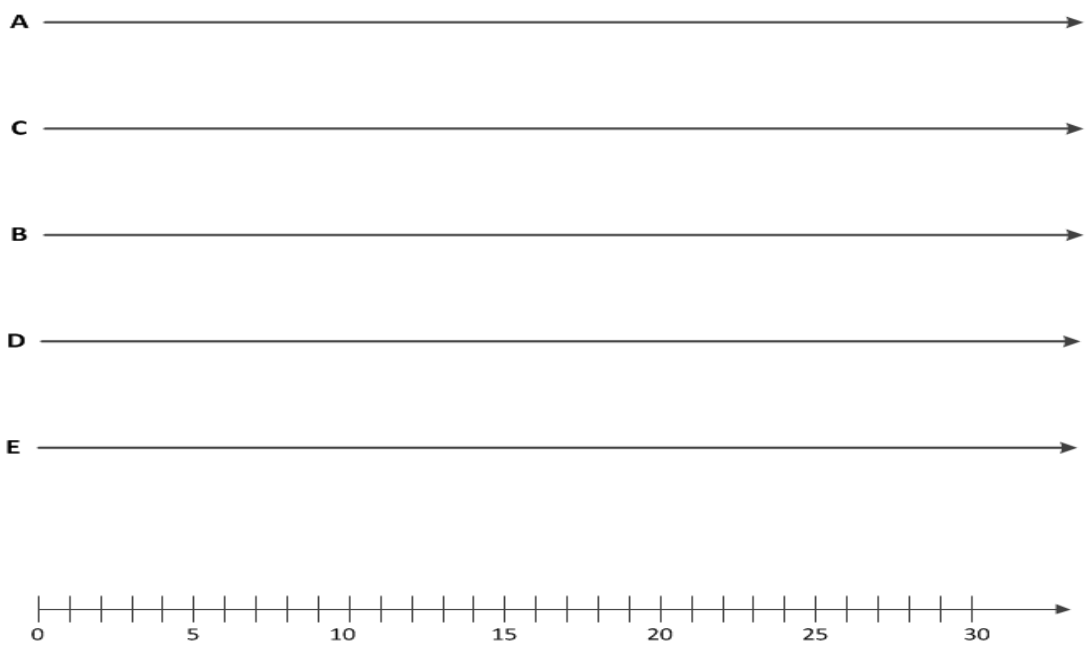


Položky - zadání

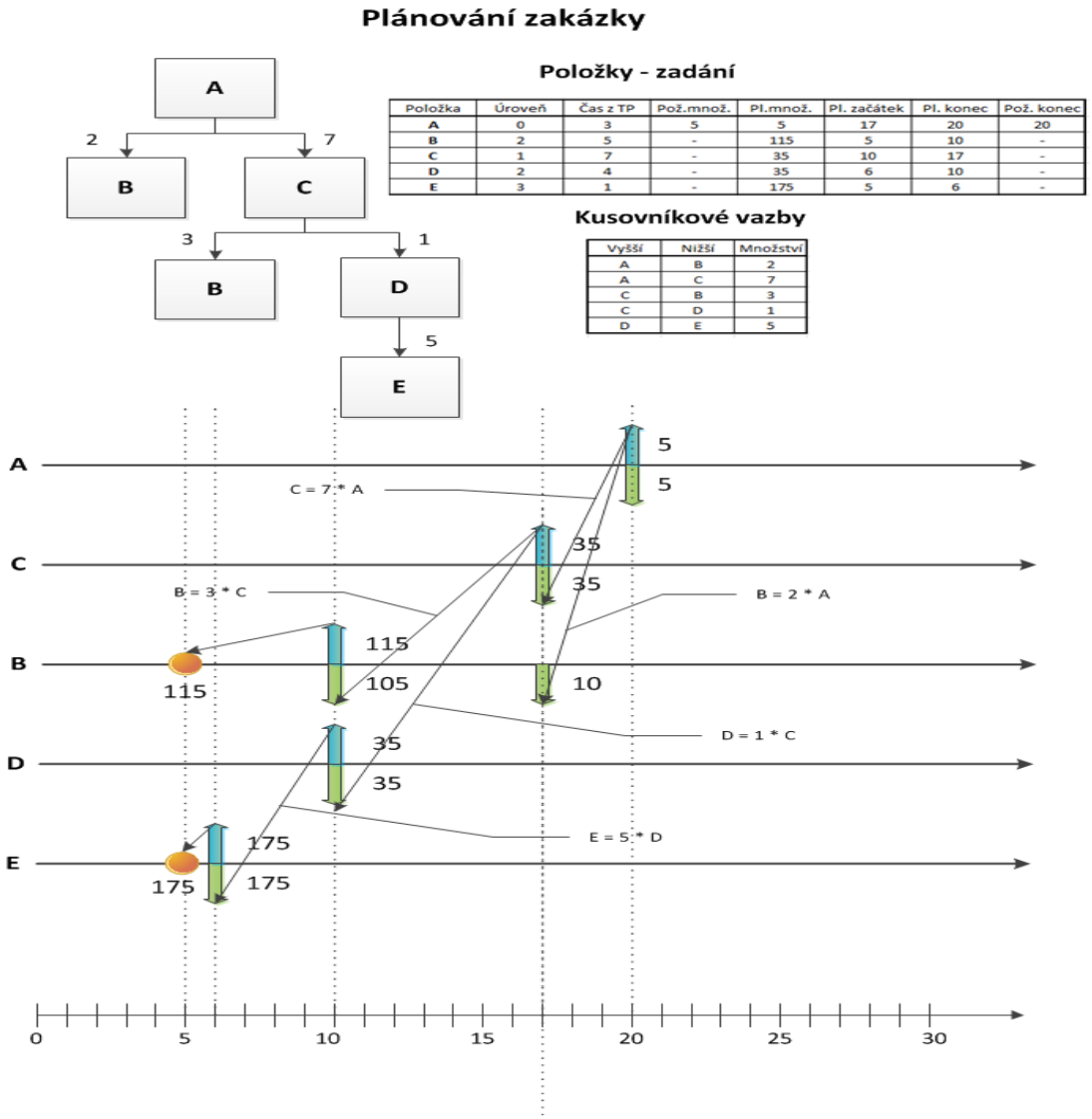
Položka	Úroveň	Čas z TP	Pož.množ.	Pl.množ.	Pl. začátek	Pl. konec	Pož. konec
A	0	3	5				20
B	2	5	-				-
C	1	7	-				-
D	2	4	-				-
E	3	1	-				-

Kusovníkové vazby

Vyšší	Nižší	Množství
A	B	2
A	C	7
C	B	3
C	D	1
D	E	5



Příklad na plánování zakázky - řešení



- Na shledanou