

# **Použití SQL v RPG**

Vladimír Župka, 2025

Použití SQL v RPG	1
Předkompilátor SQL pro jazyk RPG.....	4
Zápis příkazů v RPG programu .....	5
Statické a dynamické příkazy .....	6
Postup příkazů pro SELECT .....	6
Postup příkazů pro dynamický SELECT bez parametrů (markerů) .....	6
Postup příkazů pro dynamický SELECT s parametry (markery).....	6
Postup příkazů pro ostatní dynamické příkazy bez parametrů (markerů) .....	7
Postup příkazů pro ostatní dynamické příkazy s parametry (markery).....	7
Soubory pro příklady.....	8
Soubor CENIKP – Ceník materiálu .....	8
Soubor STAVYP – Stavy materiálových zásob .....	8
Soubor OBRATP – Obraty materiálu .....	8
Stavy a obraty – statické SQL příkazy SELECT, UPDATE.....	9
Chybové stavy.....	10
Aktualizace příkazem MERGE.....	11
Lokální SQL tabulka v podproceduře .....	12
Dynamický SELECT s markery .....	13
Dynamický UPDATE – EXECUTE IMMEDIATE .....	14
Dynamický příkaz – PREPARE, EXECUTE, marker.....	15
Příkaz k vyvolání programu.....	15
Dlouhá jména .....	16
Vytvoření SQL tabulky a naplnění záznamy.....	16
Výpis cen zboží do job logu.....	18
Datové typy RPG a SQL.....	19
Neodpovídající typy .....	20
Trigger – zvýšení ceny nejvýše o 10% při aktualizaci záznamu .....	21
Trigger SQL.....	21

Trigger externí (RPG).....	23
Trigger externí – registrace .....	24
Uložená procedura – hodnota všeho materiálu ve skladě .....	25
Procedura v SQL .....	25
Volání procedury v programu – CALL .....	26
Procedura externí – realizace programem.....	28
Procedura externí – definice .....	29
User defined function (UDF) – vrací jednotkovou cenu materiálu .....	30
Funkce v SQL .....	30
Volání funkce v programu .....	30
Funkce externí – realizace podprocedurou .....	31
Funkce externí – definice.....	32
User defined table function (UDTF) – vrací tabulku čísel a dat objednávek .....	33
Volání funkce v programu OBJDAT_P .....	33
User defined table function (UDTF) – SQL skript .....	34
User defined table function (UDTF) – RPG podprocedura .....	35
PL/SQL script pro externí RPG funkci OBJDAT_F .....	35
Modul OBJDAT_F pro servisní program OBJDAT_F.....	37

# Předkompilátor SQL pro jazyk RPG

*Zdrojový typ* **SQLRPGLE**

*Kompilace* **CRTSQLRPGI** – Create SQL ILE RPG Object

*Source member*

```
CRTSQLRPGI OBJ(PROG) SRCFILE(QRPGLESRC) SRCMBR(*OBJ)
```

*Stream file*

```
CRTSQLRPGI OBJ(PROG) SRCSTMF(' /home/vzrpg72/prog.SQLRPGLE ')
```

*Parametry překladu*

```
CRTSQLRPGI OBJ(PROG) SRCFILE(QRPGLESRC) SRCMBR(*OBJ)
```

```
    COMMIT(*CHG)                                *NONE *ALL ...
```

```
    OBJTYPE(*PGM)                                *MODULE *SRVPGM
```

```
    CLOSQLCSR(*ENDACTGRP)                        *ENDMOD    kdy se uzavře kurzor
```

```
    ...
```

```
    DFTRDBCOL(*NONE)                             předvolené SQL schema (default collection)
```

```
    DYNDFTCOL(*NO)                                *YES – DFTRDBCOL také pro dynamické příkazy
```

```
    DBGVIEW(*NONE)                                *SOURCE, *STMT, *LIST
```

```
    TOSRCFILE(QTEMP/QSQLTEMP1)                  kam přijde upravený zdroj pro kompilaci
```

```
    OPTION(*SYS | *SQL                            naming convention
```

```
            *JOB | *SYSVAL | *PERIOD | *COMMA
```

```
    ...)
```

# Zápis příkazů v RPG programu

*Ve volném formátu*

EXEC SQL *příkaz* ;

*V pevném formátu*

C/EXEC SQL

C+ *příkaz*

C/END-EXEC

Příkaz SET OPTION představuje alternativní a doplňkové volby pro SQL příkazy. Některé jsou shodné s parametry předkompilátoru.

SET OPTION

COMMIT = \*CHG \*NONE \*ALL ...

DATFMT = \*JOB \*ISO ...

DATSEP = \*JOB \*PERIOD \*COMMA \*DASH \*BLANK

DFTRDBCOL = \*NONE

LANGID = \*JOB \*JOBRUN CSY ENU DEU ...

NAMING = \*SYS \*SQL

SRTSEQ = \*JOB \*HEX \*JOBRUN \*LANGIDUNQ \*LANGIDSHR

TIMFMT = \*HMS \*ISO \*EUR \*USA \*JIS

TIMSEP = \*JOB \*COLON \*PERIOD \*COMMA \*BLANK

EXEC SQL SET OPTION LANGID = CSY, SRTSEQ = \*LANGIDSHR,  
DATFMT = \*ISO, COMMIT = \*NONE ;

# Statické a dynamické příkazy

Statické příkazy jsou zapsány přímo v příkazu EXEC SQL.

Dynamické příkazy jsou zapsány v textové proměnné.

## **Postup příkazů pro SELECT**

```
DECLARE SCROLL kurzor CURSOR FOR SELECT ...  
OPEN kurzor  
FETCH FIRST FROM kurzor INTO :proměnná, ... (NEXT, LAST, PRIOR, ...)  
CLOSE kurzor
```

## **Postup příkazů pro dynamický SELECT bez parametrů (markerů)**

```
text-příkazu = 'SELECT ... '  
PREPARE připravený-příkaz FROM :text-příkazu  
DECLARE SCROLL kurzor FOR připravený-příkaz  
... jako výše
```

## **Postup příkazů pro dynamický SELECT s parametry (markery)**

```
text-příkazu = 'SELECT ... WHERE xyz BETWEEN ? AND ? ... '  
PREPARE připravený-příkaz FROM :text-příkazu  
DECLARE SCROLL kurzor FOR připravený-příkaz  
OPEN kurzor USING :proměnná1, :proměnná2  
... jako výše
```

### ***Postup příkazů pro ostatní dynamické příkazy bez parametrů (markerů)***

```
text-příkazu = 'UPDATE ... SET ... '  
EXECUTE IMMEDIATE :text-příkazu
```

### ***Postup příkazů pro ostatní dynamické příkazy s parametry (markery)***

```
text-příkazu = 'UPDATE ... SET ... WHERE xyz BETWEEN ? AND ?'  
PREPARE připravený-příkaz FROM :text-příkazu  
EXECUTE připravený-příkaz USING :proměnná1, :proměnná2
```

# Soubory pro příklady

## Soubor CENIKP – Ceník materiálu

A				UNIQUE
A	R	CENIKPF0		
A		<u>MATER</u>	5	COLHDG('Číslo' 'mater.')
A		CENAJ	10P 2	COLHDG('Cena/j.')
A		NAZEV	30	COLHDG('Název zboží')
A	K	MATER		

## Soubor STAVYP – Stavy materiálových zásob

A				UNIQUE
A	R	STAVYPF0		
A		ZAVOD	2	COLHDG('Záv')
A		SKLAD	2	COLHDG('Skl')
A		<u>MATER</u>	5	COLHDG('Číslo' 'mater.')
A		<b>MNOZ</b>	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		<u>MATER</u>	5	COLHDG('Číslo' 'mater.')
A		<b>MNOBR</b>	10P 2	COLHDG('Množství obratu')
A	K	ZAVOD		
A	K	SKLAD		
A	K	MATER		



# Stavy a obraty – statické SQL příkazy SELECT, UPDATE

Program STAOBR\_SQL

```
**free
```

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

```
Dcl-DS stavy ExtName('*LIBL/STAVYP') End-DS; // host variables
```

```
Dcl-DS obraty ExtName('*LIBL/OBRATP') qualified End-DS; // odstraní duplicitu
```

```
Exec SQL declare CS cursor for
```

```
select ZAVOD, SKLAD, MATER, MNOZ from STAVYP;
```

```
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;
```

## Chybové stavy

SQLSTATE		SQLCODE
0	Operace byla <b>úspěšná</b> , bez varování nebo výjimky.	0
1503	Počet výsledných sloupců je větší než poskytnutý počet proměnných.	+000, +030
2000	Nastala jedna z výjimek: <ul style="list-style-type: none"><li>- Výsledek příkazu SELECT INTO nebo subselektu v příkazu INSERT je <b>prázdná</b> tabulka.</li><li>- Počet řádků určených v hledacím příkazu UPDATE nebo DELETE je <b>nula</b>.</li><li>- Pozice kurzoru v příkazu FETCH je za posledním řádkem výsledné tabulky.</li><li>- <b>Orientace</b> příkazu FETCH je nesprávná.</li></ul>	100
7001	<b>Počet</b> proměnných neodpovídá počtu parametrů (markerů)	-313
9000	<b>Trigger</b> v SQL příkazu selhal.	-723
42703	Zjištěn <b>nedefinovaný</b> sloupec nebo jméno parametru.	-205, -206, -213, -5001

# Aktualizace příkazem MERGE

Program STAOBM\_SQL

```
**free
```

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

```
Exec SQL MERGE INTO STAVYP S      // statický příkaz
      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
-- when NOT MATCHED then
--      INSERT (ZAVOD, SKLAD, MATER, MNOZ)
--      values(OBR.ZAVOD, OBR.SKLAD, OBR.MATER, OBR.SUMA_OBRATU)
;

return;
```

# Lokální SQL tabulka v podproceduře

Program LOC\_SQL

```
**free
  // hlavní procedura volá podproceduru
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('CENIKP') qualified end-DS;
  dcl-PI *N packed(10: 2); // rozhraní podprocedury
    material like(cenik_ds.MATER) CONST; // nebo VALUE
end-PI;

Exec SQL select CENAJ into :cenik_ds.CENAJ from CENIKP
      where MATER = :material;
if sqlstate >= '02000';
  cenik_ds.CENAJ = -1;
endif;
return cenik_ds.CENAJ;

end-PROC;

.....
```

# Dynamický SELECT s markery

Program DYNSEL\_MK

```
**free
```

```
Dcl-S sel_stmt      char(500) inz;
```

```
Dcl-S spodni       packed(5) inz( 25 );
```

```
Dcl-S horni        packed(5) inz( 300 );
```

```
Dcl-DS *N  ExtName('CENIKP')  End-DS;
```

```
sel_stmt = 'select MATER, CENAJ, NAZEV from CENIKP +  
           where CENAJ between ? and ? +  
           order by CENAJ asc +  
           for read only' ;
```

```
Exec SQL prepare PREP from :sel_stmt ;
```

```
Exec SQL declare CUR cursor for PREP;
```

```
Exec SQL open CUR using :spodni, :horni;
```

```
Exec SQL fetch CUR into :MATER, :CENAJ, :NAZEV;
```

```
DoW sqlstate < '02000';
```

```
    snd-msg *info MATER + ' ' + %editc(CENAJ: 'K') + ' ' + NAZEV;
```

```
    Exec SQL fetch CUR into :MATER, :CENAJ, :NAZEV;
```

```
EndDo;
```

```
Exec SQL close CUR;
```

```
return;
```

# Dynamický UPDATE – EXECUTE IMMEDIATE

Program DYNEX\_IM

```
Dcl-S DYNST          Char(500) Inz;  
Dcl-S Limit          Packed(9:2) Inz(500.00);
```

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

```
DYNST = 'update CENIKP set CENAJ = CENAJ * 1.10 +  
        where CENAJ < '  
DYNST = %trim(DYNST) + ' ' + %char(Limit) ;
```

```
Exec SQL execute immediate :DYNST;
```

```
return;
```

# Dynamický příkaz – PREPARE, EXECUTE, marker

Program DYNEX\_MK

```
Dcl-S DYNST          Char(500) Inz;
```

```
Dcl-PI *n;
```

```
    Limit Packed(9:2);  // parametr z CMD příkazu  
End-PI;
```

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

```
DYNST = 'update CENYD_T set CENAJ = CENAJ * 1.10  +  
        where CENAJ < ? ';
```

```
Exec SQL prepare STMT from :DYNST;
```

```
Exec SQL execute STMT using :Limit;
```

```
return;
```

## ***Příkaz k vyvolání programu***

```
CMD          PROMPT('Volání SQL programu DYNEX_MK')
```

```
PARM         KWD(LIMIT) TYPE(*DEC) LEN(9 2) +  
             DFT(250) PROMPT('Limit ceny:')
```

# Dlouhá jména

## *Vytvoření SQL tabulky a naplnění záznamy*

Program DL\_JM1

```
Exec SQL set option COMMIT = *NONE, DECMPT = *COMMA ;
```

```
Exec SQL CREATE OR REPLACE TABLE CENY_ZBOZI
      ( CISLO_ZBOZI      CHAR(5)          UNIQUE,
        CENA_ZA_JEDNOTKU DEC(12, 2),
        NAZEV_ZBOZI      CHAR(50)
      ) ;
```

```
Exec SQL INSERT INTO CENY_ZBOZI values ('00001', 8,99, 'PIŠKOTY OPAVIA');
```

```
Exec SQL INSERT INTO CENY_ZBOZI values ('00003', 1.25, 'Prádelní šňůra');
```

```
Exec SQL INSERT INTO CENY_ZBOZI values ('00004', 10.50, 'Ponožky pánské tmavé');
```

```
Exec SQL INSERT INTO CENY_ZBOZI values ('00005', 120.00, 'Tričko bílé');
```

```
Exec SQL INSERT INTO CENY_ZBOZI values ('00006', 10.55, 'Ponožky pánské bílé');
```

...

```
return;
```



## ***V protokolu o kompilaci najdeme odpovídající systémová jména***

První 4 znaky zůstávají, přidá se pořadové číslo.

```
...  
  
CENA_ZA_JEDNOTKU      * * * *      COLUMN  
                        8  
CENA_ZA_JEDNOTKU      6              COLUMN FOR CENA_00001 IN CENY_ZBOZI  
  
CENA_00001            6              DECIMAL(12,2) COLUMN IN CENY_ZBOZI  
  
...
```

## ***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_ZA_JEDNOTKU FOR COLUMN SYSTEM NAME kratší-jméno DEC(12, 2),  
        NAZEV_ZBOZI FOR COLUMN SYSTEM NAME kratší-jméno CHAR(50)  
      );
```

## ***Výpis cen zboží do job logu***

Program DL\_JM2

```
Dcl-DS *N ExtName('CENY_ZBOZI') End-DS; // proměnné – host variables
```

```
Exec SQL declare CS cursor for
      select CISLO_ZBOZI, CENA_ZA_JEDNOTKU, NAZEV_ZBOZI
      from   CENY_ZBOZI
      order by CISLO_ZBOZI ;
```

```
Exec SQL open CS;
```

```
Exec SQL fetch from CS into :CISLO00001, :CENA_00001, :NAZEV00001 ;
```

```
dow sqlstate < '02000';
      snd-msg CISLO00001 + ' ' + %char(CENA_00001) + NAZEV00001;
      Exec SQL fetch from CS into :CISLO00001, :CENA_00001, :NAZEV00001 ;
enddo;
```

```
Exec SQL close CS;
```

```
return;
```

## Datové typy RPG a SQL

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
BINDEC(1-4:0)	SMALLINT
BINDEC(5-9:0)	INTEGER
FLOAT(4)	FLOAT(24) / REAL
FLOAT(8)	FLOAT(53) / FLOAT
DATE(*DMY-)	DATE
TIME(*HMS.)	TIME
TIMESTAMP(n)	TIMESTAMP(n)

## Neodpovídající typy

```
dcl-S bin_1      SQLTYPE(BINARY: 50);      // CHAR(50) CCSID(*HEX);  
dcl-S varbin_1  SQLTYPE(VARBINARY: 100);  // VARCHAR(100) CCSID(*HEX);
```

*a jiné, například*

```
dcl-S FIELD_1  SQLTYPE (CLOB: 1000 );  // DCL-DS FIELD1;  
                                           //      FIELD1_LEN  UNS(10);  
                                           //      FIELD1_DATA CHAR(1000);  
                                           // END-DS FIELD1;
```

```
Exec SQL set :FIELD_1 = 'ABCD';  // příkaz SQL v RPG  
FIELD1_DATA = 'ABCD';           // příkaz v RPG
```

# Trigger – zvýšení ceny nejvýše o 10% při aktualizaci záznamu

## Trigger SQL

Skript TG\_CENA\_U

```
-- Trigger BEFORE UPDATE pro tabulku CENIKP
CREATE OR REPLACE TRIGGER TG_CENA_U BEFORE UPDATE ON CENIKP
  REFERENCING OLD ROW AS old_row
                NEW ROW AS new_row
  FOR EACH ROW                                -- spustí se u každého řádku
  MODE DB2ROW                                  -- předepsáno pro BEFORE
  WHEN (new_row.CENAJ > 100 and
        new_row.CENAJ > old_row.CENAJ * 1.10 )
  BEGIN
    SET new_row.CENAJ = old_row.CENAJ * 1.10;
    SIGNAL SQLSTATE VALUE '01H01' SET MESSAGE_TEXT =
      'Navýšení množství přesahuje 10 %. Ponechá se navýšení 10 %';
  END ;

-- DROP TRIGGER VZUPKA.TG_CENA_U;
```

1. Umístit skript do zdrojového členu TG\_CENA\_U a spustit CL příkazem  
**RUNSQLSTM SRCFILE(QSQLSRC) SRCMBR(TG\_CENA\_U)**
2. V aplikaci *IBM i Access Client Solutions* vložit skript do okna **Run SQL scripts** a spustit:

Run SQL Scripts - LAB400E1.labits.local(Db2)

Soubor Upravit Vyhledat Zobrazit Připojení Spustit Explain Monitorovat Editor Nástroje Nápověda

\*Untitled 1 x

```

1 -- Trigger BEFORE UPDATE pro tabulku CENIKP
2 CREATE OR REPLACE TRIGGER TG_CENA_U BEFORE UPDATE ON CENIKP
3   REFERENCING OLD ROW AS old_row
4     NEW ROW AS new_row
5   FOR EACH ROW -- spustí se u každého řádku
6   MODE DB2ROW -- předepsáno pro BEFORE
7   WHEN (new_row.CENAJ > 100 and
8         new_row.CENAJ > old_row.CENAJ * 1.10 )
9   BEGIN
10    SET new_row.CENAJ = old_row.CENAJ * 1.10;
11    SIGNAL SQLSTATE VALUE '01H01' SET MESSAGE_TEXT =
12      'Navýšení množství přesahuje 10 %. Ponechá se navýšení 10 %';
13  END ;
14

```

Připojen k relační databázi Db2 na LAB400E1.labits.local jako VZUPKA - 007205/QUSER/QZDASSINIT pomocí konfigurace JDBC 'Výchozí'.

[ 28. 03. 2025 17:41:01 ] Spustit vše...

```

CREATE OR REPLACE TRIGGER TG_CENA_U BEFORE UPDATE ON CENIKP REFERENCING OLD ROW AS old_row NEW ROW AS new_row FOR EACH
ROW MODE DB2ROW WHEN (new_row.CENAJ > 100 and new_row.CENAJ > old_row.CENAJ * 1.10 ) BEGIN SET new_row.CENAJ = old_row.CENAJ *
1.10; SIGNAL SQLSTATE VALUE '01H01' SET MESSAGE_TEXT = 'Navýšení množství přesahuje 10 %. Ponechá se navýšení 10 %'; END

```

✓ Příkaz byl úspěšně proveden (1 137 ms = 1,137 s)

Zprávy Prostředí

Job: 007205/QUSER/QZDASSINIT JDBC Configuration: Výchozí User: VZUPKA Řádky: 14 Ř: 13 Sl: 9

## **Trigger externí (RPG)**

Program TG\_CENA\_U

```
**free
  // vstupní parametry z databázového systému
Dcl-PI *n;
  Buf          LikeDS(TrgBuffer);
  Len          Uns(10);
End-PI;

  // Trigger buffer (1. parametr) - obsahuje data nového záznamu
Dcl-DS TrgBuffer          Len(1000);
  // Statická oblast
  FileName                Char(10);
  LibraryName              Char(10);
  MemberName               Char(10);
  TrgEvent                  Char(1);
  TrgTime                   Char(1);
  CmtLckLvl                 Char(1);
  // Dynamická oblast
  OldRecOffset            Uns(10)  pos(49);
  OldRecLen                  Uns(10);
  NewRecOffset            Uns(10)  pos(65);
  NewRecLen                   Uns(10);
End-DS;
```

```

//    Starý záznam (before update)
Dcl-DS OldRecord   ExtName('CENIKP') Based(OldRecPtr) Qualified End-DS;
//    Nový záznam (before insertion)
Dcl-DS NewRecord   ExtName('CENIKP') Based(NewRecPtr) Qualified End-DS;

Dcl-S OldRecPtr          Pointer; // Prázdný ukazatel na Starý záznam
Dcl-S NewRecPtr          Pointer; // Prázdný ukazatel na Nový záznam

OldRecPtr = %Addr(Buf) + Buf.OldRecOffset; // Adresa Starého záznamu
NewRecPtr = %Addr(Buf) + Buf.NewRecOffset; // Adresa Nového záznamu

If (NewRecord.CENAJ > 100 and NewRecord.CENAJ > OldRecord.CENAJ * 1.10) ;
    NewRecord.CENAJ = OldRecord.CENAJ * 1.10 ;
EndIf;

Return;

```

### ***Trigger externí – registrace***

```

ADDPFTRG FILE(CENIKP) TRGTIME(*BEFORE) TRGEVENT(*UPDATE)
    PGM(TG_CENA_U) RPLTRG(*YES) TRG(TG_CENA_U) TRGLIB(*FILE)
    ALWREPCHG(*YES)

```



# Uložená procedura – hodnota všeho materiálu ve skladě

## Procedura v SQL

Skript PR\_CEN (v jazyku PL/SQL)

```
CREATE OR REPLACE PROCEDURE CENA_MAT
( in CISLO_ZAVODU  CHAR(2),          -- číslo závodu
  in CISLO_SKLADU  CHAR(2),          -- číslo skladu
  out CASTKA_CELKEM decimal (11, 2), -- součet částek cena krát množství
  out POCET_MATER  integer )         -- počet druhů materiálu na skladě
language SQL reads sql data
begin
  declare SUMA  decimal (11, 2) default 0.00;
  declare POCET integer default 0;
  for
    select CENAJ, MNOZ
      from STAVYP as S
      join CENIKP as C on C.MATER = S.MATER
     where SKLAD = CISLO_SKLADU and ZAVOD = CISLO_ZAVODU
  do
    set SUMA = SUMA + (CENAJ * MNOZ) ;    -- součet celkem
  end for;
  select count(distinct MATER) into POCET -- počet druhů materiálu
    from STAVYP;
  set CASTKA_CELKEM = SUMA ;
  set POCET_MATER = POCET ;
end
```

Ralizace skriptu uloženého ve zdrojovém členu:

```
RUNSQLSTM SRCFILE(QSQLSRC) SRCMBR(PR_CEN)
```

## ***Volání procedury v programu – CALL***

Program PR\_CEN\_C

```
Dcl-F QPRINT    printer(120);
```

```
// vstupní parametry uložené procedury
```

```
Dcl-PI *n;
```

```
    zavod                Char(2);
```

```
    sklad                Char(2);
```

```
End-PI;
```

```
// výstupní parametry uložené procedury
```

```
Dcl-S suma                Packed(11:2);
```

```
Dcl-S pocet                Int(10);
```

```
Exec SQL  CALL  CENA_MAT ( :zavod, :sklad, :suma, :pocet );
```

```
Except DETAIL;    // tisk výsledků
```

```
*inlr = *on;
```

OQPRINT	E	DETAIL	1		
O					'Celková cena materiálu '
OQPRINT	E	DETAIL	1		
O					'Závod: '
O		<b>zavod</b>		+1	
OQPRINT	E	DETAIL	1		
O					'Sklad: '
O		<b>sklad</b>		+1	
OQPRINT	E	DETAIL	1		
O					'Celkem: '
O		<b>suma</b>	P	+1	
OQPRINT	E	DETAIL	1		
O					'Počet materiálů: '
O		<b>pocet</b>	P	+1	

Celková cena materiálu

Závod: 01

Sklad: 01

Celkem: 58140.52

Počet materiálů: 12

## ***Procedura externí – realizace programem***

Program PR\_CEN\_R

```
**free
  // Parametry uložené procedury
Dcl-PI *n;
  Zavod          Char(2);      // in
  Sklad          Char(2);      // in
  Castka         Packed(11:2); // out
  Pocet_mater    Int(10);      // out
End-PI;
  // Pracovní proměnné
Dcl-S Celkem     Packed(11:2);
Dcl-S Pocet      Int(10:0);

Exec SQL declare CUR cursor for
  select sum(CENAJ * MNOZ) as CELKEM
    from STAVYP as S
    join CENIKP as C on C.MATER = S.MATER
   where ZAVOD = :Zavod and SKLAD = :Sklad ;

Exec SQL open CUR;
Exec SQL fetch from CUR into :Celkem;
DoW sqlstate = '00000';
  Castka = Castka + Celkem; // out
  Exec SQL fetch from CUR into :Celkem;
EndDo;
Exec SQL close CUR;
```

```
Exec SQL select count(distinct MATER) into :Pocet_mater -- out
        from STAVYP;  
Return;
```

## ***Procedura externí – definice***

Skript PR\_CEN\_E

```
CREATE OR REPLACE PROCEDURE CENA_MAT  
    ( in CISLO_ZAVODU    CHAR(2),          -- číslo závodu  
      in CISLO_SKLADU    CHAR(2),          -- číslo skladu  
      out CASTKA_CELKEM decimal (11, 2),   -- součet částek cena krát množství  
      out POCET_MATER    integer )         -- počet druhů materiálu na skladěRPGLE                                -- jazyk ILE RPGMAIN                            -- procedura je hlavní programexternal name PR_CEN_R                       -- realizuje uloženou proceduru
```

# User defined function (UDF) – vrací jednotkovou cenu materiálu

## ***Funkce v SQL***

Skript FU\_CEN

```
CREATE OR REPLACE FUNCTION VRAT_CENU_MATERIALU (MATERIAL CHAR(5))  
  RETURNS DECIMAL (10, 2)  
  LANGUAGE SQL  
  BEGIN  
    RETURN (select CENAJ from CENIKP  
            where MATER = MATERIAL) ;  
  END;
```

## ***Volání funkce v programu***

Program FU\_CEN\_C

```
Dcl-DS cenik_ds extname('CENIKP') qualified End-DS;
```

```
Dcl-S  cena_materialu Like(cenik_ds.CENAJ) Inz;
```

```
Dcl-PI *N;
```

```
    material Like(cenik_ds.MATER); // vstupní parametr
```

```
End-PI;
```

```
// volám SQL funkci
```

```
Exec SQL set :cena_materialu = VRAT_CENU_MATERIALU ( :material );
```

```
Dsply cena_materialu;
```

```
Return;
```

## ***Funkce externí – realizace podprocedurou***

Program FU\_CEN\_R

```
**free
ctl-opt nomain;
dcl-PROC FU_CEN_R  export;
    dcl-F CENIKP keyed; // příp. STATIC – nechá soubor otevřený
    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(e) material CENIKP cenik_ds; // čte do datové struktury
    if not %found();
        cenik_ds.CENAJ = -1;
    endif;
    return cenik_ds.CENAJ;
end-PROC FU_CEN_R;
```

- 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.

### *Servisní program*

```
CRTSQLRPGI OBJ(FU_CEN_R) SRCFILE(QRPGLESRC) SRCMBR(FU_CEN_R) COMMIT(*NONE)  
OBJTYPE(*SRVPGM) DBGVIEW(*SOURCE)
```

### *Modul volajícího programu*

```
CRTSQLRPGI OBJ(FU_CEN_C) SRCFILE(QRPGLESRC) SRCMBR(FU_CEN_C) COMMIT(*NONE)  
OBJTYPE(*MODULE) DBGVIEW(*SOURCE)
```

### *Vytvoření volajícího programu připojením servisního programu*

```
CRTPGM PGM(FU_CEN_C) MODULE(*PGM) BNDSRVPGM((FU_CEN_R))
```

### *Vyvolání volajícího programu*

```
CALL PGM(FU_CEN_C) PARM('00001')
```

## **Funkce externí – definice**

Skript FU\_CEN\_E

```
CREATE OR REPLACE FUNCTION VRAT_CENU_MATERIALU (MATERIAL CHAR(5))  
  RETURNS DECIMAL (10, 2)  
  LANGUAGE RPGLE  
  PARAMETER STYLE GENERAL  
  NOT DETERMINISTIC NO SQL  
  PROGRAM TYPE SUB -- funkci realizuje podprocedura  
  NOT FENCED -- není vázána na stejnou úlohu  
  NO FINAL CALL -- není první a poslední volání  
  EXTERNAL NAME FU_CEN_R(FU_CEN_R) -- servisní program a podprocedura
```



# User defined table function (UDTF) – vrací tabulku čísel a dat objednávek

Přijímá dva parametry a vrací tabulku obsahující čísla a data objednávek v rozmezí parametrů.

## *Volání funkce v programu OBJDAT\_P*

```
Dcl-PI *n;    // vstupní parametry
    Dcl-S Datum1 Date;
    Dcl-S Datum2 Date;
End-PI;

Dcl-S COBJ    Char(6); // host variables pro fetch
Dcl-S DTOBJ   Date;

Exec SQL declare CUR cursor for
    select * from TABLE ( OBJDAT_F(:Datum1 , :Datum2 ) ) as DAT_F;

Exec SQL open CUR ;

    // vypíšu všechny záznamy vybrané tou funkcí
Exec SQL fetch CUR into :COBJ, :DTOBJ ;
DoW sqlstate = '00000';
    SND-MSG COBJ + ' ' + %char(DTOBJ); // výpis do joblogu
    Exec SQL fetch CUR into :COBJ, :DTOBJ ;
EndDo;

Exec SQL close CUR;
Return;
```

## ***User defined table function (UDTF) – SQL skript***

PL/SQL skript OBJDAT\_F

Skript generuje uživatelskou funkci, která přijímá dva parametry a *vrací tabulku* se dvěma sloupci, COBJ (číslo objednávky) a DTOBJ (datum objednávky), ze záznamů vybraných mezi dvěma daty.

```
CREATE OR REPLACE FUNCTION OBJDAT_F (DATUM1 DATE, DATUM2 DATE)
  RETURNS TABLE
  ( COBJ CHAR(6) CCSID 870,
    DTOBJ DATE )
LANGUAGE SQL
BEGIN
  RETURN select COBJ, DTOBJ from OBJHLA_T
         where DTOBJ between DATUM1 and DATUM2
         order by COBJ;
END;
```

Skript umístíme do zdrojového členu, např. OBJDAT\_F a spustíme CL příkazem

```
RUNSQLSTM SRCFILE(QSQLSRC) SRCMBR(OBJDAT_F)
```

Výsledkem skriptu je **servisní program** OBJDA T\_F v zadané knihovně. Ten je nutné spojit s modulem OBJDAT\_P, aby vznikl **program OBJDAT\_P**:

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

## ***User defined table function (UDTF) – RPG podprocedura***

1. Vytvoříme skript pro externí uživatelskou funkci, která přijímá dva parametry a vrací tabulku předepsaného tvaru.

### ***PL/SQL script pro externí RPG funkci OBJDAT\_F***

```
CREATE OR REPLACE 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
  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
```

2. Vytvoříme zdrojový **modul** OBJDAT\_F s procedurou (funkcí) OBJDAT\_F. Funkce OBJDAT\_F přijímá dva parametry určující rozmezí datumů. Vybere objednávky z tabulky OBJHLA\_T v daném rozmezí a **vytvoří tabulku** s čísly objednávek a s daty. Seznam bude uspořádán podle čísla objednávky vzestupně.

3. Vytvoříme *spojovací text* (binder source) v souboru QSRVSRC

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

4. Vytvoříme *servisní program* OBJDAT\_F s procedurou (funkcí) OBJDAT\_F. Zdrojový **modul OBJDAT\_F** se kompiluje příkazem

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

5. Vytvoříme *program* OBJDAT\_P s připojeným servisním programem OBJDAT\_F

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

## **Modul OBJDAT\_F pro servisní program OBJDAT\_F**

```
Ctl-Opt nomain;
Dcl-Proc OBJDAT_F      Export;
  Dcl-DS OBJHLA_T Ext  Template ; // host variables
End-DS;

// Datová struktura dat přetrvávajících mezi voláními funkce
Dcl-DS ScratchDS      Template;
  ScrLen              Int(10:0) Inz(%Size(ScratchDS));
  Cntr                Packed(6:0)  Inz(0);
End-DS;

// Procedure interface
Dcl-PI *N;                                     // parametry
  Datum1              Date;                    // in
  Datum2              Date;                    // in
  COBJ_PAR            Like(COBJ);              // out
  DTOBJ_PAR           Like(DTOBJ);             // out
  Datum1_ind          Int(5:0);                // in
  Datum2_ind          Int(5:0);                // in
  COBJ_PAR_ind        Int(5:0);                // out
  DTOBJ_PAR_ind       Int(5:0);                // out
  SQLSTATE_PAR        Char(5);                 // out
  Func_name           VarChar(517);            // in
  Spec_name           VarChar(128);            // in
  Message_text        VarChar(1000);           // out
  Scratchpad          LikeDS(ScratchDS);       // inout
  CallType             Int(10:0);              // in
End-PI;
```

```

If Calltype = -1;          // OPEN
    Exec SQL declare CUR cursor for
        select COBJ, DTOBJ
        from OBJHLA_T
        where DTOBJ between :Datum1 and :Datum2
        order by COBJ;
    Exec SQL open CUR;
ElseIf Calltype = 0;      // FETCH
    // Aby kurzor přetrval jednotlivá volání, je nutné při kompilaci
    // zadat parametr CLOSQLCSR(*ENDACTGRP)
    Exec SQL fetch next from CUR into
        :COBJ_PAR :COBJ_PAR_ind,
        :DTOBJ_PAR :DTOBJ_PAR_ind ;
ElseIf Calltype = 1;      // CLOSE
    Exec SQL close CUR;
EndIf;

End-Proc OBJDAT_F;

```

Poznámka 1: Čítač *Scratchpad.Cntr* zde není použit, mohl by sloužit např. k omezení nebo tisku počtu vracených řádků.

Poznámka 2: Velmi důležitý je parametr CLOSQLCSR s hodnotou \*ENDACTGRP zadaný při kompilaci modulu; zachovává otevřený kurzor mezi jednotlivými voláními procedury (Open, Fetch, Close). Kurzor se zavře až při ukončení aktivační skupiny. Předvolená hodnota \*ENDMOD by způsobila, že kurzor by se zavřel po každém volání procedury (nejen při volání Close).

Poznámka 3: Jednotlivá volání jsou realizována jako vlákna (threads). V nich nejsou dostupné hodnoty proměnných pro výpis paměti (dump). Ladění je ale možné pomocí programu STRDBG.