

6. SÍŤOVÝ DATOVÝ MODEL



Cíl Po prostudování celé kapitoly budete umět

- rozumět pojmům síťového datového modelu
- vysvětlit základní rozdíl mezi relačním a datovým modelem
- definovat data v tomto modelu
- vysvětlit princip práce s daty v síťovém modelu
- vyhledávat informace v síťové databázi pomocí procedurálního jazyka



Čas ke studiu: 3 hodiny

6.1. Základní pojmy síťového datového modelu



Výklad

□ Historie síťového datového modelu

Již víme, že historicky prvním datovým modelem v databázové technologii byl model hierarchický a pak jeho zobecnění, model síťový. Nejprve vznikaly samostatné implementace síťových SŘBD. V roce 1971 pak byl definován pracovní skupinou pro databáze DBTG (Data Base Task Group) při sdružení CODASYL (Conference on Data System Languages) návrh architektury SŘBD síťového modelu. Od té doby se stal standardem pro prakticky všechny implementace.

U nás bylo v té době velmi málo počítačů vůbec, většinou jen ve velkých výrobních podnicích a pro zpracování dat se převážně používal jazyk Cobol. Síťový model se příliš nerozšířil. Mohli jsme se setkat například se síťovými databázovými systémy

IDMS na počítačích řady IBM 360, IBM 370, EC 1045, EC 1055,
DBMS na počítačích řady SMEP, DEC, VAX,
IMAGE na počítačích Hewlet-Packard, ADT.

Standard CODASYL se týkal především implementace síťového SŘBD: deklarací - příkazů pro definice dat a JMD - způsobu práce a příkazů pro manipulace s daty.

□ Význam síťového modelu v současnosti

Proč vlastně ještě probíráme síťový model? I když se v současnosti příliš nepoužívá, snad jen ve starších dosud užívaných IS, nejsou principy tohoto modelu zastaralé. Některé metody zabudované do standardu je velmi užitečné znát i dnes. Například použití jednoznačného databázového klíče, v SŘBD s datovými typy ukazatel realizace vazeb pomocí setů atd. jsou metody aktuální i u současných objektově-relačních databází. Mohou tak využít výhod relačního modelu a současně některých metod modelu síťového.

□ Základní pojmy síťového datového modelu

Slovník síťového modelu vychází z programovacího jazyka Cobol. Standard zahrnuje tyto hlavní pojmy a zásady:

Logickému modelu databáze se říká **schema**, externím schématům **subschemata**. Na úrovni definice schématu se typ entity nazývá **typ záznamu** (RECORD), jeho atributy se nazývají **komponenty**. V databázi samé se jednotlivé entity nazývají **výskyty záznamu** příslušného typu.

Databáze se skládá z řady výskytů záznamů každého z deklarovaných typů.

Výskyty záznamů dokonce nemusí být různé, tj. jedna entita může být v databázi reprezentována více výskyty záznamů. Tyto výskyty jsou rozlišeny pouze hodnotou **databázového klíče**, který je systémem automaticky přidělován každému výskytu záznamu v okamžiku jeho uložení do databáze. Jeho hodnota jednoznačně identifikuje každý výskyt záznamu vůči jiným výskytům záznamů i jiného typu, tedy v rámci celé databáze.

Vyskytují se i deklarace typů záznamu bez komponent. Zdánlivě prázdné výskyty těchto záznamů pak ve skutečnosti obsahují ukazatele na jiné výskyty záznamů.

□ Realizace vazeb v síťovém datovém modelu

Datový model síťový se liší od relačního hlavně způsobem realizace vztahů mezi entitami.

Síťový model definuje pouze binární vztahy typů 1:1 a 1:N mezi dvěma typy záznamů R a S. Ostatní typy vztahů se předem, na úrovni konceptuální rozloží (v tom je shoda s RDM). Tento vztah se nazývá množina neboli **set**. Set je definován pomocí svého vlastníka a členů. Typ záznamu R se nazývá vlastníkem (OWNER) setu S, S je členem (MEMBER) setu S. Ve schématu je pro každý vztah definován **typ setu**. Je mu přiděleno jméno, definován typ záznamu, který je vlastníkem a typ záznamu, který je členem setu.

Síťový datový model realizuje vztah pomocí ukazatelů na vazební entity. Ke každé tabulce je připojena systémová část s tolika odkazy, ke kolika jiným typům záznamů je záznam vázán (= kolika setů je účastníkem, vlastníkem nebo členem).

V databázi je vztah reprezentován řadou **výskytů setu**. Výskyt setu obsahuje právě jeden výskyt záznamu vlastníka a právě ty výskyty záznamů člena setu, které jsou s vlastníkem výskytu setu v příslušném vztahu. Výskyt setu může obsahovat pouze výskyt záznamu vlastníka (prázdný výskyt setu) a množiny členů dvou výskytů téhož typu setu jsou disjunktní.

Výskyty setů se realizují zřetězeným seznamem: z vlastníka na prvního člena, členové setu mezi sebou, z posledního člena zpět na vlastníka. Pak procházení celé množiny je velmi rychlé, bez zbytečných přenosů záznamů mezi diskem a pamětí. Pokud jsou členy setu navíc umístovány „blízko sebe“, může být počet přenosů ještě menší, pokud je více členů množiny umístěno v jednom fyzickém bloku.

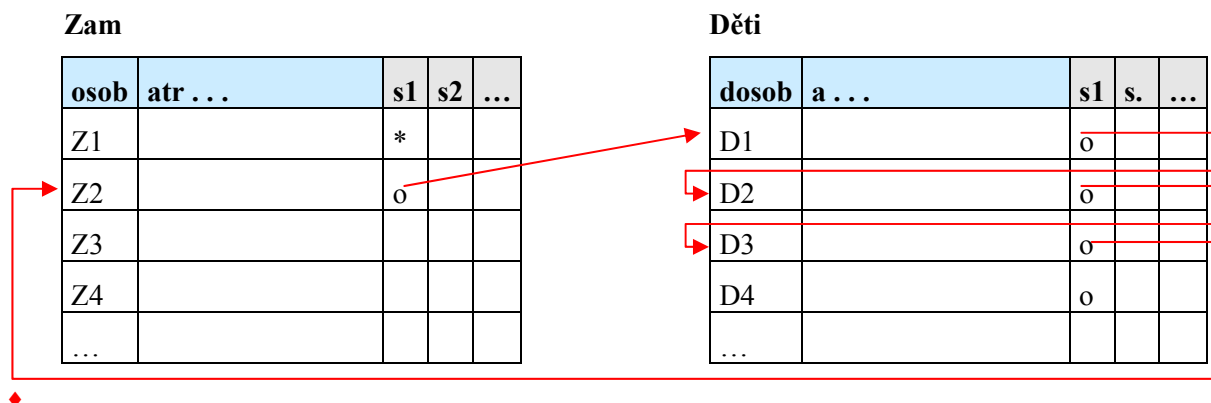
Pro zařazení záznamů do setů platí:

- výskyt setu obsahuje právě jeden výskyt záznamu vlastníka a právě ty výskyty záznamů členů setu, které jsou s vlastníkem výskytu setu v příslušném vztahu,
- výskyt setu může obsahovat pouze výskyt záznamu vlastníka (prázdný výskyt setu),
- množiny členů dvou výskytů téhož typu setu jsou disjunktní

Příklad 6.1.

*Na následujícím obrázku jsou dvě tabulky **Zam** a **Děti** ve vztahu 1:M, tedy Zam je vlastníkem, Děti členem setu zde nazvaného **rod**. První systémový sloupec v tabulkách Zam i Děti je deklarován pro set rod. První zaměstnanec nemá žádné děti, druhý má 3 děti, a to D1, D2, D3.*

Další systémové sloupce jsou určeny pro další typy setů, kterých se oba typy záznamů mohou účastnit.



□ Zobrazení záznamů, vazeb a jejich typů – konceptuální schéma

Graficky se znázorňují jednotlivé prvky modelu takto (jde o jiný způsob zakreslení konceptuálního schématu, než pomocí ERD):

typ záznamu:

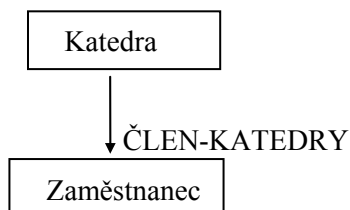
Katedra

typ setu (hrana od vlastníka ke členu)

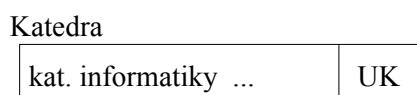
jméno typu vlastníka

jméno typu setu

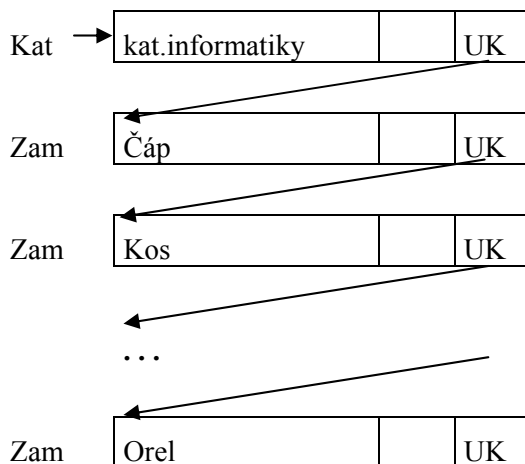
jméno typu členu



výskyt záznamu



výskyt setu

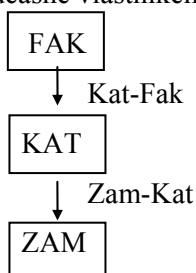


□ Pravidla pro definování setů

Pro vlastníka a členy setu platí:

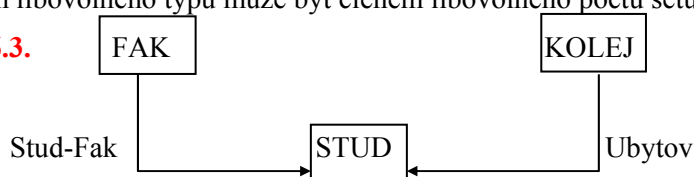
1. Tentýž typ záznamu může být současně vlastníkem jednoho setu a členem v jiném setu.

Příklad 6.2.



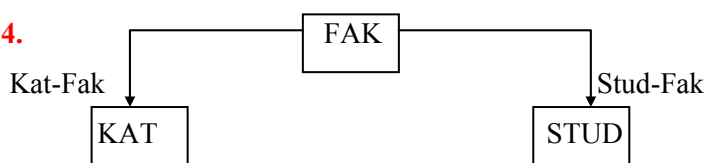
2. Záznam libovolného typu může být členem libovolného počtu setů.

Příklad 6.3.



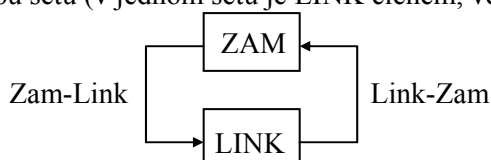
3. Záznam libovolného typu může být vlastníkem libovolného počtu setů.

Příklad 6.4.



4. Vlastník a člen setu nemohou být záznamy téhož typu. Unární vztah 1:N se realizuje prostřednictvím dalšího pomocného typu záznamu (který má pouze spojovací roli a nazveme jej LINK) a pomocí dvou typů setů (v jednom setu je LINK členem, ve druhém vlastníkem).

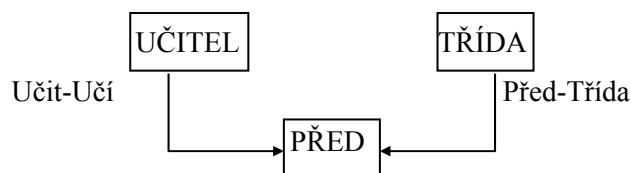
Příklad 6.5.



Pro vztah „přímý nadřízený“ mezi záznamy typu Zaměstnanec se zavede typ záznamu LINK bez uživatelských komponent. Každý výskyt záznamu příslušející vedoucímu pracovníkovi je vlastníkem výskytu setu Zam-Link s právě jedním členem typu LINK a ten je pak vlastníkem výskytu setu Link-Zam, jehož členy jsou podřízení dotčeného vedoucího pracovníka.

5. Vztah typu M:N není možno realizovat přímo a realizuje se (již na úrovni konceptuálního schématu) pomocí dvou vazeb typu 1:M prostřednictvím spojovacího typu záznamu. Ten je členem v obou typech setů.

Příklad 6.6.



Mezi Učitelem a Třídou má vztah „Učí“ kardinalitu M:N. Zavedeme pomocný spojovací typ záznamu Předmět, který na rozdíl od LINKu z minulého případu má vlastní komponenty Název a Hodin. Vazbu pak realizují dva typy setů Učit-Učí a Před-Třída. Vlastníky setů jsou Učitel a Třída, v obou je Předmět členem. Existence vztahu mezi Učitelem a Třídou je vyjádřena přítomností výskytu záznamu Předmět (ten předmět, který příslušný učitel učí v příslušné třídě).

Takto je možno realizovat i vztah typu M:N mezi záznamy téhož typu.

6.2. Jazyk pro definici dat

□ Deklarace schématu a subschémat

Je-li provedena datová analýza, tj. navrženy typy entit a vazeb, převádí se konceptuální datový model celé databáze na databázový model pomocí deklarací (odpovídají JDD).

Deklarace schématu (= databázového schématu celé databáze) má strukturu:

```
SCHEMA NAME IS jméno_schématu
RECORD SECTION
  deklarace_záznamů
SET SECTION
  deklarace_setů
```

Z daného schématu je možno deklarovat libovolný počet subschémat (= dílčích pohledů na databázi, odpovídajících aplikačním programům). Subschéma může být sdíleno více aplikacemi, různá subschémata se mohou v databázi překrývat.

V deklaraci se uvede jméno subschématu a schématu, z něhož je odvozeno. Dále platí

- v deklaraci záznamů můžeme vynechat některé komponenty původních záznamů, některé záznamy nebo některé sety.
- komponentám záznamů je možno přiřadit jiný datový typ, než měly ve schématu, můžeme změnit pořadí komponent uvnitř záznamu, případně můžeme objektům ze schématu přiřadit jiná jména.
- při deklaraci záznamů a setů již neuvádíme jejich vlastnosti stanovené ve schématu.

Deklarace subschématu má strukturu:

```
SUBSCHEMA jméno_subschématu WITHIN jméno_schématu
RECORD SECTION
  deklarace_záznamů
SET SECTION
  deklarace_setů
```

□ Deklarace typů záznamů

Při definování struktury databáze se používá syntaxe z jazyka Cobol. Datové typy jsou definovány pomocí typu a délky (například PIC 9(n) nebo PIC X(n)), jsou povoleny úrovně definice (01, 02, ... před komponentou znamenají hloubku úrovně atributu), tedy neatomické položky.

Součástí definice mohou být vlastnosti, které upřesňují způsob umístění nových záznamů do databáze. Mimo automatického umístění samotným SRBD je možné volit umístění záznamů některého typu pomocí hašovací funkce nebo volit umístění vlastníka a členů setů blízko sebe, aby jejich vyhledávání a přenos byly minimalizovány.

Deklarace typu záznamu má tvar:

RECORD NAME IS jméno_typu_záznamu	
LOCATION MODE IS SYSTEM	... umístí sám systém
DIRECT	... přímo dle DB-klíče
CALC proc USING sez-pol	... přímo dle vybrané položky
VIA	... členy setu "blízko" vlastníka
01 <komponenta 1> PIC 9(n)	
01 <komponenta 2> PIC X(n)	
02 <komponenta 3> PIC X(n)	
...	
01 <komponenta 5> PIC X(n)	
...	

Příklad 6.7.

Deklarace typu záznamu s neatomickým atributem adresa

Zaměstnanec(osob, jméno, adresa(ulice, obec, PSČ), plat)

a s hašováním podle osobního čísla osob. Hašovací procedura dává adresu umístění záznamu:

adr = proc_zam(osob)

```

RECORD NAME IS Zaměstnanec
LOCATION MODE IS CALC proc_zam USING osob
  01 osob      PIC 9(6)
  01 jméno     PIC X(20)
  01 adresa    PIC X(n)
    02 ulice   PIC X(20)
    02 obec    PIC X(20)
    02 PSC     PIC 9(5)
  01 plat      PIC 9(8)

```



□ Deklarace typů setů

Také set musíme předem deklarovat. Přitom je opět možno definovat (mimo vlastníka a člena setu) řadu vlastností setu, jako způsob zařazování nového člena do setu (= uspořádání setu), povinnost členství ve vztahu, zda může člen měnit vlastníka apod.

SET NAME IS jméno_typu_setu	
OWNER IS jméno_typu_vlastníka	
ORDER IS FIRST	... uložení v setu jako první člen
LAST	... poslední člen
NEXT	... před aktuální člen
PRIOR	... za aktuální člen
SORTED BY DEFINES KEYS	... uložení zatříděné podle klíče
PERMANENT	... není možno měnit pořadí člena
MEMBER IS jméno_typu_člena MANDATORY	... povinný člen, může měnit vlastníka
FIXED	... nemůže měnit vlastníka
OPTIONAL	... nepovinný člen
AUTOMATIC	... začlenění výskytu záznamu provede systém automaticky i do setu
MANUAL	... začlenění výskytu záznamu provede ručně uživatel v aplikaci
DUPLICATE IS NOT ALLOWED	... jednoznačnost třídícího klíče

Příklad 6.8.

Deklarace databáze FAKULTA s následujícími entitami a vazbami.

Typy záznamů: Učitel (ču, jméno, funkce, plat)
 Předmět (čp, název, ročník)
 Úvazek (ču, čp, hodin)
 Student (jméno, město, ulice)
 Kolej (název)

Typy setů: UČITEL_UČÍ (Učitel, Úvazek)
 PŘEDMĚT_UČÍ (Předmět, Úvazek)
 BYDLÍ (Kolej, Student)

Hašovací procedury pro přímé adresování tabulek Učitel a Předmět se jmenují hash_ču (ču) a hash_čp (čp).

SCHEMA NAME IS FAKULTA**RECORD SECTION****RECORD NAME IS Učitel**

LOCATION MODE IS CALC hash_ču USING ču IN Učitel
 02 ču PIC XXX
 02 jméno PIC X(15)
 02 funkce PIC X(10)
 02 plat PIC 99

RECORD NAME IS Předmět

LOCATION MODE IS CALC hash_čp USING čp IN Předmět
 02 čp PIC XXX
 02 název PIC X(15)
 02 ročník PIC 9

RECORD NAME IS Úvazek

LOCATION MODE IS SYSTEM-DEFAULT
 02 ču PIC XXX
 02 čp PIC XXX
 02 hodin PIC 99

...

SET SECTION**SET NAME IS Učitel_Učí**

OWNER IS Učitel
 ORDER IS PERMANENT SORTED BY DEFINED KEYS
 MEMBER IS Úvazek MANDATORY AUTOMATIC
 KEY IS ASCENDING čp IN Úvazek
 DUPLICATE ARE NOT ALLOWED
 SET SELECTION IS THRU Učitel_Učí
 IDENTIFIED BY CALC-KEY

SET NAME IS Učí_Předmět

OWNER IS Předmět
 ORDER IS PERMANENT SORTED BY DEFINED KEYS
 MEMBER IS Úvazek MANDATORY AUTOMATIC
 KEY IS ASCENDING ču IN Úvazek
 DUPLICATE ARE NOT ALLOWED
 SET SELECTION IS THRU Učí_Předmět
 IDENTIFIED BY CALC-KEY

...

Příklad 6.9.

Část deklarace subschématu pro mzdovou agendu.

SUBSCHEMA PENIZE WITHIN FAKULTA**RECORD SECTION**

```
01 učitel
  02 ču      PIC X(3)
  02 honorář PIC 99
01 úvazek
  02 čp      PIC X(3)
  02 ču      PIC X(3)
  02 hodin   PIC 99
```

SET SECTION

```
SET Učitel_Učí
```

```
...
```



6.3. Jazyk pro manipulaci s daty

□ Pracovní oblast uživatele

Jazyk pro manipulaci s daty je koncipován jako množina příkazů vnořených do hostitelského jazyka. Často to býval COBOL nebo PL/1. V další části textu budeme pro lepší srozumitelnost předpokládat, že hostitelským jazykem je Pascal, i když to zpravidla tak není.

Pro každý aplikační program (v paměti jich může současně být více) je v paměti počítače zvláštní oblast, která se nazývá **pracovní oblast uživatele** (user work area). V každé uživatelské oblasti jsou umístěny:

- prostor pro uchování jednoho výskytu každého typu záznamu, který uživatelský program používá; v programu se na tento prostor odkazuje jmény záznamů, případně jejich komponent podle deklarací v subschématu;
- ukazatele na aktuální objekty (currency pointer) ve formě databázového klíče, ke kterým patří: CURRENT OF typ_záznamu
- ukazatele na naposledy zpracovávané výskyty záznamů pro každý typ záznamu: CURRENT OF SET typ_setu
- odkazy na naposledy zpracovávané výskyty záznamů v každém setu: CURRENT OF PROGRAM
- odkaz na naposledy zpracovávaný výskyt záznamu celkově bez ohledu na jeho typ nebo příslušnost k setu;
- stavové proměnné, obsahující popis stavu, v němž se systém právě nachází; jsou to např. **db-status** obsahuje 0 po bezchybně provedené operaci nebo číslo chyby, **db-set-name \ db-record-name** lokalizují místo, kde nastala chyba, **db-data-name** obsahuje název chyby

Příklad 6.10.

Pracovní oblast pro databázi FAKULTA:

Paměť pro jeden výskyt každého typu záznamu, používají se jako programové proměnné	{	ču jméno ...	proměnné záznamu Učitel
		čp název ...	proměnné záznamu Předmět
		ču čp hodin	proměnné záznamu Úvazek
Tabulka ukazatelů aktuálních výskytů záznamů v tabulkách	{	DB-KEY program	CURRENT OF PROGRAM
		DB-KEY učitel	CURRENT OF Učitel
		DB-KEY předmět	CURRENT OF Předmět
		DB-KEY úvazek	CURRENT OF Úvazek
		...	CURRENT OF ...
Tabulka ukazatelů aktuálních výskytů záznamů v setech	{	DB-KEY U-Ú	CURRENT OF Učitel_Učí
		DB-KEY U-P	CURRENT OF Učí_Předmět
		...	CURRENT OF ...
Systémové proměnné		DB-STATUS	
		DB-record-name	
		DB-set-name	
		DB-data-name	



□ **Seznam příkazů JMD**

□ **Základní 4 databázové operace**

- FIND** vyhledá výskyt záznamu v DB a definuje jej za aktuální záznam programu, aktuální záznam příslušného typu záznamu a případně aktuálním záznamem setů, v nichž je záznam zařazen; jinými slovy databázový klíč vyhledaného výskytu záznamu je uložen do položek CURRENT OF PROGRAM, příslušného CURRENT OF typ a příslušných CURRENT OF SET
- GET** přesune do uživatelské oblasti (na místo rezervované pro tento typ záznamu) nalezený aktuální záznam programu
- STORE** uloží nový výskyt záznamu do DB, definuje jej za aktuální záznam programu, aktuální záznam příslušného typu záznamu a za aktuální záznamy setů, do nichž je případně automaticky zařazen
- MODIFY** modifikuje aktuální záznam programu
- ERASE** vymaže aktuální záznam programu z DB

□ Základní operace se sety

CONNECT	vloží aktuální záznam programu do výskytu setu
DISCONNECT	vyjme aktuální záznam programu z výskytu setu
RECONNECT	přesune aktuální záznam do jiného setu
ORDER	setřídí set

□ Další databázové příkazy

READY	otevře pracovní uživatelskou oblast pro práci s DB
FINISH	ukončí práci s pracovní uživatelskou oblastí
IF	možnost testu v posledním vyhledaném záznamu
USE	nastavení reakce při výjimečném stavu DB
ACCEPT	přesun hodnot systémových DB proměnných do položek v programu
KEEP	uzamkne výskyt záznamu pro ostatní uživatele
FREE	odemkne výskyt záznamu

Příkazy FIND a STORE tedy mění obsah tabulky běžných ukazatelů, ostatní příkazy pracují s aktuálním záznamem jako s operandem.

□ Provádění základních databázových operací

Dále probereme nejdůležitější příkazy JMD poněkud podrobněji. Pro lepší srozumitelnost použijeme v těchto příkazech poněkud zjednodušené syntaxe JMD i upravených tvarů hostitelského jazyka.

Označíme:	T	... typ záznamu
	S	... typ setu
	DBK	... db-klíč
	POL	... položky záznamu

Prakticky na příkladech uvidíme, že často nebude nutné přenášet datové záznamy do paměti, protože pomocí ukazatelů je možno přecházet mezi záznamy jen prostřednictvím DBK. Proto jsou příkazy pro vyhledání záznamu (zatím jsme tuto operaci chápali jako vyhledání a přenos do paměti současně) rozděleny na vlastní vyhledání FIND a přenos GET. Příkazem GET se tak přenáší jen skutečně potřebná data., která se předcházejícím použitím příkazu FIND našla.

□ Načtení záznamu z databáze do paměti

Příkaz GET má formát

GET T

Pokud aktuálním záznamem programu je výskyt záznamu typu T (předtím nalezený některou variantou příkazu FIND), přenese příkaz GET tento výskyt záznamu do uživatelské pracovní oblasti a tím jej zpřístupní programu ke zpracování.

□ Vyhledání záznamu

Příkaz FIND je základním příkazem JMD, umožňuje různými způsoby vyhledávat výskyt záznamu v DB a definovat jej aktuálním záznamem programu.

Příkaz FIND má několik variant:

- **Vyhledání záznamu podle hodnoty databázového klíče:**

FIND T RECORD BY DB-KEY DBK

Příklad 6.11.

Do proměnné typu databázový klíč si uschováme hodnotu klíče z tabulky ukazatelů:

```
klíč:=CURRENT OF učitel;
```

...

```
FIND učitel RECORD BY DB-KEY klíč;
GET učitel;
```



- **Vyhledání záznamu podle CALC-klíče:**

FIND T RECORD BY CALC_KEY

Před provedením této varianty příkazu FIND je nutné dosadit hodnotu položkám definovaným ve schématu jako CALC-klíče pro záznam typu T.

Příklad 6.12.

V záznamu UČITEL je jako CALC-klíč definována položka ČU. Načtení záznamu učitele s ČU="U1" provedeme příkazy

```
učitel.ČU:="U1";
FIND učitel RECORD BY CALC-KEY;
GET učitel;
```

Příkaz vezme v pracovní oblasti hodnoty definované jako CALC, z nich spočítá CALC-KEY a najde příslušný výskyt záznamu.



- **Vyhledání dalších výskytů záznamů se stejnými hodnotami CALC- klíčů:**

FIND DUPLICATE T RECORD BY CALC-KEY

Příklad 6.13.

Předpokládejme, že při deklaraci záznamu Úvazek jsme použili způsob umístění pomocí CALC-KEY, povolili více výskytů záznamů daného typu se stejnými hodnotami CALC-KEY a ČU je definováno jako CALC-KEY tohoto záznamu a ČP je definováno jako CALC-KEY záznamu Předmět. Pak názvy všech předmětů, které učí učitel U1 zjistíme takto:

```
Úvazek.ČU:="U1";
FIND Úvazek RECORD BY CALC-KEY;
WHILE (DB-STATUS=0) DO
  BEGIN
    GET Úvazek;
    Předmět.ČP:= Úvazek.ČP;
    FIND Předmět RECORD BY CALC-KEY;
    GET Předmět;
    PRINT (Předmět.ČP, Předmět.název);
    FIND DUPLICATE Úvazek RECORD BY CALC-KEY
  END;
```

Všimněme si, že cyklus hledání v DB zastavujeme testem na DB-STATUS, kam příkaz FIND DUPLICATE vloží jedničku, jestliže se další výskyt hledaného záznamu nenašel. ♦

- **Selektivní vyhledávání podle předem zadaných hodnot položek:**

FIND T RECORD USING POL

Příklad 6.14.

V záznamu STUDENT hledáme ulici, kde bydlí student NOVÁK

```
Student.jméno:="NOVÁK";
FIND Student RECORD USING jméno;
GET Student;
PRINT(Student.ulice);
```



- **Vyhledání dalších záznamů se stejnými hodnotami zadané položky**

FIND DUPLICATE T RECORD USING POL

Příklad 6.15.

Chceme vypsat všechna jména studentů, kteří bydlí na koleji.

```
Student.město:="Opava";
FIND Student RECORD USING město ;
WHILE (DB-STATUS=0) DO
  BEGIN
    GET Student ;
    PRINT(Student.jméno) ;
    FIND DUPLICATE Student RECORD USING město
  END;
```



CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probrané případy použití příkazu FIND a GET pojmenované:

```
Find by DB – key.exe
Find by CALC – key.exe
Find using pol.exe
Find duplicate using pol.exe
```

- **Vyhledání záznamů uvnitř výskytu setu**

Pokud při hledání výskytu záznamu víme, do kterého setu záznam patří, hledání se výrazně urychlí. Hledat můžeme buď v právě aktuálním setu (CURRENT SET) nebo v setu daném vlastníkem (OWNER). Při hledání se pak prochází jen krátký zřetězený seznam vlastníka a členů setu.

- **Vyhledání vlastníka aktuálního výskytu zadaného setu:**

FIND OWNER OF CURRENT S SET

- **Vyhledání prvního člena aktuálního výskytu zadaného setu:**

FIND FIRST T RECORD IN CURRENT S SET

- **Vyhledání následujícího výskytu záznamu uvnitř aktuálního výskytu setu**

FIND NEXT T RECORD IN CURRENT S SET

Příklad 6.16.

Vypište seznam všech předmětů, které učí učitel Horák.

Nejprve je vyhledán výskyt záznamu Učitel pro učitele Horáka a tím se zároveň stal aktuálním výskytem setu Učitel_Učí ten výskyt setu, jehož vlastníkem je učitel Horák. Vyhledáme první členský záznam tohoto výskytu setu Učitel_Učí a v cyklu do vyčerpání všech členů provádíme tyto akce:

- (1) *vyhledáme vlastníka lokalizovaného záznamu Učitel_Učí v rámci setu Učí_Předmět; vlastníkem je záznam Předmět;*
- (2) *vytiskneme název předmětu;*
- (3) *vyhledáme následující výskyt záznamu Učitel_Učí v aktuálním výskytu setu Učí_Předmět.*

```
Učitel.jméno="Horák";
FIND Učitel RECORD USING jméno;
FIND FIRST Úvazek RECORD IN CURRENT Učitel_učí SET
WHILE (DB-STATUS=0) DO
  BEGIN
    FIND OWNER OF CURRENT Předmět_učí SET;
    GET Předmět;
    PRINT (Předmět.název);
    FIND NEXT Úvazek RECORD IN CURRENT Učitel_učí SET
  END;
```



- **Selektivní vyhledávání záznamů uvnitř aktuálního výskytu setu:**

FIND T RECORD IN CURRENT S SET USING POL

FIND DUPLICATE T RECORD IN CURRENT S SET USING POL

Příklad 6.17.

Vypište čísla všech učitelů, kteří učí předmět „Logika“ 4 hodiny týdně.

```
Předmět.název="logika";
FIND Předmět RECORD USING název;
Úvazek.hodin=4;
FIND Úvazek RECORD IN CURRENT Před-učí SET USING hodin;
WHILE (DB-STATUS = 0) DO
  BEGIN
    GET Úvazek;
    PRINT (Úvazek.ČU);
    FIND DUPLICATE Úvazek RECORD IN CURRENT Před-učí SET USING hodin;
  END;
```





CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probrané případy použití příkazu FIND uvnitř setu pojmenované:

Find owner.exe

Find xxxxxxxxxx.exe

Find xxxxxxxxxx.exe

➤ Vyhledání záznamů podle hodnot z tabulky aktuálních ukazatelů

Následující příkazy nahrazují dosazovací příkaz, prováděný s aktuálními ukazateli záznamů a setů v rámci pracovní oblasti uživatele.

Nastavení aktuálního záznamu nějakého typu záznamu za aktuální záznam programu:

FIND CURRENT OF T RECORD

Nastavení aktuálního záznamu některého typu setu za aktuální záznam programu:

FIND CURRENT OF S SET

Příklad 6.18.

Máme zjistit, zda učitel U1 učí předmět s názvem LOGIKA a pokud ano, v jakém rozsahu.

```

Učitel.ČU:="U1";
FIND Učitel RECORD BY CALC-KEY;
neučí:=TRUE;
FIND FIRST Úvazek RECORD IN CURRENT Učitel-učí SET;
WHILE (DB-STATUS=0) AND neučí DO
  BEGIN
    FIND OWNER OF CURRENT Učí_předmět SET;
    GET Předmět;
    IF Předmět.název = "LOGIKA"
      THEN BEGIN
        FIND CURRENT OF Úvazek RECORD;
        GET Úvazek;
        PRINT(Úvazek.hodin);
        neučí:=FALSE
      END
    ELSE FIND NEXT Úvazek RECORD IN CURRENT Učitel-učí SET
  END; ♦

```

□ Ukládání nových záznamů

Příkaz STORE

Při ukládání výskytu záznamu do DB nejprve vytvoříme výskyt záznamu v pracovní uživatelské oblasti a odtud ho příkazem STORE do DB uložíme. Příkaz má tvar

STORE T

Ukládaný záznam se stane aktuálním záznamem programu a je zařazen do všech setů, v nichž je deklarován jako člen typu AUTOMATIC.

Příklad 6.19.

Do databáze FAKULTA přidáme další výskyt záznamu učitele do záznamu UČITEL:

```
učitel.jméno:="NOVOTNÝ";
učitel.ČU:=123;
učitel.funkce:="Docent";
STORE učitel ; ♦
```

□ Modifikace záznamů

Příkaz MODIFY má formát

MODIFY T

Celý postup při modifikaci komponent výskytu záznamu znamená nejprve záznam v DB najít příkazem FIND, načíst do pracovní oblasti příkazem GET, tam opravit příslušné komponenty záznamu a konečně pomocí příkazu MODIFY zapsat opět zpátky.

Příklad 6.20.

Student Novotný se přestěhoval do ulice Zelené 23.

```
Student.jméno:="NOVOTNÝ";
FIND Student USING jméno ;
GET Student ;
Student.ulice:="Zelená 23" ;
MODIFY Student ; ♦
```

□ Zrušení záznamů

Příkaz ERASE má formát

```
ERASE T [ PERMANENT ]
        [ SELECTIVE ]
        [ ALL ]
```

vymaže výskyt záznamu z DB.

ERASE T zruší aktuální záznam programu, pokud není vlastníkem neprázdného výskytu nějakého setu. Je-li uvedeno PERMANENT a aktuální záznam programu je vlastníkem výskytů nějakých setů, pak jsou spolu s uvedeným záznamem zrušeny také všechny členské záznamy, jejichž druh členství je MANDATORY. SELECTIVE navíc zruší všechny členské záznamy s druhem členství OPTIONAL, pokud nejsou zařazeny v žádném jiném setu. ALL zruší všechny členské záznamy bez ohledu na druh členství a zařazení v jiných setech.

**CD-ROM**

Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probrané případy použití příkazů STORE, MODIFY, ERASE pojmenované:

```
Store.exe
Modify.exe
Erase.exe
```

□ Manipulace se záznamy v rámci setu

Kromě modifikace dat ve výskytech záznamů je možno také **modifikovat struktury setů**, připojovat nové členské záznamy, rušit je (ze setu, ne z DB), případně přesouvat z jednoho setu do jiného. Mechanismus připojování záznamů však je složitější. Při definici setu můžeme zadat, zda se budou záznamy připojovat

AUTOMATIC	... příkazem STORE
MANUAL	... příkazem CONNECT

Mimo to můžeme předepsat, na kterou pozici do setu se nově vkládaný záznam umístí: FIRST, LAST, NEXT, PRIOR, SORTED

➤ Zařazení aktuálního záznamu programu za aktuální výskyt zadaného setu

Příkaz CONNECT má formát

CONNECT T TO S

Příkaz nelze použít pro druh členství MANDATORY AUTOMATIC.

Příklad 6.21.

Uvažme set BYDLÍ se záznamem typu Kolej (CALC-klíč je název) jako vlastníkem a se záznamem typu Student (CALC-klíč je jméno) jako členem. Máme zařadit studentku se jménem Kosová Helena na kolej Opletalovu.

```
Kolej.název:="Opletalova";
FIND Kolej RECORD USING název;
Student.jméno:="Kosová Helena";
FIND Student RECORD USING jméno;
CONNECT Student TO Bydlí;
```

➤ Vyřazení aktuálního záznamu programu z výskytu zadaného setu

Příkaz DISCONNECT má formát

DISCONNECT T FROM S

Druh členství v setu musí být OPTIONAL.

Příklad 6.22.

Vyloučení studenta Jana Skřivánka z koleje Mánesova

```
Kolej.název:="Mánesova";
FIND Kolej RECORD USING název;
Student.jméno:="Skřivánek Jan";
FIND Student RECORD USING jméno;
DISCONNECT Student FROM bydlí;
```

➤ Přesunutí aktuálního výskytu záznamu z původního výskytu do zadaného výskytu setu.

Příkaz RECONNECT má formát


RECONNECT T TO S

Příkaz přesune aktuální výskyt záznamu z původního výskytu setu do výskytu setu aktuálního výskytu setu zadaného typu.

Příklad 6.23.

Student Novotný se přestěhoval z koleje Opletalovy do koleje Mánesovy.

```
Student.jméno:="Novotný";
FIND Student RECORD USING jméno;
Kolej.název:="Mánesova";
FIND Kolej RECORD USING název;
RECONNECT Student TO bydli
```


CD-ROM

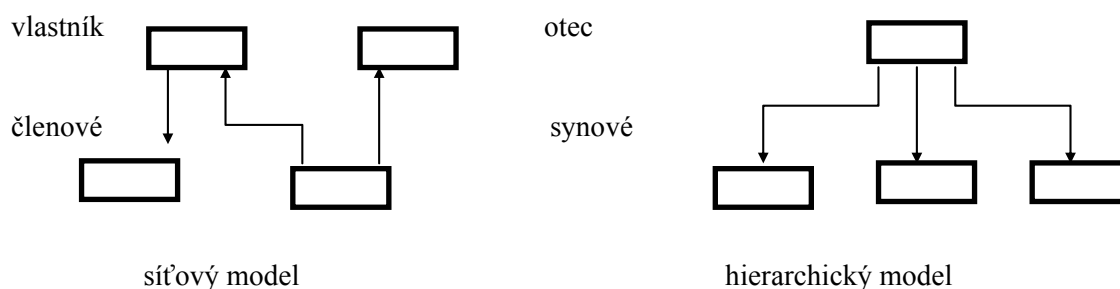
Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probrané případy použití příkazů CONNECT, DISCONNECT a RECONNECT pro setu pojmenované:

- Connect.exe
- Disconnect.exe
- Reconnect.exe

6.4. Hierarchický datový model

Hierarchický model byl používán hlavně v počátcích rozvoje databázových systémů. Je to vlastně zjednodušený síťový model. Zobrazíme-li záznamy jako uzly grafu a vazby realizované pomocí odkazů jako hrany grafu, odpovídá databázi síťového modelu obecný graf. Databázi sestavené na principech hierarchického modelu odpovídá graf zvaný strom (graf bez cyklů) nebo les (množina stromů).

Prakticky to znamená, že záznam nemůže patřit do více než jednoho setu. Při popisování hierarchického modelu se používá rodinné terminologie. Místo vlastník se užívá pojmu otec, místo členové synové.



Rozdíl mezi síťovým a hierarchickým modelem v terminologii modelu hierarchického můžeme tedy formulovat takto: v hierarchickém modelu má každý syn právě jednoho rodiče, v síťovém modelu jich může mít více.

V literatuře je často uváděn hierarchický model jako samostatný třetí typ (vedle relačního a síťového). My jej nechápeme jako nový typ a nebudeme se jím podrobněji zabývat.



Shrnutí pojmů 6

Standard pro práci se síťovým datovým modelem CODASYL.

Schéma, subschéma, typ záznamu, výskyt záznamu, typ setu, výskyt setu. Databázový klíč.

Realizace vazeb v SDM.

Jazyk pro definici dat, deklarace databáze, typů záznamů, typů setů.

Pracovní oblast uživatele.

Jazyk pro manipulaci s daty. Příkaz pro vyhledání záznamu a jeho typy. Načtení záznamu.

Uložení, modifikace a rušení záznamu.

Práce s prvky setu. Uložení, přemístění a zrušení člena setu.



Otázky 6.

1. Jaký je základní rozdíl mezi databází v relačním a síťovém datovém modelu?
2. Jak se realizují vazby v síťovém modelu?
3. Co vše se definuje pomocí JDD?
4. Co je a k čemu je pracovní oblast uživatele?
5. Jak se v síťovém datovém modelu manipuluje s daty na rozdíl od relačního modelu?
6. Jaké možnosti vyhledávání dat jsou v síťovém modelu?
7. Jak je možno urychlit vyhledávání pomocí setů?
8. Co je hierarchický datový model a jak se liší od síťového?



Úlohy k řešení 6.

1. V databázi FAKULTA deklarované v odstavci 6.2.

Typy záznamů: Učitel (ču, jméno, funkce, plat)
 Předmět (čp, název, ročník)
 Úvazek (ču, čp, hodin)
 Student (jméno, město, ulice)
 Kolej (název)

Typy setů: UČITEL_ UČÍ (Učitel, Úvazek)
 PŘEDMĚT_ UČÍ (Předmět, Úvazek)
 BYDLÍ (Kolej, Student)

vyhledejte následující informace:

- a) Jméno a funkci učitele s číslem Č17.
- b) Seznam učitelů, kteří jsou docenty.
- c) Seznam předmětů 3. ročníku.
- d) Seznam všech studentů bydlících na koleji „Studentská“.

- e) Úvazek učitele Horáka – předmět a počet hodin.
- f) Seznam učitelů, kteří učí předmět Databáze.
- g) Jména a adresy všech studentů.