Nová podoba jazyka RPG

Vladimír Župka, 2020

Vývoj tvaru jazyka RPG

Formulářový zápis programu

1959	RPG	děrné štítky
1969	RPG II	děrné štítky
1975	RPG II	diskety
1978	RPG III	terminály, disky

Výrazy v některých rubrikách formuláře C

1994	RPG IV	výrazy v příkazech (EVAL, IF, DOW, DOU)

Volný zápis ve formuláři C

2001	ILE RPG	úseky výpočtů ve volném tvaru (/FREE, /END-FREE) - verze 5.1
2010	ILE RPG	smíšené výpočty (ve volném nebo formulářovém tvaru) - verze 7.1

Alternativa k formulářům H, F, D, P - smíšený zápis

2013	ILE RPG	(CTL-OPT, DCL-F, DCL-xx, END-xx) - verze 7.2

Zcela volný zápis programu

2016	ILE RPG	úvodní příkaz **FREE - verze 7.3

Program ve smíšeném zápisu (od verze 7.2)

```
1....678.1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
      // komentář volného tvaru až od sloupce 7
       * komentářový řádek

     FCENIK     IF   E           K DISK

     D  pocet          C                   const(10)
          DCL-DS struktura;   // příkaz až od sl. 8 nebo dále
             znaky char(10);
             cislo int(10);
          END-DS struktura;

          // popisy souborů a dat lze zapsat v libovolném pořadí
          DCL-F STAVYP  // příkaz zapsaný ve více řádcích
                keyed
                usage(*update);

          if *in03;
     C                    GOTO       KONEC
          endif;
          // ...
     C     KONEC          TAG
          return;
```

Program ve zcela volném zápisu (od verze 7.3)

```
1...                                                                    ...
**FREE
CTL-OPT dftactgrp(*no);

DCL-F CENIK keyed;

    DCL-C pocet const(10);

DCL-F TISK printer(*ext)  // komentář může být na každém řádku příkazu
           usage(*output)     // komentář 2
           oflind(*in55);     // komentář 3
   // příkazy mohou začínat kdekoliv v řádku.
     /INCLUDE QRPGLESRC,PROTOTYPY
    DCL-S soucet packed(9: 2) inz(0); // jednoduchá proměnná

// náhrada GOTO, TAG
dou *in03;
    leave;
    // ...
enddo;
return;
```

Nahrazení formulářů (od verze 7.2)

Formulář		*Volný zápis*	
6	24		
H		CTL-OPT	Řídicí volby pro kompilátor
F		DCL-**F**	Popis souboru
D	C	DCL-**C**	Pojmenovaná konstanta
	S	DCL-**S**	Samostatná proměnná
	DS	DCL-**DS**	Datová struktura
		DCL-SUBF	Podpole (subfield)
		END-DS	Konec datové struktury
	PI	DCL-**PI**	Rozhraní procedury (procedure interface)
		DCL-PARM	Parametr
		END-PI	Konec rozhraní procedury
	PR	DCL-**PR**	Prototyp procedury
		DCL-PARM	Parametr
		END-PR	Konec prototypu
C		příkaz ;	Výpočty
P	B	DCL-**PROC**	Začátek procedury
	E	END-PROC	Konec procedury
I		/INCLUDE nebo /COPY	
O		/INCLUDE nebo /COPY	

Použití souboru

Formulář *Volný zápis*

17	20				
I		DCL-F	filename	DISK	USAGE(*INPUT)
I	A			**DISK**	USAGE(*INPUT: *OUTPUT)
				DISK	USAGE(*INPUT: *UPDATE)
U				**DISK**	USAGE(*INPUT: *UPDATE: *DELETE)
U	A			**DISK**	USAGE(*INPUT: *UPDATE: *DELETE: *OUTPUT)
O				**DISK**	USAGE(*OUTPUT)
O	A			**DISK**	USAGE(*OUTPUT)
O		DCL-F	filename	**PRINTER**	USAGE(*OUTPUT)
C		DCL-F	filename	**WORKSTN**	USAGE(*INPUT: *OUTPUT)

Typy dat

Formulář		*Volný zápis*	*Hodnoty*
40	44		
A		CHAR (n)	1 - 19773104
A	VARYING	VARCHAR (n {:m})	1 - 19773104 : 2|4
C		UCS2 (n)	1 - 8386552 (počet dvoubajtů)
C	VARYING	VARUCS2 (n {:m})	1 - 8386552 : 2|4
P		PACKED (n {:m})	1 - 63 : 0 - 63
S		ZONED (n {:m})	1 - 63 : 0 - 63
I		INT (n)	3, 5, 10, 20
U		UNS (n)	3, 5, 10, 20
B		BINDEC (n {:m})	1 - 9 : 0 - 9
F		FLOAT (n)	4, 8
N		IND	
D		DATE {(*DMY-) }	*ISO, ...
T		TIME {(*HMS.)}	*ISO, ...
Z		TIMESTAMP {(n)}	0 - 12 des. místa u sekund
*		POINTER	

Datové struktury

Nepojmenovaná podpole

```
DCL-DS status;    // API QDCLCFGD (List Configuration Descriptions)
   *n   char(10) inz('*EQ');
   *n   char(10) inz('*ACTIVE');
END-DS;
```

Datová struktura bez podpolí

```
dcl-DS dstr LEN(100) END-DS;    // END-DS je součástí příkazu
```

Program status data structure

```
dcl-DS pgmStat PSDS qualified;
   status *STATUS;
   routine *ROUTINE;
   library CHAR(10) POS(81);          // pozice čítaná od 1
   internal_job_id CHAR(16) POS(380); // od verze 7.4
   system_name CHAR(6) POS(396);      // od verze 7.4
end-DS;
```

Překrytí v datové struktuře

Přektytí podpole daného jménem a pozicí

```
dcl-ds *n;            // nepojmenovaná struktura
   field1 CHAR(10) inz('1234567890');
   array1 CHAR(1) dim(5) OVERLAY(field1);
   array2 CHAR(1) dim(5) OVERLAY(field1: 6); // pozice v podpoli
end-ds;

dcl-ds DATUM;
   RRRRMMDD zoned(8);
   RRRR zoned(4) POS(1);                // pozice od začátku struktury
   MM   zoned(2) OVERLAY(RRRR: *NEXT); // následující pozice
   DD   zoned(2) OVERLAY(MM: *NEXT);
end-ds;
```

Poznámka: V OVERLAY nesmí být jméno struktury (DATUM). Místo toho použijeme POS(1).

Překrytí oblasti od podpole daného jménem (od verze 7.4)

```
dcl-ds struct;
   string  CHAR(100);
   P1      CHAR(5);
   P2      CHAR(5);
   P3      CHAR(5);
   array   CHAR(5)  dim(3)  SAMEPOS(P1); // array překrývá P1, P2, P3
end-ds;
```

Externě popsaná datová struktura

```
dcl-DS STAVYP  EXT  end-DS;   // END-DS je součástí příkazu

dcl-DS EXTDS1  EXTNAME('STAVYP') end-DS;

dcl-DS STAVY_DS LIKEREC(STAVYPF0: *input); // bez END-DS, automaticky qualified
dcl-F  STAVYP; // pro LIKEREC musí být definován soubor


// *EXTDFT znamená předvolené hodnoty z DDS
dcl-DS extds2 EXTNAME('CENIKP') INZ(*EXTDFT) qualified; // INZ je až za EXTNAME
    material EXTFLD('MATER');     // přejmenování
    cena     EXTFLD  INZ(2.00);   // změna hodnoty (EXTFLD bez parametru)
    dodatek char(10) inz('ABC');  // až po existujících polích
end-DS;

extds2.dodatek = 'DEF';  // kvalifikace jménem struktury
```

Poznámka: Konstantní jména objektů v parametrech je nutné psát s apostrofy!

Data area (datová oblast)

DTAARA { (name) } jako samostatná proměnná nebo podpole

```
dcl-S  ds_area  char(100) DTAARA; // proměnná ds_area, objekt '*LIBL/DS_AREA'

dcl-DS data_struct;
   area  char(100) DTAARA ('DS_AREA');  // podpole area, objekt '*LIBL/DS_AREA'
end-DS;
```

DTAARA { {*AUTO} {*USRCTL} {(name)} } jako datová struktura

```
dcl-S  areaName char(21) inz('VZRPG72/DS_AREA'); // jméno oblasti (i z parametru)
...
dcl-DS  data_struct  DTAARA (areaName); // objekt 'VZRPG72/DS_AREA'
   podpole   char(10);
   podpole2  zoned(5);
end-DS;

dcl-DS *N DTAARA (*AUTO); // Local data area (*LDA)
   podpole   char(10);
   podpole2  char(5);
end-DS;
```

Poznámka: Konstantní jména objektů v parametrech je nutné psát s apostrofy!

Vektory

Proměnlivá dimenze (od verze 7.4)

Vektory, tabulky, vektory datových struktur.

DIM ({ *AUTO | *VAR : } num-const | *CTDATA)
definuje vektor s maximálním počtem prvků a nulovým aktuálním počtem prvků.

%ELEM (array-name : { *ALLOC | *KEEP | *MAX })
zjišťuje nebo mění aktuální počet prvků.

*AUTO - automaticky zvyšuje aktuální počet prvků.

```
dcl-s AUTO_ARR char(3) DIM(*AUTO: 1000);          // 0 prvků

auto_arr(100) = 'abc';          // 100 prvků
auto_arr(50) = 'abc';           // 100 prvků
%ELEM(auto_arr) = 25;           // 25 prvků
auto_arr(*NEXT) = 'abc';        // 26 prvků
%ELEM(auto_arr: *ALLOC) = 10;   // nechá 26 prvků
%ELEM(auto_arr: *ALLOC) = 120;  // zvýší na 120 prvků
num = %ELEM(auto_arr: *ALLOC);  // num = 120
num = %ELEM(auto_arr: *MAX);    // num = 1000
```

*VAR - aktuální počet prvků určuje jen funkce %ELEM

```
dcl-s VAR_ARR  char(1) INZ('*') DIM(*VAR: 1000);   // 0 prvků

%ELEM(var_arr) = 3;           // |*|*|*|            // 3 prvky
var_arr(1) = 'a';             // |a|*|*|
var_arr(2) = 'b';             // |a|b|*|
var_arr(3) = 'c';             // |a|b|c|
%ELEM(var_arr) = 1;           // |a|
%ELEM(var_arr: *KEEP) = 2;    // |a|b|    // zachová hodnoty prvků
%ELEM(var_arr) = 3;           // |a|b|*|  // zde předvolenou hvězdičku
```

*CTDATA - počet prvků je dán počtem řádků za **CTDATA.

```
dcl-s  VEKTOR  packed(5: 2)
       DIM(*CTDATA) CTDATA PERRCD(1) EXTFMT(S);     // 3 prvky

**CTDATA  vektor - 5 míst, z toho dvě desetinná
00100
00200
00300
```

Vnořená struktura, šablona, pointer

```
dcl-ds info_t TEMPLATE qualified; // šablona
   jmeno  varchar(25);

   dcl-ds addresa DIM(2); // vektor vnořené datové struktury
      ulice  varchar(25);
      mesto  varchar(25);
   end-ds;

end-ds;

dcl-ds informace LIKEDS(info_t) DIM(2) BASED(basPtr);

dcl-s basPtr POINTER;    // ukazatel na datovou strukturu
basPtr = %ALLOC(%size(informace: *all));  // získám paměť ze systému

informace(2).jmeno = 'Štěpán Řihošek';
informace(2)  .addresa(1).mesto = 'Praha 9';
informace(2)  .addresa(1).ulice = 'Křižíkova 45';
informace(2)  .addresa(2).mesto = 'Praha 2';
informace(2)  .addresa(2).ulice = 'Korunní 81';
```

Program s externím jménem souboru

Soubor CENIKP - **Ceník** materiálovych zásob

```
     A                                      UNIQUE
     A          R CENIKPF0
     A            MATER          5         COLHDG('Číslo' 'mater.')
     A            CENA          10  2      COLHDG('Cena' 'jedn.')
     A            NAZEV         50         COLHDG('Název' 'materiálu')
     A          K MATER
```

```
dcl-PI *N;  // rozhraní volaného programu EXTF
   filename char(21); // vstupní parametr (plné jméno souboru)
end-PI;
dcl-F ceny                        // jméno pro program
      EXTFILE(filename)           // jméno pro výpočet
      EXTDESC('*LIBL/CENIKP');    // jméno pro kompilaci
read ceny;
dsply nazev;
*inlr = *on;
```

Volání z RPG

```
dcl-PR EXTF EXTPGM;  // prototyp
   *N char(21);  // vstupní parametr (plné jméno souboru)
end-PR;
dcl-S qualifName char(21) inz('VZRPG_FREE/CENIKP');
CALLP EXTF(qualifName);
return;
```

Volání z CL

```
CALL EXTF 'VZRPG_FREE/CENIKP'
```

nebo

```
/* OVRDBF je silnější */
OVRDBF     FILE(CENIKP) TOFILE(VZRPG_FREE/CENIKP)
CALL       PGM(EXTF) PARM('VZRPGIV/CENIKP')
```

Program bez RPG cyklu

```
ctl-OPT MAIN (LINPGM)   // jmenuje hlavní proceduru modulu
        dftactgrp(*NO); // kvůli podproceduře PROC1
dcl-F  CENIKP;          // globální soubor generuje popisy I a O

dcl-PROC LINPGM;        // lineární hlavní procedura (program)
    dow not %eof;
        read  CENIKP;
        dsply MATER;
    enddo;
    PROC1();       // volání podprocedury PROC1
end-PROC;

dcl-PROC PROC1;    // podprocedura bez parametrů a návratové hodnoty
    dsply ('ABC');
end-PROC;
```

Volání z RPG s prototypem

```
dcl-PR LINPGM EXTPGM end-PR;  // prototyp
CALLP LINPGM();    // volání programu (CALLP nepovinné, závorky povinné)
```

Volání z CL

```
CALL PGM(LINPGM)
```

Lokální soubor v podproceduře

```
  // hlavní procedura volá podproceduru VRATIT_CENU
ctl-OPT dftactgrp(*no);  // kvůli podproceduře v modulu
dsply %editc(VRATIT_CENU('00001'): 'P'); // volání podprocedury
return;

  // podprocedura vrací cenu pro číslo materiálu
dcl-PROC VRATIT_CENU export;
    dcl-F CENIKP keyed;  // příp. i STATIC
    dcl-DS cenik_ds likerec(CENIKPF0); // je nutná datová struktura

    dcl-PI *N packed(10: 2);    // rozhraní podprocedury
        material like(cenik_ds.MATER) CONST; // nebo VALUE
    end-PI;

    chain material  CENIKP  cenik_ds;  // čte do datové struktury
    return cenik_ds.CENA;              // vrací hodnotu
end-PROC;
```

- Lokální soubor negeneruje popisy I a O s proměnnými.
- Pro datová pole je nutné použít datovou strukturu nebo funkci %fields u UPDATE.
- Soubor se otevře vždy při vstupu do podprocedury. Uzavírá se při výstupu z podprocedury a proměnné zanikají.
- Klíčové slovo STATIC u souboru nechá soubor otevřený a zachová jeho data pro příští volání.

Soubor jako parametr volání

Společný /INCLUDE člen FILPAR_C pro oba moduly

```
// Šablona pro soubor
  dcl-f infile_t TEMPLATE extdesc('CENIKP') block(*yes); // soubor pro kompilaci

// Šablona pro formát souboru
  dcl-ds infmt_t TEMPLATE likerec(CENIKPF0);

// Prototyp procedury zpracující soubor
  dcl-pr PROC_INFILE;
     *n    LIKEFILE(infile_t);  // soubor ze šablony
     *n    likeds(infmt_t);     // formát ze šablony jako sdružená proměnná
  end-pr;
```

Modul FILPAR (hlavní program)

```
/include FILPAR_C

// Definuji soubor pomocí LIKEFILE ze šablony
  dcl-f inp_file LIKEFILE(infile_t) extfile('CENIKP');

// Definuji formát souboru ze šablony jako datovou strukturu.
// Tu potřebuji v proceduře jako operand příkazu READ.
  dcl-ds inp_fmt likeds(infmt_t);

// volám proceduru PROC_INFILE, která zpracuje soubor
  PROC_INFILE (inp_file: inp_fmt);
  return;  // a končím
```

Modul FILPAR_P (podprocedura)

```
ctl-opt NOMAIN;
/include FILPAR_C

dcl-proc PROC_INFILE                export;

    dcl-pi *n; // rozhraní procedury bez jména
       file  LIKEFILE(infile_t);  // soubor
       fmt   LIKEDS(infmt_t);     // formát - datová struktura
    end-pi;

    // soubor se otevře při vstupu do procedury
    read file fmt;  // operace READ musí použít datovou strukturu
    dow not %eof;
       dsply (fmt.MATER + ' ' + %editc(fmt.CENA: 'P'));
       read file fmt;
    enddo;
    // soubor je lokální, zavře se automaticky
end-proc PROC_INFILE;
```

Spojení modulů do programu

```
CRTPGM PGM(FILPAR) MODULE(FILPAR FILPAR_P)
```

Použití SQL v RPG

Dlouhá SQL jména

```
// Vytvoření tabulky CENY_ZBOZI (v běžné knihovně) a naplnění záznamy
exec sql SET OPTION COMMIT = *NONE, DECMPT = *COMMA ;

exec sql CREATE OR REPLACE TABLE CENY_ZBOZI
         ( CISLO_ZBOZI CHAR(5),
           CENA_JEDN   DEC(12, 2),
           NAZEV_ZBOZI CHAR(50)
         );
exec sql INSERT INTO CENY_ZBOZI values ('00001', 8,99, 'PIŠKOTY OPAVIA');
...
return;
```

```
// Zpracování tabulky CENY_ZBOZI (v běžné knihovně)
DCL-DS CENY ExtName('*LIBL/CENY_ZBOZI') END-DS;  // Host variables

Exec SQL declare CS cursor for
         select CISLO_ZBOZI, CENA_JEDN, NAZEV_ZBOZI
         from   CENY_ZBOZI
         order by CISLO_ZBOZI ;
Exec SQL open CS;
Exec SQL fetch from CS into :CISLO00001, :CENA_JEDN, :NAZEV00001 ;
dow sqlstate < '02000';
    dsply CISLO00001;
    dsply CENA_JEDN;
    dsply NAZEV00001;
    Exec SQL fetch from CS into :CISLO00001, :CENA_JEDN, :NAZEV00001 ;
enddo;
Exec SQL close CS;

return;
```

Alternativně si můžeme volit vlastní systémová jména

```
exec sql CREATE OR REPLACE TABLE CENY_ZBOZI FOR SYSTEM NAME  kratší-jméno
         ( CISLO_ZBOZI FOR COLUMN SYSTEM NAME  kratší-jméno  CHAR(5),
           CENA_JEDN   FOR COLUMN SYSTEM NAME  kratší-jméno  DEC(12, 2),
           NAZEV_ZBOZI FOR COLUMN SYSTEM NAME  kratší-jméno  CHAR(50)
         );
```

Kompilace modulu s příkazy SQL

Zdrojový typ je SQLRPGLE. Příkaz pro kompilaci je CRTSQLRPGI. Typ objektu je určen parametrem OBJTYPE.

- *Zdrojový člen* v souboru QRPGLESRC:

```
CRTSQLRPGI OBJ(SQLPROG) SRCFILE(QRPGLESRC) SRCMBR(*OBJ)
           OBJTYPE( *PGM | *MODULE | *SRVPGM )
```

- *Stream file* v IFS adresáři /HOME/VZRPG72:

```
CRTSQLRPGI OBJ(SQLPROG) SRCSTMF('/home/vzrpg72/sqlprog.SQLRPGLE')
           OBJTYPE( *PGM | *MODULE | *SRVPGM )
```

Datové typy z SQL

Odpovídající typy

RPG	SQL
CHAR(n)	CHAR(n)
VARCHAR(n:2)	VARCHAR(n)
UCS2(n)	GRAPHIC(n)
VARUCS2(n:2)	VARGRAPHIC(n)
PACKED(n:m)	DECIMAL(n, m)
ZONED(n:m)	NUMERIC(n, m)
INT(5)	SMALLINT
INT(10)	INTEGER
INT(20)	BIGINT
UNS(n)	-
BINDEC(1-4:0)	SMALLINT
BINDEC(5-9:0)	INTEGER
FLOAT(4)	FLOAT(24) / REAL
FLOAT(8)	FLOAT(53) / FLOAT
IND	-
DATE(*DMY-)	DATE
TIME(*HMS.)	TIME
TIMESTAMP(n)	TIMESTAMP(n)
POINTER	-

Neodpovídající typy

```
dcl-S bin1    SQLTYPE(BINARY: 50);     // CHAR(50) CCSID(*HEX);
dcl-S varbin1 SQLTYPE(VARBINARY: 100); // VARCHAR(100) CCSID(*HEX);
```

aj. např.:

```
dcl-S field1  SQLTYPE (CLOB: 1000 ); // DCL-DS FIELD1;
                                     //    FIELD1_LEN  UNS(10);
                                     //    FIELD1_DATA CHAR(1000);
                                     // END-DS FIELD1;

exec SQL set :field1 = 'ABCD'; // příkaz SQL v RPG
field1_data = 'ABCD';          // příkaz v RPG
```

Dodatek 1

Kompilace RPG modulu v IFS adresáři

```
CRTBNDRPG PGM(RPGPROG) SRCSTMF('/home/vzrpg72/rpgprog.RPGLE')
CRTRPGMOD MODULE(RPGPROG) SRCSTMF('/home/vzrpg72/rpgprog.RPGLE')
```

- Kompilační příkazy CRTBNDRPG a CRTRPGMOD obsahují parametr TGTCCSID, který umožní určit způsob zakódování zdrojového textu.
- Je-li zdrojový text kódován v Unicode (UTF-8 nebo UCS-2), umožní jeho kompilaci volba **TGTCCSID(*JOB)**. Tato možnost je míněna pro zdrojové texty umístěné v IFS.
- Fyzický zdrojový soubor nelze vytvořit s kódem Unicode.

Třídění a vyhledávání ve vektoru datových struktur

Soubor CENIKP - **Ceník** materiálovych zásob

```
     A                                      UNIQUE
     A          R CENIKPF0
     A            MATER          5         COLHDG('Číslo' 'mater.')
     A            CENA          10  2      COLHDG('Cena' 'jedn.')
     A            NAZEV         50         COLHDG('Název' 'materiálu')
     A          K MATER
```

```
dcl-f CENIKP keyed;
dcl-s  idx   int(5);
dcl-c  pocet const(5);

dcl-ds STR LIKEREC(CENIKPF0) DIM(pocet) inz;

dow not %eof and idx < pocet;
   idx += 1;
   read CENIKP STR(idx);  // čte do položky struktury STR
enddo;

SORTA(D) STR(*).CENA; // setřídí STR sestupně podle ceny
SORTA STR(*).NAZEV;   // setřídí vzestupně podle názvu
idx = %LOOKUP(10.99: STR(*).CENA);   // zjistí index podle ceny
SORTA(D) %SUBARR(STR(*).CENA: 2: 3); // setřídí jen položky 2 až 4
idx = %LOOKUP('00002': STR(*).MATER; // index podle materilálu
```

Poznámka: Nelze použít funkce %LOOKUPxx s relačním operátorem, protože vektory nelze označit jako ASCEND nebo DESCEND.

Program DTAAREA4

```
dcl-pi DTAAREA4;
    areaName char(21);
end-pi;

//dcl-S  areaName char(21) inz('VZRPG72/DS_AREA');

dcl-DS area DTAARA (areaName); // objekt 'VZRPG72/DS_AREA'
   podpole  char(10);
   podpole2 zoned(5);
end-DS;

IN *lock area;          // zamknu
podpole  = 'aaaaaaaaaa'; // naplním podpole
podpole2 = 33333;
OUT area;               // přepíšu
IN area;                // přečtu
UNLOCK area;            // odemknu

return;
```

Volání systémového programu QCMDEXC s prototypem

```
// referenční soubor pro popis polí
DCL-F QADSPFFD usropn extfile('QTEMP/DSPFFD');

dcl-s cmdText char(200);
dcl-s cmdLen  packed(15: 5);

// prototyp volání programu QCMDEXC
DCL-PR QCMDEXC EXTPGM;
    *n  char(200);      // nepojmenovaný parametr
    *n  packed(15: 5);
END-PR;

// sestavím parametry pro progam QCMDEXC
cmdText = 'DSPFFD FILE(VZRPG72/CENIKP) OUTPUT(*OUTFILE) ' +
          'OUTFILE(QTEMP/DSPFFD)';
cmdLen = %len(%trimr(cmdText));

// volání programu QCMDEXC - provede příkaz DSPFFD
CALLP QCMDEXC (cmdText: cmdLen);

open QADSPFFD;   // otevřu soubor s popisy polí (teď už existuje)
read QADSPFFD;   // čtu první popis datového pole
...
close QADSPFFD;  // zavřu soubor popisů polí
```

Použití šablony TEMPLATE

Platí pouze při kompilaci

V popisu souborů

```
dcl-F  file_t  TEMPLATE  extdesc('CENIKP');
dcl-F  file  LIKEFILE(FILE_T);
```

V popisu dat

```
dcl-S  var_t  ...  TEMPLATE;
dcl-S  var  LIKE(var_t);

dcl-DS struct_t  TEMPLATE  likerec(CENIKPF0);
dcl-DS struct  LIKEDS(struct_t)  INZ(*LIKEDS);

dcl-PR procname;
   *n     LIKEFILE(file_t);
   *n     LIKEDS(struct_t);
end-pr;

dcl-PI *n;
   file   LIKEFILE(file_t);
   struct LIKEDS(struct_t);
end-pi;
```

Obtížně převoditelné příkazy

Příkaz KLIST

Pevný formát

```
FSTAVYP    UF   E             KDISK
C                    MOVEL     '01'           ZAVOD
C                    MOVEL     '01'           SKLAD
C                    MOVEL     '00001'        MATER

C      KLIC          CHAIN     STAVYP

C      KLIC          KLIST
C                    KFLD                     ZAVOD
C                    KFLD                     SKLAD
C                    KFLD                     MATER
```

Volný formát

```
dcl-f  STAVYP keyed;

dcl-ds klicStavu LIKEREC(STAVYPF0: *KEY);

klicStavu.ZAVOD = '01';
klicStavu.SKLAD = '01';
klicStavu.MATER = '00001';

CHAIN   %KDS(klicStavu : 3) STAVYP;
```

```
ZAVOD = '01';
SKLAD = '01';
MATER = '00001';

CHAIN   (ZAVOD: SKLAD: MATER) STAVYP;
```

Příkaz CALL a PLIST

Pevný formát

```
c                   CALL      'PGM1'
c                   PARM      'XXX'         A                10
c                   PARM      'YYY'         B                10
```

```
c                   CALL      'PGM1'        PARAMS

c     PARAMS        PLIST
c                   PARM      'XXX'         A                10
c                   PARM      'YYY'         B                10
```

Volný formát - nutno doplnit deklarace a příkazy

```
DCL-PR PGM1 EXTPGM; // prototyp volání
    A char(10);
    B char(10);
END-PR;

DCL-S A char(10);  // přenesené deklarace skutečných parametrů
DCL-S B char(10);

A = 'XXX';         // hodnoty skutečných parametrů
B = 'YYY';

~~CALLP~~ PGM1 ( A : B );
```

Volaný program

```
c     *ENTRY        PLIST
c                   PARM                    A              10
c                   PARM                    B              10

c     A             dsply
c     B             dsply
c                   return
```

```
DCL-PI *N ;         // program interface
    A char(10);
    B char(10);
END-PI;

dsply A;
dsply B;
return;
```

Direktivy pro kompilátor

Zapisují se ve formulářovém zápisu od sloupce **7**, ve volném tvaru od sloupce **1**.

Organizace protokolu o kompilaci

/TITLE, /EJECT, /SPACE

Začlenění úseků zdrojového textu

/INCLUDE, /COPY

Podmínění úseků zdrojového textu

/DEFINE condition-name, /UNDEFINE condition-name,
/IF {NOT} DEFINED(condition-name),
/ELSEIF {NOT} DEFINED(condition-name),
/ELSE,
/ENDIF,
/EOF

```
/DEFINE FIXED
...
DCL-S name
  /IF DEFINED(FIXED)
      CHAR(10);
  /ELSE
      VARCHAR(10)
  /ENDIF
;
```

Parametr DEFINE v příkazech CRTRPGMOD a CRTBNDRPG slouží jako /DEFINED až pro 32 podmínek.
Předvolená jména jsou *VnRnMn, *ILERPG, *CRTBNDRPG, *CRTRPGMOD.

Dočasné nastavení a obnovení hodnot předvolených v CTL-OPT

/SET, /RESTORE

Začátek a konec volného zápisu výpočetních příkazů

~~/FREE, /END-FREE~~

Dodatek 2

Lokální SQL tabulka v podproceduře - alternativa k lokálnímu souboru

```
 // hlavní procedura volá podproceduru VRATIT_CENU
ctl-OPT dftactgrp(*no);  // kvůli podproceduře v modulu
dsply %editc(VRATIT_CENU('00001'): 'P'); // volání podprocedury
return;

  // podprocedura vrací cenu pro číslo materiálu
dcl-PROC VRATIT_CENU export;
    dcl-ds cenik_ds extname('VZRPG72/CENIKP') qualified end-ds;

    dcl-pi *n packed(10: 2);    // rozhraní podprocedury
        material like(cenik_ds.MATER) CONST; // nebo VALUE
    end-pi;

    exec SQL declare CS cursor for
            select CENA
            from CENIKP
            where MATER = :material;
    exec SQL open CS;
    exec SQL fetch from CS into :cenik_ds.CENA;

    return cenik_ds.CENA;
end-PROC;
```

Plnění subfile z databáze

```
dcl-f STAVYW2 workstn sfile(DATA: idx); // display file

dcl-ds STAVYP extname('STAVYP') end-ds; // host variables
dcl-s ZAV char(2); // host variable pro výběr závodu (z obrazovky)
dcl-s idx int(10);

Exec SQL declare CURSOR cursor for
    select ZAVOD, SKLAD, MATER, MNOZ from STAVYP
        where ZAVOD >= :ZAV
        order by SKLAD, MATER asc ;
Exec SQL open CURSOR;
Exec SQL fetch from CURSOR into :ZAVOD, :SKLAD, :MATER, :MNOZ ;
dow not %eof; // plním subfile
    idx += 1;
    write DATA;
    Exec SQL fetch from CURSOR into :ZAVOD, :SKLAD, :MATER, :MNOZ ;
enddo;
Exec SQL close CURSOR;
...
exfmt RIDICI;
...
*inlr = *on;
```

Aktualizace stavů obraty

Soubor STAVYP - **Stavy** materiálových zásob

```
A                                                UNIQUE
A          R STAVYPF0
A            ZAVOD          2          COLHDG('Záv')
A            SKLAD          2          COLHDG('Skl')
A            MATER          5          COLHDG('Číslo' 'mater.')
A            MNOZ         10P 2        COLHDG('Množství')
A          K ZAVOD
A          K SKLAD
A          K MATER
```

Soubor OBRATP - **Obraty** materiálu

```
A          R OBRATPF0
A            ZAVOD          2          COLHDG('Záv')
A            SKLAD          2          COLHDG('Skl')
A            MATER          5          COLHDG('Číslo' 'mater.')
A            MNOBR        10P 2        COLHDG('Množství obratu')
A          K ZAVOD
A          K SKLAD
A          K MATER
```

```
exec SQL set option COMMIT = *NONE;

dcl-DS STAVY ExtName('*LIBL/STAVYP') END-DS; // host variables
dcl-DS obrat ExtName('*LIBL/OBRATP') qualified END-DS; // odstraní duplicity

exec SQL declare CS cursor for
         select ZAVOD, SKLAD, MATER, MNOZ
         from STAVYP
         order by ZAVOD, SKLAD, MATER ;
exec SQL open CS;
exec SQL fetch from CS into :STAVY;
dow sqlstate < '02000';
    exec SQL UPDATE STAVYP set MNOZ = MNOZ +
                ( select sum(MNOBR)
                  from OBRATP
                  where ZAVOD = :ZAVOD
                    and SKLAD = :SKLAD
                    and MATER = :MATER
                  group by ZAVOD, SKLAD, MATER
                )
               where ZAVOD = :ZAVOD and SKLAD = :SKLAD and MATER = :MATER;
    exec SQL fetch from CS into :STAVY;
enddo;
exec SQL close CS;
return;
```

Aktualizace příkazem MERGE

Bez použití host variables

```
exec SQL MERGE INTO STAVYP S
   USING (select O.ZAVOD, O.SKLAD, O.MATER, sum(O.MNOBR) SUMA_OBRATU
          from OBRATP O
          group by O.ZAVOD, O.SKLAD, O.MATER
         ) as OBR
   ON ( S.ZAVOD = OBR.ZAVOD
    and S.SKLAD = OBR.SKLAD
    and S.MATER = OBR.MATER )
   when MATCHED THEN
    UPDATE set S.MNOZ = S.MNOZ + OBR.SUMA_OBRATU ;
return;
```

Aktualizace příkazy RPG (bez SQL)

Příklad pro porovnání s SQL

```
DCL-F STAVYP keyed usage(*update);
DCL-F OBRATP keyed;

read STAVYP;
dow not %eof(STAVYP);
    setll (ZAVOD: SKLAD: MATER) OBRATP;
    if %equal;
        reade (ZAVOD: SKLAD: MATER) OBRATP;
        dow not %eof(OBRATP);
            MNOZ += MNOBR;
            reade (ZAVOD: SKLAD: MATER) OBRATP;
        enddo;
    endif;
    update STAVYPF0 %fields(MNOZ);
    read STAVYP;
enddo;
*inlr = *on;
```

Stored procedure

```
create procedure CENA_DODAVEK
   ( in DATUM1 date,                          -- počáteční datum
     in DATUM2 date,                          -- koncové datum
     out CASTKA_CELKEM decimal (11, 2),       -- Součet částek včetně DPH
     out CASTKA_BEZ_DPH decimal (11, 2) ,     -- Součet částek bez DPH
     out POCET_OBJEDNAVEK  integer            -- Počet objednávek v rozmezí datumů
   )
   language RPGLE                            -- jazyk ILE RPG
   parameter style general                   -- nepředávají se null-indikátory
   not deterministic
   reads SQL data
   program type main                         -- procedura je hlavní program
   external name VZSQLPGM/CENA_DOD           -- program je v jiné knihovně než data
```

Program CENA_DOD realizuje uloženou proceduru.

Z RPG programu vyvoláme uloženou proceduru příkazem

```
exec SQL call CENA_DODAVEK ( :datum1, :datum2,
                             :suma_s_dph, :suma_bez_dph, :pocet_objedn );
```

Spouštěč (trigger)

Registrace triggeru:

```
ADDPFTRG FILE(VZSQL/OBJDET_T) TRGTIME(*BEFORE) TRGEVENT(*INSERT)
     PGM(VZSQLPGM/TRGMNOBJI) RPLTRG(*YES) TRG(TRGMNOBJI) TRGLIB(*FILE)
     ALWREPCHG(*YES)
```

Trigger program:

```
dcl-PI *n;      // vstupní parametry z databázového systému
    Buf         likeDS(TrgBuffer);
    Len         like(TrgBufLen);
end-PI;
Dcl-DS TrgBuffer      Len(1000) qualified; // data nového záznamu
   ...
   NewRecOffset       Uns(10)  pos(65);
   ...
End-DS;
Dcl-S  NewRecPtr      Pointer;              // ukazatel na nový záznam
Dcl-DS NewRecord      ExtName('OBJDET_T') Based(NewRecPtr) End-DS;

NewRecPtr = %Addr(Buf) + Buf.NewRecOffset;
If MNOBJ > 1000; // Je-li nově objednané množství > 1000,
   MNOBJ = 0;    // dosadí 0.
EndIf;
Return;
```

Uživatelská funkce SQL (UDF)

- Vytvoříme externí uživatelskou funkci, která přijímá dva parametry a vrací tabulku předepsaného tvaru.

```
CREATE FUNCTION OBJDAT_F (DATUM1 DATE, DATUM2 DATE)
   RETURNS TABLE
   ( COBJ CHAR(6) CCSID 870,
     DTOBJ DATE )
   LANGUAGE RPGLE
   PARAMETER STYLE DB2SQL        -- umožňuje open, fetch, close
   NOT DETERMINISTIC
   READS SQL DATA
   PROGRAM TYPE SUB              -- funkce je ILE podprocedura
   SCRATCHPAD                    -- paměť přetrvávající jednotlivá volání
   NOT FENCED                    -- není vázána na stejnou úlohu
   NO FINAL CALL                 -- není první a poslední volání
   CARDINALITY 1000              -- přibližné omezení počtu výsledných řádků
   EXTERNAL NAME VZSQLPGM/OBJDAT_F(OBJDAT_F) -- servisní program a procedura
```

- Vytvoříme modul OBJDAT_F s procedurou (funicí) OBJDAT_F. Funkce OBJDAT_F přijímá dva parametry určující rozmezí datumů. Vybere objednávky z daného rozmezí a ***vytvoří tabulku*** s čísly objednávek a s datumy. Seznam bude uspořádán podle čísla objednávky vzestupně.

- Vytvoříme servisní program OBJDAT_F z modulu OBJDAT_F s procedurou (funkcí) OBJDAT_F. Modul se kompiluje příkazem

```
CRTSQLRPGI OBJ(VZSQLPGM/OBJDAT_F) SRCFILE(VZSQLPGM/QRPGLESRC) SRCMBR(OBJDAT_F)
           COMMIT(*NONE) OBJTYPE(*SRVPGM) CLOSQLCSR(*ENDACTGRP)
```

K tomu je nutný spojovací text (binder source) v souboru QSRVSRC

```
STRPGMEXP SIGNATURE('VER1')
EXPORT SYMBOL(OBJDAT_F)
ENDPGMEXP
```

- Vytvoříme program OBJDAT_P s připojeným servisním programem OBJDAT_F

```
CRTPGM PGM(OBJDAT_P) BNDSRVPGM((OBJDAT_F))
```

- Z programu OBJDAT_P voláme uživatelskou funkci OBJDAT_F příkazy

```
exec SQL declare CUR cursor for
         select * from TABLE ( OBJDAT_F(:datum1, :datum2 ) ) as DAT_F;
exec SQL open CUR ;
exec SQL fetch CUR into :COBJ, :DTOBJ ;
dow  sqlstate = '00000';  // dokud jsou nějaké záznamy
   except    DETAIL;
   exec SQL fetch CUR into :COBJ, :DTOBJ ;
enddo;
exec SQL close CUR;
```

- Procedura je volána vícekrát, vždy s určitým typem volání:
  - Open = -1 jednou
  - Fetch = 0 vícekrát
  - Close = 1 jednou