

UK MFF

Katedra softwarového inženýrství

INRA

Uživatelská příručka

INRA - IMPLEMENTATION OF NESTED RELATIONAL ALGEBRA

Uživatelská příručka

VERZE 1.02, 27. LEDNA 2002

© UK MFF

Katedra Softwarového Inženýrství
Malostranské náměstí 25

11800 Praha 1

Tel.: ++420-2-21914264 • Fax.: ++420-2-21914323

Projektový tým:

Vedoucí:

RNDr. Antonín Říha, CSc.

riha@ksi.ms.mff.cuni.cz

Programátoři:

Jan Adamec

jadamec@bigfoot.com

Radan Baše

rbas6040@barbora.ms.mff.cuni.cz

Pavel Šárek

pavel.sarek@seznam.cz

Josef Špidlen

jspi6600@ss1000.ms.mff.cuni.cz

Obsah

Úvod	3	Klíčová slova	24
CO JE TO INRA	3	Typy tabulek a proměnné	24
JEDNOTLIVÉ SOUČÁSTI	3	Komentáře	25
Binární spustitelné programy	3	Příkazy	25
Příručky	4	TRANSAKCE	25
Dokumentace	4	STRUKTURA PROGRAMU	26
Internetové stránky	4	DEFINICE TYPU	26
ROZDĚLENÍ TOHOTO MANUÁLU	5	DEFINICE DOČASNÉ TABULKY	27
INRA CLIENT	6	DEFINICE FUNKCE	28
PRVNÍ SPUŠTĚNÍ	6	Volání funkce	28
POPIS PROSTŘEDÍ	7	PROHLÍŽENÍ TABULEK	28
Popis programového menu	8	Zjištění typu tabulky	29
ZÁKLADNÍ OPERACE	11	PERZISTENTNÍ TABULKY	29
Otevření programu	11	Převedení dočasné tabulky na	
Editace Programu	11	perzistentní	30
Připojení k INRA Serveru	12	Zrušení tabulky	30
Provádění programu	12	Práva	30
RÚZNÁ NASTAVENÍ	13	Reorganizace	31
Velikost písma	13	PŘIŘAZOVACÍ PŘÍKAZ	31
Lokální nastavení	13	Operace hnížděné relační algebry	31
Konfigurace INRA Serveru	15	AKTUALIZAČNÍ OPERACE	37
Změna uživatelského hesla	15	Vložení dat	37
TABLE WIZARD	15	Aktualizace dat	38
Popis	15	Mazání dat	38
Ovládání	15	PROCEDURÁLNÍ PROGRAMOVÁNÍ	38
LADĚNÍ PROGRAMU	18	IF podmínka	38
Editace breakpointů	18	WHILE cyklus	39
Dialog pro hromadnou editaci		VĚTŠÍ PŘÍKLAD	39
breakpointů	18	GRAMATIKA PROGRAMOVACÍHO JAZYKA	44
Breakpoint	19	INRA MONITOR	50
PROGRAMOVACÍ JAZYK	20	POPIS	50
JEDNODUCHÝ PŘÍKLAD KROK ZA KROKEM	20	OVLÁDÁNÍ MONITORU	50
HNÍZDĚNÁ RELAČNÍ ALGEBRA	22	Získávání informací	51
Hnízděné relace	22	Rychlé sledování stavu	52
Hnízděná relační algebra	23	Ovládání INRA Serveru	52
ZÁKLADNÍ VLASTNOSTI JAZYKA	24		
Identifikátory	24		

Úvod

V této části příručky byste se měli stručně dozvědět základní informace o projektu Implementace Hnízděné Relační Algebry – INRA.

INRA vznikla jako studentský projekt na UK MFF, jehož cílem bylo implementovat hnízděnou relační algebru na platformě WIN32. Tato volba operačního systému měla jednak zpříjemnit práci se systémem uživatelům zvyklým na grafické prostředí Windows a zároveň umožnit studentům praktická cvičení na jejich domácích počítačích.

Co je to INRA

INRA je aplikace je typu klient-server, kde klient umožňuje interaktivní práci uživatele, spojí jej se serverem pomocí TCP/IP a předává serveru jeho požadavky. Server zajišťuje jednoduché transakční zpracování, stará se o autentifikaci uživatele, kontrolu práv jeho přístupu ke konkrétním databázovým tabulkám, zpřístupní sdílená data apod. Výsledky provedených operací pošle zpět klientovi a ten je může přehledně zobrazit. Vlastní programovací jazyk umožňuje nejen operace hnízděné relační algebry (viz kapitola 3 – Programovací jazyk) rozšířené o potenční operátor a pevný bod, ale i jednoduché procedurální programování. V aplikaci je rovněž možné graficky navrhovat schéma hnízděných tabulek. K dnes již samozřejmým vlastnostem každého integrovaného vývojového prostředí patří i možnost spouštět, ladit a krokovat program.

Jednotlivé součásti

INRA se skládá z několika binárních spustitelných programů, příruček a dokumentace.

Binární spustitelné programy

- **INRAc** INRA Client – klientské integrované vývojové prostředí, se kterým se nejčastěji setká typický uživatel. Zde je prováděna editace zdrojových textů programu, jejich spouštění a ladění pomocí zaslání na server a podobně.
- **INRAs** INRA Server – databázový server hnízděné relační algebry, ke kterému se uživatelé připojují pomocí klienta INRAc po síti s využitím TCP/IP. Server je zodpovědný za provádění databázových operací, autentifikaci

uživatelé a za udržení databáze z transakčního pohledu v konzistentním stavu. INRA Server běží jako služba na vyhrazeném počítači s MS Windows 2000, případně MS Windows NT 4.0.

- **INRAMonitor** INRA Monitor – aplikace klientského typu, která slouží ke vzdálenému monitorování stavu INRA Serveru pomocí další serverové služby INRA Controlu. Tato aplikace umožňuje pomocí TCP/IP spojení s kontrolní službou na serverovém počítači a zobrazí některé informace o stavu INRA Serveru, jako například to, kolik je aktuálně připojených uživatelů. Po přihlášení pak může uživatel s administrátorskými právy vzdáleně například zastavit, spustit či restartovat INRA Server.
- **INRAControl** INRA Control – aplikace běžící společně s INRA Serverem na serverovém počítači jako služba Windows. Tato služba umožňuje připojení jednotlivým monitorům a zprostředkovává jim informace o stavu INRA Serveru. Sama pak se serverem komunikuje a ovládá jej nepřímo pomocí manažera služeb daného počítače.
- **INRASetup** Instalační program – aplikace umožňující jednoduchou a uživatelsky příjemnou instalaci a počáteční konfiguraci zvolených komponent. Při instalaci komponent INRAs a INRAControl je nutné mít administrátorská práva pro daný počítač.
- **INRAUninstall** Odinstalovací program – aplikace umožňující čisté odebrání nainstalovaných komponent INRA.

Příručky

- **Uživatelská příručka** – příručka popisující převážně ovládání klientských částí databázového systému INRA. Je zde obsažen i přesný popis implementovaného programovacího jazyka a důležité informace pro typického uživatele.
- **Administrátorská příručka** – příručka popisující instalaci, administraci, konfiguraci a údržbu převážně INRAServeru a INRAControlu. Dále pak vzdálenou administraci prováděnou uživatelem s administrátorskými právy pomocí INRA Monitoru a konfiguračního modulu v INRA Clientovi.

Dokumentace

- **Elektronická projektová dokumentace** – dokumentace obsahující popis jednotlivých tříd, metod, atributů a podobně ve formě HTML.
- **Programátorská dokumentace** – tištěná dokumentace obsahující popis zajímavých algoritmů a způsoby řešení zajímavých problémů, popis jednotlivých rozhraní, komunikačních protokolů a podobně.

Internetové stránky

- Stránky o projektu INRA na Internetu – <http://inra.webpark.cz>.

Rozdělení tohoto manuálu

Tento manuál se skládá ze čtyř kapitol:

- **Kapitola 1: Úvod** – právě ji čtete, obsahuje stručný popis projektu, jeho rozdělení na jednotlivé programy a dostupné dokumentace.
- **Kapitola 2: INRA Client** – obsahuje popis uživatelského rozhraní aplikace INRA Client, ve které uživatelé vytvářejí a spouštějí své programy. Obsahuje i popis používání grafického nástroje pro vytváření typů relačních tabulek a informace pro užívání breakpointů při ladění programů.
- **Kapitola 3: Programovací jazyk** – zde naleznete podrobný popis programovacího jazyka určeného pro vytváření programů doplněný o příklady.
- **Kapitola 4: INRA Monitor** – obsahuje popis uživatelského rozhraní aplikace pro vzdálenou kontrolu stavu a konfigurace INRA Serveru.

INRA Client

V následující části příručky se dozvíte vše o tom, jak ovládat, používat a konfigurovat toto integrované vývojové prostředí.

První spuštění

Spuštění INRA Clienta provedete po instalaci (instalace je popsána v Administrátorské příručce) jednoduše tak, že zvolíte zástupce programu INRAc v startovací nabídce menu Windows.

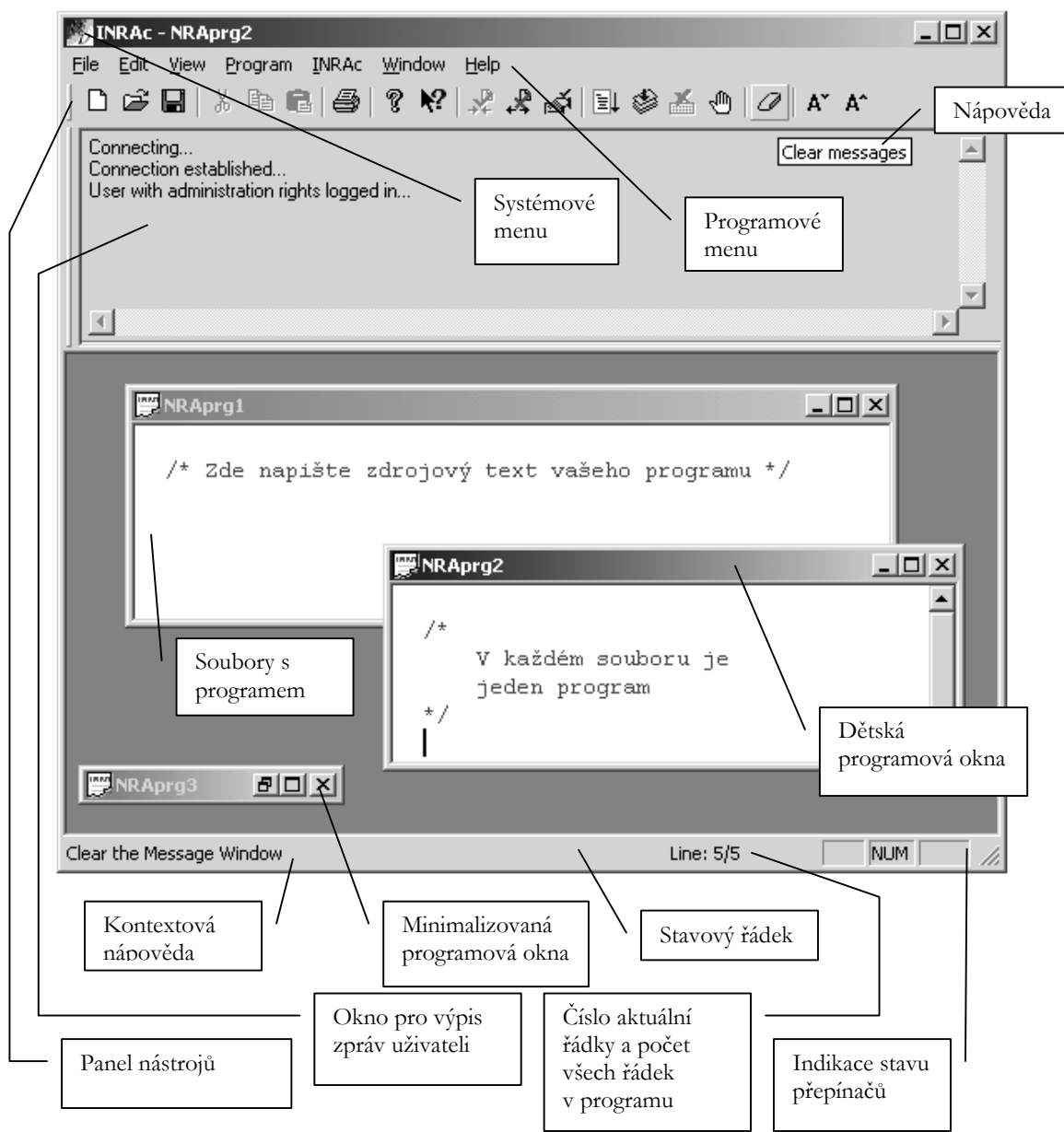
Hned poté se zobrazí úvodní obrazovka obsahující základní informace o spouštěném programu a jeho verzi. Program se v této době načítá do paměti.



V případě, že INRAc je na počítači správně nainstalován, dojde po prvním spuštění ke zkopírování globálního systémového nastavení do osobního nastavení daného uživatele, které je pak možné upravovat. Jestliže INRAc na počítači vůbec nainstalován není, dojde při prvním spuštění k automatickému vyvolání konfiguračního dialogového okna, kde je možné nastavení pro daného uživatele zadat. Takto je umožněno, aby byl tento klientský program spuštěn i na počítači, kde není řádně instalován (informace o potřebných knihovnách a ActiveX prvcích lze nalézt v Administrátorské příručce).

Popis prostředí


Okno programu INRAc lze vizuálně i logicky rozdělit na několik částí. Jako u většiny programů zhotovených pro MS Windows nalezneme zde systémové menu s ikonkou a názvem programu doplněným případně o jméno aktuálně otevřeného programu. Dále následuje typické programové menu, jehož podrobnému popisu se budeme věnovat dále. Pod ním se nachází panel nástrojů pro rychlé provedení častých operací. Hlavní okno programu obsahuje dvě části, jednu vyčleněnou pro výpis chybových, informačních, ladících a jiných zpráv a druhou pro programy napsané ve vlastním jazyce pro hnízděnou relační algebru. V této části je možno otevřít i více programů a přepínat mezi nimi. Okno je dole ukončeno stavovým řádkem, který v levé části zobrazuje aktuální kontextovou nápovědu pro položky menu a panelu nástrojů. V jeho pravé části lze zjistit číslo aktuálního řádku a sloupce v editovaném programu a stav přepínačů kláves Num Lock, Scroll Lock a Caps Lock. Stavový řádek a panel nástrojů lze případně skrýt výběrem příslušné nabídky v programovém menu. Popis prostředí je pro lepší orientaci znázorněn také na následujícím obrázku:




Popis programového menu

V následující části bude uveden jednoduchý popis menu sloužící spíše jako přehled. Záleží ale na kontextu vaší práce, které položky jsou momentálně zobrazeny a které jsou aktivní v tom smyslu, že je lze vybrat.

Menu **File**


 **New** Otevře nové okno a vytvoří nový prázdný program Ctrl + N
hnízděné relační algebry.

 **Open** Otevře existující program pomocí standardního dialogu Ctrl + O
pro otevírání souborů ve Windows. Soubory programů
pro hnízděnou relační algebru mají standardně
koncovku „.nra“, ale jedná se o obvyčejné textové
soubory se zdrojovým kódem, takže je možné je
editovat i v jiném textovém editoru a zde pouze ladit.





V případě správně nainstalovaného programu můžete
soubory „.nra“ také otevřít tak, že na ně poklepete
například v Průzkumníku MS Windows a INRAc se
automaticky otevře s daným programem. V případě
otevřeného INRAc je nový program možné otevřít také
technikou „Drag and Drop“ přenesením jména souboru
z jiného programu nad okno INRAc.

Close Uzavře aktuálně editovaný program. V případě, že
v programu jsou provedeny změny a program není
uložen, nechá si uzavření potvrdit.

 **Save** Uzavře a uloží editovaný program. V případě, že ještě Ctrl + S
není zadáno jméno souboru, do kterého se má
programu uložit, je nutné ho zadat.

Save As Systém se zeptá na jméno souboru, do kterého má F12
program uložit a uloží jej. S programem se dále pracuje
pod novým jménem.







 **Print** Vytiskne program. V případě zvolení položky z menu, Ctrl + P
zeptá se program na parametry tisku. V případě použití
ikonky dojde k automatickému tisku na výchozí tiskárně
s výchozím nastavením. Tisk z programu je
optimalizován pro tisk na papír velikosti A4 na výšku.

 **Print
Preview** Zobrazí náhled stránek tak, jak budou vytištěny.

**Print
Setup** Otevře dialogové okno k zadání parametrů tisku.

- Recent Files** Na dalším místě bude zobrazeno 6 naposledy otevřených souborů daným uživatelem. Při zvolení některého z nich dojde k jeho rychlému otevření.
- Exit** Ukončí program INRAc. V případě, že některé soubory nejsou uloženy, INRAc se zeptá, zda je chcete uložit.




Menu **E**dit

-  **U**ndo Vrátí zpět naposledy provedenou editační operaci. Program si pamatuje pouze jednu poslední editační operaci a tu lze tedy vrátit zpět. Ctrl + Z
-  **C**ut Vystřihne aktuálně vybraný kód z programu do schránky Windows ve formě čistého textu. Ctrl + X
-  **C**opy Zkopíruje aktuálně vybraný kód z programu do schránky Windows ve formě čistého textu. Ctrl + C
-  **P**aste Vloží text ze schránky Windows do programu na aktuální pozici kurzoru. V případě, že v programu je označena část kódu, je tento nahrazen obsahem schránky. Ctrl + V
-  **B**reakpoints Otevře dialog na editaci ladících bodů v programu. Jednotlivé body lze do programu vkládat vždy na konkrétní řádky a každý ladící bod může být buď aktivován nebo deaktivován. Alt + F9
-  **C**lear **M**essage **W**indow Příkaz vymaže okénko sloužící pro výpis zpráv uživatelů. Toto okénko je v některých situacích vymazáno i automaticky. (Například při spuštění programu se zprávy o překladu vypisují do čistého okna).
- T**able **W**izard Příkaz spustí pomocníka pro vytváření definic tabulek. Ctrl + T




Menu **V**iew

- T**oolbar Zobrazí nebo skryje panel nástrojů.
- S**tatus **B**ar Zobrazí nebo skryje stavový řádek.
- A*** **D**ecrease **L**etters Zmenší písmo používané v aktuálním okně.
- A*** **I**ncrease **L**etters Zvětší písmo používané v aktuálním okně.
- S**yntax **H**ighlighting Zapne nebo vypne barevné zvýrazňování syntaxe v programu.

Menu **P**rogram

-  **R**un Přeloží program a spustí jeho vykonání na INRA Serveru. Při běhu může dojít k běhové chybě (neexistující atribut, tabulka, ...). Na takové selhání programu je uživatel upozorněn dialogovým oknem. Místo výskytu chyby je graficky zvýrazněno. Při úspěšném dokončení programu je jenom vypsáno hlášení do okna zpráv a změní se stav tlačítka Stop Running. Spustit program lze pouze pokud jsme přihlášení k nějakému serveru. F5
-  **C**ompile Přeloží a zkontroluje program. Popis nalezených chyb je zobrazen v okně zpráv. První nalezená chyba je ve zdrojovém textu graficky zvýrazněna. Toto zvýraznění je zrušeno při první editační operaci zadané uživatelem. Některé chyby mohou být odhaleny až za běhu programu. F7
-  **S**top **R**unning Zastaví program běžící na INRA Serveru, případně přeruší čekání na odpověď. Podle toho, jestli je ikona aktivní, lze poznat, že běží nějaký program. V této době nelze v INRAc editovat.

Menu **I**NRAc

-  **C**onnect Vyvolá dialog pro připojení k INRA Serveru.
-  **D**isconnect Zruší spojení k INRA Serveru.
-  **S**erver **C**onfiguration Umožní administrátorovi provádět vzdálenou konfiguraci INRA Serveru.
- S**ettings Umožní editovat lokální nastavení INRA Clienta pro aktuálního uživatele Windows. Alt + F7
- C**hange **P**assword Umožní každému uživateli změnit své vlastní heslo

Menu **W**indow

- N**ew Window Otevře nové okno pro editaci aktuálního programu.
- C**ascade Utrídí všechna okna s programy kaskádovitě.
- T**ile Rozdělí okna tak, aby byla všechna nad sebou ve stejně velikých pruzích a vyplňovala celou oblast určenou pro tato okna.

Arrange Icons

V případě, že jsou okna s programy minimalizována do ikon, seřadí tyto ikonky.

Seznam otevřených oken

Zde bude zobrazen seznam aktuálně otevřených programů a výběrem některého z nich se jeho okno přesune do popředí, případně obnoví z minimalizované formy.

Menu Help

Help Topics

Otevře soubor s nápovědou.

F1



About INRAc

Zobrazí základní informace o programu INRA Client.



Tato ikonka a klávesová zkratka způsobí vyvolání rychlé nápovědy k jednotlivým ovládacím prvkům. Způsobí změnu kurzoru na šipku s otazníkem a po následném kliknutí na ovládací prvek zobrazí nápovědu přímo k němu.

Shift + F1

Základní operace

Otevření programu

Program je možné otevřít tak, jak je každý zvyklý pomocí výběru z menu **File Open**. Stejně tak lze asociovaný program s koncovkou **.nra** otevřít například poklepnutím v Průzkumníku nebo přetažením jména souboru nad pracovní plochu programu INRA Client. Založit nový soubor lze samozřejmě například pomocí výběru **File New** a hned je možné začít programovat. Programové soubory jsou obyčejné textové soubory a k jejich editaci lze tedy použít i Vaš oblíbený jiný textový editor a soubory v INRAc teprve následně otevřít k tomu, aby je bylo možné je zkompileovat a spustit na INRA Serveru.

Editace Programu

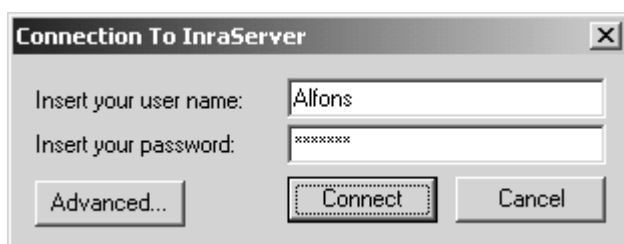
Editace programu je prováděna podobně jako ve většině textových editorů. Text programu je vlastně čistý text, který je možné psát klávesnicí tak, jak je normální uživatel zvyklý. Rozdíl oproti normálnímu textovému editoru je například v tom, že po stlačení klávesy Enter kurzor nepřejde na začátek nového řádku, ale rovnou odskočí automaticky podle předchozí řádky, což programátorovi usnadní přehledné členění zdrojového textu.

Při normálním nastavení pak program graficky zvýrazňuje syntaxi, konkrétně zobrazí definovanou barvou klíčová slova, jinou komentářové části, další pak řetězce uzavřené v uvozovkách a obyčejnou zbytek normálního textu. Nastavení těchto barev lze provést pomocí dialogu **INRAc Settings**.

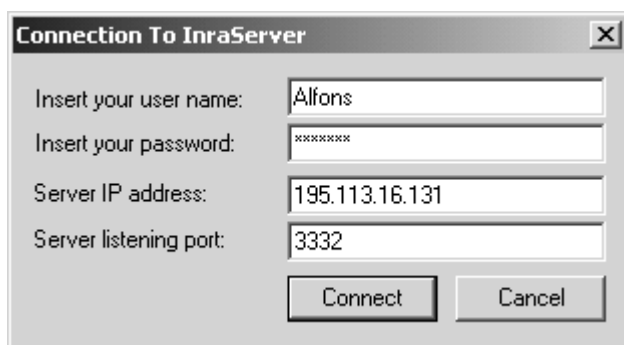
Některé editační operace lze provádět intuitivně pomocí myši a výběru příkazu z menu nebo panelu nástrojů. Jedná se zejména o příkazy práce se schránkou Windows a například také příkaz vrácení se o krok zpět.

Připojení k INRA Serveru

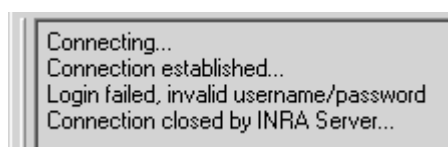
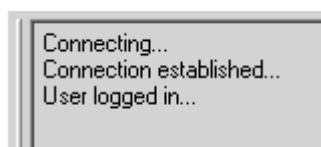
Připojení k INRA Serveru je třeba provést před spuštěním vlastního programu na tomto databázovém serveru. Tuto akci lze provést nejrychleji výběrem příslušné ikonky na panelu nástrojů nebo také pomocí výběru z menu **INRAc Connect**. Příkaz vyvolá zobrazení přihlašovacího dialogu, ve kterém musíte zadat své přihlašovací jméno a heslo, které Vám přidělil administrátor při zřizování Vašeho účtu.



Program se pokusí spojit se serverem na adrese a portu, které jsou dány Vaším lokálním nastavením. V případě, že často měníte databázové servery, na které se připojujete, lze stlačením tlačítka **Advanced** toto nastavení dočasně změnit.



Dialog ukončíte stlačením tlačítka **Connect** nebo můžete celou akci zrušit pomocí **Cancel**. Dále se pak uživateli v okénku pro zprávy zobrazí informace o tom, jak pokus o připojení dopadl. Za normálních okolností se uživatel dozví, že spojení se podařilo, ale v případě nějaké chyby bude zobrazena příslušná chybová hláška. Možné výstupy jsou tedy například tyto (vlevo je úspěšné přihlášení, vpravo neúspěšné):



Provádění programu

V případě, že jste přihlášení na server, stačí po napsání programu vybrat z menu **Program Run** (případně stlačit klávesu F5 nebo vybrat odpovídající tlačítko na panelu nástrojů) a program bude přeložen do binární formy a zaslán na server, kde budou

jednotlivé příkazy interpretovány. V případě, že chcete pouze program zkusit přeložit a zkontrolovat syntaktické chyby, stačí když vyberete **Program Compile** z menu (případně stlačíte F7 nebo použijete ikonku na panelu nástrojů) a příslušný překladač zkontroluje váš program bez toho, aby byl odeslán k vykonání na server.

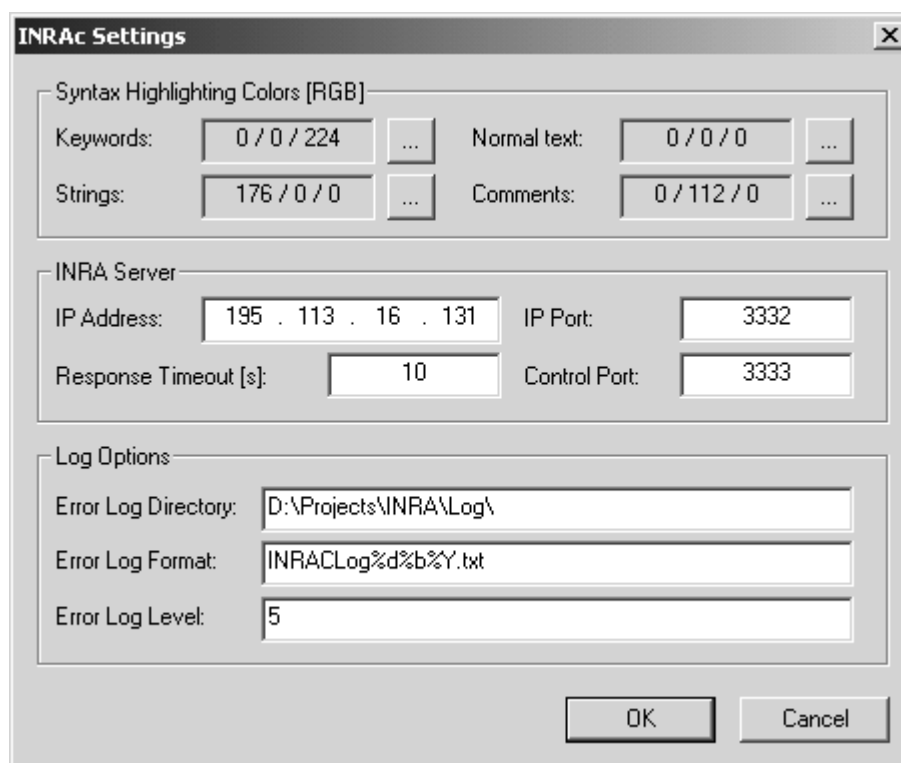
Různá nastavení

Velikost písma

V programu je standardně vytvořeno 5 různě velikých neproporcionálních fontů (Courier New), které může uživatel měnit dle libosti vybráním položky **Increase – Decrease Letters** v menu **View** nebo kliknutím na příslušnou ikonku panelu nástrojů. Nastavení je provedeno lokálně pro každý editovaný soubor zvlášť.

Lokální nastavení

Lokální nastavení programu INRA Client lze vyvolat pomocí menu **INRAc Settings** nebo pomocí klávesové zkratky **Alt + F7**. V případě, že nemáte INRAc řádně nainstalován, dialog s tímto nastavením se Vám zobrazí automaticky při prvním startu programu. Zadané nastavení je pak platné pouze pro daného uživatele systému Windows, protože je zaneseno do jeho uživatelského profilu. V případě korektní instalace programu na počítač se globální nastavení zkopíruje při prvním spuštění do uživatelského profilu a ten je pak sám sobě může měnit. Samotné nastavení se provádí v následujícím dialogu:



Konfigurační dialog má rozděleny položky celkem do 3 částí.

Nastavení barev pro zvýrazňování syntaxe

V první části lze nalézt nastavení barev pro zvýrazňování syntaxe. Zadat lze barvu klíčových slov, řetězců uzavřených do uvozovek, normálního textu a komentářů. V každém políčku je barva zobrazena ve formátu RGB, ale nelze ji editovat přímou změnou těchto hodnot. Editace se provádí po kliknutí na tlačítko vedle příslušného editačního okénka v zobrazeném dialogu pro výběr barvy (použit je standardní dialog Windows pro zvolení barvy).

Nastavení spojení k INRA Serveru

V této části je třeba nastavit především IP adresu počítače, kde je spuštěna služba INRA Serveru a případně INRA Controlu. Každá tato služba poslouchá na TCP/IP na určitém portu, který je nutné také zadat. Při standardní instalaci serverových služeb lze v těchto místech zanechat přednastavené hodnoty. Poslední položkou, která je zde uvedena, je časový limit na odpověď serveru. Limit se netýká všech jeho operací, protože některé mohou být časově náročnější a v tom případě je necháno na uživateli, kolik času INRA Serveru dá, než se jej rozhodne přerušit. Je nutné upozornit, že nastavení IP Adresy a kontrolního portu je sdílené s aplikací INRA Monitor a změny se tedy promítnou i v této aplikaci. V případě, že INRA Monitor nevyužíváte, není nastavení kontrolního portu důležité, samotný INRA Client jej nevyužívá.

Nastavení chybového logu

Zde nastavujete celkem 3 položky, jednak cestu do adresáře, ve kterém má být zřízen chybový aplikační log, dále pak formát jména souboru a úroveň chybových zpráv. Zkontrolujte především, že cesta do adresáře chybového logu je platnou cestou z hlediska vašeho počítače. Formát chybového logu je popis jména souboru, kde jsou obsaženy zkratky složené ze znaku procento a jednoho dalšího písmene. Tyto zkratky se jsou pak systémem zaměněny podle aktuálního data a času a z výsledného řetězce je vytvořeno jméno souboru, do kterého je zapisováno. Tímto lze tedy například nastavit, že soubor chybového logu se bude měnit jednou za den, týden, hodinu a podobně. Seznam nejčastěji používaných zkratk je tento:

- %b Zkrácený název měsíce
- %B Plný název měsíce
- %d Den v měsíci dekadicky jako číslo (01 – 31)
- %H Hodina ve 24 hodinovém formátu (00 – 23)
- %I Hodina ve 12 hodinovém formátu (01 – 12)
- %m Měsíc dekadicky jako číslo (01 – 12)
- %Y Rok se stoletím dekadicky jako číslo (00 – 99)

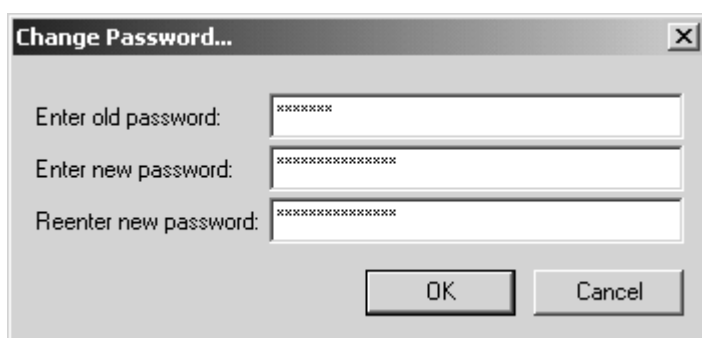
Úplný seznam těchto zkratk lze nalézt v programátorské dokumentaci. Poslední položka, kterou zde lze nastavit je úroveň chybových hlášení. Tato úroveň je číslo od 0 do 5 a určuje, které zprávy si přejete v souboru nalézt. Úroveň 0 v tomto případě znamená, že nechcete vidět vůbec nic, úroveň 1 zahrnuje chybová hlášení, ..., až úroveň 5 zaručí, že všechny zprávy budou vypsaný. Úroveň 0 není doporučována vzhledem k tomu, že v případě nějaké chyby pak není možné zpět zjistit, co se vlastně stalo. Stejně tak úroveň 5 není zcela dobrým řešením, protože v tomto případě dochází například také k hexadecimálnímu výpisu všech paketů, které procházejí mezi klientem a serverem, což je časově náročné a může dojít ke znatelnému zpomalení práce.

Konfigurace INRA Serveru

V případě, že je uživatel přihlášen k INRA Serveru a má administrátorská práva, může pomocí dialogu **INRAc Server Configuration** upravovat parametry běhu serveru, přidávat uživatele a ubírat uživatele a podobně. Popis tohoto nastavení však obsahově nespadá do této příručky a proto je uveden podrobně pouze v Administrátorské příručce.

Změna uživatelského hesla

Každý přihlášený uživatel může změnit své heslo pomocí dialogu **Change Password...** z menu **INRAc**. V tomto dialogu je třeba zadat jednak staré heslo a také dvakrát stejné nové heslo a potvrdit stlačením tlačítka OK.



Server v případě úspěšné změny odpoví:

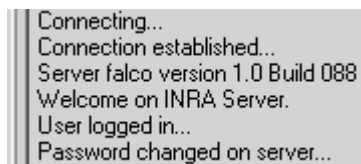


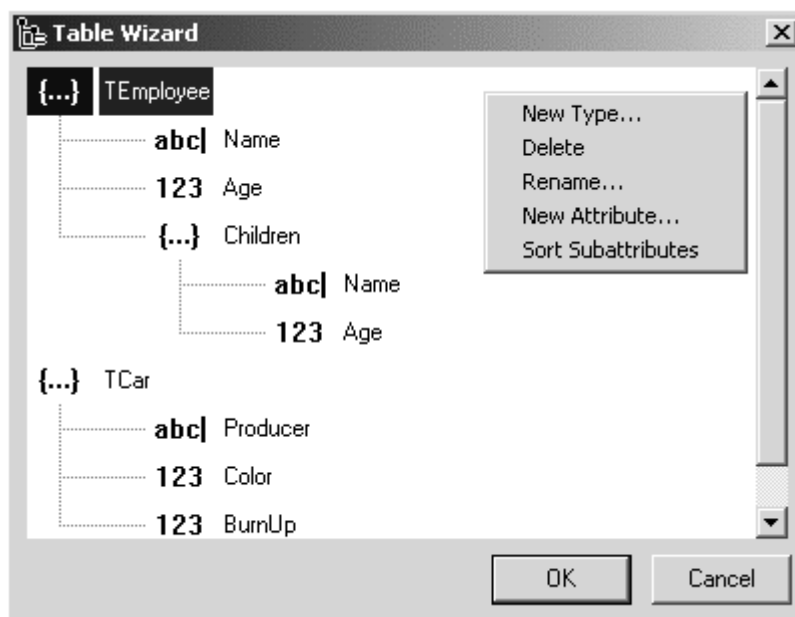
Table Wizard

Popis

Průvodce vytvářením tabulek je dialogová aplikace, která při spuštění převezme vaše definice typů tabulek (případně začne od začátku), dovolí vám graficky myší editovat jejich strukturu a nakonec vaše potvrzené změny zanesou zpět do kódu vašeho programu.

Ovládání

Průvodce vytváření tabulek spustíte výběrem **Table Wizard** z menu **Edit** nebo stlačením horké klávesy **CTRL + T**. Zobrazí se Vám dialog, ve kterém jsou pouze dvě tlačítka, **OK** a **Cancel** na potvrzení nebo vrácení provedených změn. V tomto dialogu se nachází jeden centrální ovládající prvek, který graficky zobrazuje definované typy tabulek a jejich stromovou strukturu.



Veškeré editační operace lze provádět jednotně vybráním uzlu, kterého se akce týká, a kliknutím pravým tlačítkem myši. To vyvolá kontextové menu, které obsahuje právě všechny v daném okamžiku povolené akce. Jejich výčet je tento:

Přidání nového typu tabulky

Tabulku přidáte vybráním **New Type** z kontextového menu. Dojde k zobrazení jednoduchého dialogového okna, ve kterém stačí zadat jméno typu a potvrdit akci tlačítkem **OK**. Název typu musí být mezi všemi jednoznačný a musí se skládat z písmen základní ASCII tabulky, případně může obsahovat i znak podtržení nebo číslice, ale měl by začínat písmenem. Za dva shodné názvy se v tomto případě považují i názvy lišící se pouze velikostí jednotlivých písmen (v programovacím jazyku by ovšem šlo o dva odlišné názvy typů).

Smazání typu nebo atributu

Stačí kliknout pravým tlačítkem na příslušný uzel ve stromě definic a vybrat **Delete** z kontextového menu, případně jednoduše stlačit klávesu **Delete**.

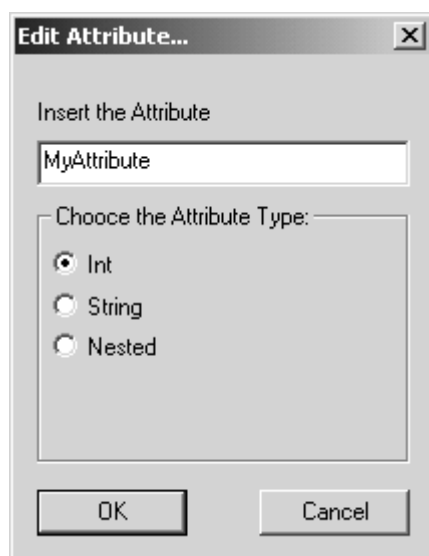
Přejmenování typu nebo atributu

Tuto akci provedete jednoduše tak, že kliknete na daný uzel a vyberete **Rename** z kontextového menu, případně jednoduše stlačíte klávesu **F2**. Při přejmenování platí samozřejmě stejná pravidla pro jméno jako při vytváření nové tabulky nebo atributu.

Přidání atributu do typu tabulky

Atribut lze přidat, jestliže aktuálně vybraný uzel představuje nějakou tabulku nebo hnížděný atribut, který tedy může mít další podatributy. Akci v tomto případě provedete vybráním **New Attribute** z kontextového menu. Dojde k zobrazení jednoduchého dialogového okna, ve kterém je nutné zadat jméno a typ atributu a potvrdit akci tlačítkem **OK**. Název atributu musí být mezi všemi atributy na stejné úrovni jednoznačný a musí se skládat z písmen základní ASCII tabulky, případně může obsahovat i znak podtržení nebo číslice, ale měl by začínat písmenem. Za dva shodné

názvy se považují i názvy lišící se pouze velikostí jednotlivých písmen (v programovacím jazyce by ale šlo o dva různé atributy).



V dialogu tedy vyplníte jméno nového atributu a zvolíte jeho typ. Podporované typy tedy jsou celá čísla, dále řetězce s maximální délkou 80 znaků, a typ hnížděného atributu.

Editace atributu

Typ a jméno atributu můžete ještě dodatečně změnit vybráním položky **Edit** v kontextovém menu. Zobrazí se Vám stejný výše uvedený dialog s přednastavenými aktuálními hodnotami. Tyto hodnoty zde můžete změnit. Speciální změny, jako třeba změna z hnížděného atributu na normální, při které dochází automaticky ke ztrátě všech podatributů, je nutné explicitně potvrdit. Program se v tomto případě dotáže, zda změnu opravdu provést.

Setřídění atributů

Posledním možným příkazem je abecední setřídění atributů typu nebo jiného hnížděného atributu. Příkaz tyto atributy seřadí podle abecedy pro Váš lepší přehled, nicméně v relační algebře není pořadí atributů definováno a není tedy zaručeno, že při různých akcích s tabulkami zůstane tak jak bylo zadáno. Server totiž může tabulky různě optimalizovat, přesouvat paměťové stránkách a vyplňovat mezery po smazaných attributech a záznamech.

Velikost celého dialogu průvodce pro návrh struktury tabulky lze měnit myší tažením za pravý dolní okraj dialogu. Je však definována jistá minimální mez, kterou tento dialog musí mít, aby bylo smysluplné jej ještě zobrazovat, a v případě, že se pokusíte dialog zmenšit ještě více, naskočí Vám automaticky jeho nejmenší možná velikost.

Ladění programu

Pro zjednodušení Vaší práce při ladění programu je v INRA Clientovi zabudována podpora přerušovacích bodů (breakpointů). Každý breakpoint je vždy svázán s určitým řádkem v programu, přesněji s první instrukcí na tomto řádku (případně s první instrukcí končící na tomto řádku u víceřádkových instrukcí). Instrukcí zde rozumíme jednoduchý příkaz ukončený středníkem. V případě nastavení breakpointu a spuštění programu je provádění zastaveno před instrukcí, na které tento breakpoint leží a je možné prohlížet obsah tabulek do té doby definovaných nebo použitých (kromě tabulek s hodnotou null). Uživatel takto může program postupně krokovat. Breakpointy umístěné mimo tělo funkce nebo hlavní blok programu jsou ignorovány. V takovém případě je v uživatelském okně zobrazeno varování.

Editace breakpointů

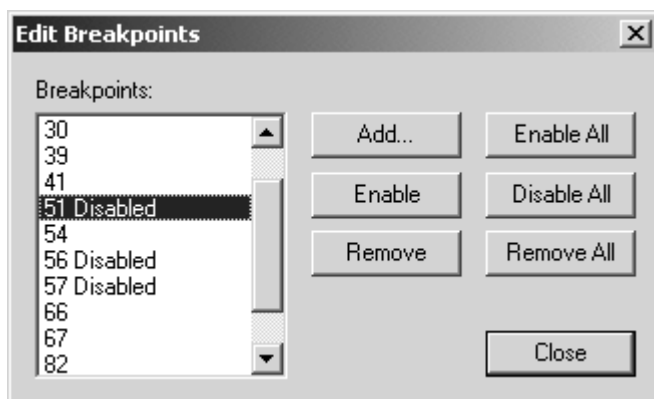
Předem je nutné upozornit, že každý breakpoint může být v jednom ze dvou stavů. Buďto je aktivován nebo deaktivován. Deaktivovaný breakpoint nemá na výkon programu žádný vliv, nicméně možnost deaktivování umožňuje uživateli například najednou dočasně vyřadit všechny definované přerušovací body, spustit program celý, a pak se zase vrátit jedním příkazem zpět do stavu, kdy všechny breakpointy byly aktivní.

```
navstevy_sel = navstevy(id_pacienta < 1000) :
```

Editaci breakpointů lze provádět dvěma způsoby. Nejjednodušší možností je nastavit kurzor v programu na řádek, jehož stav chceme změnit, a kliknout myší na ikonku ruky. První kliknutí přidá aktivní breakpoint, druhé jej deaktivuje a třetí ho pak zase zcela odstraní. Pro komplexnější správu breakpointů je možné použít následující dialog.

Dialog pro hromadnou editaci breakpointů

Dialog, ve kterém lze hromadně upravovat přerušovací body, lze vyvolat vybráním **Edit \Breakpoints** z menu nebo stlačením horké klávesy **Alt + F9**. Po této akci dojde k zobrazení následujícího dialogu:



V tomto dialogu lze nalézt seznam aktuálně definovaných breakpointů s poznámkou o tom, který je aktivní a který ne. V případě vybrání nějaké breakpointu je možné změnit okamžitě jeho stav pomocí tlačítka **Enable** případně **Disable**. Každý breakpoint lze

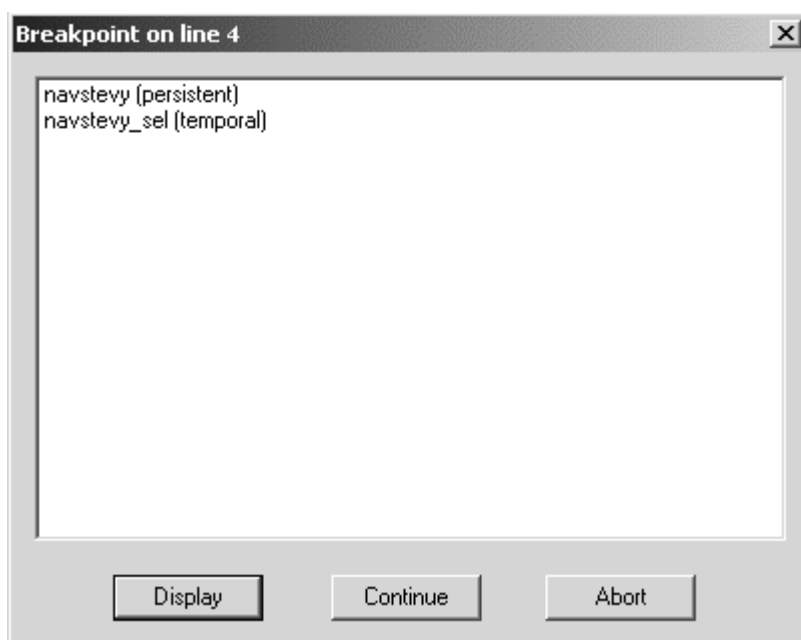
také samozřejmě smazat a to tlačítkem **Remove**. Přidání nového breakpointu lze provést jednoduše kliknutím na tlačítko **Add** a vyplněním čísla řádku v dialogu pro zadání nového breakpointu. Zároveň je možné hned určit, zda nový breakpoint má být aktivní nebo ne.

V pravé části tohoto dialogu se nalézají tři tlačítka pro hromadnou akci se všemi přerušovacími body. Tyto body lze najednou aktivovat, deaktivovat nebo odstranit. Pozor, tato akce se provádí vždy ihned a není čekáno na ukončení celého dialogu. Odstranění všech breakpointů tedy již nelze vzít zpět jinak než tím, že se všechny breakpointy postupně ručně vyrobí.

Dialog lze uzavřít pouze jediným tlačítkem **Close**.

Breakpoint

Při zastavení běhu programu na určitém breakpointu je zobrazen dialog:



V titulku je zobrazeno, na které řádce došlo k přerušení. V seznamu jsou uvedeny dosud vytvořené temporální tabulky (kromě těch s hodnotou null) a také čtené nebo zapisované persistentní tabulky. Ve funkcích jsou zobrazeny pouze tabulky dostupné v dané funkci. Napřed je uveden název a za ním v závorkách typ tabulky (persistentní, dočasná nebo parametr funkce). Uživatelem může zobrazené tabulky prohlížet buď dvojklikem na jméno tabulky, nebo vybráním jména a stlačením tlačítka Display.

Po skončení prohlížení tabulek je možné buď pokračovat v provádění programu tlačítkem Continue, nebo provádění programu zrušit tlačítkem Abort (v tom případě jsou všechny v programu dosud provedené změny vráceny do původního stavu).

Programovací Jazyk

Tato část příručky je věnována popisu programovacího jazyka, který je v programu využíván pro zápis hrůzděné relační algebry.

Jednoduchý příklad krok za krokem

V této části je podrobný návod jak vytvořit a spustit velmi jednoduchý program v relační algebře. Popis v této sekci je pouze informativní, podrobný popis používaných příkazů lze najít v dalších částech této kapitoly.

Trochu podrobněji: vytvoříme program, který vytvoří temporální datovou tabulku s námi definovaným obsahem. Jednoduchou selekcí z ní vybereme některé řádky, projekcí některé sloupce a výsledek zobrazíme.

Po spuštění INRA Clienta vytvořte nový soubor příkazem **File\New** (alternativně tlačítkem na panelu nástrojů nebo klávesovou zkratkou **Ctrl + N**). Otevře se nové okno se zatím nepojmenovaným dokumentem, který začneme editovat.

První potřebnou částí programu je definice typu tabulky. Tato definice se provádí nezávisle na vytvoření tabulky, která definovaný typ používá. Typ lze potom použít pro více různých tabulek. Napište následující kód:

```
type TZamestnanec =  
{  
    int id;  
    string jmeno;  
    int plat;  
}
```

Tím definujeme typ, který má tři sloupce (atributy): `id` s (atomickým) typem celé číslo (4 byte), `jmeno` s (také atomickým) typem řetězec (posloupnost znaků délky nejvýše 80) a číslem reprezentovaný `plat`.

Nyní definujeme dočasnou (temporální) tabulku s typem `TZamestnanec` a rovnou do ní vložíme dva záznamy (řádky). Zadejte za definicí typu text:

```

table zamestnanci : TZamestnanec =
{
    1, "Jan Novak", 11499;
    2, "Petr Placak", 11501;
}

```

Jednotlivé položky (atributy) záznamu jsou odděleny čárkami, celý záznam je ukončen středníkem. (Poznámka: zamestnanci je ve skutečnosti proměnná, která může dostat nový obsah. Takto vytvořená tabulka existuje pouze po dobu běhu programu. Na persistentní bychom ji mohli změnit přidáním příkazu „save zamestnanci“ do hlavního bloku programu. V tomto případě by bylo důležité, že název tabulky neobsahuje velká písmena. Tento příkaz by selhal, pokud by aktuální uživatel již tabulku daného jména měl. Takto vytvořenou persistentní tabulku bychom potom v nějakém jiném programu použili pouhým napsáním jejího jména.)

Nyní napíšeme hlavní blok programu (main block):

```

{
    Vysl = zamestnanci(plat > 11500); // selekce (1)
    Vysl = Vysl[id, jmeno];          // projekce (2)
    display Vysl;
}

```

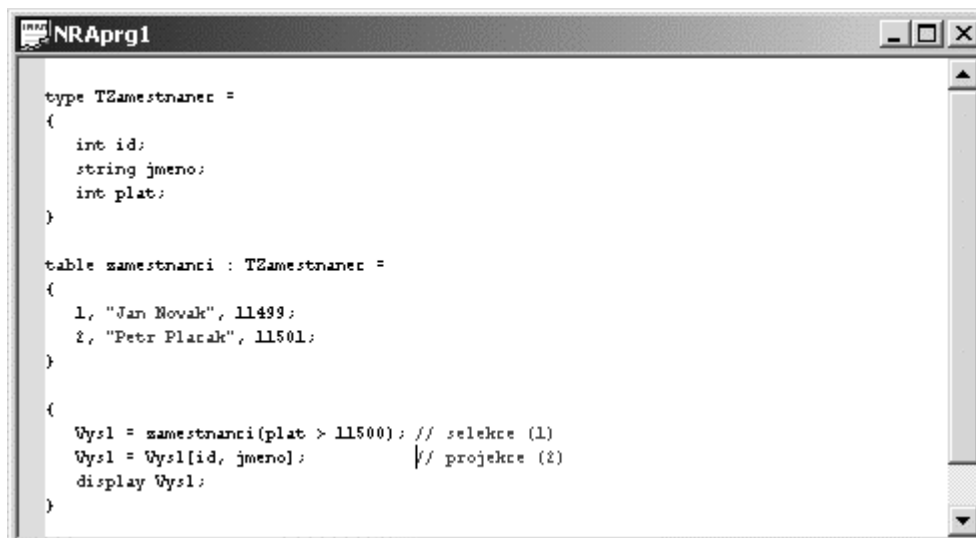
První příkaz je selekce, která vybere z tabulky zamestnanci záznamy s hodnotou ve sloupci plat větší než 11500. Text za dvěma obrácenými lomítky je komentář. Výsledek operace je přiřazen do tabulky Vysl, jejíž typ je určen implicitně podle přiřazovaných dat. Druhý řádek z tabulky Vysl vybere pouze sloupce id a jmeno (projekce). Výsledek je potom přiřazen do proměnné Vysl, jejíž předchozí obsah je tím ztracen. Poslední příkaz způsobí zobrazení tabulky Vysl. První dva řádky jsme mohli napsat do jednoho příkazu například (s použitím negativní projekce) takto:

```

Vysl = zamestnanci(plat > 11500)[^ plat]; // místo 1 a 2

```

Okno s programem by nyní mělo vypadat přibližně takto:



```

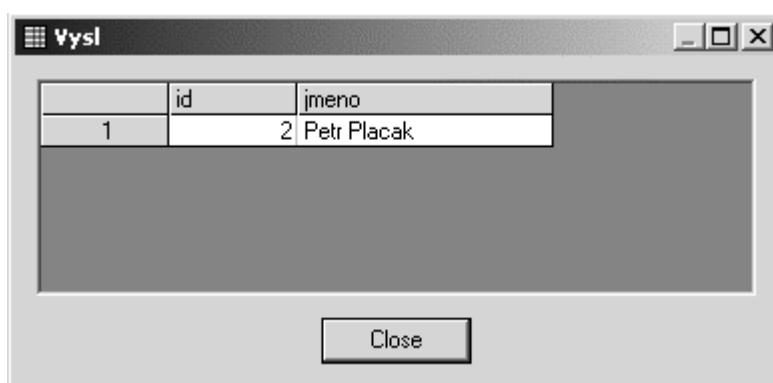
NRAprg1
type TZamestnanec =
{
    int id;
    string jmeno;
    int plat;
}

table zamestnanci : TZamestnanec =
{
    1, "Jan Novak", 11499;
    2, "Petr Placak", 11501;
}

{
    Vysl = zamestnanci(plat > 11500); // selekce (1)
    Vysl = Vysl[id, jmeno];          // projekce (2)
    display Vysl;
}

```

Teď se přihlásíme k INRA Serveru příkazem **INRAc\Connect** (viz 2. kapitola). Nyní (vytvořený dokument musí být aktivní) spustíte program příkazem **Program\Run** (nebo klávesou **F5**). V okně zpráv se objeví hlášení o úspěšné kompilaci a zahájení provádění programu na serveru. Po nějaké době by měl klient začít stahovat data tabulky ze serveru (pokud zobrazení dialogu selže, možná máte chybně provedenou instalaci a chybí Vám ActiveX prvek MSFlexGrid, viz Administrátorská příručka) a nakonec by se měl objevit tento dialog:



(Poznámka: případné hnížděné tabulky (hnížděné atributy, atributy s neatomickým typem) bychom zobrazili dvojklikem na nápis „Nested table“.) Dialog uzavřeme tlačítkem Close. Tím program skončil a v okně zpráv se zobrazí hlášení o úspěšném ukončení.

Hnížděná relační algebra

Hnížděné relace

Teorie hnížděných relací vznikla jako reakce na nedostatky běžně používaného relačního modelu, který je kritizován hlavně pro svou strukturální chudobu (neexistence datového typu s vnitřní strukturou, nutnost štěpení objektů), z čehož vyplývá velká vzdálenost relačního schématu od konceptuálního návrhu. Základní myšlenou hnížděného (nested) přístupu je opuštění požadavku první normální formy relace, tj. elementárnosti datových typů.

Relace v první normální formě – někdy pro zdůraznění nazývaná plochá (flat) relace – se často vysvětluje jako vzniklá z elementární domény použitím dvou konstruktorů – *n*-tice a množina: je to množina *n*-tic. U nenormalizovaných relací se může zmíněná dvojice konstruktorů použít opakovaně – hodnotou (hnížděného) atributu může být (hnížděná) relace, takže obecně dostáváme hierarchickou (stromovou) strukturu.

Jednoduchý příklad pro ilustraci: mějme dvouatributové schéma, kde první atribut je **atomický** (elementární, tedy na příklad číslo nebo řetězec) a druhý **hnížděný**, sestávající se opět z jednoho atomického a jednoho hnížděného. Oborem hodnot všech atomických atributů A, B, C necht' je zde pro jednoduchost množina celých čísel. Výsledné schéma může vypadat následovně:

$$\{A: \text{Int}, A1: \{ B: \text{Int}, A2: \{ C: \text{Int} \} \}$$

V tomto zápisu, který je podobný zápisu používaném v INRA Clientu, je tedy nutné pojmenovávat i hnížděné atributy. Existují i jiné varianty, které jsou spolu s přesnými definicemi přednášeny na MFF UK v předmětu „Současné databázové modely“ nebo uvedeny v literatuře (např. J. Paredaens: The structure of the relational database model, Springer, 1989).

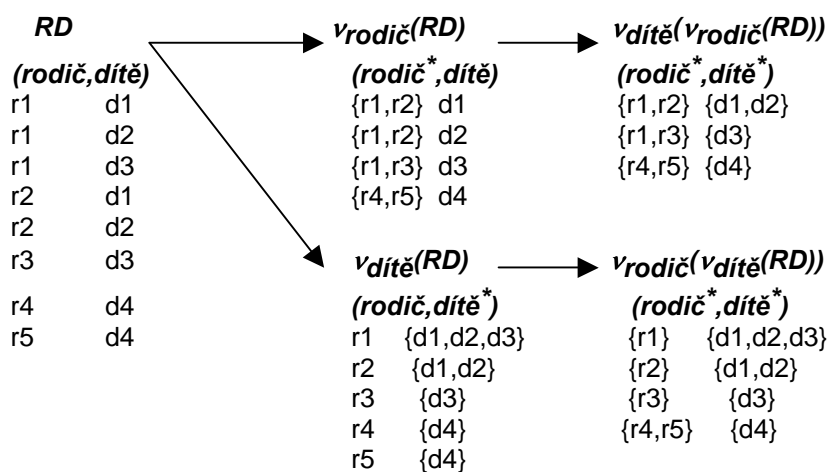
Hnížděná relační algebra

Hnížděná relační algebra je rozšířením běžné relační algebry pro hnížděné relace. K operacím sjednocení, rozdíl, kartézský součin, přejmenování, selekce a projekce přibývají další dvě operace týkající se hnížděných atributů. Jedná se o **hníždění** a **odhníždění** – tedy vytvoření (resp. odebrání) struktury nad množinou atributů. Jejich definice podle Paredaense:

Nechť X je podmnožina atributů ze schématu Ω . Pak **hníždění** $\nu(\omega; X)$ je instance nad schématem $\Omega' = \Omega \setminus X \cup \{X\}$ rovná
 $\{t \in T_{\Omega'} \mid \exists t' \in \omega: t[\Omega \setminus X] = t'[\Omega \setminus X] \ \& \ t(\{X\}) = \{t'[\{X\}]\} = \{t'[\{X\}] \mid t' \in \omega \ \& \ t'[\Omega \setminus X] = t'[\Omega \setminus X]\}$

Nechť $X \in \Omega \setminus U$ (mimo atomické atributy). Pak **odhníždění** $\mu(\omega; X)$ je instance nad schématem $\Omega' = \Omega \setminus \{X\} \cup X$ rovná
 $\{t \in T_{\Omega'} \mid \exists t' \in \omega: t[\Omega \setminus \{X\}] = t'[\Omega \setminus \{X\}] \ \& \ t[X] \in t'(\{X\})\}$

Příklad: vztah rodič – dítě v relaci s názvem RD:



je vidět, že hníždění není komutativní

Oproti relační algebře se také musí upravit přesná adresace atributů, protože v několika různých větvích schématu se mohou vyskytnout dva atributy se stejným jménem. V INRA Clientu se atributy adresují obvyklou tečkovou notací, v příkladu na předchozí straně s atomickými atributy A, B a C je možné se na atribut C dostat zápisem A1.A2.C.

Pro detailnější popis hnížděné relační algebry odkazujeme opět na předmět „Současné databázové modely“.

Základní vlastnosti jazyka

Identifikátory

Identifikátor je posloupnost začínající písmenem a pokračující písmenem, číslicí nebo podtržítkem. Identifikátory nemohou obsahovat české znaky, pouze 7bit ASCII. Nesmí být shodné s žádným klíčovým slovem. Všechny identifikátory jsou case-sensitive, záleží u nich, jestli použijeme malé nebo velké písmeno. Dva identifikátory lišící se jen velikostí písmen jsou různé.

Klíčová slova

U klíčových slov jazyka nezáleží na použití malých nebo velkých písmen, jsou case-insensitive. Klíčová slova budou v následujícím textu vždy zvýrazněna tučně. Seznam všech 63 klíčových slov:

abort	double	nest	set
addcolumn	else	nested	sorted
aggregation	fixedpoint	not	string
and	for	null	subset
arithmetic	from	or	sum
asc	function	powerset	table
avg	if	propersubset	tables
by	insert	public	to
call	int	reorganize	type
count	intersect	return	union
crossproduct	into	rightnone	unnest
delete	max	rightread	update
desc	member	rights	values
difference	min	rightundefined	where
display	naturaljoin	rightwrite	while
distinct	neg	save	

Typy tabulek a proměnné

Základním objektem, se kterým jazyk pracuje jsou tabulky. Pouze tabulky mohou být přiřazovány do proměnných.

INRA Server umožňuje pracovat se dvěma druhy relací (tabulek): s perzistentními a dočasnými. Perzistentní tabulky jsou uchovány na serveru a můžete k nim kdykoli přistupovat (pokud na to má právo). Naproti tomu, dočasné tabulky fyzicky existují jen po dobu běhu programu a slouží pro dočasné uchovávání výsledů nějaké operace a následné použití v další. Přístup k perzistentním tabulkám je zajištěn pomocí zámků a nemůže tedy dojít ke kolizím, pokud by s tabulkou pracovalo více uživatelů současně.

Každá tabulka je identifikována a adresována jménem (tj. identifikátorem). Identifikátory persistentních tabulek musí být povinně malými písmeny. Tato jména musí být pro tabulky jednoho uživatele jednoznačná. Při přístupu k persistentním tabulkám jiných vlastníků (vlastníkem tabulky je uživatel, který ji vytvořil) je třeba jméno tabulky prefixovat jménem uživatele a dvojtečkou:

```
jmeno_vlastnika_tabulky : jmeno_tabulky
```

Temporální tabulky (resp. proměnné pro ně) můžeme pojmenovat libovolným identifikátorem. Tabulky jsou lokální v bloku, kde jsou vytvořeny. Tzn. jsou viditelné v tomto bloku a jeho podblocích. (Tabulky s explicitně zadaným typem, viz dále, jsou přístupné v main bloku a jeho podblocích.) V podblocích nelze jména z nadřazených bloků zakrýt. Při opuštění bloku, kde byla proměnná poprvé definována tato proměnná zaniká. Pro předávání výsledků z podbloků (např. u if) do nadřazeného bloku je proto třeba použít jméno (proměnnou) z nadřazeného bloku. Proto existuje možnost vytvořit proměnnou bez obsahu a typu:

```
jmeno_proměnné = null;
```

Null tabulky lze pouze zapisovat, nikoli číst. Předchozím příkazem lze také smazat obsah z již existující proměnné.

Je možné, že dočasné tabulky mohou zakrýt persistentní tabulky. V takovém případě se lze k persistentní tabulce dostat prefixováním jménem vlastníka (tj. aktuálního uživatele).

Komentáře

Komentáře jsou stejné jako v C++, je možné použít buď komentář do konce řádku pomocí // nebo libovolný kus textu (i přes více řádků) ohraničit /* */.

Příklad:

```
// komentář do konce řádku
tabulka1 = tabulka2; /* tady začíná komentář
...
tady komentář končí */
tabulka3 = tabulka1 naturaljoin tabulka2<attr -> new_name>;
```

Příkazy

Každý příkaz musí být ukončen středníkem. Základním příkazem je přiřazení, které začíná jménem dočasné tabulky a za rovnítkem následuje přiřazovaný výraz. Ostatní příkazy začínají klíčovými slovy. Existují také dva příkazy pro řízení toku programu: if a while (po těch je ovšem středník nepovinný). Bloky příkazů jsou uzavřeny ve složených závorkách. Podrobnější popis jednotlivých příkazů je dále v této kapitole.

Předem upozorníme, že většina kontrol se provádí až za běhu programu na serveru. Například existence a odpovídající typ atributu, existence persistentní tabulky. Kompilace programu na klientovi odhalí pouze některé sémantické chyby (a samozřejmě všechny lexikální a syntaktické).

Transakce

Transakcí se rozumí jeden celý program zasláný z klienta na server. V rámci transakce se automaticky zamykají persistentní tabulky (pro čtení nebo zápis), které program potřebuje. Pokud se při vykonávání programu vyskytne chyba (například odkaz na neexistující atribut, apod.), server právě probíhající transakci zruší, a tím pak všechny

tabulky, které do té doby program změnil, vrátí zpět do původního stavu. Pokud program skončí bez chyby, jím provedené změny se stávají trvalými.

Transakci můžete také explicitně ukončit selháním (změny se nepotvrdí) v kódu programu pomocí příkazu:

```
Abort;
```

Například:

```
if (count(tbl1) > 1000)
{
    abort;          // pokud je v tbl1 více než 1000 řádků
}
```

Struktura programu

Program je strukturován do dvou částí. V první jsou umístěny definice typů, funkcí a těch dočasných tabulek, kterým explicitně přiřazujeme typ (ostatní dočasné tabulky vznikající při běhu programu získávají svůj typ implicitně podle přiřazovaných dat).

Definice typů, tabulek i funkcí (viz dále) může být přirozeně více. Mohou se libovolně míchat, ale je samozřejmě nutné napřed definovat typ, než ho použijeme pro nějakou tabulku. Podobně, když funkce volá nějakou jinou funkci, musí tato funkce být již definována (rekurzivní není povolena, forward deklarace proto nejsou potřeba).

```
definice
{
    vlastní kód programu, posloupnost příkazů ukončených ;
}
```

Druhou částí programu je hlavní blok programu, uzavřený ve složených závorkách. Skládá se z posloupnosti instrukcí, z nichž každá je ukončena středníkem.

Definice typu

Definice typu má následující strukturu:

```
Type jméno_definovaného_typu = { popis_typu };
```

Je tedy uvozena klíčovým slovem **Type**, po kterém je uvedeno jméno definovaného typu. Syntaxe typu tabulky je díky hníždění rekurzivní. Jednoduchými typy jsou **Int** a **String**, za kterými následuje jméno. Hnížděný atribut je uvozen klíčovým slovem **Nested** a za jménem následují složené závorky s typem. Tabulka jako taková je implicitně **Nested**, proto je vše zabaleno ve složených závorkách. Atributy jsou od sebe odděleny středníky (po definici hnížděného atributu není středník povinný).

Každý typ tabulky (tedy i hnížděného atributu) musí obsahovat alespoň jeden atribut.

Dva typy jsou navzájem kompatibilní, pokud se shoduje počet, typ i jména všech atributů.

Definice typu potom může vypadat například takto:

```
Type TPacient =
{ Int id_pacienta;
  String jmeno;
  Int datum_narozeni;
  Int cislo_pojistovny;
  Nested navstevy {
    Int cislo_navstevy;
    Int datum_navstevy;
    Nested vysetreni {
      Int kod_vysetreni; Int kod_nalezu; Int zavaznost;
    };
  };
};
```

Definice dočasné tabulky

Dočasnou (temporální) tabulku definujeme takto:

```
Table jméno_tabulky : identifikátor_typu =
{
  ... data ...
};
```

Data, která mají být v tabulce hned po jejím vytvoření jsou pro každý hnížděný atribut opět ve složených závorkách, záznamy v nich jsou odděleny středníkem a hodnoty atributů tabulky odděleny čárkami. Řetězce se ohraničují uvozovkami. Pořadí atributů je dané z definice typu. Prázdnou tabulku (i hnížděnou) vytvoříme napsáním samotných složených závorek.

Tabulky a hnížděné atributy jsou standardně multimnožiny, jeden záznam se v nich proto může vyskytovat i vícekrát.

Příklad:

```
Table pacienti : TPacient =
{ 1, "Josef Novák", 761025, 210,
  { 1, 20000517, {741, 514, 2; 745, 502, 3; 474, 513, 4; };
    2, 20000615, {741, 514, 1; 745, 503, 1; 747, 513, 4; };
    3, 20010130, {716, 512, 4; };
  };
  2, "Tomáš Vykouk", 520618, 210,
  { 1, 20001005, {741, 512, 7; 314, 452, 5; }
    2, 20001104, {741, 512, 6; 314, 452, 5; 232, 501, 3; };
  };
  3, "Pavel Maňas", 640229, 215, { /* empty */ };
};
```

Další operace manipulující přímo s daty (aktualizační operace) najdete v samostatné podkapitole dále.

Definice funkce

Často používané části kódu je výhodné zapouzdřit do funkce, kterou nadefinujeme :

```
Function jméno_funkce(jméno_par1, jméno_par2, ...)  
{  
    ... příkazy těla ...  
    return jméno_tabulky;  
}
```

Parametry funkce (oddělené čárkami) jsou celé tabulky a samotná funkce také vždy právě jednu tabulku vrací. Funkce musí mít vždy alespoň jeden parametr. Uvnitř funkce se mohou používat jen tabulky, které jí byly předány jako parametr nebo tabulky vzniklé uvnitř funkce jako výsledek nějaké operace. Parametry funkce v ní nelze nijak modifikovat. Typ parametrů funkce není nikde zadáván ani kontrolován. Skutečný stav se zjistí až za běhu programu podle typů předaných skutečných parametrů.

Posledním příkazem těla funkce musí být příkaz **return** s parametrem, který označuje, jaká tabulka se má vrátit jako výsledek. Na jiném místě **return** být nemůže.

Volání funkce

Z programu je možné funkci zavolat příkazem **call** se syntaxí:

```
Call jméno_funkce(parametr_1, param_2, ...)
```

Za klíčovým slovem **call** (pozor, při jeho vynechání se kompilátor bude snažit kompilovat selekci) následuje identifikátor funkce a za ním v závorkách skutečné parametry (tabulky) oddělené čárkami. Jejich počet musí odpovídat počtu parametrů v definici funkce.

Výsledkem je jedna tabulka. Výše uvedenou konstrukci lze použít ve výrazu u přiřazení kdekoliv, kde by mohlo být jméno tabulky. Například:

```
Ukoly = Call PridelPraci(zamestnanci, Projekty);
```

Volat funkci z jiné funkce je možné, ale volaná funkce musí být v programu definovaná již dříve. Rekurze není umožněna.

Prohlížení tabulek

Pomocí příkazu **display** si můžete prohlížet obsah tabulky. Po stažení dat, které můžete zrušit tlačítkem Abort na dialogu (to ale způsobí abort celého programu), se zobrazí dialogové okno s atributy na nejvyšší úrovni v tabulce. Na obsah hnízděných atributů (Nested table) se dostanete poklepnutím (double click levým tlačítkem myši) na jednotlivé buňky tabulky (s nápisem „Nested table“).

	cislo_navste	datum_navs	vysetreni
1	1	20000517	Nested table
2	2	20000615	Nested table
3	3	20000718	Nested table
4	4	20000919	Nested table

Je možné nechat si tabulku zobrazit i setříděnou, přidáním sekce **sorted by** s vyjmenováním sloupců tabulky a způsobem třídění.

Syntaxe:

Display jméno_tabulky;

Display jméno_tabulky **sorted by** atr1 **Asc/Desc**, atr2 **Asc/Desc**;

Při třídění se uvádějí atributy podle významnosti od nejdůležitějších spolu a jestli se má třídít vzestupně – **Asc** nebo sestupně – **Desc**. Pořadí zobrazených sloupců (atributů) není možné ovlivnit.

Příklad:

```
display pacienti sorted by datum_narozeni Desc,
                        navstevy.datum_navstevy Asc,
                        jmeno Asc;
```

Zjištění typu tabulky

Dialogové okno se schématem libovolné tabulky se zobrazí příkazem:

display type for jméno_tabulky;

Strukturu tabulky ale nejde pomocí tohoto dialogu nijak měnit.

Perzistentní tabulky

Perzistentní tabulky (tabulky, které jsou uchovány trvale na serveru) vyžadují speciální operace pro jejich využívání jinými uživateli. U perzistentní tabulky můžete například ostatním uživatelům nadefinovat přístupová práva. Pokud vy taková práva k cizí tabulce (tj. tabulce jiného uživatele) máte a budete je chtít ve svém programu použít, musíte ji označovat „jméno uživatele: jméno tabulky“.

Poznámka: každý identifikátor tabulky v programu, který není znám (tabulka v něm není vytvořena), je považován za identifikátor vaší perzistentní tabulky na serveru.

Převedení dočasné tabulky na perzistentní

Vytvoření nové perzistentní tabulky se provádí příkazem **save**:

```
Save jméno_dočasné_tabulky;
```

Tento příkaz vytvoří na serveru pod aktuálním uživatelem perzistentní tabulku se jménem a obsahem, které měla dočasná tabulka.

Nahradit obsah již uložené perzistentní tabulky nějakou jinou tabulkou lze příkazem:

```
Save jméno_tabulky to jméno_existující_persistentní_tabulky;
```

K tomu je samozřejmě potřeba právo zápisu na měněnou perzistentní tabulku (viz dále).

Zrušení tabulky

Perzistentní tabulku může zrušit buď její vlastník nebo uživatel s administrátorskými právy pro správu serveru příkazem **delete**.

```
Delete jméno_tabulky;
```

Práva

Vzhledem k tomu, že je možné přistupovat k perzistentním tabulkám jiných uživatelů, má každá perzistentní tabulka definovaná přístupová práva. Jako vlastník můžete nastavit práva jednotlivým uživatelům a pak zvlášť pro zbylé uživatele (**public**). Možnosti jsou „nedefinováno“, „žádná práva“, „jen čtení“ a „čtení i zápis“. Pokud má uživatel u sebe právo „nedefinováno“, použije se hodnota u záznamu **public**. Vlastník tabulky a každý uživatel s administrátorskými právy pro správu serveru má vždy právo „čtení i zápis“. Po vytvoření nové tabulky je pro **public** nastaveno právo „žádná práva“, práva jednotlivých uživatelů jsou „nedefinována“.

Vlastník může použít k nastavení práv příkaz **Set Rights**.

```
Set Rights jméno_tabulky ( typ_práva -> jméno_uživatele );  
nebo  
Set Rights jméno_tabulky ( typ_práva -> public );
```

kde typ práva je jedno z **RightWrite**, **RightRead**, **RightNone** a **RightUndefined**. (Poznámka: typ **RightWrite** znamená právo „čtení i zápis“, jednotlivé typy se navzájem vylučují – každý uživatel má nastaveno právě jedno právo.)

Pro výpis již definovaných práv k nějaké tabulce může její vlastník použít příkaz **Display Rights**, který po zpracování zobrazí dialogové okno se seznamem definovaných uživatelů a jejich práv. Stejný příkaz, ale použitý na cizí tabulku, zobrazí jaká práva k ní máte.

K vypsání všech tabulek ode všech uživatelů, ke kterým máte jakékoli právo, existuje příkaz **Display Tables** (bez parametrů). Seznam tabulek se opět zobrazí ve zvláštním dialogovém okně.

```
Display rights for jméno_tabulky;  
Display tables;
```

Reorganizace

Reorganizace slouží jen pro příležitostnou minimalizaci prostoru potřebného k uchování perzistentní tabulky, při které se odstraní nevyužitá místa vzniklá při mazání dat.

```
Reorganize jméno_tabulky;
```

Přiřazovací příkaz

Základní operací je přiřazovací příkaz se syntaxí:

```
temporální_tabulka = výraz_relační_algebry;
```

Na levé straně je identifikátor existující dočasné tabulky (proměnné) nebo (pro tabulku) dosud nepoužitý identifikátor. Případný starý obsah proměnné je zahozen a po provedení přiřazení dostane nový obsah. Typ proměnné se automaticky přizpůsobí obsahu.

Na pravé straně je (potenciálně velmi složitý) výraz rozšířené relační algebry, jehož popis následuje dále. Nejjednoduším výrazem je jméno již existující tabulky, například:

```
pacienti_kopie = pacienti;
```

Operace hnížděné relační algebry

Většina operací vrací jako svůj výsledek tabulku. Proto můžeme z takové operace skládat a vytvářet tak složitější výrazy. Precedence jednodušších operací z následující kapitoly je předem stanovena, u ostatních je implicitně dána jejich syntaxí, která jakékoli nejasnosti vylučuje.

Množinové operace

Základní operace relační algebry mají velice jednoduchou syntaxi, proto je popíšeme jen stručně a nebudeme jednotlivě rozebírat na příkladech. Všechny mají dva parametry.

U operací průnik, rozdíl a sjednocení musí mít tabulky naprosto stejné schéma (jména i typy), u kartézského součinu naopak na nejvyšší úrovni disjunktní a přirozené spojení vyžaduje, aby atributy na nejvyšší úrovni se shodnými jmény měly i stejné typy.

Průnik:	intersect
Rozdíl:	difference (nebo -)
Sjednocení:	union (nebo +)

Kartézský součin: **crossproduct** (nebo #)
Přirozené spojení: **naturaljoin** (nebo *)

Všechny vyjmenované operátory jsou asociativní doleva a mají dvě úrovně priorit. Na nižší jsou **union**, **intersect** a **difference**, na vyšší úrovni jsou **crossproduct** a **naturaljoin**.

Příklad:

```
v = (a union b) # c;
```

Výrazem vyhodnotitelným na jednu tabulku je například složitější výraz uzavřený v kulatých závorkách, název existující tabulky (dočasné i perzistentní), přejmenování, projekce, selekce, zahníždění, odhníždění, zdvojení, eliminace duplikátů, potenční operátor, aritmetika a agregace. Dále však pro přehlednost budeme u zmíněných operací psát, jakoby byly samostatné instrukce.

Převedení na množinu

Za zvláštní pozornost stojí operátor **distinct**, který převede multimnožinu na množinu (eliminace duplicitních záznamů). Jako parametr může dostat i výraz uzavřený v kulatých závorkách.

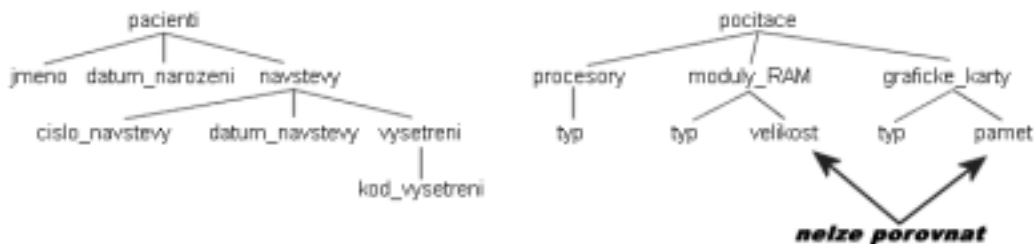
Příklad:

```
v = (distinct (a union b)) # c;
```

Selekce

Selekce je určena k výběru záznamů z tabulky, resp. z jejich hnížděných atributů, na základě splnění podmínky. V selekci je možné uvést i několik podmínek, které mohou být ve složitém booleovském výrazu vytvořeným infixovými spojkami **and** a **or**, prefixovou spojkou **not** a kulatými závorkami. Jednotlivé podmínky slouží k porovnání atributu s danou hodnotou nebo s jiným atributem, jejich typy (u hnížděných atributů schémata) musí navzájem odpovídat podle druhu podmínky, což je důležité hlavně pro složitější podmínky jako MEMBER a podmnožina.

Dalším omezením při porovnávání dvou atributů je jejich vzájemná poloha ve schématu. Pokud si představíme schéma tabulky jako strom s kořenem reprezentujícím celou tabulku (viz obrázek níže), rodiče obou atributů musí ležet na jedné větvi stromu. Na příkladu vlevo na obrázku je možné porovnávat jakoukoli dvojici kompatibilních atributů, ale na složitějším schématu vpravo tak tomu už není. Selekcii však v takovém případě lze provést po předchozích úpravách, například po odhníždění.



Syntaxe: podmínky se uvádějí v kulatých závorkách za názvem tabulky (resp. výrazem)

```
vysl = jméno_tabulky ( podmínka );
```

Podmínka je booleovský výraz z relačních operátorů, které se vždy zapisují:

```
první_parametr relační_operátor druhý_parametr
```

Přitom povolené relační operátory jsou `==`, `!=`, `>`, `<`, `>=`, `<=`, **member**, **not member**, **subset**, **not subset**, **probersubset** a **not probersubset**. Jedním z parametrů musí být vždy atribut (příp. s cestou) a druhým buď opět atribut nebo číslo či řetězec. Operátor **subset** a další za ním vyjmenované je možné použít jen se dvěma atributy.

Dalším povoleným relačním operátorem (nebo spíše celou sadou) je porovnání na počet řádků hnížděného atributu se syntaxí:

```
count(atribut) porovnání číslo
```

(pořadí lze samozřejmě i prohodit) nebo

```
count(atribut1) porovnání count(atribut2)
```

kde za *porovnání* může stát jeden z `==`, `!=`, `>`, `<`, `>=` a `<=`.

Poslední možností je varianta operátoru **member** (nebo **not member**), která má na levé straně nějaký atomický atribut (nebo konstantu) a na pravé straně cestu ke hnížděnému atributu (případně s projekcí), která dává jako výsledek množinu atomických hodnot:

```
atribut/konstanta member hnížděný_atribut[atomický_atribut]
```

Výraz v selekci lze pak popsat gramatikou:

```
výraz ->   relační operátor  
          |   výraz and výraz  
          |   výraz or  výraz  
          |   not výraz  
          |   ( výraz )
```

Příklady selekce:

```
v = zam(zarazení == "delnik" or plat < 10000);  
v = zam(count(detí) <= 0);  
v = zam("Petr" member detí[křestní_jm]);
```

Projekce

Projekce má za parametr seznam atributů vstupní tabulky v hranatých závorkách za názvem tabulky, které mají zůstat ze vstupní tabulky. Pokud je v seznamu hnížděný atribut, je ve schématu nechán i celý jeho podstrom.

```
vysl = jméno_tabulky [atr1, atr2, ..., atrN];
```

Příklad:

```
navstevnici = pacienti[jmeno, navstevy.datum_navstevy];
```

Výsledná tabulka bude mít schéma {jmeno, navstevy:{datum_navstevy}}, tedy u každého pacienta získáme množinu s daty jeho návštěv.

Variantou je negativní selekce, která naopak říká, které atributy se mají ze schématu vyjmout:

```
vysl = jméno_tabulky [ ^ atr1, atr2, ..., atrN];
```

Změny struktury

Mimo projekci existují další možnosti, jak změnit schéma tabulky. Opakem projekce je přidání sloupce příkazem **AddColumn**. Je možné přidávat jen atributy atomických typů, hnízděné atributy vznikají například při hnízdění.

Buď můžeme přidávat na nejvyšší úroveň tabulky:

```
vysl = AddColumn typ nový_atribut to jméno_tabulky;
```

nebo do určeného hnízděného atributu na libovolné úrovni hnízdění:

```
vysl = AddColumn typ nový_atribut to jméno_tabulky.atribut;
```

Jako typ jsou přípustná pouze klíčová slova **String** a **Int**. Hodnoty nového sloupce budou ve všech řádcích tabulky nulové (resp. prázdný řetězec).

Dále je možné měnit jména atributů. Jeden specifikovaný atribut v tabulce lze přejmenovat příkazem

```
vysl = jméno_tabulky <staré_jméno -> nové_jméno>;
```

Místo jména tabulky můžete uvést libovolný výraz. Staré jméno může být i cesta k nějakému zahnížděnému atributu, v tomto případě se již u nového jména celá cesta neopakuje, uvádí se jen nové jméno atributu na listové úrovni.

Hromadné přejmenování provedeme příkazem:

```
vysl = jméno_tabulky <* -> nějaký_prefix *>;
```

Tento příkaz přejmenuje všechny atributy na nejvyšší úrovni tabulky tím způsobem, že před jejich původní název přidá specifikovaný prefix. Mezera mezi prefixem a hvězdičkou není povinná.

Pomocí operace zdvojení tabulky (**double**) můžeme zdvojit každý řádek na nejvyšší úrovni tabulky. Ta je tedy dvakrát širší, má ale stejný počet řádků. Nově přidáné sloupce mají jména jako originály, před které je přidán zadaný prefix. Na místě jména tabulky může být složitější výraz v kulatých závorkách.

```
vysl = Double jméno_tabulky (prefix_pro_kopie_sloupců);
```

Hnízdění a odhnízdění

Syntaxe dvou klíčových operací pro hnízděnou relační algebru je následující.

Hnízdění:

```
vysl = Nest jméno_tabulky (seznam_atributů -> nový_atribut);
```

Seznam atributů nesmí být prázdný, jednotlivé atributy jsou odděleny čárkami. Všechny atributy v tomto příkazu musí být na nejvyšší úrovni tabulky, tj. nesmí jít o cesty obsahující tečku.

Odhnízdění atributu na nejvyšší úrovni:

```
vysl = Unnest jméno_tabulky (odhnizďovaný_atribut);
```

Po odhnízdění nesmí vzniknout na nejvyšší úrovni duplicitní atribut. Pokud tak nastane, je oznámena chyba. Řešením je přejmenování jednoho z atributů. Na místě jména tabulky u obou operací může být složitější výraz v kulatých závorkách.

Příklad navazující na příklad z projekce:

```
t = unnest navstevnici (navstevy);  
denik = nest t (jmeno -> pacienti);
```

Celý příklad můžeme napsat i do jediného výrazu (s menší přehledností):

```
denik = nest  
      (unnest (pacienti[jmeno, navstevy.datum_navstevy])  
       (navstevy)  
      ) (jmeno -> pacienti);
```

Výsledná tabulka denik bude mít schéma {datum_navstevy, pacienti: {jmeno}}, tedy u každého data, kdy přišel alespoň jeden pacient, budeme mít seznam jmen přijatých pacientů.

Pevný bod

Pevný bod je ideální operací pro konstrukci tranzitivního uzávěru. Přesnou definici můžete najít v předmětu „Současné databázové modely“, my si vystačíme s jejím zjednodušením. Pevný bod aplikuje na vstupní tabulku daný výraz tak dlouho, až je v jedné iteraci výsledek roven vstupu. (Poznámka: pevný bod tedy můžeme jednoduše simulovat WHILE cyklem.) Pro zabezpečení konečnosti výpočtu server provede jen určité nastavené množství iterací (implicitně 1000). Připomínáme, že zpracování každého programu můžete přerušit tlačítkem Stop running.

Pro tuto operaci si připravíme dvě tabulky (schémata popsána jen symbolicky):

```
vybrana_vysetreni: { kod_vysetreni }  
diag_postup: {kod_vysetreni, predchozi_vysetreni{ kod_predchozi}}
```

Pomocí pevného bodu můžeme získat tabulku, kde budou kódy všech vyšetření, které je nutné provést před námi vybranými.

Syntaxe – jsou dvě možnosti:

```
vysl = FixedPoint vstupní_tabulka = výraz;  
nebo  
vysl = FixedPoint vstupní_tabulka = call funkce(parametry);
```

Výraz, který je popisem operátoru, vzhledem ke kterému pevný bod počítáme, by měl operovat se vstupní tabulkou, do které je v každé iteraci dosazena hodnota z předchozího kroku. Pokud není možné napsat požadované operace do jednoho výrazu (například aritmetické operace nevracejí tabulku a nemohou být tak ve výrazu použité), musíme použít druhou variantu syntaxe a místo do výrazu napsat příkazy do funkce.

Příklad:

```
unnest_diags = unnest diag_postup (predchozi_vysetreni);  
fx = fixedpoint vybrana_vysetreni =  
    distinct ( (vybrana_vysetreni * unnest_diags)  
                [kod_predchozi]  
                <kod_predchozi -> kod_vysetreni>  
                union vybrana_vysetreni  
            );
```

Další příklad užití pevného bodu je na konci této (třetí) kapitoly v implementaci Dijkstrova algoritmu.

Potenční operátor

Potenční operátor vytvoří množinu (tabulku) všech podmnožin vstupní tabulky. Výsledná tabulka tak bude mít schéma s na nejvyšší úrovni s jediným hnížděným atributem, jehož jméno se musí uvést jako parametr. Přesněji: předpokládejme, že vstupní tabulka má schéma Ω . Výsledná tabulka tedy bude mít schéma $\{\text{nový_atribut: } \Omega\}$

Při této operaci vzniká tabulka s velkým počtem řádků, přesněji 2^n , kde n je počet záznamů vstupní tabulky. Proto server omezuje počet záznamů vstupní tabulky maximálně na 20.

Syntaxe:

```
vysl = Powerset jméno_tabulky (jméno_nového_kořenového_atr);
```

Na místě jména tabulky může být složitější výraz v kulatých závorkách.

Aritmetické operace

Pomocí aritmetických operací můžete vypočítávat nové hodnoty atributů v rámci jednoho hnížděného atributu (nebo tabulce) na jediné úrovni hníždění.

Syntaxe:

```
vysl = Arithmetic tabulka (atr1 operátor atr2 -> atribut);
```

Všechny atributy musí být typu `int`. Možné operátory jsou `+`, `-`, `*`, `/` (celočíselné dělení), `%` (modulo).

Pokud dojde k dělení nulou, je výsledek nedefinován. Na tento případ není uživatel nijak upozorněn.

Příklad na tabulce `vyrazy` s typem `{ Int pocet_op; Nested op {Int in_1; Int in_2; Int vysl; } }`:

```
vysl = Arithmetic vyrazy (op.in_1 + op.in_2 -> op.vysl );
```

Agregační operace

Agregační operátory `sum`, `avg`, `count`, `max` a `min` pracují s hnížděnými atributy (nebo tabulkou), který obsahuje alespoň jeden atomický atribut typu `Int` a alespoň jeden hnížděný atribut, ve kterém je opět nejméně jeden atribut typu `Int`. Pomocí agregačních operátorů můžeme z hnížděného atributu vybrat nějakou hodnotu a tu uložit do atomického. Na příkladu u výrazů by nás například mohlo zajímat, kolik je u každého výrazu podvýrazů, jaká je maximální hodnota atributu podvýrazů.C, apod.

Syntaxe:

```
vysl = Aggregation tabulka (operátor agr_atr -> atribut);
```

Všechny atributy musí být typu `int`. Možné operátory jsou `sum`, `avg`, `count`, `max` a `min`.

Příklad na tabulce `vyrazy` z aritmetických operací:

```
vysl = Aggregation vyrazy(count op.in_1 -> pocet_op);
```

Aktualizační operace

V této části jsou popsány příkazy podobné SQL příkazům `INSERT`, `UPDATE` a `DELETE`.

Vložení dat

Vkládat řádky do tabulky lze příkazem:

```
Insert Into jméno_tabulky Type jméno_typu Values { data };
```

Tento příkaz vkládá data do zadané tabulky na nejvyšší úroveň. Společně se jménem tabulky je potřeba zadat i její správný typ. Data jsou zadávána ve stejném formátu jako při definici tabulky.

```
Insert Into jméno_tabulky.cesta_k_hnížděné_tabulce Type  
jméno_typu Values { data } Where ( podmínka );
```

Tato varianta vloží data do těch hnížděných atributů, které jsou určeny cestou za **into** a zároveň splňují podmínku za **where** (stejná syntax jako u selekce). Zadaný typ musí odpovídat typu hnížděného atributu.

Podmínky ve **where** se musí vázat k atributům, které jsou na cestě do modifikovaného hnížděného atributu.

Aktualizace dat

Příkazem:

```
Update jméno_tabulky.cesta_k_hnížděmu_atributu  
Set atomický_atribut_hnížděho_atributu = nová_hodnota  
Where ( podmínka );
```

nastavíme hodnotu atomických atributů ve vybraných hnížděných atributech na zadanou konstantu. Na **where** podmínku zde platí stejné omezení jako u **insert**. Podmínkou za **where** určujeme jednak, které hnížděné tabulky budeme modifikovat, ale také, které řádky v těchto tabulkách budeme měnit. Cesta k hnížděnému atributu za jménem tabulky (i s tečkou) je nepovinná, lze pracovat přímo s tabulkou.

Mazání dat

Vybrané řádky (**where** podmínka funguje stejně jako u **update**) vymažeme příkazem

```
Delete From jméno_tabulky.cesta_k_hnížděmu_atributu  
Where ( podmínka );
```

Cesta k hnížděnému atributu za jménem tabulky (i s tečkou) je nepovinná, lze pracovat přímo s tabulkou.

Procedurální programování

Programovací jazyk umožňuje také použít jednoduché procedurální konstrukce, jako je IF podmínka, WHILE cyklus a definovat si vlastní funkce (o nich je psáno již dříve – viz sekce Definice funkcí).

IF podmínka

Syntaxe je obvyklá, přitom else větev je nepovinná:

```
if ( podmínka )  
{  
    příkazy blok1  
}  
else  
{  
    příkazy blok2  
}
```

Pokud je podmínka splněna, pokračuje zpracování blokem1, jinak se přejde do bloku2 (pokud je přítomen). Jako podmínka se dají použít různé vztahy dvou tabulek jako celku (`==`, `!=`, `member`, `subset`, `probersubset`, `not member`, `not subset`, `not probersubset`) a porovnání počtu řádek tabulky (`count` – podobně jako u selekce) s konstantou nebo počtem řádek jiné tabulky. Vyjmenované relační operátory je možné podle libosti kombinovat logickými spojkami `not`, `and` a `or` (stejně jako u selekce).

WHILE cyklus

Syntaxe:

```
while( podmínka )
{
    blok
}
```

Nejdříve se vyhodnotí podmínka (stejná jako u `if`). Pokud je splněna, provádí příkazy bloku tak dlouho, až splněna nebude. Pro zabezpečení konečnosti výpočtu server provede jen určité nastavené množství iterací (implicitně 1000). Připomínáme, že vždy můžete výpočet přerušit tlačítkem Stop running.

Příklad – náznak simulace pevného bodu přes tabulku t:

```
{
    new_t = t;
    t = empty;
    while(new_t != t)
    {
        t = new_t;
        // dále získání tabulky new_t z t
    }
}
```

Větší příklad

Za spolupráce externího programátora Petra Andrše jsme pro vás připravili program, který na zadaném ohodnoceném grafu vypočítá nejkratší cestu z daného vrcholu grafu do všech ostatních. Používá se přitom známý a efektivní Dijkstrův algoritmus.

```
// D I J K S T R U V   A L G O R I T M U S
```

```
// typy pro pomocne tabulky
```

```
type tint = {
    int cislo;
};
```

```
type tstring = {
    string retezec;
};
```

```
// typ tabulky vrcholu. V distance se pamatuje vzdalenost od start vrcholu
// a final je boolska hodnota urcujici, zda hodnota dostance byla jiz
// definitivne stanovena.
```

```
type vertex = {
```

```

    string name;
    int distance;
    int final;
};

// Mnozina hran - start, cil a delka
type edge = {
    string start;
    string end;
    int length;
};

// konstatnta nula
table nula : tint = {
    0;
};

// konstanta jedna
table jedna : tint = {
    1;
};

// konstanata nekonecno
table nekonecno : tint = {
    1000000;
};

// vychodzi vrchol, je definitivni a vzdalenost vrcholu od sama sebe je nula
table origin : vertex = {
    "Tokushima",0,1;
};

// seznam hran
table edges : edge = {
    "Aioi","Deai",21;
    "Aki","Nangoku",26;
    "Anabuki","Koyadaira",25;
    "Anabuki","Sadamitsu",12;
    "Anabuki","Waki",1;
    "Anan","Tachibana",5;
    "Deai","Kito",21;
    "Deai","Sawatani",13;
    "Donari","Waki",17;
    "Hanoura","Anan",7;
    "Hanoura","Wajiki",24;
    "HigashiIyayama","NishiIyayama",10;
    "Hiketa","Itano",11;
    "Hiketa","Shirotori",9;
    "Hiwasa","Aioi",14;
    "Hiwasa","Kainan",29;
    "Ikeda","Kawanoe",27;
    "Ikeda","NishiIyayama",30;
    "Ikeda","Otoyo",45;
    "Ishii","Kamiyama",20;
    "Ishii","Kamojima",9;
    "Itano","Donari",14;
    "Kainan","Deai",63;
    "Kainan","Toyo",17;
    "Kamibun","Koyadaira",16;
    "Kamibun","Yamakawa",20;
    "Kamikatsu","Sawatani",23;
    "Kamiyama","Kamibun",9;
    "Kamojima","Donari",5;
    "Kamojima","Yamakawa",13;
};

```

```

"Katsuura", "Kamikatsu", 20;
"Komatsujima", "Hanoura", 7;
"Komatsujima", "Katsuura", 15;
"Komatsujima", "Sanagouchi", 15;
"Kotohira", "Ikeda", 29;
"Kotohira", "Sakaide", 15;
"Kotohira", "Toyohama", 22;
"Kotohira", "Zentsuji", 8;
"Koyadaira", "Minokoshi", 25;
"Mima", "Ikeda", 24;
"Mima", "Kotohira", 33;
"Minokoshi", "HigashiIyayama", 33;
"Minokoshi", "Sadamitsu", 49;
"Muroto", "Nahari", 29;
"Nahari", "Aki", 15;
"Nangoku", "Kito", 74;
"Nangoku", "Kochi", 9;
"Naruto", "Hiketa", 18;
"Naruto", "Itano", 16;
"Otoyo", "HigashiIyayama", 47;
"Otoyo", "Nangoku", 32;
"Sadamitsu", "Ikeda", 23;
"Sadamitsu", "Mima", 1;
"Sakaide", "Zentsuji", 9;
"Sanagouchi", "Kamiyama", 10;
"Sawatani", "Kamibun", 22;
"Shionoe", "Kotohira", 29;
"Shionoe", "Takamatsu", 23;
"Shirotori", "Donari", 22;
"Shirotori", "Takamatsu", 35;
"Tachibana", "Hiwasa", 21;
"Tachibana", "Wajiki", 20;
"Takamatsu", "Kotohira", 30;
"Takamatsu", "Sakaide", 25;
"Tokushima", "Ishii", 11;
"Tokushima", "Itano", 15;
"Tokushima", "Komatsujima", 8;
"Tokushima", "Naruto", 12;
"Tokushima", "Sanagouchi", 20;
"Toyo", "Muroto", 32;
"Toyo", "Nahari", 42;
"Toyohama", "Kawanoe", 13;
"Wajiki", "Aioi", 10;
"Waki", "Mima", 11;
"Waki", "Shionoe", 20;
"Yamakawa", "Anabuki", 8;
"Zentsuji", "Toyohama", 24;
};

```

/ popis standardního řešení problému hledání nejkratší cesty v grafu*

Klasický algoritmus pro tento problém pochází od Dijkstry. Používáme v něm pro každý vrchol u číselnou proměnnou $L(u)$ a logickou proměnnou $P(u)$ a pro jednoduchost zapisujeme předpokladáme, že množina vrcholů je množina $\{1, \dots, n\}$ a že hledáme nejkratší cestu z 1 do n :

```

01 begin
02   L(1) := 0; P(1) := true;
03   for u := 2 to n do
04     begin
05       if (1,u) je hrana then L(u) := dist(1,u) else L(u) := Inf;

```

```

06     P(u) := false;
07     end;
08     while existuje vrchol u takovy, ze P(u) = false do
09     begin
10         zvol jako w vrchol, pro nejz P(w) = false a L(w) je minimalni
11         P(w) := true;
12         for vsechny vrcholy v takove, ze P(v) = false a (w,v) je hrana do
13             if L(v) > L(w) + dist(w,v) then L(v) := L(w) + dist(w,v);
14     end;
15 end;

L je representovana sloupcem distance, P sloupcem final

*/

// v nasledujicim vzdy komenar popisuje co dela nasledujici blok prikazu
// az do dalsiho komentare

// funkce provede jednu iteraci cyklu dijkstrova algoritmu (radky 08 - 14)
FUNCTION dstep(vertices2,nekonecno2,jedna2,nula2,edgesbi2)
{
// vyselekti se nedefinitivni vrcholy a zahrnizdi podle vsech atributu -
// vznikne tedy tabulka 1 x 1 s hnizdenym, atributem kde je cela puvodni
// tabulka
a = nest (vertices2(final==0)) (name,final,distance -> vrcholy);
// do "a1" se naopak nacpou definitivni vrcholy
a1 = vertices2(final==1);
// k "a" ( ktera ma rozmer 1x1 ) se prida novy sloupec se jmenem cislo
// a hodnotou representujici nekonecno
b = a crossproduct nekonecno2;
// do policka pridaneho v minulem kroku se spocita minimum ze atributu
// distance pres vsechny vrcholy
b = aggregation b (min vrcholy.distance -> cislo);
// odhnizdime
c = unnest b (vrcholy);
// a nechame jen vrchloly, jejichz distance se rovna spoctenemu minimu
// a odstranime pomocny sloupec
o = c(cislo==distance) [name,final,distance];

// nyni je v o radek z vertices pro ktery plati final = 0 a distance je
// minimalni.
// pokud je takovych vice, jsou tam vsechny tj. sekvenci "a" .. "o" byl
// odsimulovan radek 10
// tyto vrcholy se v tomto kroku stanou definitivni. tedy "o" je w
// z algoritmu

// zarazeni vrcholu z w mezi definitivni - sloupec final s nulami
// se odstrani, prinasobi
// se novy s jednickami a prejmenuje na final - radek 11
p = (o[name,distance] crossproduct jedna2<cislo ->
final>) [name,final,distance];
// k vrcholům z w se prispojı hrany, ktere z nich vedou
// sdistance - definitivni vzdalenost start vrcholu
// edistance - stara jeste nedefinitivni vzalenost end vrcholu
q = ((p[name,distance]<distance -> sdistance)<name -> start) naturaljoin
edgesbi2) [start,end,length,sdistance];
// dale prispojime ke hranam vedoucım z vrcholu z w jeste jednou tabulku
// vsech vrcholu
// abychom k nim dostali stare hodnoty distance (do edistance)
// a vytvorime novy sloupec, do ktereho se vypocitaji nove vzdalenosti
// do techto vrcholu pres vrchol z w
r = ( ( ( (q naturaljoin vertices2<name -> end><distance -> edistance>)
        (final==0)
        ) [end,length,edistance,sdistance]

```

```

        ) crossproduct nula2
    ) <cislo -> ndistance>;
// vypocteme do ndistance vzdalenosti do vrcholu pres vrcholy z w
r = arithmetic r (sdistance + length -> ndistance);
// ponechame jen vrcholy, do kterych je nova vzdalenost pres w mensi
// nez vzdalenost stara
s = r(ndistance < edistance);
// transformujeme vrcholy z promenne s na schema tabulky vertices
t = ((s[end,ndistance]<end -> name><ndistance -> distance>)
    crossproduct nula2<cislo -> final>)[name,final,distance];
// pridame k puvodni tabulce vertices radky pro vsechny vrcholy,
// kde doslo behem teto
// iterace vypoctu ke zmene hodnoty nejakeho atributu
u = distinct( vertices2 union p union t);
// pokud pro nejaky vrchol existuje radek s final=0 i final=1,
// pro takove vrcholy se radky s final=0 odstrani
v = (nest u (final -> hnizdo)) crossproduct nula2;
v = aggregation v (max hnizdo.final -> cislo);
w = v[name,distance,cislo]<cislo -> final>;
// pokud pro nejaky vrchol existuje vice radku s ruznymi
// hodnotami distanece,
// pro takove vrcholy se ponechaji radky s nejmensi hodnotou distance
x = (nest w (distance -> hnizdo)) crossproduct nula2;
x = aggregation x (min hnizdo.distance -> cislo);
xvertices = x[name,final,cislo]<cislo -> distance>;
// nyní je v xvertices opet pro kazdy vrchol jen jeden radek.
// vracime novou mnozinu vrcholu
return xvertices;
} // --- function dstep() ---

// --- main ---
{
// vytvoreni orientovaneho grafu tim, ze kazda neorientovana hrana je
// nahrazena dvema orientovanymi
edgesbi = distinct ( edges union edges<* -> n*><nstart -> end><nend ->
start><nlength -> length> );
// vytvori ze seznamu hran seznam vsech vrcholu
vertices = distinct ( edges[start]<start -> name> union edges[end]<end ->
name> );
// odebrani pocatecniho vrcholu ze seznamu vsech vrcholu
vertices[name] difference origin[name];
// inicializace pro ostatni vrcholy (final=0, distance=nekonecno)
vertices = distinct ( ( ( ( (vertices crossproduct nekonecno)
    <cislo -> distance>
    ) crossproduct nula)<cislo -> final>
    ) union origin);
// nastaveni pocatecnich vzdalenosti do vrcholu, do kterych se lze dostat
// z pocatecniho vrcholu
vertices = ( nest ( vertices union
    ( ( ( origin crossproduct edgesbi)
        (name == start)[end,length]<end -> name>
        <length -> distance>
    ) crossproduct nula
    ) <cislo -> final>
    ) ( distance -> distances ) )
    crossproduct nula;
vertices = aggregation vertices (min distances.distance -> cislo);
vertices = vertices[name,final,cislo]<cislo -> distance>;
// initialized

// spusteni cyklu - dokud se neco meni, stale se iteruje. pokud jsou
// vsechny vrcholy
// definitivni, vypocet se ukonci a ve vertices mame nejkratsi vzdalenosti
// do vsech vrcholu

```

```

vysledek = fixedpoint vertices = call
dstep(vertices, nekonecno, jedna, nula, edgesbi);

// zobrazeni vysledku
vysledek = vysledek[^final];
display vysledek sorted by distance asc;

} // --- main ---

```

Gramatika programovacího jazyka

Tato část obsahuje úplnou LALR(1) gramatiku programovacího jazyka. Pravidlo začíná jménem neterminálu s dvojtečkou a končí středníkem. Varianty v rámci jednoho pravidla jsou odděleny znakem '|'. /* empty */ znamená, že neterminál se může redukovat na prázdné slovo.

Neterminály začínají malým písmenem a jsou psány kurzívou. Terminály složené z písmen začínají velkým písmenem (v programu lze použít stejné slovo psané libovolně vzhledem k velikosti písmen, protože klíčová slova nejsou case-sensitive), terminály složené ze speciálních znaků jsou uvedeny v apostrofech. Ke klíčovému slovu CrossProduct existuje varianta '#', pro And jsou ekvivalentní '&' a '&&', pro Or '|' a '||' a stejné jako Not je Neg a '!'.

Iniciální symbol, základní struktura programu

```

program
    :      definitions program_body
    ;

definitions
    :      /* empty */
    |      definitions type_def
    |      definitions table_def
    |      definitions function_def
    ;

```

Definice typu

```

type_def
    :      Type Identifier '=' type optional_semicolon
    ;

type
    :      '{' attrib_list '}'
    ;

attrib_list
    :      /* empty */
    |      attrib_list Int Identifier ';'
    |      attrib_list String Identifier ';'
    |      attrib_list Nested Identifier type optional_semicolon
    ;

```

Pomocné neterminály – cesty, seznamy, volitelný středník, ...

```

optional_semicolon
    :      /* empty */
    |      ';'
    ;

attribute_path
    :      Identifier

```

```

        |      attribute_path '.' Identifier
;
attribute_list
    :      attribute_path
    |      attribute_list ',' attribute_path
;
identifier_list
    :      Identifier
    |      identifier_list ',' Identifier
;
attribute_path_or_null
    :      /* empty */
    |      '.' attribute_path
;

```

Identifikátory tabulek

```

table_identifier
    :      simple_table_identifier
    |      Identifier ':' simple_table_identifier
;
simple_table_identifier
    :      Identifier
;

```

Definice tabulky

```

table_def
    :      Table simple_table_identifier ':'
    |      Identifier '=' value optional_semicolon
;
value
    :      '{' value_list '}'
;
value_list
    :      /* empty */
    |      value_list IntegerValue ','
    |      value_list StringValue ','
    |      value_list value ','
    |      value_list IntegerValue ';'
    |      value_list StringValue ';'
    |      value_list value ';'
;

```

Definice funkce

```

function_def
    :      Function Identifier '(' function_parameters ')'
    |      function_body optional_semicolon
;
function_parameters
    :      identifier_list
;
function_body
    :      '{' instruction_list Return table_identifier ';' '}'
;

```

Tělo programu

```

program_body
    :      '{' instruction_list '}' optional_semicolon
;

```

```

instruction_list
    : /* empty */
    | instruction_list instruction_with_semicolon
    | instruction_list while_statement
    | instruction_list if_statement
;

instruction_with_semicolon
    : instruction ';'
;

instruction
    : /* empty */
    | simple_table_identifer '='
      table_or_fixed_point_expression
    | simple_table_identifer '=' Null
    | Abort
    | insert_statement
    | update_statement
    | delete_from_statement
    | display_statement
    | save_statement
    | Delete table_identifer
    | set_rights_statement
    | Reorganize table_identifer
;

table_or_fixed_point_expression
    : fixed_point_expression
    | table_expression
;

```

While cyklus

```

while_statement
    : While '(' cond_expr ')' '{' instruction_list '}'
;

```

If podmínka

```

if_statement
    : If '(' cond_expr ')' '{' instruction_list '}' else_block
;

else_block
    : /* empty */
    | Else '{' instruction_list '}'
;

```

Pevný bod

```

fixed_point_expression
    : FixedPoint simple_table_identifer '=' table_expression
;

```

Výrazy pracující s tabulkami

```

table_expression
    : table_literal
    | table_expression union_symbol table_expression
    | table_expression Intersect table_expression
    | table_expression difference_symbol table_expression
    | table_expression CrossProduct table_expression
    | table_expression natural_join_symbol table_expression
;

```

```

natural_join_symbol
    :      NaturalJoin
    |      '*'
;

difference_symbol
    :      Difference
    |      '-'
;

union_symbol
    :      Union
    |      '+'
;

table_literal
    :      table_term
    |      table_literal '<' attribute_path '->' Identifier '>'
    |      table_literal '<' '*' '->' Identifier '*' '>'
    |      table_literal '[' attribute_list ']'
    |      table_literal '[' '^' attribute_list ']'
    |      table_literal '(' sel_cond_expr ')'
    |      Nest table_term '(' identifier_list '->' Identifier ')'
    |      Unnest table_term '(' Identifier ')'
    |      Double table_term '(' Identifier ')'
    |      Distinct table_term
    |      PowerSet table_term '(' Identifier ')'
    |      AddColumn Int Identifier To table_term
    |      AddColumn Int Identifier To table_term '.' attribute_path
    |      AddColumn String Identifier To table_term
    |      AddColumn String Identifier
    |      To table_term '.' attribute_path
    |      Arithmetic table_term '(' attribute_path
    |      arithmetics_op attribute_path '->' attribute_path ')'
    |      Aggregation table_term '(' aggregation_op
    |      attribute_path '->' attribute_path ')'
;

arithmetics_op
    :      '+'
    |      '-'
    |      '*'
    |      '/'
    |      '%'
;

aggregation_op
    :      Min
    |      Max
    |      Avg
    |      Sum
    |      Count
;

table_term
    :      table_identifier
    |      '(' table_expression ')'
    |      function_call
;

function_call
    :      Call Identifier '(' parameter_list ')'
;

parameter_list
    :      parameter_list ',' table_identifier
    |      table_identifier
;

```

Aktualizační příkazy

```
insert_statement
:      Insert Into table_identifier Type Identifier Values value
|      Insert Into table_identifier '.' attribute_path Type
      Identifier Values value Where '(' sel_cond_expr ')'
;

update_statement
:      Update table_identifier attribute_path_or_null Set
      Identifier '=' IntegerValue Where '(' sel_cond_expr ')'
|      Update table_identifier attribute_path_or_null Set
      Identifier '=' StringValue Where '(' sel_cond_expr ')'
;

delete_from_statement
:      Delete From table_identifier attribute_path_or_null
      Where '(' sel_cond_expr ')'
;
```

Příkaz display

```
display_statement
:      Display          table_identifier
|      Display          table_identifier Sorted By sort_attr_list
|      Display Tables
|      Display Rights For persistent_table_identifier
|      Display Type For table_identifier
;

sort_attr_list
:      attribute_path Asc
|      attribute_path Desc
|      sort_attr_list ',' attribute_path Asc
|      sort_attr_list ',' attribute_path Desc
;
```

Příkaz save

```
save_statement
:      Save simple_persistent_table_identifier
|      Save table_identifier To persistent_table_identifier
;
```

Příkaz set rights

```
set_rights_statement
:      Set Rights persistent_table_identifier
      '(' rights_identifier '->' table_identifier ')'
|      Set Rights persistent_table_identifier
      '(' rights_identifier '->' Public ')'
;

rights_identifier
:      RightNone
|      RightRead
|      RightWrite
|      RightUndefined
;
```

Booleovský výraz použitelný v selekci

```
sel_cond_expr
:      sel_relation_operator
|      sel_cond_expr And sel_cond_expr
|      sel_cond_expr Or sel_cond_expr
|      Not sel_cond_expr
|      '(' sel_cond_expr ')'
;
```

```

sel_relation_operator
:   attribute_path comparison_op attribute_path
|   attribute_path Member attribute_path
|   attribute_path Not Member attribute_path
|   attribute_path Subset attribute_path
|   attribute_path Not Subset attribute_path
|   attribute_path ProperSubset attribute_path
|   attribute_path Not ProperSubset attribute_path
|   attribute_path Member attribute_path '[' Identifier ']'
|   attribute_path Not Member attribute_path
    '[' Identifier ']'
|   attribute_path comparison_op IntegerValue
|   IntegerValue comparison_op attribute_path
|   IntegerValue Member attribute_path
|   IntegerValue Not Member attribute_path
|   IntegerValue Member attribute_path '[' Identifier ']'
|   IntegerValue Not Member attribute_path '[' Identifier ']'
|   Count '(' attribute_path ')' comparison_op IntegerValue
|   IntegerValue comparison_op Count '(' attribute_path ')'
|   Count '(' attribute_path ')' comparison_op
    Count '(' attribute_path ')'
|   attribute_path comparison_op StringValue
|   StringValue comparison_op attribute_path
|   StringValue Member attribute_path
|   StringValue Not Member attribute_path
|   StringValue Member attribute_path '[' Identifier ']'
|   StringValue Not Member attribute_path '[' Identifier ']'
;

comparison_op
:   '=='
|   '!='
|   '<'
|   '>=' // nebo '<='
|   '>'
|   '<=' // nebo '<='
;

```

Booleovský výraz do podmínky if a while příkazu

```

cond_expr
:   cond_relation_operator
|   cond_expr And cond_expr
|   cond_expr Or cond_expr
|   Not cond_expr
|   '(' cond_expr ')'
;

cond_relation_operator
:   table_identifier '==' table_identifier
|   table_identifier '!=' table_identifier
|   table_identifier Member table_identifier
|   table_identifier Not Member table_identifier
|   table_identifier Subset table_identifier
|   table_identifier Not Subset table_identifier
|   table_identifier ProperSubset table_identifier
|   table_identifier Not ProperSubset table_identifier
|   IntegerValue comparison_op Count '(' table_identifier ')'
|   Count '(' table_identifier ')' comparison_op IntegerValue
|   Count '(' table_identifier ')' comparison_op
    Count '(' table_identifier ')'
;

```



INRA Monitor

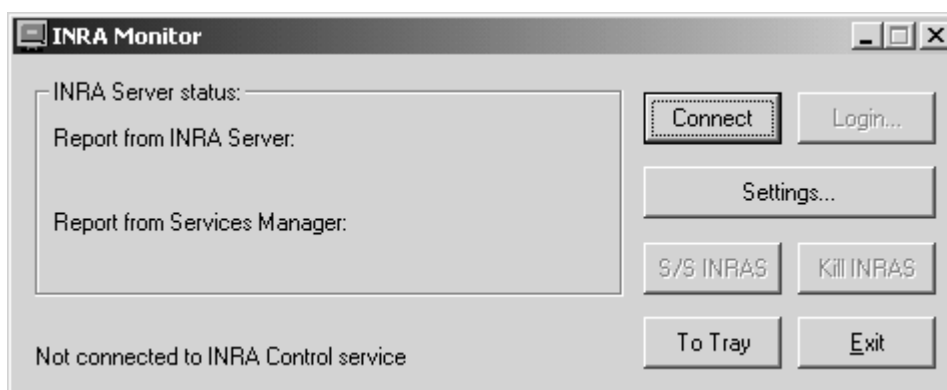
Tato kapitola popisuje funkce a ovládání pomocného programu pro kontrolování a sledování stavu INRA Serveru.

Popis

INRA Monitor je aplikace, která slouží ke vzdálenému sledování stavu INRA Serveru. Tato aplikace se vlastně klientem z hlediska TCP/IP, který se připojuje k serverové aplikaci INRA Control spuštěné na stejném počítači jako INRA Server. INRA Control pak svým klientům – monitorům – zasílá informace o stavu INRA Serveru. Monitor je po přihlášení s administrátorskými právy možno také použít například k restartování služby INRA Serveru.

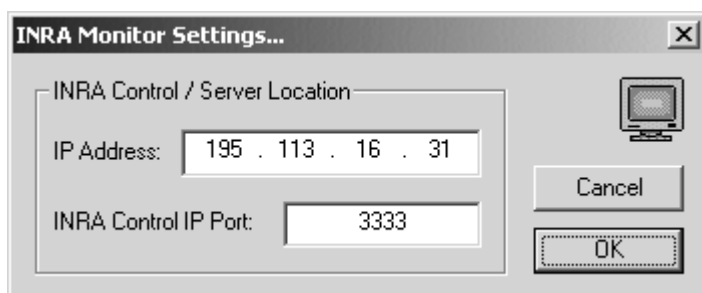
Ovládání Monitoru

Po spuštění INRA Monitoru se zobrazí dialog, který představuje celou aplikaci.



První akcí, kterou je nutné provést, je přihlášení ke kontrolní službě stlačením tlačítka **Connect**. Program přihlášení automaticky provede použitím vašeho nastavení adresy a portu, kde lze nalézt INRA Control. Nastavení této adresy je sdíleno s aplikací INRA Client pro specifikaci adresy počítače s INRA Serverem. V případě, že program není korektně instalován, dojde stejně jako při prvním spuštění INRA Clienta k zobrazení konfiguračního dialogu, který lze také ručně vyvolat pomocí tlačítka **Settings**.

V tomto dialogu je nutné zkontrolovat, zda adresa a port služby INRA Control jsou správně nastaveny.



Špatné nastavení se projeví tím, že se vůbec nepodaří spojit se službou INRA Control. V případě, že se spojení nepodaří, je nutné zkontrolovat, zda máte zde uvedeno správné nastavení, zda existuje spojení k cílovému počítači a případně zda a na jakém portu je spuštěna služba INRA Control. (Více informací lze nalézt v Administrátorské příručce.)

Získávání informací

Po připojení dostane typicky po několika málo vteřinách informaci o aktuálním stavu INRA Serveru. K získávání informací není třeba být přihlášen. Získávané informace jsou celkem trojího druhu.

Údaj o stavu spojení ke kontrolní službě

Tento údaj lze nalézt na dialogu vlevo dole a může nabývat pouze 2 hodnot:

- Connected to INRA Control service
- Not connected to INRA Control service

Informace od manažera služeb na cílovém počítači

Tento údaj je dekodovanou odpovědí manažera služeb na stav služby INRA Server. Možné jsou následující odpovědi:

- INRA Server service running
- INRA Server service stopped
- INRA Server service start pending
- INRA Server service stop pending
- Unknown state of the INRA Server service
- Can't query INRA Server service status
- Can't open INRA Server service (Not installed?)
- Can't open the service manager

Informace od INRA Serveru

Za normálních okolností se INRA Control ptá INRA Serveru na aktuální počet připojených uživatelů. V případě, že server rozumně odpoví, je tato informace cenná pro správce vzhledem ke sledování vytíženosti serveru a v případě, že ne, signalizuje tento stav nějaký problém. Možné zprávy v tomto místě jsou:

- $\langle X \rangle$ users logged on the INRA Server (kde $\langle X \rangle$ je celé číslo)
- INRA Server response timeout
- INRA Server window not found

Rychlé sledování stavu

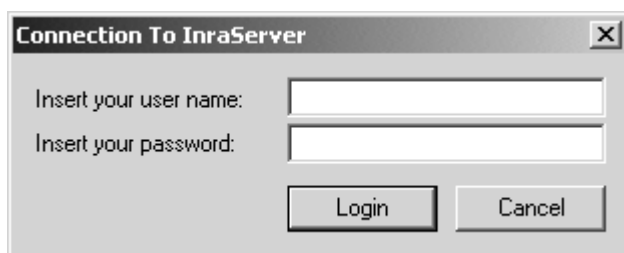
Stlačením tlačítka **To Tray** na INRA Monitoru se jeho okno schová do malé ikonky v pravém dolním rohu vaší pracovní plochy. Ikonka pak sama mění svůj vzhled podle toho, zda INRA Server v pořádku běží nebo zda se vyskytl nějaký potenciální problém.



Takto můžete pohodlně stav sledovat aniž by vám Monitor překážel na pracovní ploše.

Ovládání INRA Serveru

K ovládání INRA Serveru je nutné se přihlásit pomocí tlačítka **Login** a v následném dialogu zadat přihlašovací jméno a heslo:



Po ověření korektnosti a administrátorských práv máte možnost ovládat vzdáleně službu INRA Serveru. Tlačítkem **Start INRAS** zadáte příkaz ke spuštění služby. (Akci je nutné potvrdit) INRA Control poté odpoví informací, jak pokus o nastartování dopadl. Možné odpovědi jsou následující:

- INRA Server starting OK.
- Starting error, can't open the Service Manager.
- Starting error, can't open the INRA Server Service.
- Starting error, INRA Server request timeout.
- Starting error, some other error.
- Starting error, INRA Server service marked for delete.
- Starting error, service database locked.
- Starting error, INRA Server service already running.
- Starting error, INRA Server service executable file not found.

Jestliže je INRA Server spuštěn, je možné, aby jej administrátor zastavil. Provést tuto akci lze tlačítkem **Stop INRAS**. (Akci je nutné potvrdit) INRA Control poté odpoví informací, jak pokus o zastavení služby dopadl. Možné odpovědi jsou následující:

- INRA Server stopping OK.

- Stopping error, can't open the Service Manager.
- Stopping error, can't open the INRA Server Service.
- Stopping error, INRA Server not active.
- Stopping error, INRA Server request timeout.
- Stopping error, some other error.

Jestliže je INRA Server spuštěn, tedy jestliže běží jeho komunikační vlákno, ale server jako takový je z nějakých příčin zablokován. (Například došlo k závažné chybě v relačním jádře), je možné, aby administrátor dal pokyn k okamžitému ukončení procesu INRA Serveru tlačítkem **Kill INRAS**. (Akci je také nutné potvrdit. Serverový proces není přímo zabit, ale pomocí komunikačních prostředků Windows je mu INRA Controlem poslán příkaz k okamžitému ukončení procesu, server pak již nečeká na uvolnění žádných kritických sekcí a nesnaží se ani uvést databázi do konzistentního stavu, ale neprodleně se ukončí. Rollback nedokončených transakcí je zajištěn až po opětovném startu serveru) INRA Control odpoví všem přihlášeným Monitorům informací o aktuálním stavu.