

ALL_PB168

#DB #uvod #entity-relationship-model #ER-model

Organizační údaje

- projekty ve skupinách max 3
- etapy v rámci bloků - 4 (1 týden)
- hodnocení ano/ne :(
- Kladné hodnocení musíte mít u min 3
 - Prázdné odevzdání není odevzdání
- email pokud o prodloužení času
- Písemná zk (opravdu písemná)
 - 6 příkladů - otevřené
 - max 21b (11 a víc ok :)
- projekt vybíráno podle sumy UČA Knihovna/Hotel

Úvod

- IS informační systém
 - Sys pro poskytování informací uživ a k tomu potřebné nástroje pro sběr a správu dat
 - Informace jsou data v kontextu
- Návrh IS
 - SW není problém ale lidi
 - Lidé - nutné získat, vyškolit, prosazovat jejich prospěch
 - Uživatelé nemohou využít IS sami
 - nelze ale využít bez nich, nutno rozumět potřebám
- Realizace na profese a úkoly
- Databázové sys
 - Vyhledávatelnost dat
 - Architektura dat. sys
 - Datové modely, Struktura dat sys, relační databáze (SQL, tabulky, vztahy mezi nimi, klíče), objektové databáze
 - Zpracování transakcí - kroky pro vykonání ? (pro commit, dá se reversnout, stojí to nějakou režii ale)
 - HI. - konsistentní strukturu dat
 - aktuální info, všichni vidí to samé / dva stejné requesty nebudou jiné, uživatel nemůže zadat hodnotu mimo normu (neplatnou hodnotu),
 - Vztahy

- primární, cizí klíč
- většinou podle id
-

Student

UČO	Jméno
1	x
2	y
3	z

Zápis

UČO	Kod
1	PB168
3	PB168

Předmět

Kod	Naezv
PB168	Z. DB.
IB000	Mat. z. i.

- Datový model
 - sada nástrojů pro popis dat
 - a vztahů mezi nimi
- Jazyk pro definici dat
 - DDL, poskytuje výrazy na definici schéma databáze
 - Kompilátor DDL vygeneruje množinu tabulek
- SQL
 - neprocedurální jazyk
 - Např.:
 - z prezentace
- Schéma (struktura (logický, fyzická))/Databáze (aktuální datový obsah)/Databázový systém (Implementace konkrétního dat. modelu včetně dalšího SW (MySQL, PostgreSQL...))

Entitně-relační model/diagramy

- ER model, komunikační nástroj se zadavatelem/zákazníkem
- používá vazební tabulky, ... (vazební tabulku jsme použili na "Zápis" minule)
- vždy? ukázán primární klíč (unikátní id)
- dvojitá čára = musí být ve vztahu
- šipka = nejvýše jednoho (tam kam ukazuje)
- Příklad - půjčka v bance
 - klient si žádá o půjčku (účel, kolik, info o klientovi)
 - banka schvaluje půjčku
 - data/procesy
- DFD - prezentace
 - data-flow-diagram
- Cardinality - 1-1, 1-N, N-1, N-N
- Atributy - entita je reprezentována atributy
 - mají svoje jméno (unikátní v entitě), Doména (schválené hodnoty)
 - Typy: (prezentace)
 - Simple attribute - single value
 - Composite - atribut z atributů (name -> first_name, last_name...)
 - Multi-valued (více emailů v jediném atributu)
 - Derived (Odvozený) - age, given date_of_birth (neukládá se? (dá se vypočítat))
- Obdelník = entita
- Diamant = relationship set
- Elipsy = attributes
- Lines = link
- Attributy:
 - Double ellipses = Multivalued attributes (další tabulka vlastně)
 - Dashed (přerušovaná) ellipses = derived attributes
 - Underline (podtržení) = primary key (! každá tabulka by ho měla mít (v diamantu jde i kompoziční, který se skládá ze svou atributů (např. dva cizí klíče)))

#DB #ER-model #entity-relationship-model

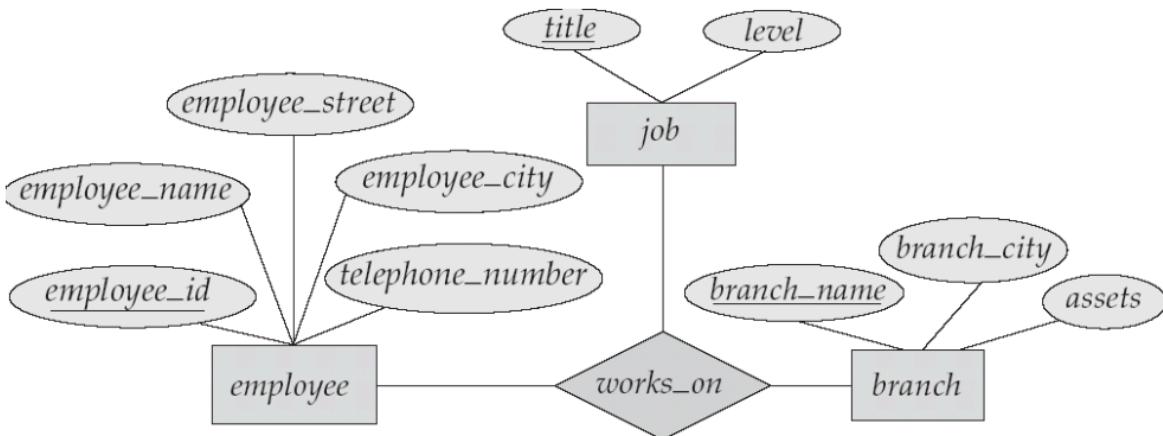
<https://is.muni.cz/auth/el/fi/podzim2025/PB168/index.qwarp?prejit=4928369>

<https://is.muni.cz/auth/el/fi/podzim2025/PB168/um/slides02-erd.pdf?predmet=1654908>

One-To-Many Relationship (1-N)

- customer <---borrower --- loan
 - může mít anonymní customery
 - právě jednoho zákazníka (?)
- customer <---borrower === loan

E-R Diagram with a Ternary Relationship



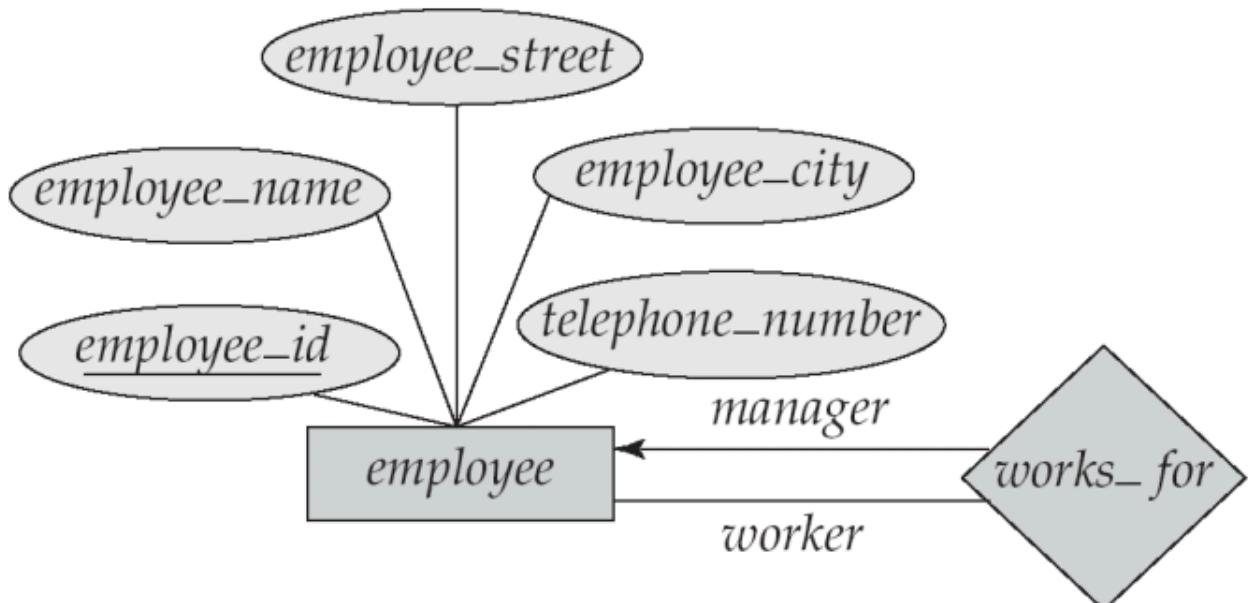
- many to many to many
s -> k job
- jedna práce k "employee" který může být na několik pobočkách

E-R Diagram with Aggregation

- sémantika, která zdůrazňuje je mezi tímto množinou a tímto vztahem

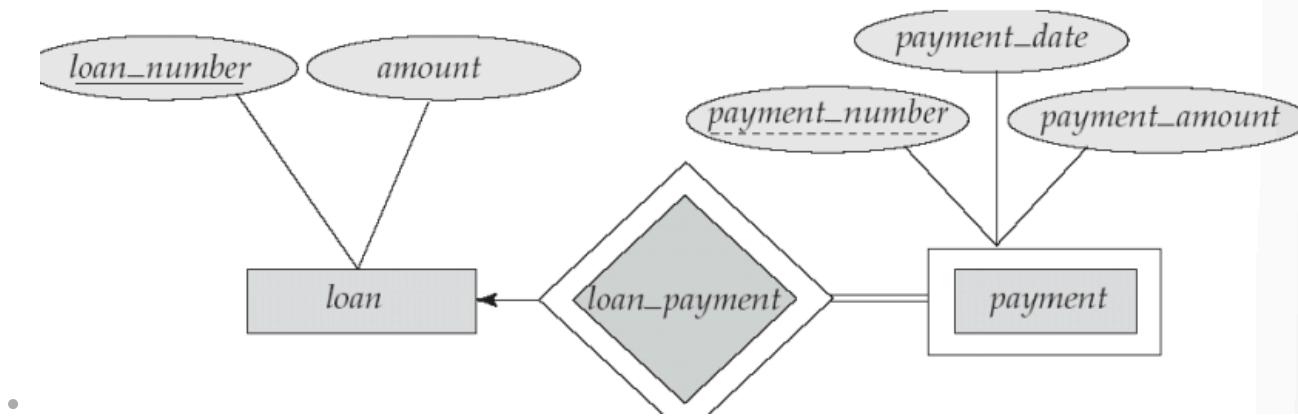
Roles

- jak entita reaguje na nějaký vztah
- label linek



Slabá entitní množina

- dvojitě zakreslená a primární klíč? je podtržen čárkovaně
- nemůže existovat samostatně



Design Issues

- str 37

Cvičení 1

[Cvičení1-obchod](#)

[Cvičení1-Uni_system](#)

#dfd #IS

<https://is.muni.cz/auth/el/fi/podzim2025/PB168/um/is/DFD.pdf>

Konceptováno:

- Datová vrstva (relační datová vrstva) <> Informační systém <> Data se zobrazují (výstupy)(uživatelská vrstva)

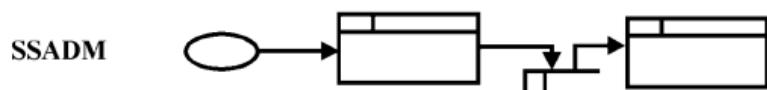
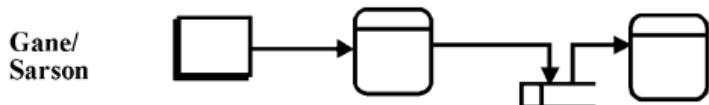
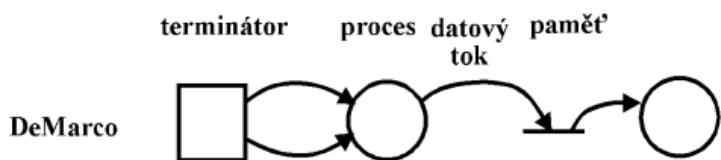
DFD - Data Flow Diagram

- modelovací nástroj -> umožňuje zobrazit systém jako síť procesů
- funkčně orientovaný pohled na systém

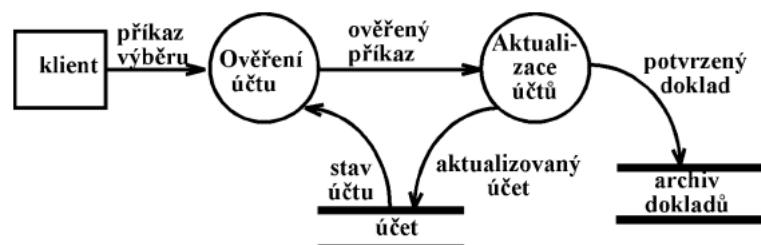
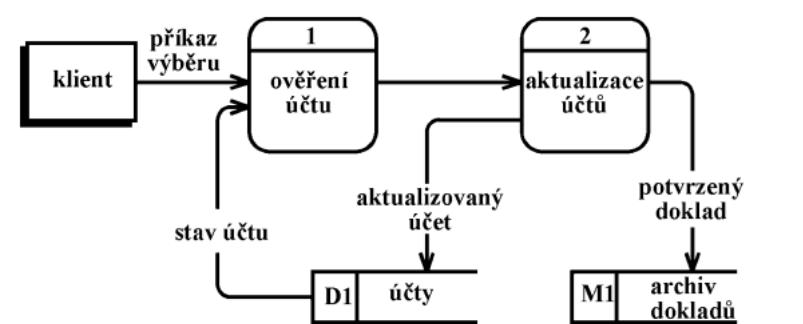
Komponenty DFD

- proces, funkce, transformace
- Paměť, datastór
- Datový tok, tok

- Terminátor, vnější entita (vstupní/výstupní sestava)



Příklad notací Gene/Sarson a Yourdon/DeMarco



- šipka z paměti do ověření představuje **čtení**
 - aktualizovaný účet = **update**
 - potvrzený doklad = **insert**

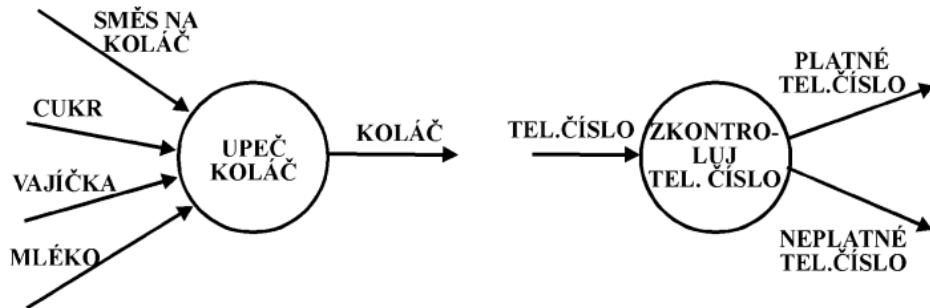
Procesy

- ukazují část systému, která transformuje určité vstupy na výstupy.
 - pojmenován jediným slovem, frází, jednoduchou větou.
 - pokud název obsahuje jméno osoby, skupiny, zařízení, pak jméno vyjadřuje, kdo provádí nějakou činnost, transformaci dat, místo toho, aby vyjádřilo podstatu transformace

Toky

- cesta, po které se pohybují datové shluky (informační pakety) z jedné části systému do druhé.
- V některých případech vyjadřují toky pohyb fyzických materiálů. U reálných systémů mohou být na DFD současně toky, které vyjadřují pohyb materiálů a dat.

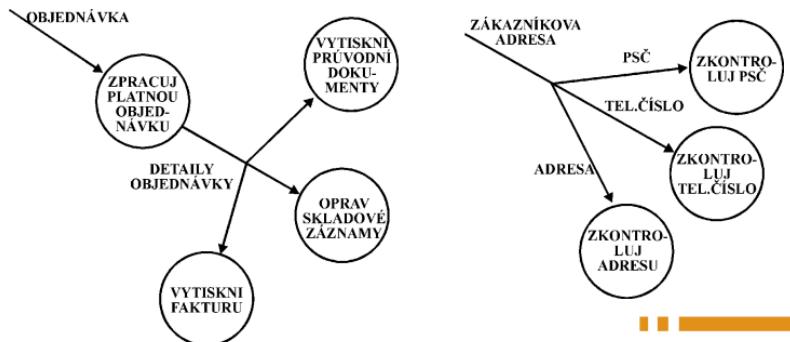
Materiálové a datové toky



- Paket má jiný význam, pokud putuje na různě pojmenovaných tocích.

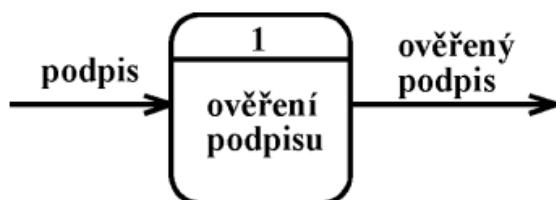
Divergující toky

- Duplikáty tejného datového paketu jsou zaslány do různých částí.



Pojmenování datových toků

- Jméno toku by mělo vyjadřovat podstatu transformace

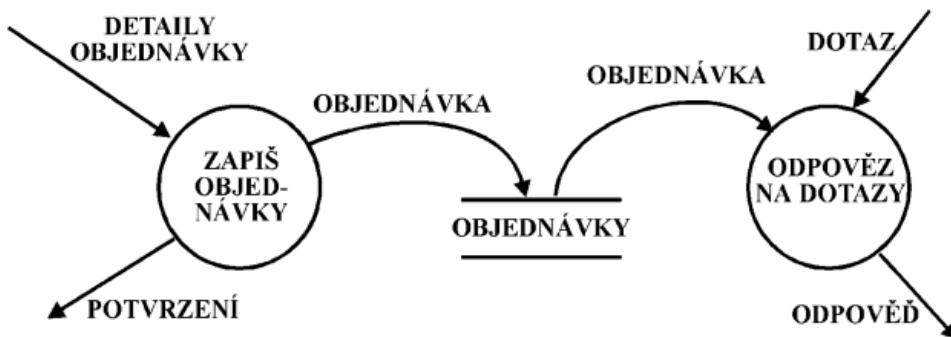


- Př.:
 - objednávka - prověřená objednávka
 - řetěz znaků - číslo v daném rozsahu
 - vyplněný formulář - dostatečně vyplněný formulář

- rastrový obraz - ekvalizovaný rastrový obraz

Paměť, esenciální paměť'

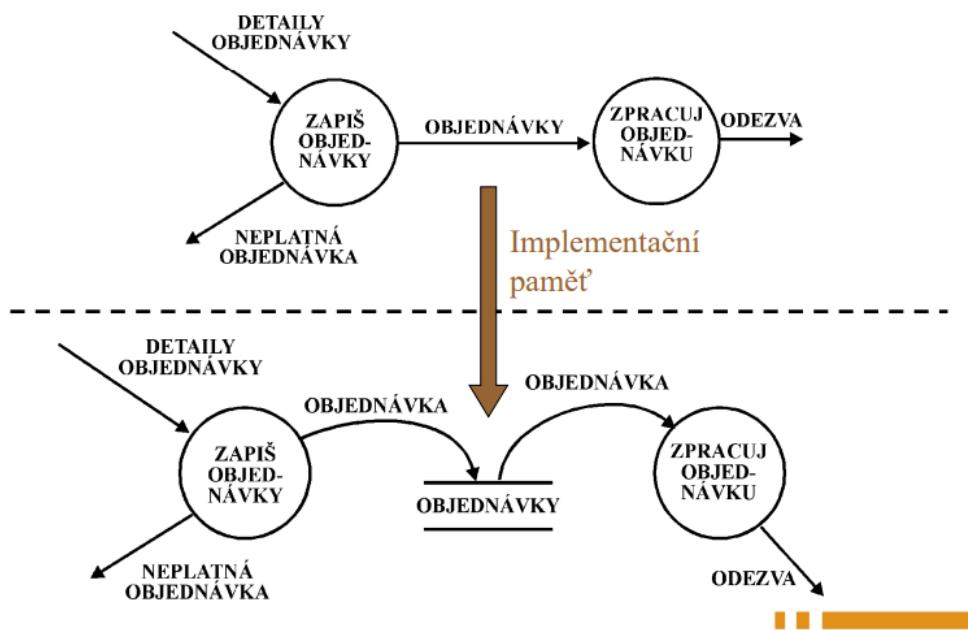
- Paměť modeluje kolekci dat v klidu. Jméno je voleno obvykle jako množné číslo jména, kterým jsou označeny pakety na tocích vedoucích do a z paměti.
- **Esenciální paměť'** - data předávaná mezi dvěma a více procesy pracujícími v různém čase.



Vlastnosti paměti

- pasivní částí systému
- data nejsou přenášena do/z paměti, pokud o to proces explicitně nepožádá
- čtení = nedestruktivní -> paměť se nemění
- tok vedoucí do paměti může mít význam zápisu, změny nebo zrušení. Může vyjadřovat následující situace:
 - Jeden nebo více nových paketů je přidáno do paměti; na konec nebo někam mezi existující pakety.
 - Jeden nebo více paketů je zrušeno, přemístěno z paměti.
 - Jeden nebo více paketů jsou přeměněny; to může znamenat změnu celého paketu nebo (častěji) části paketu, nebo obdobných částí více paketů.

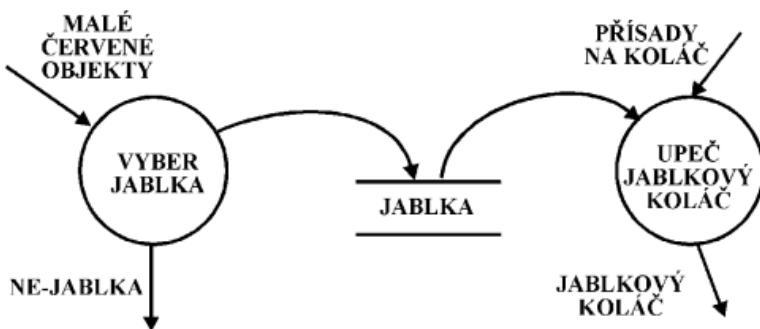
Implementační paměť'



Důvody pro implementační paměť'

- Oba procesy poběží na stejném počítači, ale není k dispozici dostatek paměti (nebo jiný HW prostředek), kterou by oba procesy použily ve stejném čase.
- Jeden nebo oba procesy budou řešeny na technickém vybavení, které není dostatečně spolehlivé. Paměť slouží jako prostředek pro uchování dosud zpracovaných částí dat a zvyšuje bezpečnost systému.
- Oba procesy budou implementovány různými programátory. Paměť pak slouží jako rozhraní mezi dvěma nezávisle řešenými subsystémy.
- Systémový analytik předvídí, že paměť bude v budoucnosti použita pro další, dosud nespecifikované funkce.

Paměti s nepojmenovanými toky

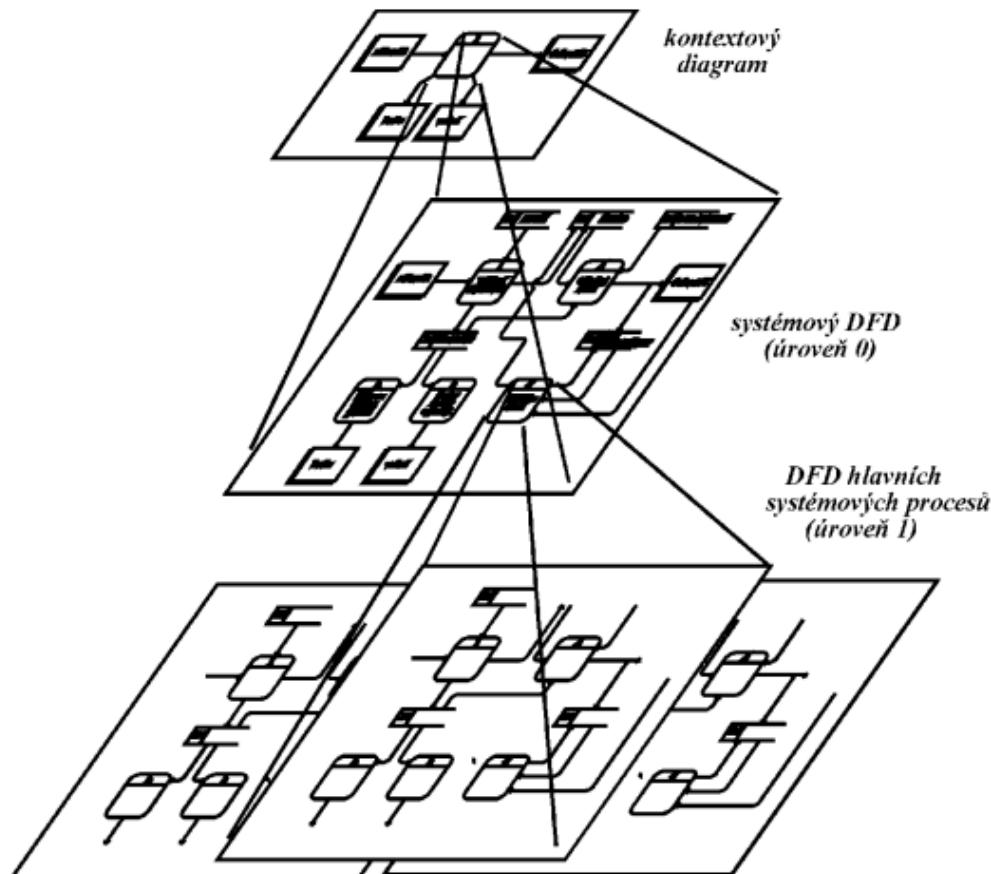


- Z analytického hlediska je nepodstatné, zda vyjadřujeme situaci, kdy:
 - jediný datový paket je čten z paměti
 - více než jeden datový paket je získán z paměti
 - je čtena pouze část paketu
 - jsou čteny části více než jednoho paketu

Terminátor

- reprezentuje externí entity, se kterými systém komunikuje
- vně modelovaného systému, toky, které je propojují s procesy (nebo pamětí) v systému, reprezentují rozhraní mezi systémem a vnějším světem.
- Žádný vztah mezi terminátory nebude ukázán na DFD.

Celková hierarchie DFD



- to co je o úrovně níže je vidět o úrovně výš
- pokud T1 vede do obousměrně do procesů v nižší úrovni může ale směřovat čtení do jiného sub-procesu/paměti než jeho modifikace (čtení třeba do sub1 a modify do sub3)

Příklad - Dekompozice zpracování žádosti

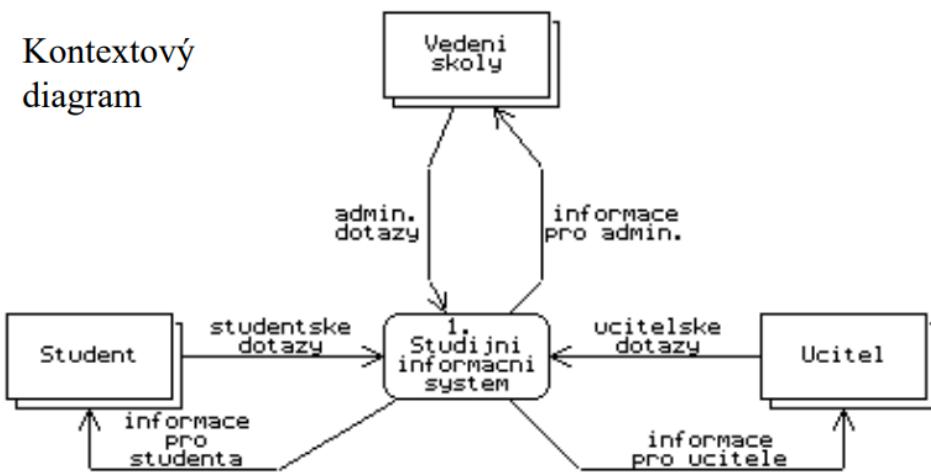
1. Vyřizování žádostí
 1. Vstupní kontrola
 2. Zpracování žádostí
 1. Určení komu patří
 2. Vlastní vyřizování
 3. Nestandardní zpracování
 3. Styk s občany

Doporučení při tvorbě DFD

- Volit **výstižná jména** pro procesy, datové toky, paměti a terminátory.

- **Systematicky číslovat** procesy.
- Překreslovať diagramy tak, aby byly co nejpestřejší. (terminátory vně a procesy vevnitř, nekřížit toky pokud možno)
- Vyhnut se příliš jednoduchým nebo příliš složitým DFD.
- Ověřit vnitřní i externí konzistenci DFD.

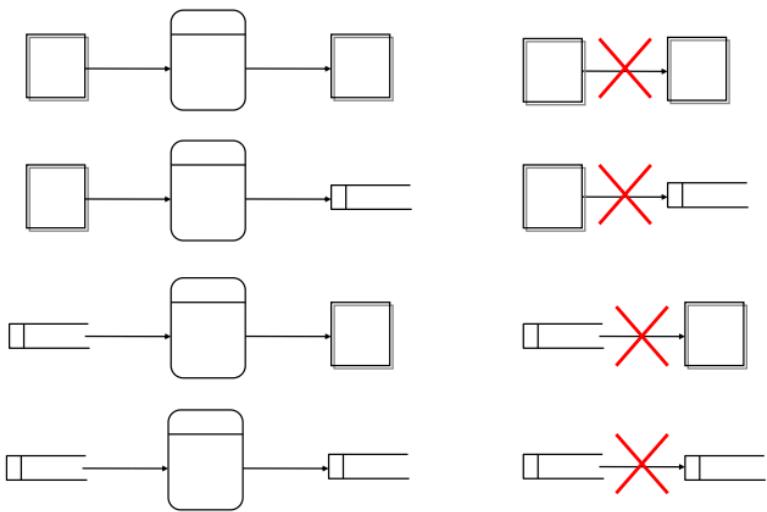
Příklad - Studijní IS



Vnitřní konzistence DFD (nedělat!)

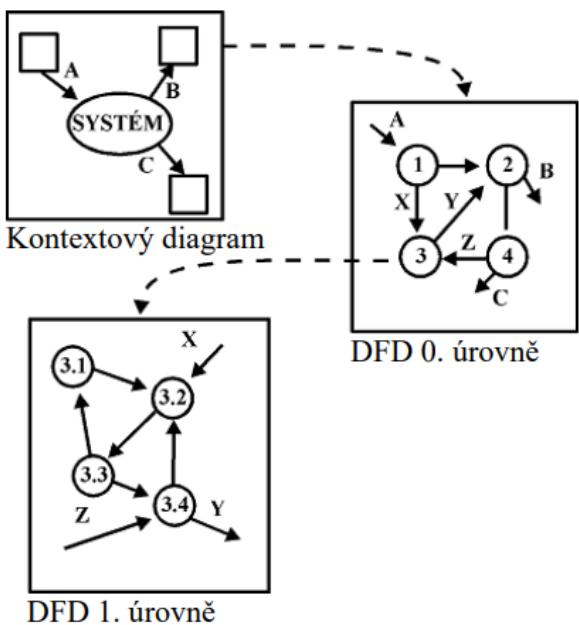
- "Černý díra" => procesy, které mají pouze vstupy a neprodukují žádná data (skryté rozhraní, ukrytý terminátor)
- "Bílý trpaslíci" => procesy, které mají pouze výstupy (skryté DB rozhraní, ukrytý terminátor)
- neoznačené toky a procesy
 - Podezření:
 - U nepojmenovaného datového toku byly seskupeny náhodně různé skupiny dat a je obtížné je pojmenovat.
 - U nepojmenovaného procesu není zřejmá funkce, rozhodnutí bylo "odloženo" na pozdější dobu.
- Paměť výhradně pro čtení či pro zápis. Má smysl pouze na rozhraní mezi systémem a terminátorem (většinou považováno za chybu)

Vztahy podrobené analýze



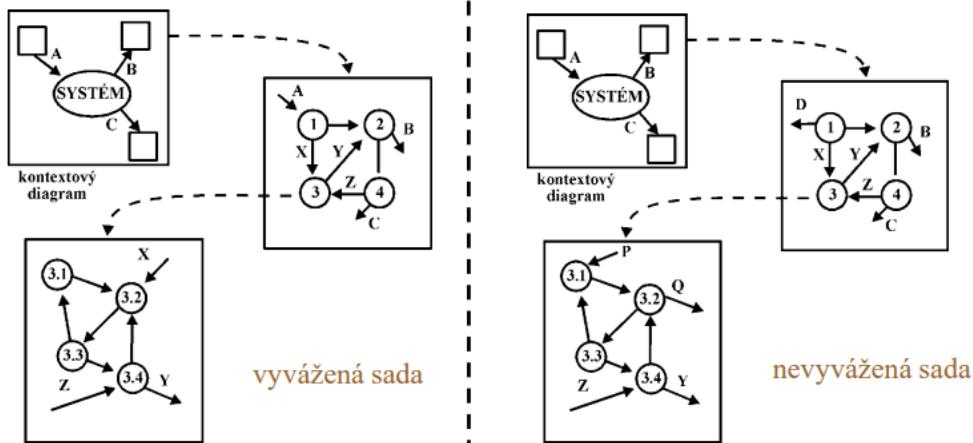
Hierarchie DFD

- DFD 0.úrovně:
 - pohled na hlavní funkce a na rozhraní mezi těmito funkcemi

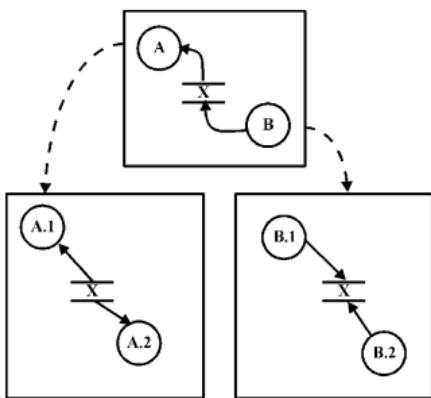


Pravidla tvorby víceúrovňových DFD

- číslování procesu se přenáší na nižší úrovně (n, n.1, n.2,...)
- Jméno procesu se stává jménem DFD na nižší úrovni
- Doporučeno DFD na A4 s průměrně 5-7 procesy a pamětí
- Systém střední velikosti bude mít cca 3-6 úrovní DFD
- Pro zajištění konzistence na jednotlivých úrovních DFD musí souhlasit vstupní a výstupní datové toky u procesů a jím odpovídajících DFD na nižších úrovních.



- Paměť zakreslíme poprvé na úrovni, kde je použita jako rozhraní mezi dvěma a více procesy. Na nižších úrovních ji zopakujeme u všech DFD, které obsahují dekompozici procesů spolupracujících s pamětí na vyšší úrovni.



Minispecifikace - Strukturovaná angličtina

IF <podmínka>,

THEN

<činnost pro platnou podmínu>

OTHERWISE

<činnost, když podmínka neplatí>

SELECT:

CASE 1 (podmínka 1): <činnost pro podmínu 1>

...

CASE n (podmínka n): <činnost pro podmínu n>

<úvodní fráze pro opakování>:

<opakování činnost>

<podmínka pro opakování>

<úvodní fráze pro opakování a podmínka opakování>:

<opakování činnost>

REPEAT UNTIL, WHILE DO, FOR EVERY DO

Příklad - Stipendia

Systematické číslování podle úrovní vnořených konstrukcí:

1. FOR EVERY student DO

 1.1 IF student složil předepsané zkoušky, THEN

 1.1.1 SELECT:

 CASE 1 (Stud. průměr =< 1.50):

 1.1.1.1 Nejvyšší stipendium.

 CASE 2 (Stud. průměr > 1.50):

 1.1.1.2 Žádné stipendium.

 OTHERWISE

 1.1.2 Žádné stipendium.

2. Další činnost

Příklad - Koktejly v letadlech

„Jestliže je let obsazen více jak z poloviny a průměrná cena letenky přesahuje 350\$, pak dáváme koktejly zdarma, pokud to není vnitrostátní let.

U vnitrostátních letů účtujeme všechny koktejly, když je podáváme.

Dáváme je, jen když je let více jak z poloviny obsazen.“

SELECT:

CASE 1 (Obsazeno z více než 50% a letenka stojí více než 350\$ a není vnitrostátní let): Podávej koktejly zdarma.

CASE 2 (Vnitrostátní let):

 Účtuj podané koktejly.

 IF Obsazeno z více než 50%, THEN Podávej koktejly.

 OTHERWISE Nic.

Specifikace pomocí *rozhodovací tabulky*

- jednodušší nalezení rozporů a nejasností

PODMÍNKY

1. vnitrostátní let
2. obsazen nad 50%
3. cena > 350\$

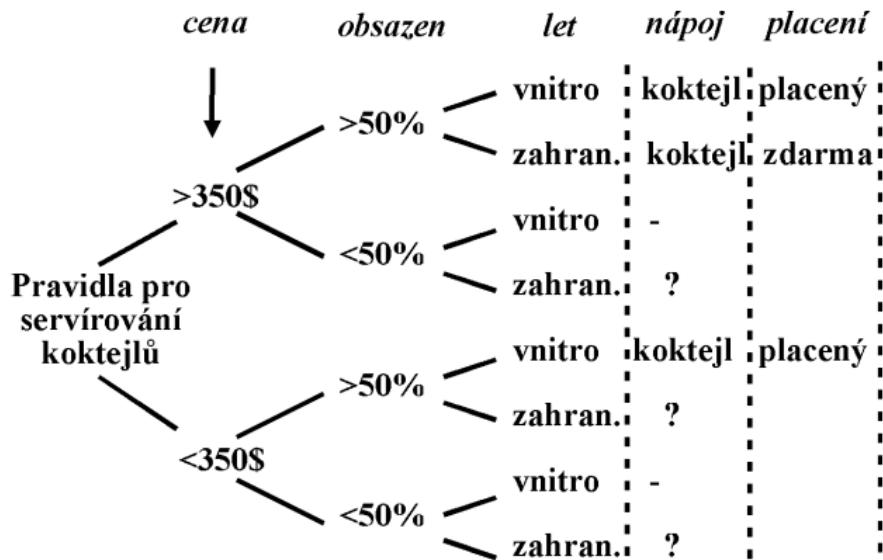
AKCE

1. servírovány koktejly
2. bezplatně

PRAVIDLA

1	2	3	4	5	6	7	8
A	N	A	N	A	N	A	N
A	A	N	N	A	A	N	N
A	A	A	A	N	N	N	N
A	A	N	?	A	?	N	?
N	A			N			

Specifikace pomocí rozhodovacího stromu



Minispecifikace - Úvodní a závěrečné podmínky

Doporučení: Nejprve popsát normální situace, poté připojit podmínky pro řešení chybových situací.

- Precondition 1 - Zákazník uvede číslo_účtu, které se shoduje s číslem účtu vedeným v paměti ÚČTY, jehož stavový_kód je nastaven na hodnotu „platný“.
- Postcondition 1 - Připravena faktura, na které jsou uvedeny číslo_účtu a prodejní_cena.
- Precondition 2 - Podmínka 1 není splněna (číslo_účtu nelze nalézt v paměti ÚČTY, nebo stavový_kód není „platný“).
- Postcondition 2 - Je sestavena chybová zpráva.

Cvičení a Úkoly

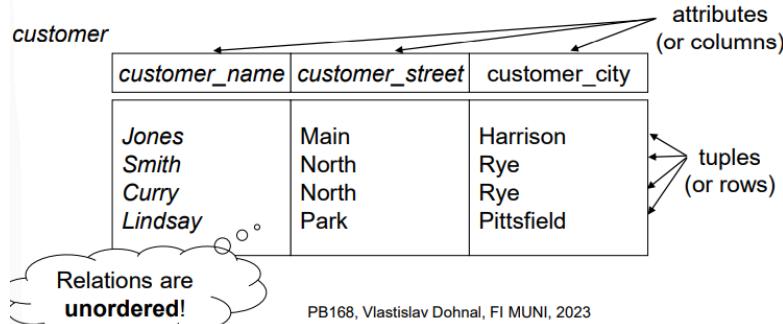
[Úkol_1_draw.excalidraw](#)

[Úkol_1_u_učitele_Ráčka](#)

Relational Model - Revision

- Given attribute names A1, A2... An
 - Schema R is an ordered tuple R=(A1, A2, ... An)
- Given attributes domains D1, D2, ... Dn corresponding to A1, A2, ... An
 - Relation r on the schema R is
 - $r \subseteq D_1 \times D_2 \times \dots \times D_n$

- $r = \text{set of } n\text{-tuples } (a_1, a_2, \dots, a_n) \text{ where } a_i \in D_i$



Query Lan. examples

- Given a relation
 - loan (loan_number, branch_name, amount)
- Query:
 - Return all loans having amount greater than \$ 1,200
- Relation algebra
 - $\sigma(amount > 1200) (Loan) \text{ or } \sigma\{amount \gt 1200\} (loan)$
- Tuple relational calculus

$$\{t | t \in \text{loan} \text{ and } t[\text{amount}] > 1200\}$$
- Domain relation calculus
 - $\{ \langle \text{loan_number}, \text{branch_name}, \text{amount} \rangle | \langle \text{loan_number}, \text{branch_name}, \text{amount} \rangle \in \text{loan} \wedge \text{amount} > 1200 \}$
- SQL
 - SELECT loan_number, branch_name, amount FROM loan WHERE amount > 1200
- Apache Pig Latin
 - A = LOAD 'loan' as (loan_number, branch_name, amount)
 - B = FILTER A BY amount > 1200
 - DUMP B;

Relational Algebra

- Procedural language
- Six operations
 - Select: σ (unární, jako fce)
 - Project: Π
 - Union: \cup (binární, napsaná mezi)
 - Set difference: $-$
 - Cartesian product: \times
 - Rename: ρ

- Principle:
 - An operation takes one or two relations as input and produce a new relation as a result.
 - So, another operation can be applied to this result.

Select

□ Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

□ $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Project

□ Relation r :

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\Pi_{A,C}(r)$

A	C	A	C
α	1	α	1
α	1	β	1
β	1	β	2
β	2		

Union

- Relations r, s :

r	A	B
	α	1
	α	2
	β	1

s	A	B
	α	2
	β	3

- $r \cup s$

	A	B
	α	1
	α	2
	β	1
	β	3

(Kdyby s bylo ABC tak se C nepřidá !) (musí být kompatibilní)

Set Difference

- Relations r, s :

r	A	B
	α	1
	α	2
	β	1

s	A	B
	α	2
	β	3

- $r - s$

	A	B
	α	1
	β	1

Cartesian-Product

- Relations r, s :

r	A	B
	α	1
	β	2

s	C	D	E
	α	10	a
	β	10	a
	β	20	b
	γ	10	b

- $r \times s$

	A	B	C	D	E
	α	1	α	10	a
	α	1	β	10	a
	α	1	β	20	b
	α	1	γ	10	b
	β	2	α	10	a
	β	2	β	10	a
	β	2	β	20	b
	β	2	γ	10	b

Composition of Operations

- Example: $\sigma_{A=C}(r \times s)$

- Relations:

r	A	B	s	C	D	E
	α	1		α	10	a
	β	2		β	10	a
				β	20	b
				γ	10	b

$r \times s$:		A	B	C	D	E
α	1	α	10	a		
α	1	β	10	a		
α	1	β	20	b		
α	1	γ	10	b		
β	2	α	10	a		
β	2	β	10	a		
β	2	β	20	b		
β	2	γ	10	b		

$\sigma_{A=C}(r \times s)$		A	B	C	D	E
α	1	α	10	a		
β	2	β	10	a		
β	2	β	20	b		

Rename and Cartesian-Product

- r (A,B,C)
- s (C, D, E)
- pro evaluaci $r \times s$, potřebujeme přejmenovat atributy

$$r \times p_{s(sC,D,E)}(s)$$

- plus existuje tečková notace

Example:

$$\sigma_{C=sC}(r \times p_{s(sC,D,E)}(s))$$

versus:

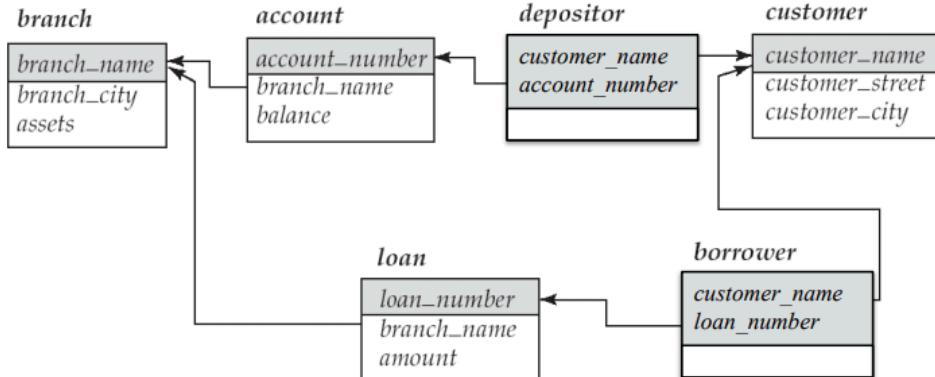
$$\sigma_{r.C=s.C}(r \times s)$$

Constant Relation

- A relation that is “manually” written as an expression.
- $\{('A-973', 'Perryridge', 1200)\}$
- Rename operation is usually applied to name the attributes.
- Pro vkládání do tabulky

Příklad Banking

- branch (branch_name, branch_city, assets)
- customer (customer_name, customer_street, customer_city)
- account (account_number, branch_name, balance)
- loan (loan_number, branch_name, amount)
- depositor (customer_name, account_number)
- borrower (customer_name, loan_number)



PB168, Vlastislav Dohnal, FI MUNI, 2023

Příklad Queries

- Vztahy:
 - loan (loan_number, branch_name, amount)
 - depositor (customer_name, account_number)
 - borrower (customer_name, loan_number)
- Find all loans of over 1200
 - $\sigma_{amount > 1200} (loan)$
- Find the loan number for each loan of an amount greater than \$1,200
 - $\Pi_{loan_number} (\sigma_{amount > 1200} (loan))$
- Find the names of all customers who have a loan, an account, or both, from the bank
 - $\Pi_{customer_name} (borrower) \cup \Pi_{customer_name} (depositor)$
- Vztahy:
 - customer (customer_name, customer_street, customer_city)
 - loan (loan_number, branch_name, amount)
 - borrower (customer_name, loan_number)
- Find the names of all customers who have a loan at the Perryridge branch

$$\Pi_{customer_name} (\sigma_{loan.loan_number = borrower.loan_number} ((\sigma_{branch_name = 'Perryridge'} (loan) \times borrower))$$

$$\Pi_{customer_name} (\sigma_{branch_name = 'Perryridge'} (\sigma_{borrower.loan_number = loan.loan_number} (borrower \times loan)))$$

- Vztahy:
 - loan (loan_number, branch_name, amount)
 - depositor (customer_name, account_number)
 - borrower (customer_name, loan_number)
- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

$$\begin{aligned} & \prod_{\text{customer_name}} (\sigma_{\text{branch_name} = \text{'Perryridge'}} \\ & \quad (\sigma_{\text{borrower.loan_number} = \text{loan.loan_number}} (\text{borrower} \times \text{loan}))) - \\ & \prod_{\text{customer_name}} (\text{depositor}) \end{aligned}$$

1.

- account (account_number, branch_name, balance)
- Find the largest account balance
 - Strategy:
 - Find those balances that are not the largest
 - Rename account relation as d so that we can compare each account balance with all others
 - Use set difference to find those account balances that were not found in the earlier step
 - query is:

$$\begin{aligned} & \prod_{\text{balance}} (\text{account}) - \prod_{\text{account.balance}} (\\ & \quad \sigma_{\text{account.balance} < d.\text{balance}} (\text{account} \times \rho_d (\text{account}))) \end{aligned}$$

Natural-Join

- přirozené spojení

Relations r, s:

r	A	B	C	D	s	B	D	E
α	1	α	a			1	a	α
β	2	γ	a			3	a	β
γ	4	β	b			1	a	γ
α	1	γ	a			2	b	δ
δ	2	β	b			3	b	ϵ

$r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Outer Join

- extention of the natural join operation

- avoids loss of information
- Missing rows in result of natural joins

<i>loan</i>			<i>borrower</i>	
<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>	<i>loan_number</i>
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	Hayes	L-155

loan \bowtie *borrower*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

Left Outer Join

<i>loan</i>			<i>borrower</i>	
<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>	<i>loan_number</i>
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	Hayes	L-155

loan \bowtie *borrower*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	null

Right Outer Join

<i>loan</i>			<i>borrower</i>	
<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>	<i>loan_number</i>
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	Hayes	L-155

loan \bowtie *borrower*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	null	null	Hayes

Full Outer Join

loan

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

borrower

<i>customer_name</i>	<i>loan_number</i>
Jones	L-170
Smith	L-230
Hayes	L-155

loan \bowtie *borrower*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	<i>null</i>
L-155	<i>null</i>	<i>null</i>	Hayes

Aggregate Functions

- avg
- min
- max
- sum
- count

□ Relation *r*

<i>A</i>	<i>B</i>	<i>C</i>
α	α	7
α	β	7
β	β	3
β	β	10

□ $G_{\text{sum}(C)}(r)$

sum
27

□ Relation *account*

<i>branch_name</i>	<i>account_number</i>	<i>balance</i>
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$G_{\text{sum}(\text{balance})}(\text{account})$

sum
3500

- Relation account grouped by *branch_name*:

<i>branch_name</i>	<i>account_number</i>	<i>balance</i>
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

branch_name \mathcal{G} *sum(balance)* (*account*)

<i>branch_name</i>	sum
Perryridge	1300
Brighton	1500
Redwood	700

- Result of aggregation does not have a fixed name
 - There is *no standard* on naming such result attributes.

- Rename operation is used to give it a name

$$\rho_x(\text{branch_name}, \text{sum_balance}) \left(\begin{array}{l} \text{branch_name} \mathcal{G} \text{sum(balance)} (\text{account}) \\ \end{array} \right)$$

<i>branch_name</i>	sum
Perryridge	1300
Brighton	1500
Redwood	700

<i>branch_name</i>	sum_balance
Perryridge	1300
Brighton	1500
Redwood	700

Modifikace tabulek

- Deletion
- Insertion
- Updating
- Značení:
 - $r \leftarrow E$
 - It can be used to store intermediate results

- $\text{temp} \leftarrow \Pi_{A, B, C}(r)$

Cvičení - After notes

2

- Uvažujte tři atributy jméno, příjmení a platová_třída.
- Mějme následující relační schémata
- R1=(jméno, příjmení, platová_třída)
- R2=(jméno, příjmení)

- R3=(příjmení, jméno, platová_třída)
- Liší se? Jak?

3**4**

- Mějme relaci

<u>kód</u>	<u>název</u>	<u>kredity</u>	<u>ukončení</u>
PB154	Základy DB systémů	3	zk
IB009	Paralelní výpočty	5	k
PV120	Informační právo	2	z
IA009	Paralelní výpočty	5	zk
PV094	Technické vybavení počítačů	3	zk

- Formulujte výraz v relační algebře pro dotazy:
 - Předměty s názvem "Paralelní výpočty"
 - Předměty s více jak dvěma kredity.

$$T_{kód}(G_{název='Par.Vy' \wedge kredity > 2}(předmět))$$

6

<u>kód</u>	<u>název</u>	<u>kredity</u>	<u>ukončení</u>
PB154	Základy DB systémů	3	zk
IB009	Paralelní výpočty	5	k
PV120	Informační právo	2	z
IA009	Paralelní výpočty	5	zk
PV094	Technické vybavení počítačů	3	zk

- Napište výraz v relační algebře, který vrací
 - kódy předmětů s názvem 'Paralelní výpočty';
 - názvy předmětů, které mají alespoň tři kredity;
 - kódy a ukončení předmětů, které mají ukončení vyšší než zápočet
 - možná ukončení jsou {'z', 'k', 'zk'}

$$T_{kód,ukončení}(G_{ukončení='k' \vee ukončení='zk'}(předmět))$$

11

- Mějme relace:
 - předmět (kód, název, kredity)
 - skupina (kód, číslo, kapacita)

- zápis (učo, kód)
- student (učo, jméno)
- Sestavte výrazy relační algebry, které vrací:
 - jména studentů, kteří mají zapsaný předmět 'PB168';

$$T_{jméno}(student \bowtie T_{učo}(G_{kód='PB168'}(Zápis)))$$

- čísla skupin předmětu s názvem 'UNIX', které mají kapacitu menší než 10;

$$T_{číslo}(G_{název='UNIX'}(G_{kapacita<10} \cap_{předmět.kód=skupina.kód} (skupina \times předmět)))$$

nebo místo x bude natural join $|><|$

- názvy a počty kreditů předmětů, které má zapsaný student 'Tomáš'.

$$G_{jméno='Tomáš'}(student \bowtie zápis \bowtie předmět)$$

- Napište výraz vracející učo studentů, kteří mají zapsané předměty 'PB154' a 'MA102' současně

$$T_{učo}(G_{kód='PB154'}(zápis)) \bowtie T_{učo}(G_{kód='MA102'}(zápis))$$

13

- Mějme relace:
 - předmět (kód, název, kredity)
 - skupina (kód, číslo, kapacita)
 - zápis (učo, kód)
 - student (učo, jméno)
- kódy a názvy předmětů, které nemá zapsaný žádný student;

$$T_{kód,název}(G_{učo \text{ is } NULL}(předmět \leftarrow \bowtie zápis))$$

- jména studentů, kteří nemají zapsaný žádný předmět; ???????

$$G_{count=0}(kód,název) Q_{count(učo)}(předmět \bowtie zápis))$$

- názvy všech předmětů, které mají alespoň dvě skupiny. ????????????

19? SQL

select kod, count()

cheat sheet

- σ (selekce)
- π (projekce)
- \times (kartézský součin)
- \cup (sjednocení)

- \neg (rozdíl)
- ρ (přejmenování)
- \cap (průnik) – lze odvodit
- \bowtie (přirozené spojení) – lze odvodit
- strukturovaný dotazovaný jazyk
- <https://is.muni.cz/auth/el/fi/podzim2025/PB168/um/slides05-sql.pdf>

Basic Query Structure

- **select** A1, A2, ..., An
- **from** r1, r2, ..., rn
- **where** C
- attribute (např.: jméno, adresa, rok...)
- relation (např.: tabulka reprezentující osoby)
- condition (např.: rok > 1970)

The select Clause

- Relation
 - *instruktor (id, name, debt_name, salary)*
- Find the names of all instructors:
 - select name
 - from instructor
- elimination of duplicates -> select distinct ... from ...
- select clause can contain arithmetic expressions
 - +, -, *, and /,
 - operating on constants or attributes of tuples
 - functions (nullif(), upper(), to_char(),...)

The where Clause

- např.: where dept_name = 'Comp. Sci.' and salary > 80'000
- např.: where salary / 12 > 6000

The from Clause

- Find the cartesian product instructor x teaches:
 - select * from instructor, teaches

instructor				teaches				
ID	name	dept_name	salary	ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2009
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
32343	El Said	History	60000	15151	MU-199	1	Spring	2010
...	22222	PHY-101	1	Fall	2009

instructor × teaches			Inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Physics	95000	10101	CS-101	1	Fall	2009			
10101	Srinivasan	Physics	95000	10101	CS-315	1	Spring	2010			
10101	Srinivasan	Physics	95000	10101	CS-347	1	Fall	2009			
10101	Srinivasan	Physics	95000	10101	FIN-201	1	Spring	2010			
10101	Srinivasan	Physics	95000	15151	MU-199	1	Spring	2010			
10101	Srinivasan	Physics	95000	22222	PHY-101	1	Fall	2009			
...
12121	Wu	Physics	95000	10101	CS-101	1	Fall	2009			
12121	Wu	Physics	95000	10101	CS-315	1	Spring	2010			
12121	Wu	Physics	95000	10101	CS-347	1	Fall	2009			
12121	Wu	Physics	95000	10101	FIN-201	1	Spring	2010			
12121	Wu	Physics	95000	15151	MU-199	1	Spring	2010			
12121	Wu	Physics	95000	22222	PHY-101	1	Fall	2009			
...

Joins

- Relations:
 - instructor (id, name, dept_name, salary)
 - course (course_id, title, dept_name)
 - section (course_id, sec_id, semestr, year)
 - teaches (id, course_id, sec_id)
- all instructors who teach courses, find names and course id of each course they teach
 - **select** name, course_id
 - **from** instructor, teaches
 - **where** instructor.id = teaches.id
- Find the course id, title, semester and year of each course offered by the “Comp. Sci.” department
 - **select** course.course_id, title, semester, year
 - **from** course, teaches, section
 - **where** course.course_id = teaches.course_id and teaches.course_id = section.course_id and teaches.sec_id = section.sec_id and dept_name = 'Comp. Sci.'

Natural Join

- "Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column"
- For relations:
 - instructor (id, name, dept_name, salary)
 - teaches (id, course_id, sec_id, semestr, year)
- **select * from** instructor **natural join** teaches;

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

Danger in natural join

- beware of unrelated attributes with same name which get equated incorrectly
- relations:
 - instructor (id, name, dept_name, salary)
 - course (course_id, title, dept_name)
 - section (course_id, sec_id, semester, year)
 - teaches (id, course_id, sec_id)
- List the names of instructors along with the titles of courses that they teach.
 - **Incorrect version** (equates course.dept_name with instructor.dept_name)
 - **select** name, title
 - **from** (instructor **natural join** teaches) **natural join** course;
 - **Correct version**
 - **select** name, title
 - **from** (instructor **natural join** teaches), course
 - **where** teaches.course_id= course.course_id
 - Another correct version
 - **select** name, title
 - **from** (instructor **natural join** teaches) **join** course **using**(course_id);

Ordering the Display of Tuples

- select name from instructor order by name
- desc / asc
 - order by name desc

Where Clause Predicates

- **between**
- Example:

- Find the names of all instructors with salary between \$90,000 and \$100,000 (that is, $\geq \$90,000$ and $\leq \$100,000$)
 - **select** name
 - **from** instructor
 - **where** salary **between** 90000 **and** 100000
- Tuple comparison
 - select name, course_id
 - from instructor, teaches
 - where (instructor.ID, dept_name) = (teaches.ID, 'Biology');

Null Values

- unknown value / does not exists
- The result of any arithmetic expression involving null is null
 - Example: 5 + null returns null
- The predicate is null can be used to check for null values.
 - Example: Find all instructors whose salary is null.
 - select name
 - from instructor
 - where salary **is null**

Duplicates

prez

Set Operations (union, intersect, except)

- Relation:
 - teaches (id, course_id, sec_id, semester, year)
- Find courses that ran in Fall 2009 or in Spring 2010
 - (select course_id from teaches where semester = 'Fall' and year = 2009)
 - union
 - (select course_id from teaches where semester = 'Spring' and year = 2010)
- Find courses that ran in Fall 2009 and in Spring 2010
 - (select course_id from teaches where semester = 'Fall' and year = 2009)
 - intersect
 - (select course_id from teaches where semester = 'Spring' and year = 2010)
- Find courses that ran in Fall 2009 but not in Spring 2010
 - (select course_id from teaches where semester = 'Fall' and year = 2009)
 - except
 - (select course_id from teaches where semester = 'Spring' and year = 2010)

Cvičení 4

2

- Příkaz SELECT jazyka SQL lze zapsat jako: SELECT A1 , ..., Ak FROM r1 , ..., rn WHERE podmínka
 - tento zápis převeďte do relační algebry
- Mějme relace:
 - předmět (kód, název, kredity)
 - zápis (učo, kód, ukončení)
 - student (učo, jméno, příjmení)
- V SQL uveďte dotazy, jejichž výsledkem jsou:
 - názvy předmětů, které mají alespoň tři kredity;
 - SELECT název FROM předmět WHERE kredity >= 3
 - názvy předmětů, které si studenti zapsali na zápočet;
 - SELECT název FROM předmět, zápis WHERE zápis.ukončení='z' and předmět.kód = zápis.kód
 - předměty, jejichž kód začíná 'PV';
 - SELECT all FROM předmět WHERE kód LIKE 'PV%'
 - předměty, které mají v názvu slovo 'angličtina';
 - SELECT all FROM předmět WHERE název LIKE '%angličtina%' or nazev LIKE '%Angličtina%'
 - jména a příjmení studentů uspořádaná podle abecedy.
 - SELECT all FROM student order by prijmení, jmeno
 - defaultně ASC

4

- Pro relaci
 - produkt

<u>kód</u>	<u>název</u>	<u>jedn_množství</u>	<u>cena_bez_DPH</u>
LCM01	ACER LCD 19"	1	10 800
RAM23	DDR2 1024MB (2x512)	2	4 980

- Vytvořte dotaz v SQL, který vrátí názvy produktů a jejich cenu včetně 20% daně.
 - SELECT nazev, cena_bez_dph * 1.21 AS end_price FROM product
- Operátor přejmenování AS: SELECT jedn_množství AS množství FROM produkt;
 -
- Upravte příklad tak, že bude vracet dva atributy pojmenované název a cena_s_DPH.
 - Samostatně: Tento dotaz převeďte do relační algebry.
 - X

5

- Uvažujte relaci test (A,B,C)
- Přejmenování relací v SQL:
 - SELECT A, B, C FROM test AS t WHERE t.A=17;
 - Zároveň lze přejmenovat i atributy:
 - SELECT nA, nB, nC FROM test AS t (nA,nB,nC) WHERE nA=17;
- Výsledkem operace SELECT je opět relace
 - tj. SELECT lze použít v části FROM:
 - SELECT p.název FROM předmět AS p,
 - (SELECT kód FROM zápis WHERE ukončení='z') AS z
 - WHERE p.kód=z.kód;

6

- Operace spojení
- Varianty INNER JOIN, [LEFT | RIGHT | FULL] OUTER JOIN
 - SELECT ... FROM r1 NATURAL INNER JOIN r2
 - SELECT ... FROM r1 INNER JOIN r2 ON podmínka
 - SELECT ... FROM r1 INNER JOIN r2 USING (seznam atributů)
- Mějme relace
 - předmět (kód, název, kredity)
 - skupina (kód, číslo, kapacita)
- Formulujte SQL dotazy:
- předmět  skupina
- pro každou seminární skupinu vypište její číslo, kapacitu a název odpovídajícího předmětu;
 - select cislo,kapacita,nazev from skupina full outer join predmet using (kod)
- vypište dvojice kód předmětu a číslo skupiny. ve výsledku se musí objevit kódy všech předmětů

7

Agregace pomocí GROUP BY

`SELECT G1, ..., Gn, F1(A1), ..., Fm(Am) FROM r GROUP BY G1, ..., Gn`

- ❑ Atributy G_i a A_j jsou ze schématu relace r
- ❑ F_i označují agregační funkce, atributy G_i jsou nepovinné
- ❑ Relační schéma výsledku je: (G₁, ..., G_n, F₁, ..., F_m)

Mějme relace předmět (kód, název, kredity)
 skupina (kód, číslo, kapacita)

Zapište výrazy v SQL, jejichž výsledkem je:

- ❑ celkový počet předmětů s kódem začínajícím 'MA';
- ❑ celková kapacita skupin předmětu 'PB154';
- ❑ kód předmětu a počet jeho seminárních skupin.
 - pro předměty mající alespoň jednu sem. skupinu
 - pro všechny předměty a uspořádejte sestupně podle počtu skupin
- select sum(kapacita) from skupina where kod='PB154'
- select kod, sum(kapacita) from skupina group by kod
- select nazev, string_agg(cislo::var_char, ','), sum(kapacita) from predmet left join skupina using kod group by kod
- select nazev, count(kapacita) from predmet left join skupina using (kod) group by kod
order by count(cislo)

9

- Vnořený SELECT v klauzuli WHERE:
 - Používaný s množinovými operátory
 - IN, NOT IN, EXISTS, > ANY (), = ANY (), SELECT ... FROM ... WHERE A IN (SELECT A FROM ...)
- Mějme relace
 - předmět (kód, název, kredity)
 - zápis (učo, kód, ukončení)
 - student (učo, jméno, příjmení)
- Formulujte SQL dotaz, který vybírá:
 - kódy předmětů, které nemá zapsaný žádný student;
 - select kod from predmet where kod not in (select kod from zapis)
 - předměty s následujícími kódy MA102, PB154, PV004;
 - select * from predmet where kod in ('MA102', 'PB154', 'PV004')
 - názvy předmětů s nejvíce kreditami;
 - select nazev from predmet where kredity=(select max(kredity) from predmet)
 - jména studentů, kteří mají zapsané alespoň dva předměty.
 -

Pokračování

Aggregate fce

- avg, min, max, sum, count
- Relations:
 - instructor (id, name, dept_name, salary)
 - teaches (id, course_id, sec_id, semestr, year)
- Find the average salary of instructors in the Computer Science department
 - select avg (salary)
 - from instructor
 - where dept_name= 'Comp. Sci.';

$\vartheta_{avg(salary)} \left(\sigma_{dept_name='Comp.Sci.'} (instructor) \right)$

- Find the total number of instructors who teach a course in the Spring 2010 semester
 - select count (distinct id)
 - from teaches
 - where semester = 'Spring' and year = 2010
- Find the average salary of instructors in each department
 - select dept_name, avg (salary)
 - from instructor
 - group by dept_name;

ID	name	dept_name	salary
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	avg
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

- Attributes in select clause outside of aggregate functions must appear in group by list
 - Erroneous query:
 - select dept_name, id, avg (salary)
 - from instructor
 - group by dept_name;

Aggregate fce - Having Clause

- Relations:
 - instructor (id, name, dept_name, salary)

- Find the names and average salaries of all departments whose average salary is greater than 42,000
 - select dept_name, avg (salary)
 - from instructor
 - group by dept_name
 - having avg (salary) > 42000;
- nebo
 - select * from (select dept_name, avg (salary) as a
 - from instructor
 - group by dept_name)
 - where a > 42000;

Null Values and Aggregates

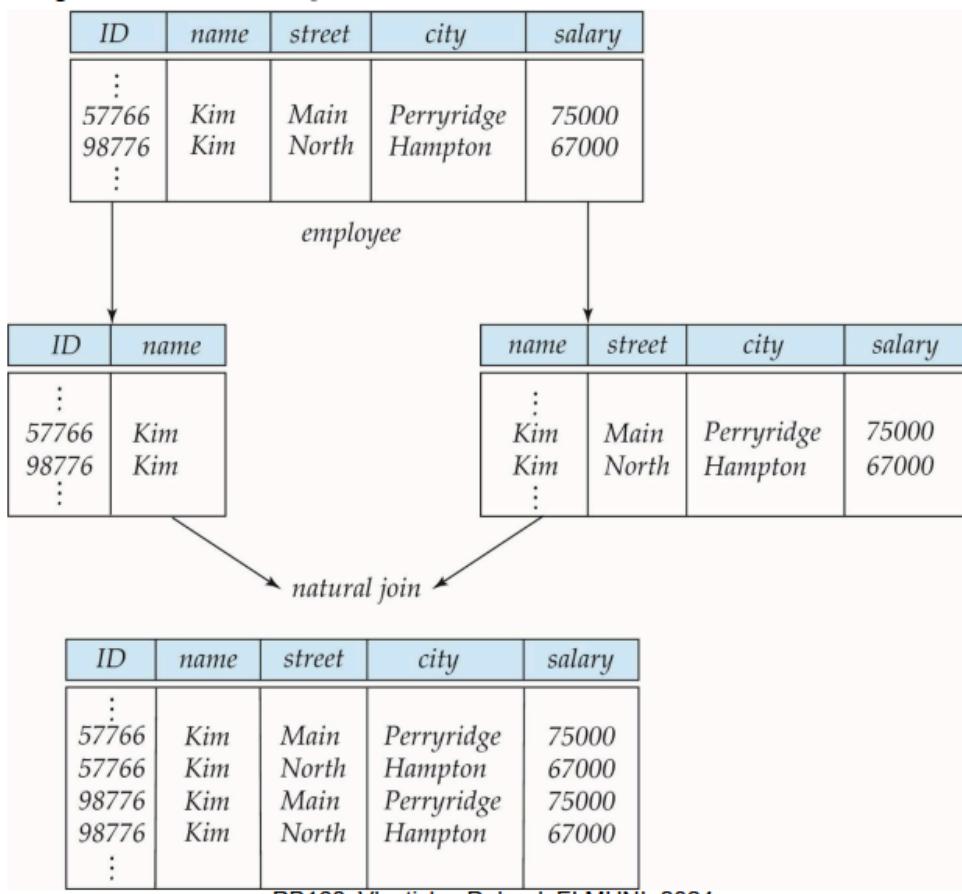
- Total all salaries:
 - select sum (salary) from instructor
- Above statement ignores null amounts
- Result is null if there is no non-null amount
- All aggregate operations except count(*) ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
 - count returns 0
 - all other aggregates return null

dpolnit str ~40 až 75

Combine smaller schemas

- Suppose we combine instructor(ID, name, salary, dept_name) and department(dept_name, building, budget) into inst_dept
- Suppose we had started with
 - inst_dept (ID, name, salary, dept_name, building, budget)
- How would we know to split up decompose into instructor and department tables?
 - Denote as a functional dependency:
 - dept_name → building, budget
- In inst_dept, because dept_name is not a candidate key, the building and budget of a department may have to be repeated.
 - This indicates the need to decompose inst_dept

Lossy decomposition



PB168, Vlastislav Dohnal, FI MUNI, 2024

example:

$$\begin{array}{l}
 R = (A, B, C) \text{ into } R_1 = (A, B) \quad R_2 = (B, C) \\
 \begin{array}{|c|c|c|} \hline A & B & C \\ \hline \alpha & 1 & A \\ \beta & 2 & B \\ \hline \end{array} \qquad \begin{array}{|c|c|} \hline A & B \\ \hline \alpha & 1 \\ \beta & 2 \\ \hline \end{array} \qquad \begin{array}{|c|c|} \hline B & C \\ \hline 1 & A \\ 2 & B \\ \hline \end{array} \\
 r \qquad \qquad \qquad \Pi_{A,B}(r) \qquad \qquad \qquad \Pi_{B,C}(r)
 \end{array}$$

$$r =? \Pi_{A,B}(r) \bowtie \Pi_{B,C}(r) \quad \begin{array}{|c|c|c|} \hline A & B & C \\ \hline \alpha & 1 & A \\ \beta & 2 & B \\ \hline \end{array}$$

Functional Dependencies

- $A \rightarrow B$
 - holds on R if and only if for any legal relation $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes A , they also agree on the attributes B . That is:

$$t_1[\alpha] = t_2[\alpha] \implies t_1[\beta] = t_2[\beta]$$

- read $A \rightarrow B$ as "B depends on A"

- example:

- Consider $r(A,B)$ with the following instance of r .

A	B
1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does NOT hold, but $B \rightarrow A$ does hold.

- K is a superkey for a relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if:

$K \rightarrow R$, and

for no $\alpha \subset K$, $\alpha \rightarrow R$

- Meaning: there is only one value for each value of K
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys.:
 - Consider the schema: inst_dept (ID, name, salary, dept_name, building, budget)
 - We expect these functional dependencies to hold:
 - $dept_name \rightarrow building$
 - $ID \rightarrow building$
 - $ID \rightarrow dept_name$
 - only one building for each department
 - but would not expect the following to hold:
 - $dept_name \rightarrow salary$
- We use functional dependencies to:
 - test relations to see if they are legal under a given set of funct. depend.
 - specify constraints on the set of legal relations

Closure of a Set of Functional Dependencies

- Given a set F of functional dependencies, there are certain other functional dependencies that are logically implied by F.
 - If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of all functional dependencies logically implied by F is the closure of F.
 - We denote the closure of F by F^+
 - F^+ is a superset of F.

Boyce-Codd Normal Form

- A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form $A \rightarrow B$
- where $A \leq R$ and $B \leq R$, at least one of the following holds:
 - $A \rightarrow B$ is trivial ($B \leq A$)

- A is a superkey for R ($A \rightarrow R$)
 - example schema not in BCNF
 - `instr_dept (ID, name, salary, dept_name, building, budget)`
 - because $\text{dept_name} \rightarrow \text{building, budget}$ holds on `instr_dept`, but dept_name is not a superkey.

Decomposing a Schema into BCNF

Suppose we have a schema R

A non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF, so we decompose R into:

- $R_1 = (\alpha \cup \beta)$
 - $R_2 = (R - (\beta - \alpha))$

In our example, *dept_name* → *building*, *budget*

- $\alpha = \text{dept_name}$
 - $\beta = \text{building, budget}$ $\text{instr_dept}(ID, name, salary, \text{dept_name}, \text{building}, \text{budget})$

and inst_dept is replaced by

 - $R_1 = (\alpha \cup \beta) = (\text{dept_name}, \text{building}, \text{budget})$
 - $R_2 = (R - (\beta - \alpha)) = (ID, name, salary, \text{dept_name})$

Third Normal Form

- A relation schema R is in third normal form (3NF) if for all: $A \rightarrow B$ in F^+
 - where $A \leq R$ and $B \leq R$, at least one of the following holds:
 - $A \rightarrow B$ is trivial (B in A)
 - A is a superkey for R
 - Each attribute A in $B - A$ is contained in a candidate key for R
 - each attribute may be in a different candidate key
 - If a relation is in BCNF, it is in 3NF Since in BCNF one of the first two conditions above must hold
 - Third condition is the minimal relaxation of BCNF to ensure dependency preservation (will see why later).

Closure of a Set of Functional Dependencies

- Given a set F of functional dependencies, there are certain other functional dependencies that are logically implied by F.
 - If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
 - The set of all functional dependencies logically implied by F is the closure of F.
 - We denote the closure of F by F^+ .
 - We can find F^+ , the closure of F, by repeatedly applying Armstrong's Axioms:
 - if $B \subseteq A$, then $A \rightarrow B$ (reflexivity)
 - if $A \rightarrow B$, then $\gamma A \rightarrow \gamma B$ (augmentation)

- if $A \rightarrow B$, and $B \rightarrow y$, then $A \rightarrow y$ (transitivity)

example:

- $R = (A, B, C, G, H, I)$
 $F = \{ A \rightarrow B$
 $\quad A \rightarrow C$
 $\quad CG \rightarrow H$
 $\quad CG \rightarrow I$
 $\quad B \rightarrow H\}$
- some members of F^+
 - $A \rightarrow H$
 - by transitivity from $A \rightarrow B$ and $B \rightarrow H$
 - $AG \rightarrow I$
 - by augmenting $A \rightarrow C$ with G , to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
 - $CG \rightarrow HI$
 - by augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI$, and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity

Shrnutí

BCND \leq 3NF \leq 2NF \leq 1NF

Cvičení Funkční závislosti

Tvorba fun. závislosti a jejich platnost

- Uvažujte evidenci zapsaných předmětů s následujícími požadavky
 - Student má jméno a své jedinečné UČO;
 - učo \rightarrow jméno
 - Předmět má název, počet kreditů a svůj jedinečný kód;
 - kód \rightarrow název, počet kreditů
 - Student si zapisuje více předmětů;
 - učo, kód \rightarrow ukončení (z vytvoření tabulky učo, kód)
 - Předmět si může zapsat více studentů;
 - učo, kód \rightarrow ukončení (z vytvoření tabulky učo, kód)
 - Student má pro zapsaný předmět zvolené určité ukončení.
 - učo, kód \rightarrow ukončení
- Formulujte funkční závislosti

Třetí normální forma

Definice 3NF: Relační schéma R je v 3NF, pokud je v 2NF a pro každý atribut A z R platí některé z následujících tvrzení:

- A je součástí některého kandidátního klíče, nebo
- A není tranzitivně závislý na žádném superklíči.
 - A je tranzitivně závislý na α , pokud $\alpha \rightarrow \beta$, $\beta \not\rightarrow \alpha$, $\beta \rightarrow A$ a A není součástí β ani α .

Rozhodněte, zda jsou následující relace v 3NF

závěrečná_práce

učo	program	obor	název_práce	vedoucí	vedoucí_katedra
12345	Inf.	Mat. inf.	Teorie sčítání	Brázdil	KTP
67890	Apl. inf.	Bioinf.	Život brouka	Lexa	KTP

$$\begin{aligned} F_1 = \{ & \text{ učo, program} \rightarrow \text{obor, vedoucí, název_práce} \\ & \text{obor} \rightarrow \text{program} \\ & \text{vedoucí} \rightarrow \text{vedoucí_katedra} \} \end{aligned}$$

student

učo	jméno	příjmení
12345	Jan	Novák
67890	Lenka	Šťastná

garant

obor	garant_oboru
Mat. inf.	Hliněný
Bioinf.	Brim

$$F_2 = \{ \text{ učo} \rightarrow \text{jméno, příjmení } \}$$

$$F_3 = \{ \text{ obor} \rightarrow \text{garant_oboru} \}$$

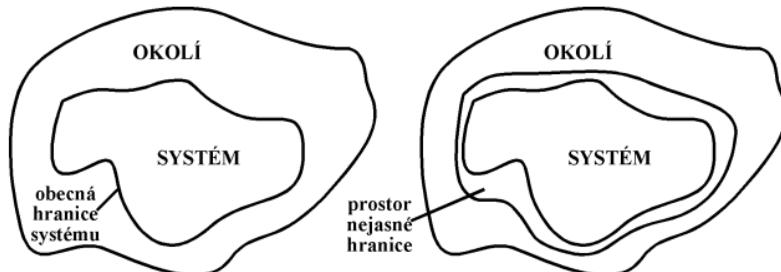
- vedoucí \rightarrow vedoucí_katedra VADÍ

Hranice mezi systémem a okolím

- Rozhraní mezi sys a okolním světem popisuje **model okolí**.
- Vnitřní části sys popisuje **model chování**.
- Úkol analýzy - stanovit, co je a co není součástí systému.
 - Yourdonova Moderní strukturovaná analýza, 1989 (YMSA):
 - Esenciální model = model okolí + model chování

Volba hranice systému

- Analytik může ovlivnit volbu hranice v prostoru nejasné hranice.



- Nebezpečí: Malý rozsah projektu identifikuje pouze symptomy, velký rozsah projektu je obtížně zvládnutelný, nebo neuspěje vůbec.

Definice okolí - Nástroje

- Účel systému
 - krátký, stručný textový dokument
 - pro pracovníky vrcholového řízení
 - Vyjadřuje strategické cíle, kterých by mělo být dosaženo po realizaci a nasazení systému.
- Kontextový diagram
 - zvláštním případem DFD
 - Obsahuje jediný proces, který reprezentuje celý systém. Zdůrazňuje tyto rysy systému:
 - terminátory - lidé a systémy z okolí, s nimiž systém komunikuje
 - data přijímaná z vnějšího světa, která mají být nějak zpracována
 - data produkovaná, které systém zasílá do vnějšího světa
 - datové paměti sdílené systémem a terminátory
 - hranice mezi systémem a ostatním světem
- Seznam událostí
 - textový výčet stimulů
 - objevují se ve vnějším světě a na něž musí systém odpovědět.

Příklad kontextového diagramu



Události

- = Seznam událostí
 - textový výčet stimulů, které se objevují ve vnějším světě a na něž musí systém odpovědět.
 - Každá událost je označena F, T nebo C.
- F (Flow)
 - tokově orientovaná událost
 - sdružená s datovým tokem (příjem jednoho nebo několika datových paketů)

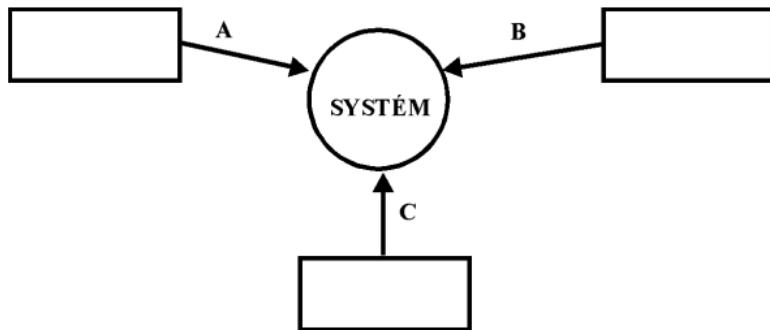
- T (Temporal)
 - časová událost
 - nastává v nějakém významném časovém bodě (absolutní nebo relativní čas)
- C (Control)
 - řídící událost, povel, signál
 - je sdružena s vnějším řídícím tokem, povelem

Příklady FTC

1. U1: Zákazník vystavuje objednávku. (F)
2. U2: Zákazník ruší objednávku. (F)
3. U3: Vedení požaduje zprávu o prodeji každé ráno. (T)
4. U4: Pokyn k vyhodnocení stavu a vývoje prodeje. (C)

Vztahy mezi datovými událostmi

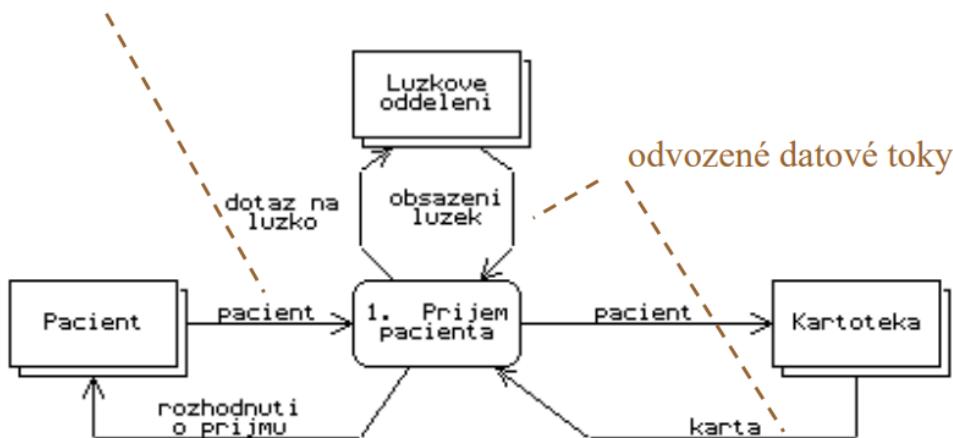
- Ne všechny vnější datové toky jsou sdruženy s F-událostmi.
- A - tok spojený s událostí
- B, C - toky explicitně vyžádané pro řešení události



•

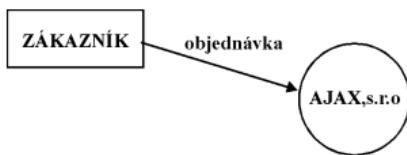
Příklad DFD - Ambulantní příjem

- F-událost: Příchod pacienta



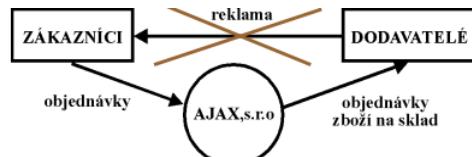
Sestavení modelu okolí

- Komunikace znázorněná na kontextovém diagramu:



a) přímá komunikace terminátor - proces

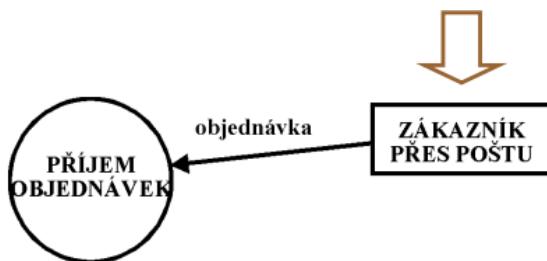
b) komunikace přes externí paměť



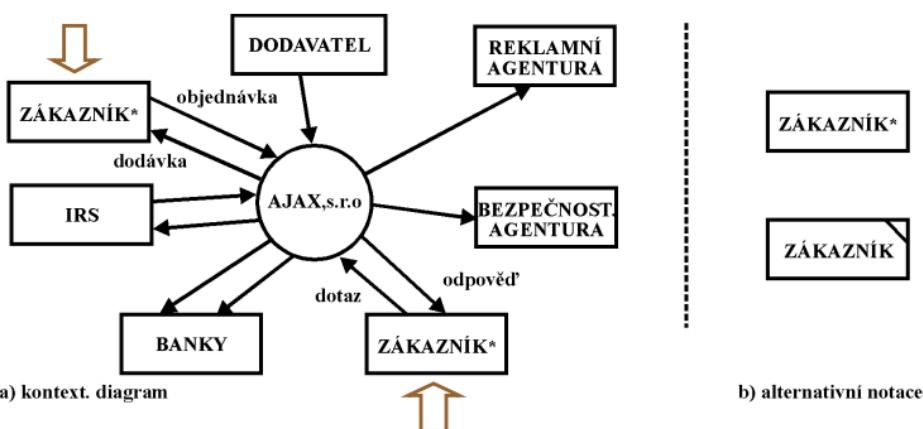
- c) komunikace terminátor - terminátor: chybný kontextový diagram

Poznámky k terminátorům

- Terminátory s velkým počtem vstupů/výstupů – kvůli přehlednosti zakreslíme vícekrát.
- Terminátor je individuální osoba (subjekt) - je třeba zjistit roli vzhledem k systému a pojmenovat terminátor podle této role.
- Rozlišíme mezi zdroji dat a nosiči (nositeli) dat. Přednost dáme pojmenování podle zdroje dat, nosič dat je „technologický prvek“, při realizaci může být použito odlišné „médium“.
- Kompromisní pojmenování terminátoru: „zdroj i nosič dat“ v jednom jméně.

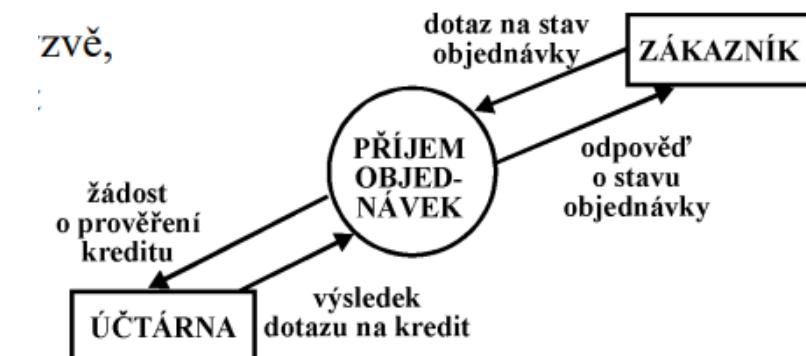


Duplikované terminátorы

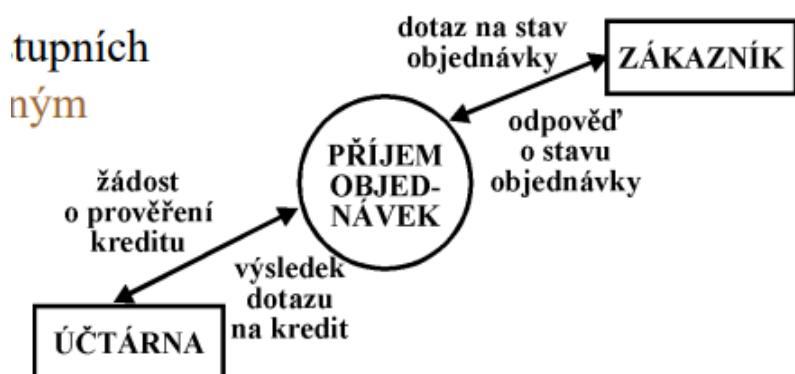


Dialogové toky

- Pokud systém posílá data až po výzvě, je tato výzva esenciální a musí být zakreslena na kontext.



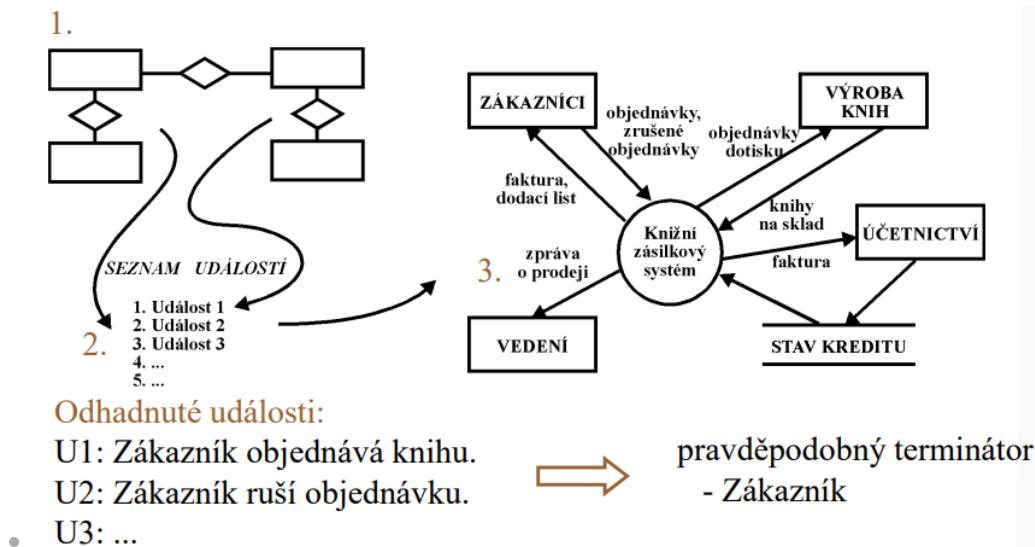
- Dvojice odpovídajících vstup-výstupních toků lze někdy nahradit obousměrným tokem.



Pořadí tvorby modelů okolí

- Varianty možných postupů:
 - Na základě informací od uživatelů lze sestavit kontextový diagram. Zkoumáním terminátorů a toků odhadneme události. Prověříme pomocí dalších modelů (procesní, datový m.).
 - Začneme s datovým modelem - ERD. Nalezneme esenciální objekty a jejich vztahy. Pro každou entitu hledáme, jaké vnější události vedou k jejímu použití, změně atd. Sestavíme předběžný seznam událostí. Pomocí něho vytvoříme kontextový diagram.
- Kontextový diagram => Seznam událostí => ...
 - ERD => Seznam událostí => Kontextový diagram => ...

Identifikace událostí pomocí ERD



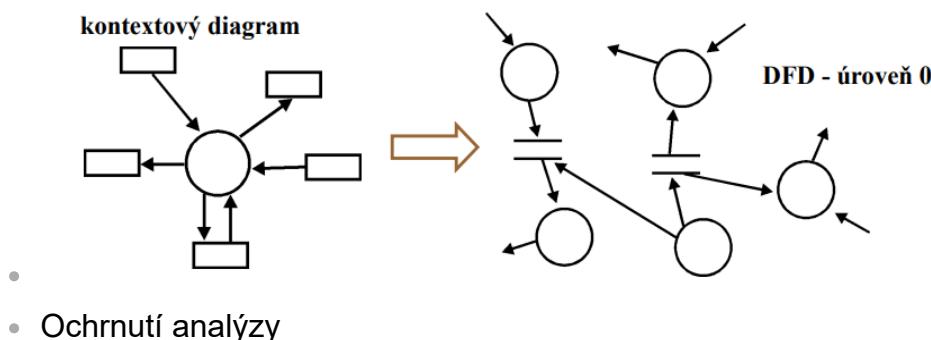
Dokončení modelu okolí

- Model okolí je nutné prověřit:
 - Každý vstupní tok na kontext. diagramu je nezbytný pro rozpoznání události, nebo pro vytvoření odezvy na událost, nebo obojí současně.
 - Každý výstupní tok je odezvou na událost.
 - Každá nečasová událost ze seznamu událostí by měla mít vstup podle něhož systém detekuje její výskyt.
 - Každá událost musí produkovat okamžitý výstup jako odezvu, nebo by měla uložit data pro pozdější výstup, nebo by měla změnit stav systému.

Vytvoření prvotního modelu chování

- Výchozí podklady - model okolí systému, který obsahuje dokumenty:
 - kontextový diagram,
 - seznam událostí,
 - dokument o účelu systému,
 - zahájena tvorba datového slovníku
- Tvoříme model chování systému, tj. dekompozici systému na jednotlivé procesy, toky a paměti uspořádané v sadě DFD a doplněné o příslušné mini specifikace a definici datových komponent.

Kritika postupu „shora dolů“



- Ochrnutí analýzy

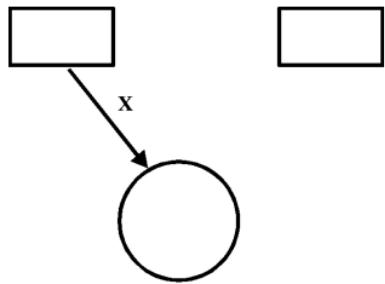
- Analytik nedokáže nalézt dekompozici na hlavní subsystémy.
- Jev 6 analytiků
 - Systém obsahuje právě tolik subsystémů, kolik je analytiků.
- Náhodné fyzické členění
 - Subsystémy tvořeny podle existujícího fyzického a organizačního členění.

Princip tvorby modelu chování

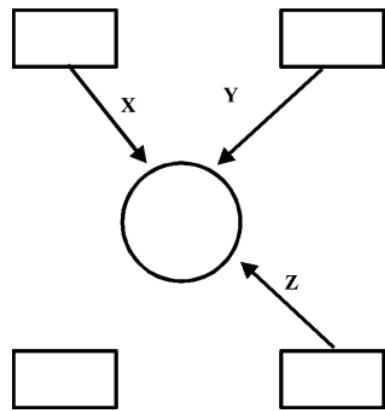
- Pro každou odezvu na událost ze seznamu událostí zakreslíme do prvotního DFD jeden proces.
- Proces pojmenujeme podle očekávané odezvy na tuto událost.
- Zakreslíme datové paměti, které modelují data nezbytná pro zpracování asynchronně probíhajících událostí.
- Doplníme odpovídající vstupní a výstupní toky.
- Vytvořený DFD ověříme proti kontextovému diagramu.

Události a průvodní toky

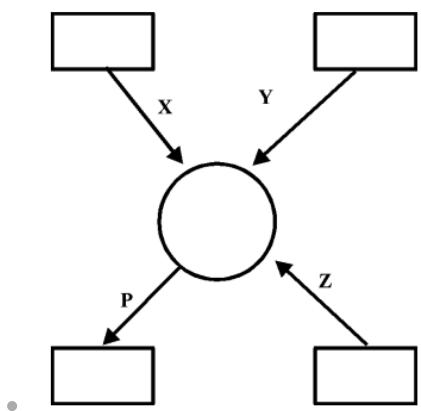
1. Výskyt události (datový tok X)



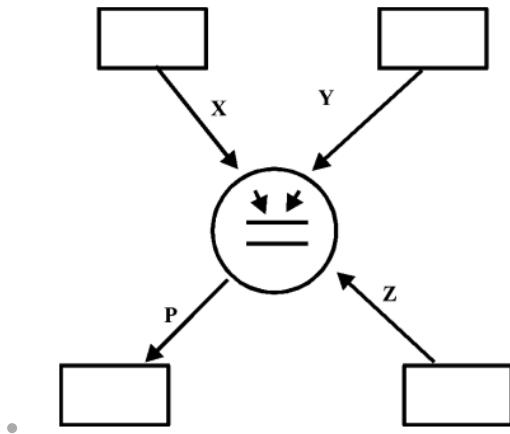
2. pro řešení události požadovány další údaje z okolí systému (Y,Z)



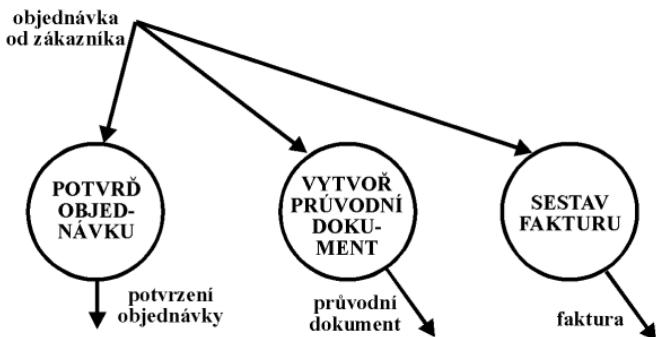
3. odezva na událost ve formě výstupního toku (P)



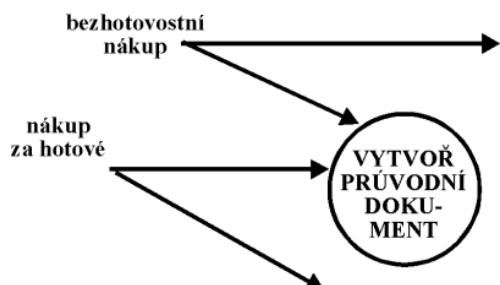
- 4. odezva na událost ve formě uložený



Události s více odezvami



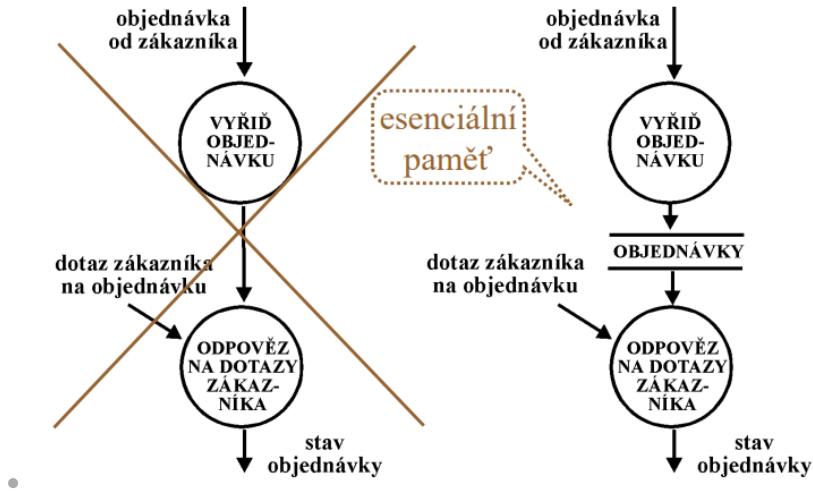
- Všechny odezvy (generující procesy) používají stejný vstupní tok.
- Všechny odezvy jsou vzájemně nezávislé.



- Odezva produkovaná procesem musí být identická pro různé události.
- Vstupní a výstupní data jsou identická pro různé události.

Propojení odezv na události

- Odezva na událost může vyžadovat data vytvořená jinou událostí.
- V esenciálním modelu pracuje každý proces nekonečně rychle.
 - Každý datový tok představuje komunikační cestu s nulovým dopravním zpožděním.
- Události, které na sobě závisí, můžeme synchronizovat jedině pomocí paměti.
 - Paměť je esenciální, je vynucena časováním vně systému.

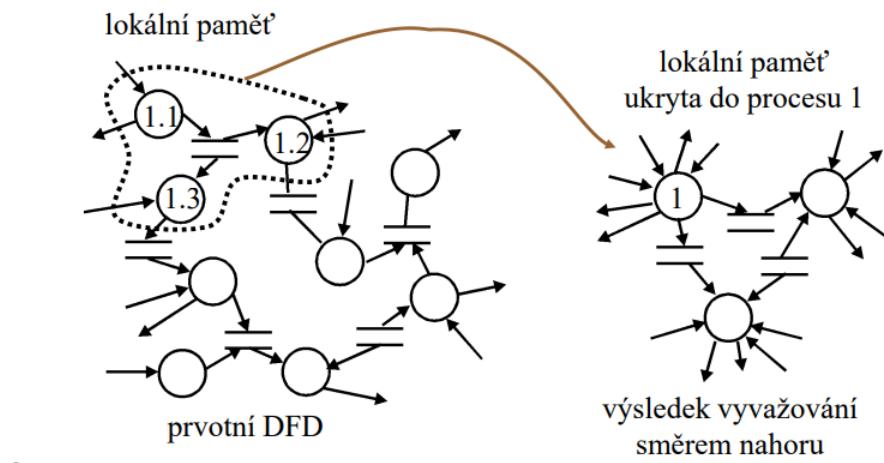


Prvotní model chování

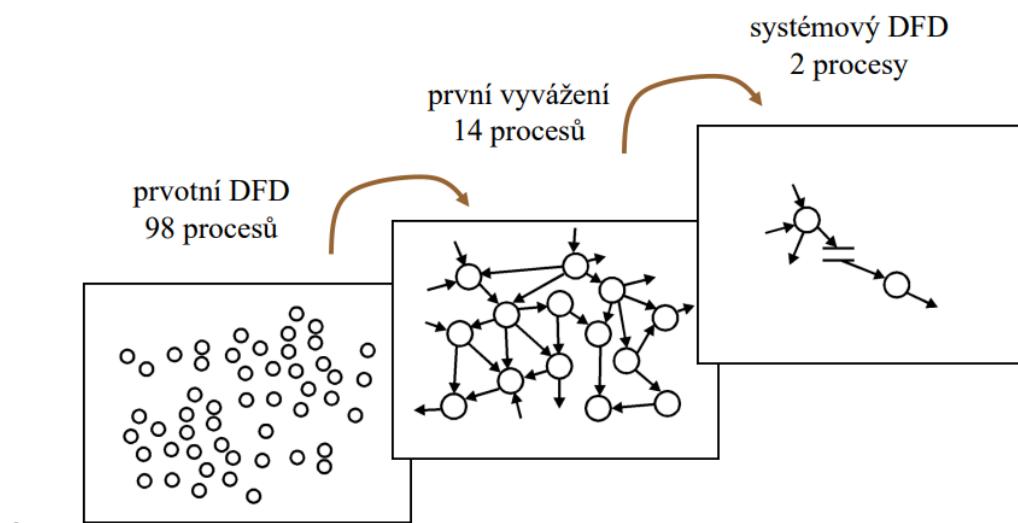
- prvotní DFD <=> prvotní ERD
- paměti <= => entity
- Kontrola:
 - Zpracovává prvotní DFD každou událost?
 - Jsou zakresleny všechny potřebné vstupy a výstupy pro každou u
 - Jsou zakreslena nezbytná propojení mezi událostmi?
- Prvotní model chování nekonzultujeme s uživatelem ! Model není uspořádaný tak, aby mohl být pochopen jako celek.

Vyvažování směrem nahoru

- Hledáme seskupení vzájemně souvisejících procesů do agregovaného procesu na diagramu vyšší úrovni.
1. Seskupení procesů má zahrnout blízké, obdobné odpovědi (obdobně pojmenované procesy).
 2. Paměti zakrýváme, pokud paměť používá pouze skupina procesů na nižší úrovni (žádný jiný proces vně této skupiny paměť nepoužívá).

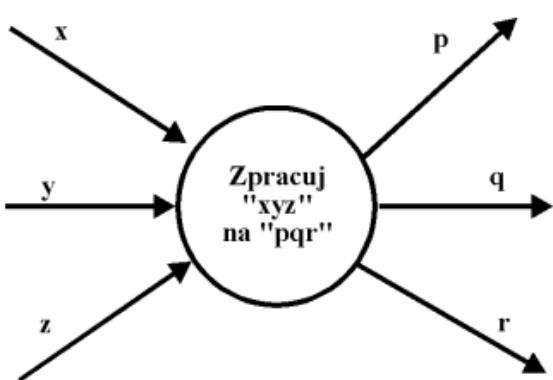


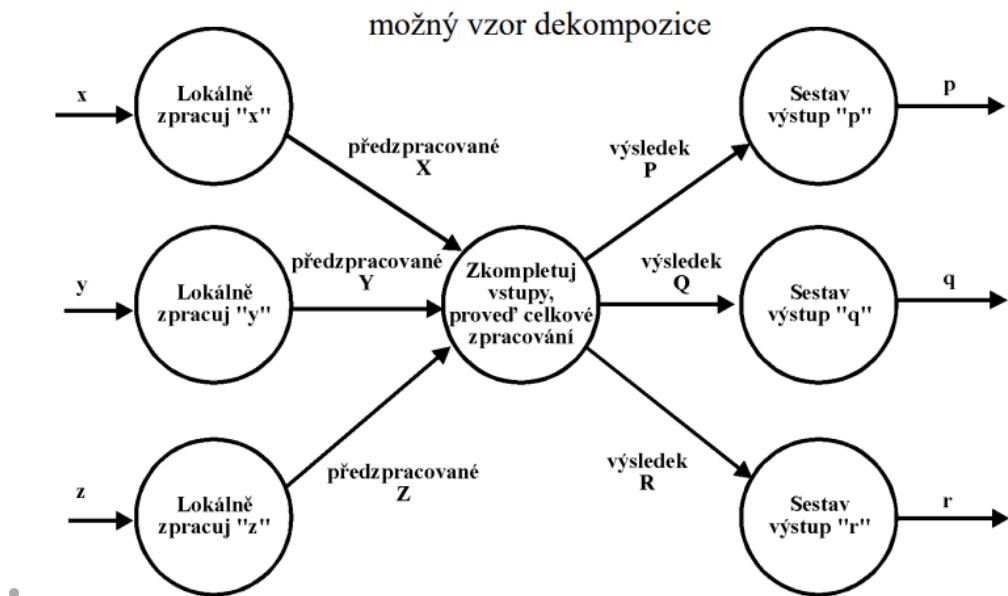
Vícenásobné vyvažování směrem nahoru



Vyvažování směrem dolů

- Čistá funkční dekompozice
 - u procesu se složitou funkcí jsou zřetelné dílčí funkce.
 - Tyto funkce vyjádříme jako procesy na nižší úrovni DFD.
- Funkční dekompozice nezřetelná
 - Pokusíme se odvodit dekompozici na základě vstupních a výstupních toků.
- Data určují směr dekompozice.





Dokončení modelu chování

- Dokončení ERD a datového modelu jako celku:
 - ERD vyvýjen obdobně jako DFD a souběžně s DFD.
 - první ERD
 - přiřazování atributů k entitám
 - identifikace nových nebo přebytečných entit
 - křížová kontrola proti DFD
- Dokončení stavového modelu:
 - Stavové diagramy plní úlohu minispecifikace řídících procesů.
 - definice všech možných stavů
 - dosažitelnost všech stavů
 - cesta z nekoncových stavů
 - reakce na všechny možné podmínky v každém stavu

Stavové modely, DFD s řízením, vyvažování modelů

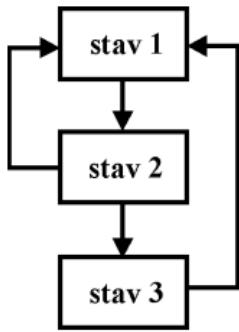
Stavy systému

- Stav:
 - Množina okolností nebo atributů charakterizujících osobu nebo věc v daném čase; způsob nebo forma existence; podmínka.
- Název obvykle označuje nějakou formu čekání systému
 - v názvu je jmenována skutečnost, která má nastat

Změny stavu

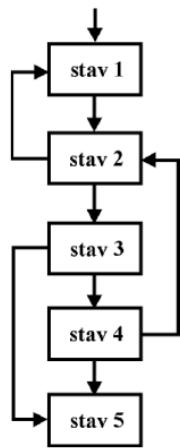
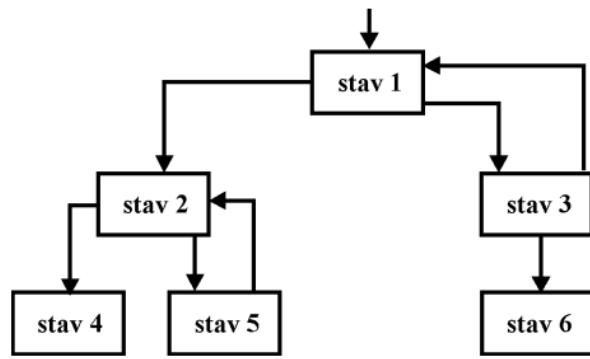
- U systému, který má striktně vymezená pravidla chování nelze přejít z jednoho stavu do libovolného dalšího stavu náhodně; rozlišujeme, které změny jsou smysluplné a

platné.



Počáteční a koncový stav

- Systém může mít jen jeden iniciální stav, ale více různých koncových stavů.

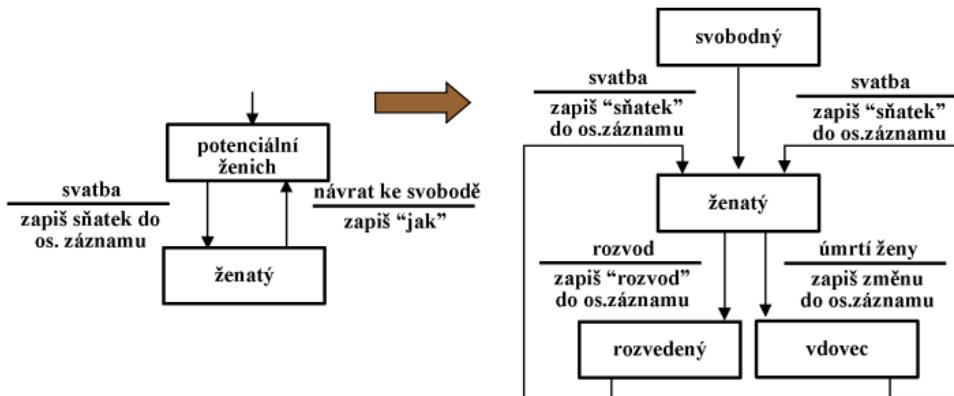


Podmínka a akce přechodu

- Změna stavu u esenciálního modelu probíhá okamžitě
- Popis přechodu**
 - podmínka**, při jejímž splnění nastává změna stavu
 - akce**, kterou systém provede při změně stavu
- Podmínka je nějaká událost ve vnějším okolí, kterou je systém schopen detektovat (signál, přerušení, příchod datového paketu).
- Systém provede přechod ze stavu:
 - „čekání na X“ do stavu „čekání na Y“
 - „činnost X“ a zahájí provádění „činnosti Y“

- Akce jsou buď odezvy do vnějšího okolí, nebo výpočty, jejichž výsledky si systém pamatuje, aby mohl později reagovat na další události.

Příklad STD - „Ze života“



- Na STD něco chybí ...

Tvorba stavového diagramu

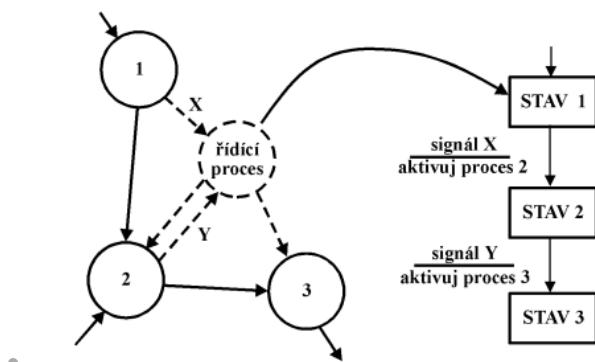
1. Identifikovat všechny možné stavy systému, reprezentovat každý z nich jako samostatný obdélník.
2. Vyšetřit všechny smysluplné přechody mezi stavů. Začít zakreslením výchozího stavu, poté metodicky sledovat vývoj do dalších stavů.
3. Prověřit konzistenci STD a hierarchii STD.
4. Prověřit konzistenci STD proti jiným modelům systému

Konzistence STD

- Byly definovány všechny stavy?
- Jsou všechny stavy dosažitelné?
- Lze opustit všechny stavy?
- Reaguje systém ve všech stavech na všechny možné podmínky?
 - Nejčastější chybou je opomenutí přechodů, které nastanou za neočekávaných podmínek

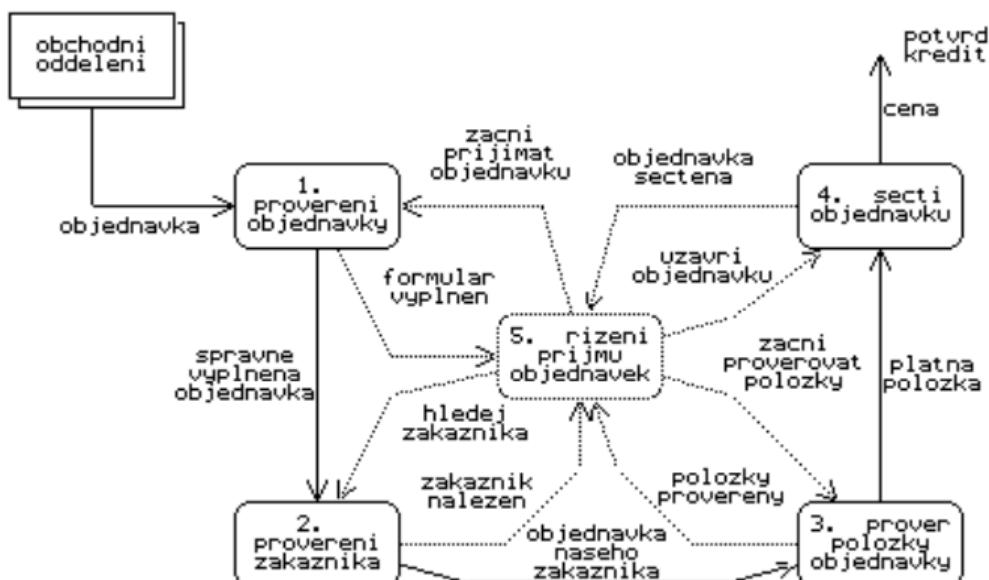
Vztah mezi CDFD a STD

- CDFD
 - Control Data Flow Diagram
 - DFD rozšířený o řídící procesy, toky a paměti.



- Každý **řídící proces** je popsán pomocí STD.
- **Vstupní signály** se uplatňují v podmínkách přechodů
- Akcím přechodů odpovídají procesy na CDFD.

Příklad CDFD - Příjem objednávek



- Úkol: STD „řízení příjmu objednávek“

Vyvažování modelů

- Souhrn probraných nástrojů strukturované analýzy:
 - Kontextový diagram
 - DFD
 - Minispecifikace procesů
 - Datový slovník - DD
 - ERD
 - STD
 - CDFD
- Vyvažování
 - **vzájemná kontrola** mezi **modely** a **uvnitř hierarchie modelů**.

Vyvažování DFD - DD

- **Každý datový tok a každá datová paměť** na DFD musí být **definovány v datovém slovníku**.
 - Pokud v něm **chybí**, jsou **považovány za nedefinované**.
- Každý datový element nebo paměť definované v datovém slovníku se musí vyskytovat někde na DFD.
 - V opačném případě jsou považovány za přebytečné, za nepoužité v systému.

Vyvažování DFD - minispecifikace

- (Pravidla se vztahují na procesy, které řeší zpracování dat)
- Každý proces na DFD musí být asociován s DFD na nižší úrovni nebo se specifikací procesu, ale nikoliv současně s oběma.
 - Pokud existují obě definice procesu, je model zbytečně a nebezpečně redundantní.
- Každá specifikace procesu musí mít sdružený proces na nejnižší úrovni.
 - Pokud chybí procesy, pak byly obvykle zrušeny při úpravách DFD.
- Musí souhlasit vstupy a výstupy. -> Vstupní a výstupní toky stejně jako paměti zakreslené u procesu na DFD by měly mít odpovídající operace ve specifikaci (GET, INPUT, DISPLAY ...).

Vyvažování DD - minispecifikace

- Pro každý odkaz ve specifikaci procesu (obvykle podstatné jméno) musí platit jedna z následujících skutečností:
 - Souhlasí se jménem datového toku nebo datové paměti připojené k procesu, který minispecifikace popisuje.
 - Může to být lokální název, explicitně definovaný ve specifikaci.
 - Položka v datovém slovníku je komponentou datového toku nebo datové paměti připojené k procesu

Vyvažování ERD - DFD + minispecifikace

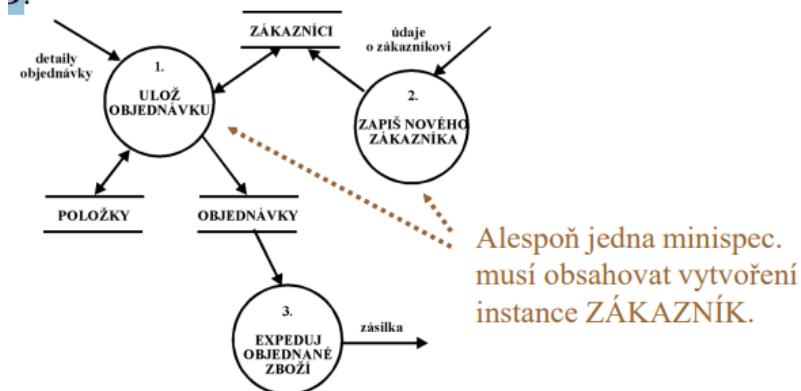
- Paměť na DFD musí odpovídat objektovému typu, relaci nebo jejich kombinaci (asociovanému objektu) na ERD.
 - Výskyt paměti bez odpovídajícího objektového typu nebo objektový typ bez odpovídající paměti jsou považovány za chybu.
- Jména objektových typů na ERD a datových pamětí na DFD musí odpovídat.
 - Konvence předpokládá použití množného čísla na DFD a jednotného čísla na ERD.
- Položky v datovém slovníku musí být aplikovatelné současně na DFD i na ERD. Datový slovník musí obsahovat současně definici objektu z ERD i paměti z DFD.

Př.: ZÁKAZNÍCI = {ZÁKAZNÍK}

ZÁKAZNÍK = jméno + adresa + tel.číslo + ...

Vyvažování ERD - DFD + minispecifikace

- Vytvoření a rušení instancí každého objektového typu a relace uvedené na ERD



- V DFD musí existovat procesy, které přiřadí hodnoty každému datovému elementu, který je atributem objektového typu

Vyvažování DFD - STD

- Pro každý řídící proces existuje stavový diagram jako jeho specifikace.
 - Ke každému stavovému diagramu musí existovat řídící proces na DFD.
- Každá podmínka ve stavovém diagramu musí odpovídat nějakému vstupnímu řídícímu toku, který vede do řídícího procesu sdruženého s příslušným STD.
- Každé akci stavového diagramu musí odpovídat nějaký výstupní řídící tok řídícího procesu sdruženého se tímto STD.

Basic concepts

- Indexing mechanisms used to speed up access to desired data.
 - author catalog in library
- Search key - id
- An index file consists of records (called index entries) of the form
 - search-key --- pointer
- Two basic kinds of indices:
 - Ordered indices**: search keys are stored in sorted order
 - Hash indices**: search keys are distributed uniformly across “buckets” using a “hash function”.

Ordered Indices

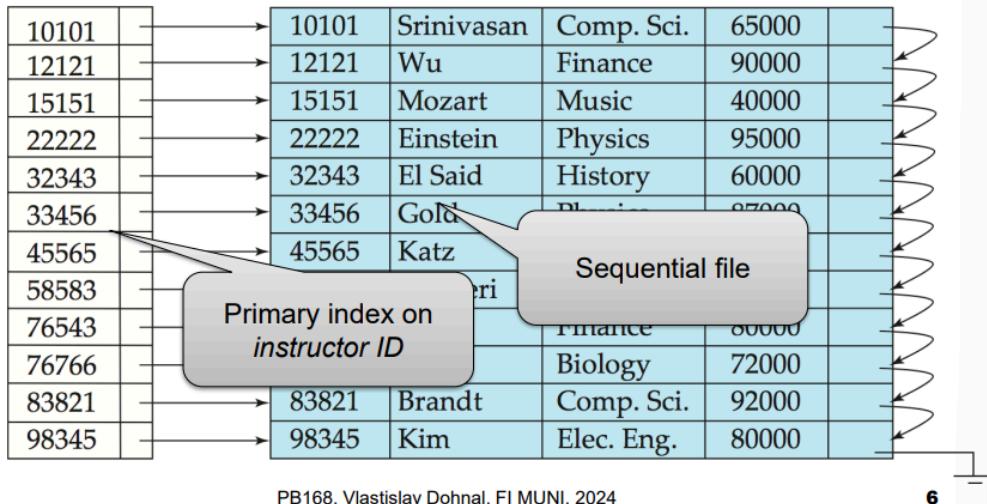
- In an ordered index, index entries are stored sorted on the search key value
- Primary index**:
 - assume a sequential file, the index whose search key specifies the sequential order of records in the file.
 - Also called **clustering index**
 - The search key of a primary index is usually but not necessarily the primary key.

- **Secondary index:**

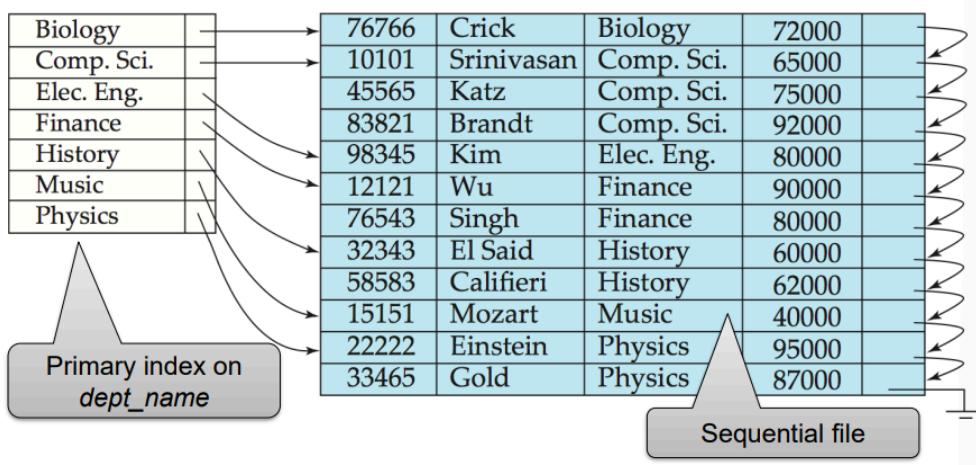
- an index whose search key specifies an order different from the sequential order of records in the file.
 - Also called **non-clustering index**

Dense Index Files (primary index)

- Dense index
 - Index record appears for every search-key value in the file.
- E.g. index on ID attr. of instructor(id, name, dept_name, salary)

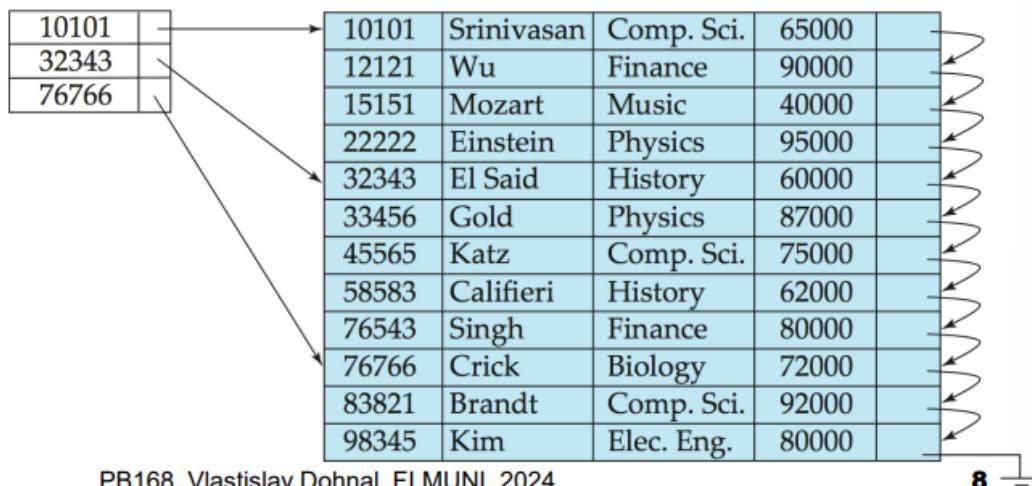


- PB168, Vlastislav Dohnal, FI MUNI, 2024
- Dense index on dept_name, with instructor file sorted on dept_name



Sparse Index Files

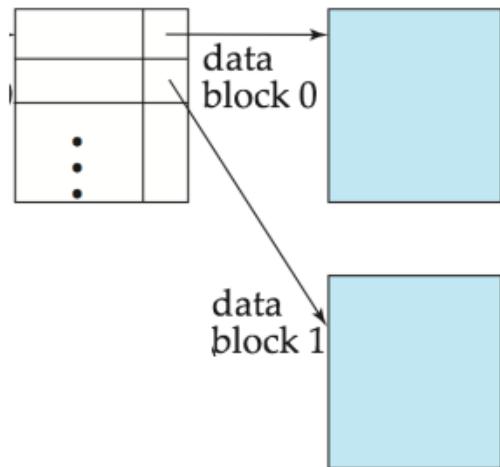
- Sparse Index
 - Contains index records for only some search-key values
 - Applicable when records are sequentially ordered on search-key



PB168, Vlastislav Dohnal, FI MUNI, 2024

8

- Compared to dense indices:
 - Less space and less maintenance overhead for insertions and deletions.
 - Generally slower than dense index for locating records.
- Good tradeoff:**
 - Sparse index with an index entry for every block in file, corresponding to least search-key value in the block.

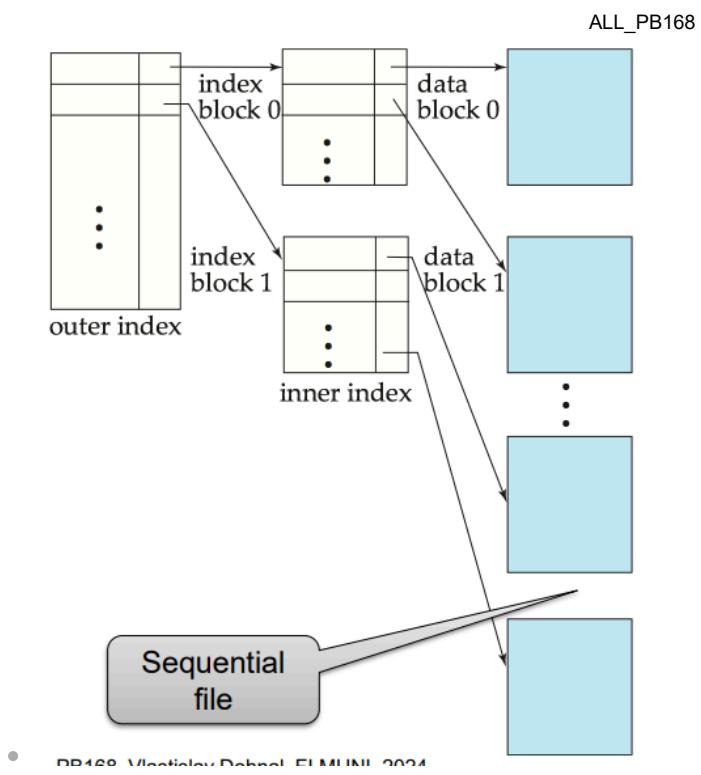


Secondary Indices

xxx <https://is.muni.cz/auth/el/fi/podzim2025/PB168/um/slides08-indexing-hashing.pdf?predmet=1654908>

Multilevel Index

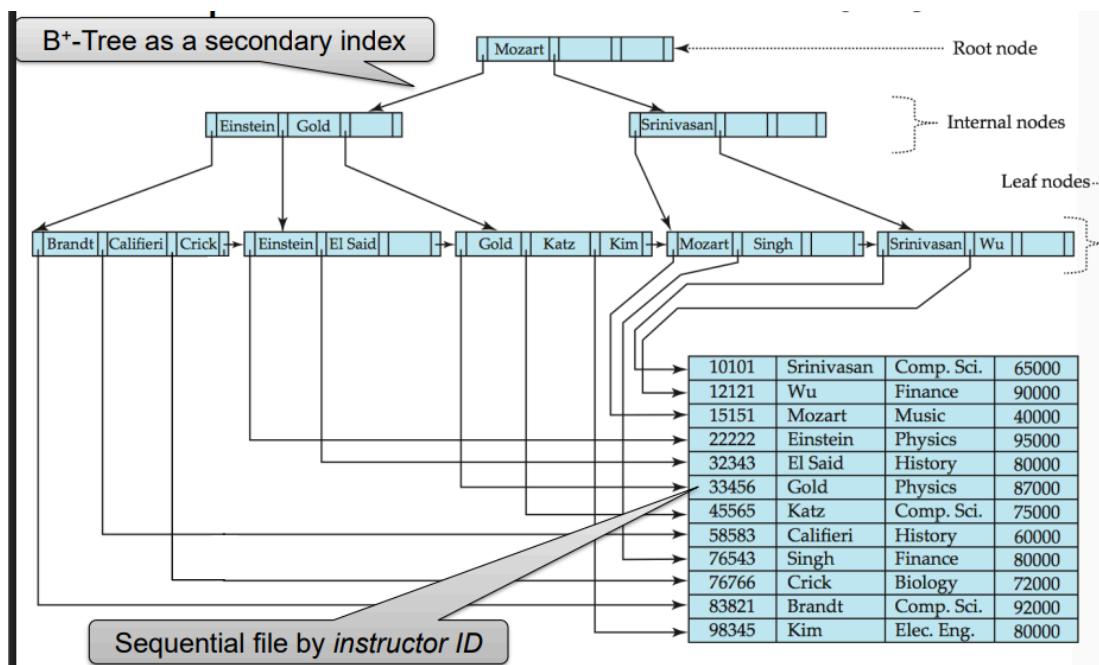
- If an index does not fit in memory, access becomes expensive.
- Solution: treat the index kept on disk as a sequential file and construct a sparse index on it.
 - outer index – a sparse index of the original index
 - inner index – the original index file



- DB168_Vigotsky_Dabhol_ELMUNI_2024

B+ - Tree Index Files

- B+tree file organization is an alternative to indexed-sequential files.
- Disadvantage of indexed-sequential files
 - Performance degrades as file grows, since many overflow blocks get created.
 - Periodic reorganization of entire file is required.
- Advantage of B+tree files:
 - Automatically reorganizes itself with small, local, changes, in the face of insertions and deletions.
 - Reorganization of entire file is not required to maintain performance.
- (Minor) disadvantage of B+trees:
 - Extra insertion and deletion overhead, space overhead



B+ Tree Index

- A B+ -tree is a rooted tree satisfying the following properties:
 - All paths from root to leaf are of the same length
 - Each node that is not a root or a leaf has between $n/2$ and n children.
 - A leaf node has between $(n-1)/2$ and $n-1$ values

B+ Tree Node Structure

- | | | | | | | |
|-------|-------|-------|---------|-----------|-----------|-------|
| P_1 | K_1 | P_2 | \dots | P_{n-1} | K_{n-1} | P_n |
|-------|-------|-------|---------|-----------|-----------|-------|

 - K_i are values of the search key
 - P_i are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes).
 - The search-key values in a node are ordered!
 - .
 - .
 - .
 - .

Hashing

- In a hash file organization, we obtain the address of a record directly from its search-key value using a hash function
 - Address is typically a bucket – a unit of storage containing one or more records
 - A bucket corresponds to a disk block
- Hash function h
 - a function from the set of all search-key values K to the set of all bucket addresses B
 - used to locate records for access, insertion as well as deletion
- Records with different search-key values may be mapped to the same bucket
 - thus entire bucket has to be searched sequentially to locate a record

bucket 0			

bucket 4			
12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 1			
15151	Mozart	Music	40000

bucket 5			
76766	Crick	Biology	72000

bucket 2			
32343	El Said	History	80000
58583	Califieri	History	60000

bucket 6			
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 3			
22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 7			

PB168, Vlastislav Dohnal, FI MUNI, 2024

Example of Hash File Organization

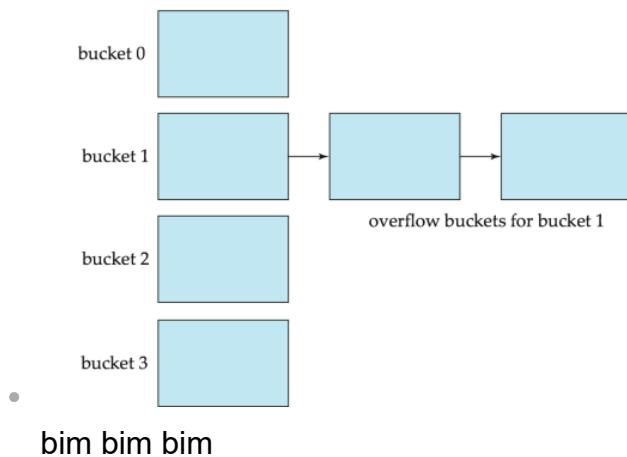
- Hash function on dept_name can be defined as:
 - The binary representation of the i-th character in the alphabet is assumed to be the integer i
 - The hash function returns the sum of the binary representations of all the characters modulo 8
 - i.e., there are 8 buckets.
- $h(\text{Music}) = 1 \quad (\text{M=13, u=21, s=19, i=9, c=3} \Rightarrow 65 \bmod 8 \Rightarrow 1)$
 - $h(\text{History}) = 2$
 - $h(\text{Physics}) = 3$
 - $h(\text{Elec. Eng.}) = 3$

Handling of bucket overflows

- Collision occurs when two different search-key values are hashed to the same address (bucket)
- Bucket overflow can occur because of
 - Insufficient bucket size
 - Skew in distribution of records.
- Although the probability of bucket overflow can be reduced, it cannot be eliminated; it is handled by using
 - Overflow buckets**
 - Collision function**

Overflow chaining

- the overflow buckets of a given bucket are chained together in a linked list.
 - This scheme is called closed hashing.



Dynamic Hashing

bim bim bim

Index Definition in SQL

- Create an index


```
create index <index-name> on <relation-name>
    (<attribute-list>)
```

 -
- Drop an index
 - `drop index <index-name>`

Classification of Physical Storage Media

- Speed of access
- Cost per unit of data
- Reliability
- Can differ:
 - Volatile storage: loses contents when power off
 - Non-volatile storage: Contents persists

Phys. St.Me.

- Cache - fastest and most costly form of storage; volatile; managed by the computer system hardware
- Main memory:
 - fast access
 - generally too small to store the entire database
 - Volatile

- Flash memory:
 - Data survives power failure
 - Data can be written at a location only once, but location can be erased and written to again
 - Writes are slow
 - USB, SSD, phone mem
- Magnetic-disk
 - read/write magnetically
 - Much slower access
- Optical storage
 - Non-volatile
 - data read optically from spinning disk using laser
 - DVD (in GB)
 - Blu-ray (in GB)
- Tape storage

Storage Hierarchy

- Primary storage
 - fastest/volatile (cache, main memory)
- Secondary storage
 - non-volatile, moderately fast (also called on-line storage)(flash, magnetic disks)
- Tertiary storage
 - non-vol., slow access, (off-line storage)(magnetic tape, optical storage)

Performance Measures of Disks

- Access time - the time it takes from when a read or write request is issued to when data transfer begins
 - Seek time - time it takes to reposition the arm over the correct track
 - Rotational latency - time it takes for the sector to be accessed to appear under the head
 - 4 to 11 milliseconds on typical disks (5400 to 15000 rpm)
- Data-transfer rate - the rate at which data can be retrieved from or stored to the disk
 - 25 to 200 MB per second max rate

Optimization of Disk-Block Access

- Block - a contiguous sequence of sectors from a single track
 - data is transferred between disk and main memory in blocks
 - sizes range from one sector (e.g., 512 bytes) to several sectors (e.g., 4 kilobytes)

- File organization - optimize block access time by organizing the blocks to correspond to how data will be accessed
 - File systems may get fragmented over time

File Organization

- The database is a collection of relations
 - Relations are stored as individual files.
 - Each file is a sequence of records.
 - A record is a sequence of fields/attributes.



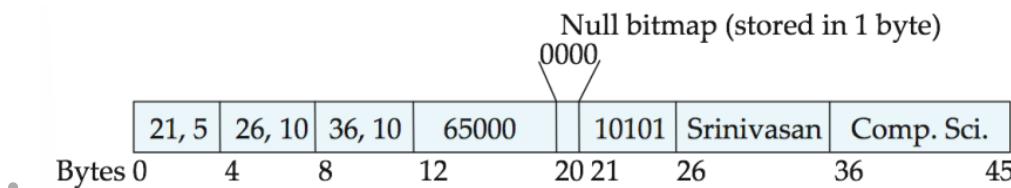
- Block ~4KB

Fixed-Length Records

- Simple approach:
 - Store record i starting from byte $n * (i - 1)$
 - where n is the size of each record.
 - Record access is simple, but records may cross blocks
 - Solution: do not allow records to cross block boundaries
- Deletion of record i
 1. move records $i + 1, \dots, n$ to $i, \dots, n - 1$
 2. move record n to i
 3. do not move records, but link all free records on a free list

Variable-Length Records

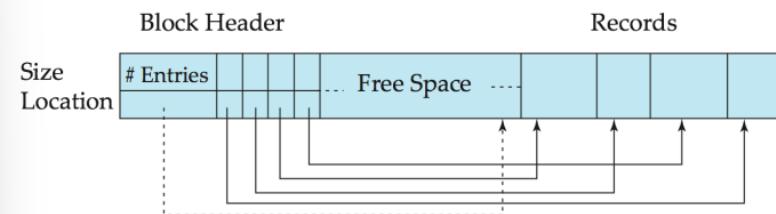
- Variable-length records are common in database systems:
 - Attributes of variable-length type, e.g. strings (varchar)
- Solution:
 - Attributes are stored in order
 - Variable-length attributes represented by fixed size (offset, length)
 - actual data stored after all fixed length attributes (at address of offset)
 - Null values represented by null-value bitmap
- Relation: instructor(id, name, dept_name)



Slotted Page Structure

- Header contains:
 - number of record entries
 - end of free space in the block
 - location and size of each record
- Records stored from the end of the block
- Record updates
 - Record can be moved around within a page
 - to keep all records contiguous with no empty space between them
 - Only entry in the header must be updated

Block on a disk:



Organization of Records in Files

- File
 - is usually divided into blocks (atomic size of DB I/O operations)
- Type of file organization:
 - Heap – a record can be placed anywhere in the file where there is space (i.e., in any block, where is a room)
 - Sequential – store records one after the other and in a sequential order, based on the value of the search key of each record
 - Hash – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed

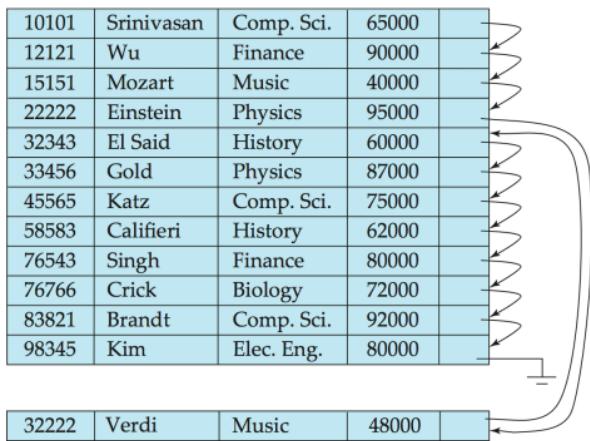
Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key
 - id

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Sequential File Organization: Updates

- Deletion – use pointer chains
- Insertion – locate the position where the record is to be inserted
 - if there is free space insert there
 - if no free space, insert the record in an overflow block
 - in either case, pointer chain must be updated
- Need to reorganize the file from time to time



- 32222 Verdi Music 48000

Data Dict. Storage

Cvičení

[Cvičení5 - B+ stromy](#)

[CvičB+stromy.excalidraw](#)