MarkOneTools

Think Then Talk

Ultimo aggiornamento: 23-Lug-2023

6 DICEMBRE 2019

Tutto (o quasi) su date e ore

Il tipo dati **Date** è stato introdotto nella V2R3 (ovvero all'incirca nel 1994), ma ancora oggi purtroppo sembra un tipo dati sconosciuto e "pericoloso" da usare.

Un po' per tradizione e un po' per pigrizia le date nelle tabelle del DB2 non sono altro che numeri scritti nei formati più svariati gestiti interamente dalla logica applicativa nei programmi.
Ma essendo per l'appunto nient'altro che campi numerici il DB2 ignora del tutto che il contenuto del campo sia una data. Il campo è un *surrogato di una data*. Quindi tutta la fatica di gestire correttamente le date è demandata alle capacità del programmatore per nulla sfruttando le potenzialità del DB2.

Esistono numerosi metodi diversi per operare con le date: codici operativi e funzioni RPG, funzioni SQL, API, comandi di sistemi operativo: almeno 98 metodi!

Lavorare con il tipo dati Date, Time o Timestamp nelle tabelle del DB2 o nei programmi RPG richiede di conoscere bene le caratteristiche di questo tipo di dati. In questo articolo cercherò di esporre in maniera organica gli argomenti che riguardano la definizione e la manipolazione di date e ore, con la speranza che un maggior numero di sviluppatori le utilizzino con disinvoltura.
Allegati all'articolo troverete 29 sorgenti con numerosi esempi. Scaricare tutti gli esempi da qui.

Scarica l'articolo in formato PDF.

Sommario

Definizione campi e variabili ▲

Tabelle ▲

Esistono 3 tipi dati per rappresentare i valori di data/ora: `date`, `time` e `timestamp`.

La rappresentazione interna di un valore data (`date`) nel DB2 è una stringa di 4 bytes che contiene un intero (il cosiddetto "scaliger number").
La lunghezza di un campo DATE (come descritto in SQLDA) può essere 6, 8 o 10 a secondo del formato.

La rappresentazione interna di un valore ora (`time`) nel DB2 è una stringa di 3 bytes. Ogni byte contiene un numero packed di 2 digits. Il primo byte rappresenta l'ora, il secondo i minuti e il terzo i secondi.
La lunghezza di un campo TIME (come descritto in SQLDA) è di 8 bytes.

La rappresentazione interna di un valore `timestamp` è una stringa compresa tra 7 e 13 bytes. I primi 4 bytes rappresentano la data, i successivi 3 l'ora e gli ultimi (da 0 a 6) la parte frazionale dei secondi.

Se nella definizione del campo si aggiunge la keyword `not null default` senza specificare nessun valore verrà assunto la data/ora/timestamp correnti al momento dell'operazione di insert.

DDS▲

Definendo una tabella con il vecchio metodo tradizionale delle DDS alla posizione 35
– il tipo data è `L`
– il tipo ora è `T`
– il tipo timestamp è `Z`
La lunghezza è determinata dal DB2 e non deve essere specificata nelle posizioni 30-34.

```
  * ESEMPIO TABELLA CON DATA/ORA/TIMESTAMP
A          R REK01
 *
A            FLDDATA         L
A            FLDORA          T
A            FLDTIMS         Z
```

La definizione dei campi tabella sarà:

```
            Tipo        Lungh  immiss Posizione        Uso      Intestazione
Campo       Tipo       campo   buffer  buffer          Campo    Colonna
FLDDATA     DATE           10      10         1        Entrambi FLDDATA
  Formato data . . . . . . . . . . . . . . . . : *ISO
  Identificativo serie caratteri  codificati:    280
FLDORA      ORA             8       8        11        Entrambi FLDORA
  Formato ora . . . . . . . . . . . . . . . . :  *ISO
  Identificativo serie caratteri  codificati:    280
FLDTIMS     TIMESTAMP   26  6      26        19        Entrambi FLDTIMS
  Identificativo serie caratteri  codificati:    280
```

SQL▲

Utilizzando invece il linguaggio SQL:
– il tipo data è `date`
– il tipo ora è `time`
– il tipo timestamp è `timestamp(n)`. La precisione `n` della parte frazionale dei secondi può essere da 0 (secondi) a 12 (picosecondi). Il valore di default è 6 (microsecondi).

```
/* ESEMPIO TABELLA CON DATA/ORA/TIMESTAMP */

create or replace table TABDDS (
  FLDDATA date not null,
  FLDORA  time not null,
  FLDTIMS timestamp not null)
  rcdfmt REK01;
```

Anno lungo 2 o 4? ▲

Senza dubbio in qualsiasi contesto (database, printer file, display file...) bisogna utilizzare campi data il cui formato preveda l'anno di 4 cifre.
Usare l'anno a due cifre limita il range di date dal 1/1/1940 al 31/12/2039! Ormai non manca tanto al 2039.
Usare l'anno a 3 cifre ovvero i formati *CYMD, *CMDY, *CDMY ampliano il range di date dal 1/1/1900 al 31/12/2899. Ma non vedo il senso di fare la fatica di ampliare l'anno da 2 a solo 3 cifre, quando è molto più comodo e universalmente condiviso che le date debbano avere l'anno di 4 caratteri.

CLP ▲

I tipi data/ora/timestamp per le variabili in un programma CL non sono disponibili.

RPG ▲

Nelle specifiche di definizione (D) le variabili di tipo data/ora/timestamp si definiscono:
– date (D)
– time (T)
– timestamp (Z)

Non è necessario specificare la lunghezza del campo.

Il timestamp ha una lunghezza di default di 26 (con 6 byte per le frazioni di secondo). La lunghezza massima può essere 32.
Da IBM i 7.2 è possibile specificare quante cifre si desidera gestire come frazione di secondo: da 0 a 12. Il default è 6 (ovvero una lunghezza totale del campo di 26).

Esempi:

```
dcl-s ts   timestamp;     // YYYY-MM-DD-hh.mm.ss.ffffff (default)
```

```
dcl-s ts   timestamp(6);  // YYYY-MM-DD-hh.mm.ss.ffffff (default)
dcl-s ts0  timestamp(0);  // YYYY-MM-DD-hh.mm.ss
dcl-s ts1  timestamp(1);  // YYYY-MM-DD-hh.mm.ss.f
dcl-s ts3  timestamp(3);  // YYYY-MM-DD-hh.mm.ss.fff
dcl-s ts12 timestamp(12); // YYYY-MM-DD-hh.mm.ss.ffffffffffff
```

N.B. la keyword `inz()` o la funzione `%timestamp` valorizzano solo le prime 3 cifre della porzione di frazione di secondo.

Il valore di default (nonché il valore minimo *LOVAL) è 1 gennaio 0001 ora 00.00.00.
Il valore massimo (*HIVAL) è 9999-12-31-24.00.00.

E' possibile inizializzare una variabile di tipo data alla data di sistema con la keyword `inz(*sys)` oppure alla data del job con `inz(*job)`.
E' possibile inizializzare una variabile di tipo ora all'ora di sistema con la keyword `inz(*sys)`.

Lunghezza campi data/ora convertiti in campi numerici

La lunghezza di campi ora è sempre 6 tranne per il formato *usa che non è convertibile in un numero.
La lunghezza di campi data dipende dal formato della data:
8 per *iso, *eur, *usa e *jis
6 per *ymd, *dmy, and *mdy
5 per *jul
La lunghezza di un campo timestamp dipende dalla porzione frazionale dei secondi (da 14 a 26).

Programma di esempio FLDLEN.RPGLE; output:

```
DSPLY  * DATE *
DSPLY  USA_date: 8
DSPLY  ISO_date: 8
DSPLY  EUR_date: 8
DSPLY  JIS_date: 8
DSPLY  YMD_date: 6
DSPLY  DMY_date: 6
DSPLY  MDY_date: 6
DSPLY  JUL_date: 5
DSPLY  DFT_date: 8
DSPLY  * ORE *
DSPLY  USA_time: 8
DSPLY  ISO_time: 6
DSPLY  EUR_time: 6
DSPLY  JIS_time: 6
DSPLY  HMS_time: 6
DSPLY  DFT_time: 6
DSPLY  * TIMESTAMP *
DSPLY  ts: 20
DSPLY  ts6: 20
DSPLY  ts0: 14
DSPLY  ts1: 15
DSPLY  ts3: 17
DSPLY  ts12: 26
```

Lunghezza campi data/ora convertiti in campi alfanumerici

La lunghezza di campi ora è sempre 8.
La lunghezza di un campo data dipende dal formato della data:
10 per *iso, *eur, *usa e *jis
8 per *ymd, *dmy, and *mdy
6 per *jul
La lunghezza di un campo timestamp dipende dalla porzione frazionale dei secondi (da 19 a 32).

Programma di esempio FLDLENA.RPGLE; output:

```
DSPLY  * DATE *
DSPLY  USA_date: 10
DSPLY  ISO_date: 10
DSPLY  EUR_date: 10
DSPLY  JIS_date: 10
DSPLY  YMD_date: 8
DSPLY  DMY_date: 8
DSPLY  MDY_date: 8
DSPLY  JUL_date: 6
DSPLY  DFT_date: 10
DSPLY  * ORE *
DSPLY  USA_time: 8
DSPLY  ISO_time: 8
DSPLY  EUR_time: 8
DSPLY  JIS_time: 8
DSPLY  HMS_time: 8
DSPLY  DFT_time: 8
DSPLY  * TIMESTAMP *
DSPLY  ts: 26
DSPLY  ts6: 26
DSPLY  ts0: 19
DSPLY  ts1: 21
DSPLY  ts3: 23
DSPLY  ts12: 32
```

Display e printer files▲

Il tipo dati (pos. 35) per data è L, per ora T, per timestamp Z. La lunghezza del campo (pos. 30-34) deve essere lasciata vuota. Per il tipo dati L il formato definito in DATFMT determina la lunghezza del campo (p.es. con *ISO è 10). Con *JOB la lunghezza è sempre 10 anche se viene mostrata una data di 8 caratteri.
Per il tipo dati T la lunghezza è determinata dalla keyword TIMFMT. Se *ISO la lunghezza è 8.
Per il tipo dati Z la lunghezza è sempre 26.

Per **mostrare un campo vuoto** a video quando il valore è impostato al default 0001-01-01 si può utilizzare la keyword MAPVAL che serve per mappare una data, ora o timestamp su un valore diverso durante le operazioni di I/O. Per esempio:

```
A            WSDATA          L   B   8 47DATFMT(*EUR)
A                                      MAPVAL(('01.01.0001' *BLANK))
```

La keyword MAPVAL è valida solo per i tipi dati data, ora o timestamp.
Si possono specificare anche più valori da mappare p.es. `MAPVAL(('0001-01-01' *BLANK) ('0001-01-02' *CUR))`.

Scarica il programma di esempio MAPVAL e il file video MAPVALV.

La keyword DATE visualizza la data corrente (solo per un campo di output). Il default è `DATE(*JOB)` ovvero la data del lavoro. Si può anche specificare `DATE(*SYS)` per mostrare la data di sistema. Il secondo parametro consente di specificare la lunghezza dell'anno: per default la lunghezza è 2 (`*Y`). Per avere l'anno di 4 caratteri bisogna specificare `*YY`. P.es. `DATE(*JOB *YY)`.
La keyword DATFMT determina il formato di visualizzazione di un campo data. Il default è *ISO. La keyword DATSEP definisce il separatore, si può specificare *JOB oppure '/', '-', '.', ',' o ' '.

La keyword TIME visualizza l'ora corrente di sistema (solo per un campo di output).
Se non specificato il formato si assume che sia `EDTWRD('0 : : ')`
Un altro esempio di maschera di editazione è `EDTWRD(' h& m& s')` per visualizzare " 9h 40m 10s".

La keyword `TIMFMT` specifica il formato di un campo di tipo time. Il default è *ISO (hh.mm.ss). Se si specifica `TIMFMT` non si può specificare anche `TIMSEP`.

Per i campi data si può specificare il **codice di editazione** (`EDTCDE`) `W` (editazione data 4 digits) o `Y`. Entrambi i codici editazione inseriscono il carattere separatore /. Per esempio:

edit code	6 caratteri	8 caratteri
W	nnnn/nn	nnnn/nn/nn
Y	nn/nn/nn	nn/nn/nnnn

Alcuni esempi di **maschere di editazione** (`EDTWRD`)per campi data:
`' / / '` p.es. 010388 -> 1/03/88
`'0 / / '` p.es. 010388 -> 01/03/88

Scarica il programma di esempio ESDATE e il file video ESDATEV.

Java ▲

I tipi dati corrispondenti per data, ora e timestamp in Java sono `java.sql.Date`, `java.sql.Time` e `java.sql.Timestamp` rispettivamente.

Riepilogo definizioni ▲

	Description	RPG IV Type[1]	RPG IV Length	RPG IV Keyword	DDS Type	DDS Keyword	SQL Type	SQL Length	Value	Format
DATE/TIME[2]	Date	D *date*	6 8 10	datfmt(*JUL)[3] datfmt(*YMD) datfmt(*DMY) datfmt(*MDY) datfmt(*ISO)[4] datfmt(*EUR) datfmt(*USA) datfmt(*JIS)	L	DATFMT DATSEP	DATE[5]		40/001 to 39/365, def. 40/001 40/01/01 to 39/12/31 def. 40/01/01 01/01/40 to 31/12/39 def. 01/01/40 01/01/40 to 12/31/39 def. 01/01/40 01/01/0001 to 31/12/9999 def. 01/01/0001 01.01.0001 to 31.12.9999 def. 01.01.0001 01/01/0001 to 12/31/9999 def. 01/01/0001 0001-01-01 to 9999-12-31 def. 0001-01-01	2 digits yy/ddd 2 digits yy/mm/dd 2 digits dd/mm/yy 2 digits mm/dd/yy 4 digits yyyy-mm-dd 4 digits dd.mm.yyyy 4 digits mm/dd/yyyy 4 digits yyyy-mm-dd
	Time	T *time*	8	timfmt(*HMS)[3] timfmt(*ISO) timfmt(*USA) timfmt(*EUR) timfmt(*JIS)	T	TIMFMT TIMSEP	TIME[7]		00:00:00 to 24:00:00, def. 00:00:00 00.00.00 to 24.00.00, def. 00:00:00 00:00 AM to 12:00 AM, def. 00:00 AM 00.00.00 to 24.00.00, def. 00:00:00 00.00.00 to 24.00.00, def. 00:00:00	hh:mm:ss hh.mm.ss hh:mm AM or hh:mm PM hh.mm.ss hh:mm:ss
	Timestamp	Z *timestamp*	26-32[8]	*ISO	Z		TIMESTAMP[9]		0001-01-01-00.00.00.000000 to 9999-12-31-24.00.00.000000 def. 0001-01-01-00.00.00.000000	YYYY-MM-DD- hh.mm.ss.mmmmmm[10]

IBM i 7.3 MK

1 In *brown italic font* the free-form syntax
2 A 0 after format indicates NO separator (ex. *ISO0)
3 Valid separator for 2 digits format: / - . , &
4 The default format for date, time e timestamp fields is *ISO
5 The internal representation is a string of 4 bytes that cointains an integer (scaliger number)
6 Valid separator: : . , &
7 The internal representation is a string of 3 bytes that cointains two packed decimal digits.
8 From 7.2 the lenght can be until 32
9 The internal representation is a string of 3 of between 7 and 13 bytes
10 The fractional part is facultative. The length could be from 1 to 12. The default is 6.

Un riepilogo completo di tutti i tipi dati disponibili nei vari linguaggi di programmazione (CLP, RPG, SQL...) è consultabile al link Ce n'è di tutti i tipi.

Parole riservate ▲

Parole riservate relative a data/ora: *CDMY, *CMDY, *CYMD, *DMY, *EUR, *HMS, *ISO, *JIS, *JOB, *JOBRUN, *JUL, *LONGJUL, *MDY, *SYS, *USA, *YMD.

Definizione del formato ▲

Per il tipo dati Date, Time e Timestamp riveste un'importanza fondamentale la **definizione del formato**. Dal formato dipende **come viene "esposto"** il dato e non come viene memorizzato internamente dal DB2. Dal formato però **dipendono** anche le caratteristiche del contenuto ovvero i valori di **default** e i **limiti** di valore minimo e massimo.
Capire bene questo concetto è fondamentale per manipolare correttamente date e ore e spesso ignorare questo argomento spesso causa non pochi grattacapi ai programmatori.

Nei programmi RPG la definizione del formato dipende dalle specifiche di controllo (H), di definizione file (F), di definizione delle variabili (D).

I formati *iso0, *ymd0, ecc. che **non** prevedono l'uso dei separatori sono disponibili da V3R7.

Combinando i formati validi e i separatori esistono 57 combinazioni di formati data e 14 per le ore.

Nome	Descrizione	Formato	Separatore[1]	Lungh.	Esempio	Minimo[2] (*LOVAL)	Massimo (*HIVAL)	Default	Range anni
DATA									
*MDY	USA: Month/Day/Year	mm/dd/yy	/-.&	8	01/15/94	01/01/40	12/31/39	01/01/40	1940-2039
*DMY	European: Day/Month/Year	dd/mm/yy	/-.&	8	15/01/94	01/01/40	31/12/39	01/01/40	1940-2039
*YMD	Year/Month/Day	yy/mm/dd	/-.&	8	94/01/15	40/01/01	39/12/31	40/01/01	1940-2039
*JUL	Julian	yy/ddd	/-.&	6	94/015	40/001	39/365	40/001	1940-2039
*ISO	International Standards Organization	yyyy-mm-dd	-	10	1994-01-15	0001-01-01	9999-12-31	0001-01-01	0001-9999
*USA	IBM USA std.	mm/dd/yyyy	/	10	01/15/1994	01/01/0001	12/31/9999	01/01/0001	0001-9999
*EUR	IBM European std.	dd.mm.yyyy	.	10	15.01.1994	01.01.0001	31.12.9999	01.01.0001	0001-9999
*JIS[5]	Japanese Industrial Standard	yyyy-mm-dd	-	10	1994-01-15	0001-01-01	9999-12-31	0001-01-01	0001-9999
*CYMD[4]	Century[3]/Year/Month/Day	cyy/mm/dd	/-.&	9	094/01/15	000/01/01[6]	999/12/31	000/01/01	1900-2899
*CMDY	Century/Month/Day/Year	cmm/dd/yy	/-.&	9	001/15/94	001/01/00	912/31/99	001/01/00	1900-2899
*CDMY	Century/Day/Month/Year	cdd/mm/yy	/-.&	9	015/01/94	001/01/00	931/12/99	001/01/00	1900-2899
*LONGJUL	Long Julian	yyyy/ddd	/-.&	8	1994/015	0001/001	9999/365	0001/001	0001-9999
*JOBRUN[7]		*DATFMT*	*DATSEP*						
ORA									
*HMS	Ore/Minuti/Secondi	hh:mm:ss	:.,&	8	14:00:00	00:00:00	24:00:00	00:00:00	
*ISO	International Standards Organization	hh.mm.ss	.	8	14.00.00	00.00.00	24.00.00	00.00.00	
*USA	IBM USA std.	hh:mm AM or hh:mm PM	:	8	02:00 PM	00:00 AM	12:00 AM	00:00 AM	
*EUR	IBM European std.	hh.mm.ss	.	8	14.00.00	00.00.00	24.00.00	00.00.00	
*JIS	Japanese Industrial Standard	hh:mm:ss	:	8	14:00:00	00:00:00	24:00:00	00:00:00	
*JOBRUN[7]			*TIMSEP*						
TIMESTAMP									
		yyyy-mm-dd-hh.mm.ss.mmmmmm		26-32	1994-01-15-14.00.00.000000	0001-01-01-00.00.00	9999-12-31-24.00.00	0001-01-01-00.00.00	

IBM i 7.3

[1] Separatore & indica che verrà inserito un blank.
Se il carattere separatore è diverso da quello di default del formato viene specificato sulla destra del formato, p.es. *YMD-
Se a destra del formato si aggiunge 0 significa che il separatore verrà soppresso, p.es. *ISO0
[2] Per le ore il valore minimo e massimo hanno il medesimo significato, ovvero la mezzanotte
[3] Di fatto il formato *JIS è identico a *ISO
[4] I formati che prevedono il secolo e il formato *LONGJUL non sono utilizzabili per memorizzare campi di tipo data in una tabella. Sono disponibili solo per consentire a chi già memorizzava le date in campi numerici con questo formato di convertirli in campi di tipo date
[5] '0' per le date fino al 1999, '1' per le date dal 2000, ..., '9' dal 2800
[6] Il range per i formati che prevedono il secolo è tra 1 gen 1900 e 31 dic 2899
[7] Può essere specificato in fattore 1 di un'operazione di MOVE, MOVEL, TEST quando un campo carattere o numerico il cui formato corrisponde agli attributi correnti del job (DATFMT, DATSEP, TIMSEP) deve essere trasferito in un campo di tipo date/time

Specifiche di controllo (H)▲

Nelle **specifiche H** le seguenti keyword condizionano l'uso di variabili data/ora:

– datedit: specifica il formato dei campi numerici con il codice editazione Y. Il separatore di default è /. I formati possono essere *DMY, *MDY, *YMD.
– datfmt: specifica il formato interno per le costanti e variabili di tipo data. In assenza di questa keyword il valore di default è *ISO.
– timfmt: specifica il formato interno per le costanti e variabili di tipo ora. In assenza di questa keyword il valore di default è *ISO.
– validate: da IBM 7.2 tramite questa keyword è possibile specificare `validate(*nodatetime)` per omettere il controllo di validità sui campi data, ora e timestamp. Specificando questa keyword le performance del programma possono migliorare. Omettendo questo controllo si potrebbero avere dei valori non corretti nei campi data, ora o timestamp. Da utilizzare con MOLTA CAUTELA.

Specifiche di definizione dei file (F)▲

Per ogni **file descritto a programma** (il cosiddetto "file interno") è possibile specificare le keyword per il formato dei campi data/ora
– datfmt: formato e separatore data
– timfmt: formato e separatore ora

Quando si definisce in un programma un **file descritto esternamente** con campi data o ora, il formato di questi campi viene **automaticamente convertito nel formato di default del**

programma.

Specifiche di definizione (D)▲

Il **formato interno di default** per le date e le ore è ***ISO**.
Il formato viene determinato secondo questa **priorità**:
1) keyword `DATE`, `DATFMT`, `TIME`, `TIMFMT` nelle specifiche D
2) keyword `DATFMT`, `TIMFMT` nella più recente direttiva `/SET`
3) keyword `DATFMT`, `TIMFMT` nella specifica H
4) formato *ISO

I valori **costanti** (definiti nelle specifiche D o utilizzati nelle specifiche di calcolo) devono avere il formato definito nella specifica H o in mancanza della specifica H devono essere scritti in formato *ISO.
N.B. Questa regola vale anche se il formato della variabile data è diverso da quello della specifica H.
Il valore costante racchiuso tra apici deve essere preceduto da `d` (data), `t` (ora), `z` (timestamp). Per esempio:

`d'xxxx-xx-xx'`

`t'xx:xx:xx'`

`z'yyyy-mm-dd-hh.mm.ss'`, opzionalmente si può indicare anche la frazione di secondo (da 1 a 12 cfre) `z'yyyy-mm-dd-hh.mm.ss.frac'`. Se frac ha meno di 6 cifre viene riempito con zeri fino a un minimo di 6 cifre.

Esempio:

```
    ctl-opt datfmt(*DMY) timfmt(*HMS);

    dcl-s DateFld date(*YMD) inz(d'29/01/19');
    dcl-s TimeFld time(*EUR) inz(t'13:00:00');
     // data con separatore -
    dcl-s BirthDate date(*MDY-);
     // LastUpd è un timestamp. UpdDate e UpdTime sono rispettivamente la porzione data e ora
del timestamp.
 //  Devono essere in formato *ISO affinché siano compatibili con il formato del timestamp
    dcl-ds;
      LastUpd timestamp;
      UpdDate date(*ISO) overlay(LastUpd);
      UpdTime time(*ISO) overlay(LastUpd:12);
    end-ds;

    dcl-s USA_date date(*usa);
    dcl-s ISO_date date(*iso);
    dcl-s EUR_date date(*eur);
    dcl-s JIS_date date(*jis);
    dcl-s YMD_date date(*ymd);
    dcl-s DMY_date date(*dmy);
    dcl-s MDY_date date(*mdy);
    dcl-s JUL_date date(*jul);
    dcl-s DFT_date date;
```

oppure per i nostalgici amanti del tradizionale formato fisso

```
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++
D USA_date        S               D   datfmt(*usa)
D ISO_date        S               D   datfmt(*iso)
D EUR_date        S               D   datfmt(*eur)
D JIS_date        S               D   datfmt(*jis)
D YMD_date        S               D   datfmt(*ymd)
D DMY_date        S               D   datfmt(*dmy)
D MDY_date        S               D   datfmt(*mdy)
D JUL_date        S               D   datfmt(*jul)
D DFT_date        S               D
```

Embedded SQL ▲

Nei programmi RPG embedded SQL i parametri `DATFMT/DATSEP/TIMFMT/TIMSEP` possono essere specificati nel comando di creazione (`CRTSQLxxx`) oppure tramite l'istruzione `SET OPTION`.
Per esempio:

```
exec sql
  set option DATFMT = '*YMD';
```

Vale la pena ricordare che l'istruzione SET OPTION deve essere **scritta all'inizio del programma**, subito dopo le specifiche di dichiarazione o comunque prima di qualsiasi altra istruzione SQL. Non è un'istruzione che viene eseguita a runtime, ma una direttiva di compilazione che viene interpretata dal precompilatore SQL.

Manipolare date e ore ▲

In un programma RPG è possibile manipolare campi data/ora tramite svariati codici operativi o BIF. Inoltre con SQL embedded è possibile sfruttare le numerose funzioni SQL a disposizione per manipolare date e ore. Se tutto ciò non fosse sufficiente esistono numerose API.

Sebbene, come ho spiegato in precedenza (cfr. par. definizione del formato), il *formato è solo un modo di rappresentare* un valore data indipendente da come il dato è gestito internamente dal DB2, bisogna porre ***attenzione a spostare valori tra variabili data con formati diversi*** in quanto si potrebbe incorrere nell'errore RNQ0114, ovvero si eccedono i valori limite minimo o massimo.

Per esempio:
Il formato *USA ha un range di valori consentito tra 01/01/0001 e 12/31/9999

```
USA_date = d'1939-09-03';
```

Spostando la data 3 settembre 1939 da un campo in formato *USA a un campo in formato *MDY si riceve un errore, perché il formato *MDY ha un range di valori consentito più ristretto rispetto al formato *USA (da 01/01/(19)40 a 12/31/(20)39)

```
MDY_date = USA_date;
```

Un'altra situazione in cui bisogna porre particolarmente attenzione al formato è nei programmi RPG embedded SQL dove si leggono tabelle del DB2 con campi data; se p.es. il record contiene nel campo data il valore di default 0001-01-01 si incorre nell'eccezione RNQ0114. Per evitare ciò occorre impostare il formato corretto della data nel comando di creazione o con l'istruzione SET OPTION (cfr. par. Embedded SQL).

```
exec sql
  set option datfmt = *ISO;
```

I campi data non possono contenere 0 perché non è una data valida. Quindi il codice operativo `CLEAR` imposta il valore minimo consentito dal formato della variabile.
Per inizializzare una variabile al suo valore minimo si può usare anche la costante figurativa `*LOVAL`, oppure per impostarla al valore massimo si usa `*HIVAL`.

I codici operativi disponibili in RPG per manipolare date e ore sono: `ADDDUR`, `SUBDUR`, `EXTRCT`, `MOVE`.

Le BIF RPG per manipolare le date e ore disponibili dalla versione V5R1 sono: `%diff`, `%subdt`, `%days`, `%months`, `%years`, `%hours`, `%minutes`, `%seconds`, `%mseconds`.

Prima di vedere in dettaglio cosa si può fare con questi codici operativi e funzioni è bene imparare a convertire altri tipi di dati in date/ore/timestamps "veri". E' bene leggere con attenzione il prossimo paragrafo "conversione" in quanto nella maggior parte dei casi si avrà a che fare con tabelle i cui campi data o ora sono dei surrogati definiti con campi numerici o alfanumerici.

Conversione ▲

Da tipo data a tipo alfanumerico ▲

La funzione **%char** converte una data, ora o timestamp in un campo alfanumerico

`%char(date|time|timestamp {: format})`

Se il primo parametro è una costante, la conversione viene eseguita a tempo di compilazione.
Il secondo parametro rappresenta il formato della data, ora o timestamp restituito. Nel risultato sono inclusi i separatori a meno che si aggiunga 0 al formato (p.es. `*iso0`).

Per esempio:

```
dcl-s DataAlfa char(10);
dcl-s MiaData date inz('2019-08-29');

  DataAlfa = %char(MiaData: *ISO0);  // 20190829
```

`*iso0` indica che il formato sarà YYYYMMDD senza separatori. Invece `*iso` prevederebbe l'uso dei separatori.

Programmi di esempio CVDAT2ALFA e CVDAT.

*ATTENZIONE: nel manuale ILE RPG reference nel capitolo della funzione %char è scritto un esempio per convertire un campo timestamp in alfanumerico senza separatori utilizzando nel secondo parametro la keyword *iso&; in realtà bisogna utilizzare la keyword *iso0.*

Da tipo data a tipo numerico ▲

La funzione **%dec** converte una data in campo numerico.

```
%dec(date time or timestamp expression {:format})
```

Il parametro `format` è disponibile da V5R3.

Per esempio:

```
dcl-s yyyymmdd packed(8);
dcl-s ddmmyy packed(6);
dcl-s MiaData date(*USA) inz(d'2019-10-20');

  yyyymmdd = %dec(MiaData : *iso);
  ddmmyy = %dec(MiaData : *dmy);
```

Si può convertire una data in campo numerico anche utilizzando le funzioni **%char** e **%uns**. Per esempio:

```
yyyymmdd = %uns(%char(MiaData : *iso0));
```

Può tornare comodo anche la funzione SQL **varchar_format**. P.es.:

```
  -- da data a campo numerico di 8 cifre
dec(varchar_format(current date), 'YYYYMMDD'), 8)
    -- da data a campo numerico di 7 cifre
dec(varchar_format(current date), 'YYYYMMDD'), 8) - 19000000
```

Anche la funzione SQL **dec** consente di effettuare la conversione da data/ora a campo numerico. P.es.:

```
dec(MyDate)
dec(MyTime)
dec(MyTimestamp)
dec(MyTimestamp, 26, 12)
dec(date(MyTimestamp))
dec(time(MyTimestamp), 6, 0)
```

Programma di esempio CVDAT2NUM(scarica).

Da altro tipo a data/ora/timestamp ▲

Le funzioni `%date`, `%time`, `%timestamp` convertono una variabile in data, ora o timestamp. Introdotte nella V5R1 sostituiscono i codici operativi `MOVE`, `MOVEL` (cfr. par. MOVE: qualcuno la usa ancora?).

%DATE

```
%date{(expression{:date-format})}
```

Se non viene specificato il *primo parametro*, viene restituita la **data corrente di sistema** (N.B. NON la data del job come restituirebbe *DATE). Cfr. anche il par. data/ora correnti.
Il *secondo parametro* specifica il formato della data in input; se non specificato si assume sia *iso. Se il primo parametro è un timestamp, oppure *DATE o UDATE il secondo parametro non è necessario.
La data restituita è sempre in formato *iso.

Per esempio per convertire un campo numerico 8,0 `DataPacked` che contiene una data in formato AAAAMMGG in un campo data `DataTrue`:

```
DataTrue = %date(DataPacked:*iso);
```

%TIME

```
%time{(expression{:time-format})}
```

Se non viene specificato il *primo parametro*, viene restituita l'**ora corrente di sistema**. Cfr. anche il par. data/ora correnti.
Il *secondo parametro* specifica il formato dell'ora in input; se non specificato si assume sia *iso. Se il primo parametro è un timestamp, il secondo parametro non è necessario.
L'ora restituita è sempre in formato *iso.

%TIMESTAMP

```
%timestamp{(char-num-expression { : *ISO|*ISO0 : {fractional-seconds}})}
%timestamp{(date-timestamp-expression { : fractional-seconds})}
```

Se non viene specificato il *primo parametro* o viene impostato a *sys, viene restituito il timestamp corrente di sistema (con solo le prime 3 cifre della porzione frazionale dei secondi). Cfr. anche il par. data/ora correnti.
Se il primo parametro è alfanumerico, si può specificare il formato nel *secondo parametro*.
Se il primo parametro è numerico l'unico formato ammesso è *iso.
Il timestamp restituito è sempre in formato *iso.

SQL way

Le funzioni precedenti sono ottime in un programma RPG, ma se si avesse bisogno di effettuare la conversione in un'istruzione SQL?
Nulla vieta di creare una funzione scalare SQL che con linguaggio SQL/PL o chiamando un programma RPG effettui la conversione.
Vediamo un esempio di conversione dal formato AAAAMMGG (campo packed di 8 cifre) a un tipo dati

data. L'esempio poi può essere poi adattato ad effettuare la conversione con formati diversi o migliorato per gestire con un'unica funzione tutti i possibili casi.
La funzione che creeremo si basa sulla funzione SQL date che restituisce un campo di tipo data da un valore alfanumerico che rappresenta una data o un timestamp (come descritto in String representations of datetime values).

Istruzione SQL per creare la funzione `cvtDate`:

```
create or replace function CVTDATE
  (InpData dec(8, 0) )
  returns date
  language sql
  contains sql
  no external action
  global deterministic
  return null on null input
  not fenced
  return date(left(char(InpData), 4) concat '-' concat substr(char(InpData), 5, 2) concat '-
' concat substr(char(InpData), 7, 2));
```

In breve la spiegazione dei parametri utilizzati per creare la funzione:

– dopo il nome della funzione sono elencati tra parentesi i parametri di input

– dopo la keyword returns si specifica il tipo di dato restituito

– language sql: indica che il body della funziona è scritto in linguaggio SQL/PL

– contain sql: la funzione non può eseguire nessuna istruzione SQL che legge o modifica dati

– no external action: indica che non vengono eseguite azioni esterne che cambiano lo stato di un oggetto non gestito dal DB2 (p.es. invio di un messaggio o la scrittura di un file su IFS)

– global deterministic: indica che a parità dei valori dei parametri di input il risultato restituito è sempre il medesimo. Questo consente di ottimizzare le performance della funzione, in quanto il DB2 può sfruttare un meccanismo di cache dei risultati

– return null on null input: se l'argomento di input è null restituisce null

– not fenced: la funzione viene eseguita nello stesso thread dell'istruzione SQL che la chiama

– dopo c'è il body della funzione: in questo caso molto semplice è composto da una sola istruzione

Altre funzioni SQL utili per effettuare la conversione del tipo data/ora/timestamp: `time`, `timestamp`, `timestamp_iso`, `timestamp_format`.

La funzione **time** restituisce un tipo dati time di un valore che rappresenta un'ora o un timestamp. Se il valore è una data time restituisce come ora la mezzanotte.

La funzione **timestamp** restituisce un tipo dati timestamp di un valore che rappresenta una data o un timestamp. Si può specificare solo il *primo parametro* come rappresentazione di un valore data o timestamp ed eventualmente nel *secondo parametro* si specifica un intero per indicare la precisione della porzione frazionale del timestamp. Altrimenti si può specificare nel *primo parametro* un valore che rappresenta una data e nel *secondo parametro* un valore che rappresenta un'ora.

Simile alla precedente è la funzione **timestamp_iso** che però può ricevere solo un parametro che rappresenta una data, un'ora o un timestamp. Se si specifica un valore che rappresenta un'ora, la porzione data viene impostata in base al registro CURRENT DATE.

La funzione **timestamp_format** (o le equivalenti **to_date**, **to_timestamp**) converte una stringa `string_expression` con formato `format_string` in un timestamp:

```
timestamp_format(string_expression, format_string, precision-constant)
```

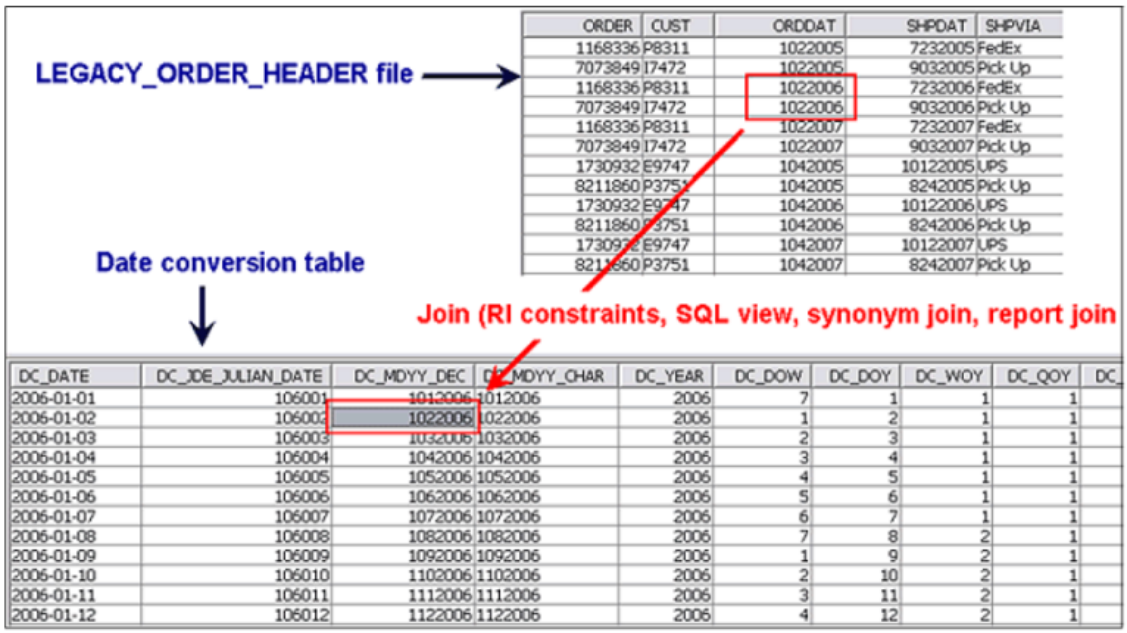
I separatori consentiti in format_string sono: `-`, `.`, `/`, `,`, `'`, `;`, `:`, `blank`.
Gli elementi del formato specificabile in format_string sono riassunti nella tab. 53 (p. 619) del manuale SQL reference 7.3. Alcuni esempi:

```
 values timestamp_format('1999-12-31 23:59:59', 'YYYY-MM-DD HH24:MI:SS');
 values timestamp_format('15/12/98 13:48', 'DD/MM/RRRR HH24:MI');
 values timestamp_format('9-3-2004 8:02', 'DD/MM/RRRR HH24:MI');
 values timestamp_format('190718', 'YYMMDD');
 values timestamp_format('2015-09-04 13:00:01', 'YYYY-MM-DD HH24:MI:SS');
 values timestamp_format('Settembre:2015:12 01:00:13', 'Month:YYYY:DD SS:MI:HH24');
 values timestamp_format('Nov:15:03 15', 'Mon:RR:DD HH24');
 ...where timestamp_format(char(DataPacked8), 'YYYYMMDD') between current date and (current
date + 30 days)
 ...where timestamp_format(char(DataPacked7+19000000), 'YYYYMMDD') between current date and
(current date + 30 days)
```

Tabella di conversione delle date▲

Un altro metodo di "conversione" di una data memorizzata nel DB2 in un campo con tipo dati non appropriato consiste nell'utilizzare una tabella di conversione delle date.
Si tratta di una **tabella "calendario"** che contiene un record per ogni giorno in uno specifico range di date.
Ogni record contiene la stessa data rappresentata in diversi modi (data giuliana, data "vera", giorno della settimana, data YYYYMMDD, data DDMMYY, excel...).
E' documentata nel redbook SG24-7214-01 "Getting Started with DB2 Web Query for i" (appendix B p. 542-552).

Potete scaricare lo zip MK1DATECNV.ZIP che contiene lo script di creazione localizzato per l'Italia ed arricchito.



ORDER	CUST	ORDDAT	SHPDAT	SHPVIA
1168336	P8311	1022005	7232005	FedEx
7073849	I7472	1022005	9032005	Pick Up
1168336	P8311	1022006	7232006	FedEx
7073849	I7472	1022006	9032006	Pick Up
1168336	P8311	1022007	7232007	FedEx
7073849	I7472	1022007	9032007	Pick Up
1730932	E9747	1042005	10122005	UPS
8211860	P3751	1042005	8242005	Pick Up
1730932	E9747	1042006	10122006	UPS
8211860	P3751	1042006	8242006	Pick Up
1730932	E9747	1042007	10122007	UPS
8211860	P3751	1042007	8242007	Pick Up

DC_DATE	DC_JDE_JULIAN_DATE	DC_MDYY_DEC	DC_MDYY_CHAR	DC_YEAR	DC_DOW	DC_DOY	DC_WOY	DC_QOY	DC_
2006-01-01	106001	1012006	1012006	2006	7	1	1	1	
2006-01-02	106002	1022006	1022006	2006	1	2	1	1	
2006-01-03	106003	1032006	1032006	2006	2	3	1	1	
2006-01-04	106004	1042006	1042006	2006	3	4	1	1	
2006-01-05	106005	1052006	1052006	2006	4	5	1	1	
2006-01-06	106006	1062006	1062006	2006	5	6	1	1	
2006-01-07	106007	1072006	1072006	2006	6	7	1	1	
2006-01-08	106008	1082006	1082006	2006	7	8	2	1	
2006-01-09	106009	1092006	1092006	2006	1	9	2	1	
2006-01-10	106010	1102006	1102006	2006	2	10	2	1	
2006-01-11	106011	1112006	1112006	2006	3	11	2	1	
2006-01-12	106012	1122006	1122006	2006	4	12	2	1	

La logica di utilizzo di questa tabella di conversione è metterla in join con la tabella contenente la data “legacy” tramite il campo più appropriato (p.es. per una data legacy memorizzata in un campo packed 8,0 in formato YYMMDD si utilizza DC_YYMD_DEC) e quindi utilizzare nel resto della query il campo data “vero” DC_DATE.

Garantito al 100% che questa tecnica offre prestazioni migliori rispetto a qualsiasi formula di conversione, soprattutto se il campo è utilizzato nella clausola where di una istruzione SQL.

Durate▲

Le funzioni RPG %days, %months, %years, %hours, %minutes, %seconds, %mseconds convertono un campo numerico in una durata. Introdotte da V5R1 sostituiscono i codici operativi SUBDUR/ADDDUR (cfr. par. Somma o sottrazione di durate).

`%days(number)`
converte un numero in una durata in giorni che può essere sommata/sottrata a una data o a un timestamp.

`%months(number)`
converte un numero in una durata in mesi che può essere sommata/sottrata a una data o a un timestamp.

`%years(number)`
converte un numero in una durata in anni che può essere sommata/sottrata a una data o a un timestamp.

`%hours(number)`
converte un numero in una durata in ore che può essere sommata/sottrata a un’ora o un timestamp.

`%minutes(number)`
converte un numero in una durata in minuti che può essere sommata/sottrata a un'ora o un timestamp.

`%seconds(number)`
converte un numero in una durata in secondi che può essere sommata/sottrata a un'ora o un timestamp.
Se si aggiungono/sottraggono secondi a un timestamp, il numero può avere anche una porzione decimale.

`%mseconds(number)`
converte un numero in una durata in microsecondi che può essere sommata/sottrata a un'ora o un timestamp.
`%seconds(.000005)` and `%mseconds(5)` rappresentano entrambi 5 microseconds.

I codici di durata che possono essere utilizzati con i codici operativi ADDDUR/SUBDUR sono: `*YEARS` o `*Y`, `*MONTHS` o `*M`, `*DAYS` o `*D`, `*HOURS` o `*H`, `*MINUTES` o `*M`, `*SECONDS` o `*S`, `*MSECONDS` o `*MS`.

Controllo di un campo data ▲

Il codice operativo TEST consente di controllare se il contenuto di un campo (**NON di tipo data/ora/timestamp** ovviamente) è una data/ora/timestamp valido.
L'*operatore di estensione* indica il tipo di controllo da effettuare: `D` per le date, `T` per le ore, `Z` per i timestamps.

Per esempio per controllare la validità di una data nel formato di default:

```
test(de) DataAlfa;
```

oppure per controllare uno specifico formato (*YMD. = anno, mese, giorno con separatore il punto):

```
test(de) *ymd. DataAlfa;
```

Si può anche ricorrere alla struttura di intercettazione degli errori monitor per controllare il buon esito di operazioni con le date. P.es.:

```
monitor;
  YMD_date = %date(DataAlfa : *iso);
on-error 112 : 113 : 114;
  // data errata
  // ...
endmon;
```

Scarica i programmi di esempio TESTDAT1.RPGLE e TESTDAT2.RPGLE e TESTDAT3.RPGLE.

Comparazione

Per comparare due variabili di tipo data **non è strettamente necessario che siano nello stesso formato**.
Il formato è una rappresentazione del valore data, quindi anche cambiando formato il valore non cambia. Ne consegue che si possono comparare correttamente due variabili data anche se sono di formato diverso.

Scarica il programma di esempio COMPDAT.RPGLE.

Somma o sottrazione di durate

I codici operativi SUBDUR/ADDDUR eseguono somme o sottrazioni su campi data, ora o timestamp. Possono essere usati nelle specifiche C a *formato fisso*. Nel *formato libero* si può usare l'operatore + o - con le funzioni di durata (p.es. `%YEARS`, `%DAYS`...). Cfr. par. durate.
ADDDUR/SUBDUR per le date aggiungono/sottraggono giorni, mesi o anni. Per le ore aggiungono/sottraggono ore, minuti o secondi. Per i timestamp aggiungono/sottraggono microsecondi. P.es. programma SOMMADUR.RPGLE e SOMMADURF.RPGLE:

```
c     MiaData       adddur    7:*DAYS       ResData
c     MiaOra        adddur    8:*HOURS      ResOra
```

Il campo risultato deve essere di tipo data, ora o timestamp.

Se la somma/sottrazione produce una data non valida (p.es. sommando anni o mesi) il risultato viene impostato all'ultimo giorno valido dello stesso mese.
Se la somma/sottrazione produce un'ora non valida viene automaticamente normalizzata. P.es. se alle 23:30:00 si somma 1 ora si otterrà le 00:30:00 e non le 24:30:00.
Il campo che contiene la durata da sommarre/sottrarre è una variabile numerica (packed, zoned o binaria). La dimensione massima è 15 digit.
L'indicatore del minore (<) viene acceso se il fattore 1 contiene una data non valida.

`SUBDUR` può anche essere utilizzato per calcolare il **tempo trascorso tra due date**, nel qual caso fattore 1 e 2 devono essere due campi data, ora o timestamp. Nel formato libero si utilizza la funzione %diff.
P.es. programma DIFFDAT.RPGLE:

```
c     DataConsegna  subdur    DataOrdine    DiffGG:*D
```

Il campo risultato deve essere un campo numerico con 0 decimali e dimensione massima 15 digit.
`SUBDUR` può calcolare la differenza oltre che ovviamente tra due campi dello stesso tipo, anche tra data e timestamp, ora e timestamp.

ADDDUR/SUBDUR vs ultimo giorno del mese o anni bisestili:

I codici operativi ADDDUR/SUBDUR hanno dei limiti chiamati "*risultati imprevisti*" di cui occorre tener conto prima di incorrere in errori di calcolo.
Si veda il seguente esempio nel programma SOMMADUR2.RPGLE:

```
c     d'28/02/02'   ADDDUR    30:*DAYS      MiaData1
 * risultato MiaData1 = 30/03/02

c     d'28/02/02'   ADDDUR    1:*MONTHS     MiaData2
 * risultato MiaData2 = 28/03/02

c     MiaData1      ADDDUR    30:*DAYS      MiaData3
 * risultato MiaData3 = 29/04/02

c     MiaData2      ADDDUR    1:*MONTHS     MiaData4
 * risultato MiaData4 = 28/04/02
```

Regole generali:
– quando si aggiungono/sottraggono mesi o anni la porzione del giorno rimane invariata se possibile.
– se però si ottiene un risultato non consistente con un tipo di data valido viene utilizzato l'ultimo giorno del mese
– qualsiasi operazione di questo tipo che ha prodotto una variazione della porzione del giorno non è reversibile. P.es. `2000-03-31 + %MONTHS(1) - %MONTHS(1)` avrà come risultato 2000-03-30
– la durata tra due date è 1 mese se sottraendo 1 mese alla data maggiore si ottiene la data minore. P.es. la differenza tra 2000-03-31 e 2000-04-30 è 0 mesi, perché 2000-04-30 – %MONTHS(1) è 2000-03-30.

Per sommare o sottrare durate in SQL si possono usare gli operatori matematici (+ o -) con etichette di durata (*labeled duration*). P.es.

```
date('2023-07-23') + 3 days
date('2023-07-23') + 3 months
```

In alternativa agli operatori matematici si possono usare le funzioni SQL: ADD_YEARS, ADD_MONTHS, ADD_DAYS, ADD_HOURS, ADD_MINUTES, ADD_SECONDS (a parte add_months queste funzioni sono state introdotte dalla versione 7.4 TR8 e 7.5 TR2).

Occorre prestare particolare attenzione alla funzione SQL add_months che consente di sommare un numero di mesi ad una data. P.es.: `add_months(current date, 3)`. Il risultato di questa funzione è simile all'utilizzo di operatori matematici con date e durate, ma ci sono alcune differenze da tenere in considerazione.
P.es. in questo caso il risultato è identico

```
date('2000-02-28') + 4 months = '2000-06-28'
add_months('2000-02-28', 4) = '2000-06-28'
```

ma

```
date('2000-02-29') + 4 months = '2000-06-29'
```

```
add_months('2000-02-29', 4) = '2000-06-30' -- somma i mesi adattando il giorno all'ultimo giorno del mese
```

Differenza tra date e ore▲

La funzione %diff calcola la differenza tra date, ore o timestamps. Introdotta nella V5R1 sostituisce il codice operativo SUBDUR (cfr. par. somma o sottrazione di durate).

```
%diff(op1 : op2 : unit {: frac })
```

Il *parametro unit* definisce con quale tipo di durata sarà restituito il risultato (p.es. *DAYS, *MONTHS...).
I *primi due parametri* devono avere un tipo dati compatibile per calcolare la differenza. E' anche possibile calcolare la differenza tra un campo data o ora e un timestamp: nel primo caso verrà usata solo la porzione data del timestamp invece nel secondo caso verrà usata solo la porzione dell'ora. Il *risultato* è sempre arrotondato in difetto e l'eventuale resto viene ignorato. P.es. una differenza di 61 minuti è uguale a 1 ora, invece una differenza di 59 minuti è uguale a 0 ore.

Per esempio: `DiffGG = %diff(DataConsegna : DataOrdine : *days);`

Altri esempi nel programma DIFFDAT.RPGLE.

La differenza tra due date può essere calcolata anche con **SQL**. Bisogna porre attenzione al calcolo della differenza con le funzioni scalari year, month o day. Si potrebbe essere tentati di pensare di utilizzare la funzione SQL che corrisponde al terzo parametro della funzione RPG `%diff`, ovvero `month` se si desidera la differenza in mesi (*months). Ma vediamo alcuni esempi (programma DIFFDATS.SQLRPGLE) per chiarire meglio la differenza tra RPG e SQL.

```
dcl-s DataOrdine date inz(d'2018-11-01');
dcl-s DataConsegna date inz(d'2019-10-20');

  // differenza in giorni
  exec sql
    set :DiffGG = days(:DataConsegna) - days(:DataOrdine);
  // risultato 353
  DiffGG = %diff(DataConsegna : DataOrdine : *days);
  // risultato 353

  // differenza in mesi
  exec sql
    set :DiffMM = month(:DataConsegna) - month(:DataOrdine);
  // risultato -1
  DiffMM = %diff(DataConsegna : DataOrdine : *months);
  // risultato 11
```

La spiegazione risiede nel fatto che la funzione SQL `month` estrae la porzione del mese da una data quindi nell'esempio mostrato con l'istruzione SQL si calcola la differenza tra il mese 11 e il mese 10 ovvero -1.

Con SQL invece è possibile calcolare un tipo di differenza tra date che non ha un equivalente immediato in RPG, ovvero la **differenza composta in anni/mesi/giorni**. Per esempio:

```
dcl-s DataOrdine date inz(d'2018-11-01');
dcl-s DataConsegna date inz(d'2019-10-20');
dcl-s DataOrdine2 date inz(d'2009-12-01');
dcl-s DataConsegna2 date inz(d'2019-01-20');
dcl-s Diff char(10);

  exec sql
    set :Diff = date(:DataConsegna) - date(:DataOrdine);
  // risultato = 1119 ovvero: 11 mm 19 gg

  exec sql
    set :Diff = date(:DataConsegna2) - date(:DataOrdine2);
  // risultato = 90119 ovvero 9 aa 01 mm 19 gg
```

Una funzione SQL simile a `days` è julian_day che restituisce il numero di giorni a partire dal 1 gennaio 4713 a.c. (ovvero l'inizio del calendario giuliano). Quindi la differenza tra due date convertite con la funzione `julian_day` restituisce il numero di giorni intercorrenti tra le due date.

La funzione SQL months_between restituisce un numero stimato di mesi tra due date. Il risultato è un tipo dati dec(31, 15). Si assume che il mese abbia 31 giorni, ma la funzione tiene anche conto se si sta calcolando la differenza tra gli ultimi giorni di mesi differenti. Per esempio:

```
dcl-s DataOrdine date inz(d'2018-11-01');
dcl-s DataConsegna date inz(d'2019-10-20');
dcl-s DataOrdine2 date inz(d'2009-12-01');
dcl-s DataConsegna2 date inz(d'2019-01-20');
dcl-s DataOrdine3 date inz(d'2008-02-29');
dcl-s DataConsegna3 date inz(d'2008-03-29');
dcl-s DataOrdine4 date inz(d'2008-02-29');
dcl-s DataConsegna4 date inz(d'2008-03-31');
dcl-s DiffMM2 packed(31:15);

  exec sql
    set :DiffMM2 = months_between(:DataConsegna, :DataOrdine);
  // risultato = 11.61290322580645
  exec sql
    set :DiffMM2 = months_between(:DataConsegna2, :DataOrdine2);
  // risultato = 109.6129032258064
  exec sql
    set :DiffMM2 = months_between(:DataConsegna3, :DataOrdine3);
  // risultato = 1.000000000000000
  // la porzione giorno delle due date è identica (29)
  //  e quindi il numero stimato di mesi differenza è 1
  exec sql
    set :DiffMM2 = months_between(:DataConsegna4, :DataOrdine4);
  // risultato = 1.000000000000000
  // per entrambe le date si tratta dell'ultimo giorno del mese
  //  e quindi il numero stimato di mesi di differenza è 1
```

La funzione SQL timestampdiff restituisce un numero stimato di intervalli del tipo definito dal primo argomento, basato sulla differenza tra due timestamp.

```
timestampdiff(interval-type, string-expression)
```

Il *primo parametro* specifica il tipo di intervallo da restituire come differenza tra due timestamp (secondi, minuti, ore, giorni...), come riepilogato alla tab. 1 del manuale SQL reference (cfr. https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/db2/rbafzscatimedifstmp.htm).
Il *secondo parametro* è la differenza tra due timestamp convertita in una stringa di 22 caratteri composta come descritto nella tab. 2 del manuale SQL reference (cfr. https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/db2/rbafzscatimedifstmp.htm). Per esempio:

```
dcl-s DataOrdine date inz(d'2018-11-01');
dcl-s DataConsegna date inz(d'2019-10-20');
dcl-s DataOrdine2 date inz(d'2009-12-01');
dcl-s DataConsegna2 date inz(d'2019-01-20');
dcl-s DataOrdine3 date inz(d'2008-02-29');
dcl-s DataConsegna3 date inz(d'2008-03-29');
dcl-s DataOrdine4 date inz(d'2008-02-29');
dcl-s DataConsegna4 date inz(d'2008-03-31');
dcl-s DiffInt int(20);

  exec sql
    set :DiffInt =
      timestampdiff(64,
        cast(cast(:DataConsegna as timestamp) -
             cast(:DataOrdine as timestamp) as char(22)));
  // risultato = 11
  // se nel primo parametro invece di 64 (mesi) si richiede la differenza in giorni (16)
  // risultato (gg) = 349

  exec sql
    set :DiffInt =
      timestampdiff(64,
        cast(cast(:DataConsegna2 as timestamp) -
             cast(:DataOrdine2 as timestamp) as char(22)));
  // risultato = 109
  // risultato (gg) = 3334

  exec sql
    set :DiffInt =
      timestampdiff(64,
        cast(cast(:DataConsegna3 as timestamp) -
             cast(:DataOrdine3 as timestamp) as char(22)));
  // risultato = 1
  // risultato (gg) = 30

  exec sql
    set :DiffInt =
      timestampdiff(64,
        cast(cast(:DataConsegna4 as timestamp) -
             cast(:DataOrdine4 as timestamp) as char(22)));
  // risultato = 1
  // risultato (gg) = 32
```

Estrazione porzione di date e ore ▲

Il codice operativo RPG EXTRCT estrae una porzione dal campo di tipo data, time o timestamp.
Si utilizzano i codici di durata (cfr. par. durate) per determinare quale porzione estrarre.
Nel *formato libero* si utilizza la funzione %subdt.

In *fattore 2* si specifica il campo da cui si desidera estrarre la porzione e il codice durata (`*D`, `*M`...).
Se fattore 2 contiene un timestamp si possono estrarre solo microsecondi; per estrarre porzioni diverse bisogna usare la funzione `%subdt`.

La funzione `%subdt` estrae porzione di una data, ora o timestamp. Sostituisce il codice operativo `EXTRCT`.

```
%subdt(value : unit { : digits { : decpos } })
```

La funzione restituisce sempre l'anno a 4 cifre anche se la data è in un formato con anno a 2 cifre.
Il giorno estratto (`*DAYS`) è sempre riferito al giorno del mese e non dell'anno, anche se la data è in formato giuliano.
Il parametro `decpos` è disponibile da 7.2.

Per esempio (programma EXTRACT.RPGLE):

```
// estrazione anno
AnnoOrdine = %subdt(DataOrdine : *y);
// estrazione mese
MeseOrdine = %subdt(DataOrdine : *m);
```

Un altro metodo per estrarre una porzione di data è sfruttare i puntatori. Nel programma EXTRACTJUL.RPGLE vediamo un esempio.

Dalla versione 7.2 TR9 o 7.3 TR5 la funzione SQL extract è stata ulteriormente arricchita e può restituire parte di una data o ora da un campo data, ora o timestamp.
Le informazioni che si possono estrarre con la funzione `extract` sono:
– epoch: numero di secondi trascorsi dalle 12:00 AM del 1970
– millenium: numero del millenio
– century: numero del secolo
– decade: numero in periodo di 10 anni
– year: risultato identico alla funzione year
– quarter: risultato identico alla funzione quarter.
– month: risultato identico alla funzione month
– week: risultato identico alla funzione week
– day: risultato identico alla funzione day
– dow: risultato identico alla funzione dayofweek
– doy: risultato identico alla funzione dayofyear
– hour: risultato identico alla funzione hour
– minute: risultato identico alla funzione minute
– second: risultato identico alla funzione second
– millisecond
– microsecond: risultato identico alla funzione microsecond

Esistono anche altre funzioni SQL che estraggono da una data informazioni “descrittive” come il nome del giorno o del mese (nella lingua appropriata):
– dayname: restituisce una stringa mixed case con il nome del giorno. I nomi sono contenuti nel messaggio CPX9034 del file QCPFMSG
– monthname: restituisce una stringa mixed case con il nome del mese. I nomi sono contenuti nel messaggio CPX3BC0 del file QCPFMSG

Altre funzioni SQL estraggono altre parti di data restituendo informazioni sul giorno relativo del mese o della settimana in maniera conforme anche allo standard ISO:
– dayofmonth: restituisce il giorno del mese
– dayofweek_iso: restituisce il numero del giorno della settimana, dove 1 è lunedì. Invece la funzione `dayofweek` restituisce 1 per il giorno di domenica
– week_iso: restituisce il numero della settimana (compreso tra 1 e 53, a differenza della funzione week che restituisce un numero compreso tra 1 e 54). La settimana inizia con lunedì. La settimana numero 1 è la prima settimana dell’anno che contiene un giovedì ovvero la prima settimana dell’anno che contiene il 4 gennaio.

MOVE: qualcuno la usa ancora? ▲

Alcuni esempi nel programma MOVEDAT.RPGLE:

```
     ctl-opt datfmt(*YMD-) timfmt(*ISO);

     dcl-s Date1 date(*MDY/) inz(D'02-10-22');
     dcl-s Date2 date(*ISO) inz(D'02-10-22');
     dcl-s Time1 time(*EUR) inz(T'13.00.00');
     dcl-s Chardate char(8) inz('03-10-22');
     dcl-s DataUSA date(*USA);
     dcl-s UpdDate date(*ISO);
     dcl-s UpdTime time(*USA);

C                    MOVE      Date1          UpdDate
 ***UpdDate = '2002-10-22'

C                    MOVE      Time1          UpdTime
 ***UpdTime = '01:00 PM'
```

```
C     *YMD-         MOVE      CharDate      DataUSA
 ***DataUSA = '10/22/2003'

C*                  MOVEL     Date2         CharDate
 *** Errore in compilazione: Chardate è troppo piccolo
```

Arrotondamento di date e ore ▲

In alcune situazioni si può avere la necessità di arrotondare un campo data o ora.

Si può fare tramite RPG e vediamo l'esempio di arrotondamento al minuto nel programma ROUNDMIN.RPGLE.

Sicuramente però la funzione SQL round_timestamp è molto più comoda: restituisce un timestamp arrotondato all'unità specificata nel *secondo parametro*. Se viene omesso il secondo parametro si assume 'DD' ovvero l'arrotondamento al giorno. I valori per il secondo parametro sono elencati nella tab. 1 del manuale SQL reference (https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/db2/rbafzscaroundts.htm)
La funzione SQL trunc_timestamp restituisce un timestamp troncato all'unità specificata nel *secondo parametro*. Se viene omesso il secondo parametro si assume 'DD' ovvero l'arrotondamento al giorno. Alcuni esempi nel programma ROUNDSQL.SQLRPGLE.

```
// arrotondamento al giorno
exec sql
  set :TimeR = round_timestamp(:Time1, 'DD');

// arrotondamento al minuto
exec sql
  set :TimeR = round_timestamp(:Time1, 'MI');
```

Altre funzioni SQL ▲

last_day restituisce la data corrispondente all'ultimo giorno del mese della data ricevuta come parametro. La funzione restituisce una data o un timestamp in maniera congruente al parametro ricevuto

```
-- restituisce 2019-11-30
select last_day(date('2019-11-16')) as "Ultimo Giorno"
     from sysibm/sysdummy1;
```

midnight_seconds restituisce il numero di secondi tra mezzanotte e l'ora specificata nel parametro. Il valore restituito è compreso tra 0 e 86400. 00:00:00 e 24:00:00 rappresentano entrambi la mezzanotte, però nel primo caso il risultato della funzione è 0 mentre nel secondo è 86400

next_day restituisce una data o timestamp che rappresenta la data del primo giorno della settimana (definito nel secondo parametro) successiva alla data del primo parametro. Le abbreviazioni per il secondo parametro sono 'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT', 'SUN' (oppure si possono

specificare anche nel linguaggio nazionale installato sul server, così come sono scritti nei messaggi CPX9034 e CPX9039).

```
-- restituisce 2019-11-18 ovvero il primo lunedì dopo il 16/11/2019
select next_day(date('2019-11-16'), 'LUN') as "LunedìSuccessivo"
     from sysibm/sysdummy1;
```

varchar_format consente di restituire un campo carattere con la rappresentazione della data/timestamp del primo parametro nel formato descritto dalla stringa immagine del secondo parametro. Cfr. paragrafo formattare date e ore.

API ▲

Esistono numerose API per manipolare date/ore e timestamps:

QWCADJTM: regolazione del fuso orario

Qp0zCvtToTimeval(): converte un machine timestamp (tipo dati _MI_time) nella corrispondente struttura timeval.

Qp0zCvtToMITime(): funzione inversa della precedente

QWCCVTDT: converte date e ora tra formati diversi o tra tipi dati diversi. Un esempio di utilizzo nel programma CVTDATAPI.RPGLE.

QWCRTVTM: reperisce le informazioni di sistema relative all'ora (UTC Coordinated Universal Time e il fuso orario)

QWCRTVTZ: reperisce le informazione di uno o più fusi orari

QWCSETTM: imposta UTC Coordinated Universal Time

API CEE

Nella categoria delle API CEE abbiamo a disposizione:

CEEDAYS: converte la data dal formato alfanumerico al formato liliano, ovvero un numero intero che rappresenta il numero di giorni dall'inizio del calendario gregoriano (15-ott-1582 cfr. wikipedia.org/wiki/Lilian_date)

CEEDATE: converte la data dal formato liliano al formato alfanumerico

CEEDYWK: calcola il giorno della settimana (un numero compreso tra 1 e 7, con 1 domenica) di una data in formato liliano

CEESECS: converte un timestamp dal formato alfanumerico al formato liliano ovvero in un numero a virgola mobile che rappresenta il numero di secondi trascorsi dall'inizio del calendario gregoriano (ovvero dalla mezzanotte del 15-ott-1582)

CEEDATM: converte il numero di secondi in formato timestamp alfanumerico

CEELOCT: restituisce la data e il timestamp del sistema locale in 3 formati: data liliana, timestamp liliano, timestamp alfanumerico

CEESECI: estrae le parti di un timestamp (anno, mese, giorno, ora, minuti, secondi, millisecondi) da un timestamp liliano

CEESCEN: imposta il secolo per le date con anni a due cifre

CEEUTCO: restituisce la differenza dell'ora locale rispetto a UTC (Universal Time Coordinated). Vedi programma di esempio TIMEZONE.RPGLE.

Nel programma CVALFA2DAT.RPGLE vediamo un esempio di utilizzo delle API CEEDATE e CEEDAYS per convertire una stringa in un campo data e per cambiare il formato.

Una importante **differenza tra le bif rpg e le api cee** riguarda le date espresse con **anni di due cifre**. Con le bif l'intervallo gestito è 1940-2039, quindi gli anni da 40 a 99 saranno preceduti da 19, invece quelli da 0 a 39 saranno preceduti da 20. Con le API CEE invece il limite del secolo è 80 anni prima della data di sistema. Quindi p.es. oggi (2019) l'intervallo è 1939-2038. Eventualmente l'API CEESCEN può essere utilizzata per impostare un intervallo di secolo diverso.

API runtime C

Le API CEE gestiscono ogni porzione di una strinda data/ora ad **eccezione del fuso orario**. A tal scopo è necessario utilizzare le API della libreria di runtime C. Gli svantaggi con le **api C** sono che:
– le api di conversione non verificano le date e ore
– non tengono conto dell'impostazione del sistema operativo per il fuso orario e l'ora legale

asctime(), asctime_r(), ctime(), ctime_r(), ctime64(), ctime64_r(): converte un timestamp in una stringa alfanumerica di 26 car. con il formato "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n". P.es. Sat Jul 16 02:03:55 1994\n\0 (con \n carattere di nuova riga e \0 carattere null).
`ctime()` è equivalente a `asctime(localtime(&anytime))`.

clock(): restituisce il timestamp del processore. Utile per calcolare con precisione i tempi di esecuzione di un programma.

difftime(), difftime64(): calcola la differenza in secondi tra due timestamp.

gmtime(), gmtime_r(), gmtime64(), gmtime64_r(): divide un timestamp in porzioni in una struttura dati

localtime(), localtime_r(), localtime64(), localtime64_r(): converte un timestamp nell'ora locale e lo divide in porzioni in una struttura dati

mktime(), mktime64(): funzione inversa rispetto a `gmtime`

time(), time64(): restituisce l'ora corrente in secondi (ovvero il numero di secondi a partire da EPOCH ovvero 1/gen/1970)

strptime(): converte una stringa in timestamp

strftime(), wcsftime(): converte in una data/ora formattata

strptime(), wcsptime(): converte una stringa in una data/ora formattata

API Unix-Type

Tra le API di tipo Unix abbiamo a disposizione.

System clock APIs:

adjtime(): regola l'ora di sistema

ftime(): reperisce UTC corrente

gettimeofday(): reperisce UTC corrente compreso delle informazioni sul fuso orario

settimeofday(): imposta UTC corrente

Software clock APIs:

Qp0zAdjTime(): regola il software clock

Qp0zGetTimeofDay(): reperisce il software clock

Qp0zSetTimeofDay(): imposta il software clock

N.B. i componenti di sistema non si basano sul software clock ma sul system clock. Le API che lavorano con il software clock non dovrebbero essere utilizzate.

Esempi vari ▲

Cambio formato con MULT

Riporto un metodo molto sintetico e storicamente importante (soprattutto negli stati uniti) per convertire una data in formato YMD memorizzata in un campo numerico 6, 0 nel formato MDY. Una riga di codice rigorosamente scritta con il codice operativo MULT in formato fisso!

```
     // data ora in input
     dcl-s wDataI packed(6);
     // data output
     dcl-s pmDataO packed(6);

C      wDataI        mult      100,0001      pmDataO
```

Scarica il programma completo CVTMULT.RPGLE.

Calcolare il primo e l'ultimo giorno del mese con RPG

Nel programma LASTDAY.RPGLE (tratto dal manuale del corso IBM da RPG/400 a System i ILE) vediamo come calcolare l'ultimo giorno del mese con i codici operativi RPG time, adddur, subdur, extrct.

Un metodo alternativo utilizzando le BIF RPG è:

```
dcl-s LastDay date;

LastDay = (%date() - %days(%subdt(%date():*days)-1)) +
           %months(1) - %days(1);
```

Invece per calcolare il primo giorno del mese:

```
dcl-s FirstDay date;

FirstDay = %date - %days(%subdt(%date:*days)) + %days(1);
```

Calcolare il numero del giorno della settimana con RPG

Il programma DAYWEEK.RPGLE calcola il numero del giorno della settimana (con primo giorno della settimana lunedì = 1)

```
dcl-s MiaData date inz(*sys);
dcl-s GiornoSettimana zoned(1);
dcl-s DataRif date inz(d'0001-01-01');

GiornoSettimana = %rem(%diff(MiaData : DataRif : *DAYS) : 7) + 1;
```

Formattare date e ore ▲

In questo capitolo vediamo come esporre campi di tipo data/ora/timestamp utilizzando il formato che si desidera.
Esistono per questo scopo diversi metodi disponibili sia tramite keyword nei display files o printer files sia tramite BIF RPG o funzioni SQL.

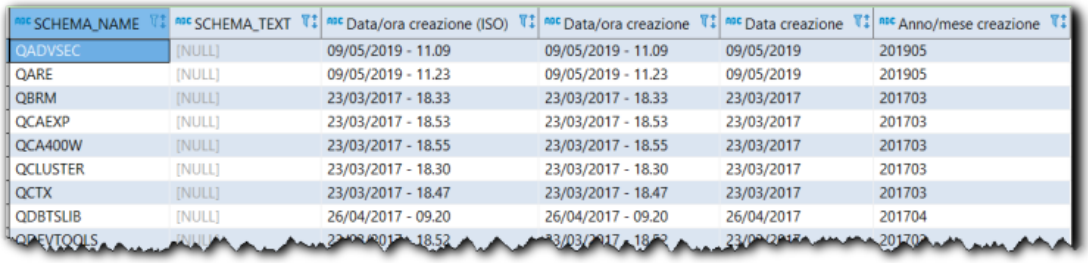
Il riepilogo dei vari formati disponibili è esposto al paragrafo definizione del formato.

Per le keyword di display e printer files che condizionano il formato dei campi data/ora si veda quanto già scritto sopra al paragrafo display e printer files.

Per formattare un campo data/ora/timestamp è molto comoda la funzione SQL varchar_format.
Attenzione: la funzione restituisce un tipo dati carattere a lunghezza variabile. Quindi se per esempio in una istruzione SQL si desidera esporre un campo timestamp (p.es. UltAggRek) formattato e ordinare il risultato sempre per il campo UltAggRek nella clausola select si utilizzerà la funzione varchar_format, ma nella clausola order by bisogna utilizzare il campo "originale" UltAggRek della tabella.

I separatori validi che possono essere specificati nella stringa di formattazione (secondo parametro) sono: `-`, `.`, `/`, `,`, `'`, `;`, `:`, `blank`.

```
-- per esempio
select SCHEMA_NAME, SCHEMA_TEXT,
   varchar_format(CREATION_TIMESTAMP, 'DD/MM/IYYY - HH24.MI') "Data/ora creazione (ISO)",
   varchar_format(CREATION_TIMESTAMP, 'DD/MM/YYYY - HH24.MI') "Data/ora creazione",
   varchar_format(CREATION_TIMESTAMP, 'DD/MM/IYYY') "Data creazione",
   varchar_format(CREATION_TIMESTAMP, 'YYYYMM') "Anno/mese creazione"
  from QSYS2/SYSSCHEMAS
  where SYSTEM_SCHEMA_NAME like 'Q%'
  order by SYSTEM_SCHEMA_NAME, CREATION_TIMESTAMP desc;
```



SCHEMA_NAME	SCHEMA_TEXT	Data/ora creazione (ISO)	Data/ora creazione	Data creazione	Anno/mese creazione
QADVSEC	[NULL]	09/05/2019 - 11.09	09/05/2019 - 11.09	09/05/2019	201905
QARE	[NULL]	09/05/2019 - 11.23	09/05/2019 - 11.23	09/05/2019	201905
QBRM	[NULL]	23/03/2017 - 18.33	23/03/2017 - 18.33	23/03/2017	201703
QCAEXP	[NULL]	23/03/2017 - 18.53	23/03/2017 - 18.53	23/03/2017	201703
QCA400W	[NULL]	23/03/2017 - 18.55	23/03/2017 - 18.55	23/03/2017	201703
QCLUSTER	[NULL]	23/03/2017 - 18.30	23/03/2017 - 18.30	23/03/2017	201703
QCTX	[NULL]	23/03/2017 - 18.47	23/03/2017 - 18.47	23/03/2017	201703
QDBTSLIB	[NULL]	26/04/2017 - 09.20	26/04/2017 - 09.20	26/04/2017	201704
QDEVTOOLS	[NULL]	23/03/2017 - 18.52	23/03/2017 - 18.52	23/03/2017	201703

```
   -- 15/06/2019 - 23.29
select varchar_format(now(), 'DD/MM/IYYY - HH24.MI')
  from SYSIBM/SYSDUMMY1;

   -- sabato 15/06/2019 - 23.31
select varchar_format(now(), 'day DD/MM/IYYY - HH24.MI')
  from SYSIBM/SYSDUMMY1;

   -- sab 15 giugno 2019
select varchar_format(now(), 'dy DD month IYYY')
  from SYSIBM/SYSDUMMY1;

   -- sab 15 giu 2019 - 24ª settimana - 2° quadrimestre
select varchar_format(now(), 'dy DD mon IYYY - WW') concat 'ª settimana' concat ' - ' concat
varchar_format(now(), 'Q') concat '° quadrimestre'
  from SYSIBM/SYSDUMMY1;
```

Alcuni elementi di formattazione sono preceduti dalla lettera **I**: per esempio IYYY indica l'anno ISO, invece YYYY è l'anno. Quale la differenza?
Vediamo con un esempio e ricordiamo che la la prima settimana dell'anno secondo lo standard ISO è quella che contiene il 4 di gennaio.

```
values(varchar_format(date('2018-12-31'), 'IYYY')); --> restituisce 2019
values(varchar_format(date('2018-12-31'), 'YYYY')); --> restituisce 2018
```

Alcuni esempi con diversi formati di ora e i valori di mezzogiorno e mezzanotte:

```
select
  varchar_format(timestamp('2000-01-01-00.00.00.000000'), 'HH'),   --> 00
  varchar_format(timestamp('2000-01-01-12.00.00.000000'), 'HH'),   --> 12
  varchar_format(timestamp('2000-01-01-24.00.00.000000'), 'HH'),   --> 12
  varchar_format(timestamp('2000-01-01-00.00.00.000000'), 'HH24'), --> 00
  varchar_format(timestamp('2000-01-01-12.00.00.000000'), 'HH24'), --> 12
  varchar_format(timestamp('2000-01-01-24.00.00.000000'), 'HH24')  --> 24
  from SYSIBM/SYSDUMMY1;
```

Data e ora correnti

Nella definizione di una variabile di tipo data è possibile inizializzarla alla data di sistema con la keyword `inz(*sys)` oppure alla data del job con `inz(*job)`.

```
dcl-s DataJob date inz(*job);
dcl-s DataSys date inz(*sys);
```

oppure in formato fisso

```
D DataJob         S               D   Inz(*job)
D DataSys         S               D   Inz(*sys)
```

Invece per una variabile di tipo ora è possibile inizializzarla all'ora di sistema con la keyword `inz(*sys)`.
Cfr. il paragrafo definizioni variabili – RPG.

Il codice operativo `TIME` restituisce l'ora di sistema se il campo risultato è di 6 cifre, oppure restituisce la data e ora di sistema se il campo risultato è di 12 o 14 cifre. Se i campi risultato sono di tipo data, ora o timestamp il codice operativo time restituisce il valore corrispondente.

`UDATE` e `*DATE` restituiscono la **data del job** rispettivamente di 6 e 8 cifre.

ATTENZIONE: con la "data del job" si intende la data in cui è iniziato il job. Spesso e volentieri le variabili *DATE o UDATE si usano impropriamente come data corrente. Se un job inizia prima di mezzanotte del 15/01/2019 e prosegue fino al giorno successivo, le variabili *DATE e UDATE conterranno sempre il valore 15/01/2019.

N.B. i campi *DATE e UDATE sono campi numerici e NON campi di tipo data. Per poterli utilizzare in operazioni di durata (ADDDUR, SUBDUR) occorre convertirli in campo di tipo data. Per esempio:

```
dcl-s DataCorrente date;

DataCorrente = %date(*date);
```

Il *formato* della data di **DATE, UDATE* è determinato dalla keyword DATEDIT nella specifica H (i formati possibili sono 3). Se non è specificata i default sono rispettivamente *USA e *MDY.

DATEDIT	formato UDATE	formato *DATE
*MDY	*MDY	*USA (mmddyyyy)
*DMY	*DMY	*EUR (ddmmyyyy)
*YMD	*YMD	*ISO (yyyymmdd)

Altre costanti figurative disponibili in RPG: `UMONTH`, `*MONTH`, `UDAY`, `*DAY`, `UYEAR`, `*YEAR`. Anche questi sono tutti campi numerici e non di tipo data.

Nella program status data structure (SDS) ci sono a disposizione:

data del job (8 A): posizioni 191-198. Data in cui il job è stato immesso nel sistema. Formato identico a *date

secolo (2, 0 S): posizioni 199-200. Primi due caratteri dell'anno di 4 cifre.

data del job (6, 0 S): posizioni 270-275. Data in cui il job è stato immesso nel sistema. Formato identico a udate

data di sistema (6, 0 S): posizioni 276-281. Data di sistema. Formato identico a udate.

ora di sistema (6, 0 S): posizioni 282-287. Formato hhmmss.

L'API QWCRSVAL restituisce i valori di sistema, tra cui data e ora correnti.

Le funzioni `%date()`, `%time()` e `%timestamp()` restituiscono rispettivamente la data o l'ora di sistema (non del job).

La funzione `%timestamp(*sys)` restituisce il timestamp corrente di sistema con parte frazionale dei secondi (solo le prime 3 cifre sono valorizzate). P.es. 2014-06-27-01.02.03.421000.
Se non si desidera la parte frazionale dei secondi si può usare la funzione `%timestamp(*sys : 0)`. P.es. 2014-06-27-01.02.03.

In SQL ci sono alcuni special registers che restituiscono data e ora correnti:
– `CURRENT DATE`: restituisce la data di sistema
– `CURRENT TIME`: restituisce l'ora di sistema in formato time
– `CURRENT TIMESTAMP(n)`: restituisce la data/ora di sistema in formato timestamp con la precisione della frazione di secondi determinata dal valore di n.

```
values(current date); --> 2019-11-17

values(current time); --> 00.59.30

values(current timestamp(6)); --> 2019-11-17 00:59:38.528262

values(current timestamp(12)); --> 2019-11-17 00:59:46.528674507812
```

In alternativa all'uso degli special register esistono anche altre funzioni SQL che restituiscono il medesimo risultato:
`CURDATE` –> CURRENT DATE
`CURTIME` –> CURRENT TIME
`NOW(n)` –> CURRENT TIMESTAMP(n). Il parametro n può assumere un valore da 1 a 12 per specificare

la precisione della porzione frazionale dei secondi. Il default è 6.Se non specificato il valore di default di n è 6.

Gli special registers o le altre funzioni se vengono utilizzati più di una volta nella stessa istruzione sono basati su una sola lettura del clock di sistema.

Si consiglia di utilizzare gli special register invece delle funzioni per garantire la massima portabilità dell'istruzione SQL.

Data/ora e valori nulli ▲

Dalla V3R7 è possibile gestire nel database campi con valori nulli.
Con i campi numerici (surrogati del tipo data) solitamente una data "nulla" viene memorizzata con valore zero.
Ma se si usa nel database un campo data "vero" esso deve contenere una data valida. Il valore zero non è una data valida (si riceverebbe errore RNQ0112).
Quindi per lasciare un campo data vuoto bisogna o inizializzarlo per convenzione a una data fittizia che non sarà mai una data reale dei dati contenuti nel database (p.es. 0001-01-01 oppure 1900-01-01 o ...) oppure utilizzare i valori null.
Per consentire l'inserimento del valore null in un campo del database bisogna aggiungere nelle keyword delle DDS `ALWNULL`, oppure se si usa SQL per creare la tabella non bisogna specificare "`not null`". Per default un campo di una tabella creata con SQL consente l'inserimento di valori null.
Quando si ordinano i record per un campo che può contenere valori null, il valore null è l'ultimo (ovvero il valore massimo).

Costanti figurative per il valore null:
– nelle DDS: `*NULL`
– in OPNQRYF: `%NULL`
– in RPG: `*null`

Per gestire i valori nulli in un programma **RPG** bisogna prima di tutto aggiungere nella specifica di controllo (H) la keyword `alwnull(*usrctl)`.

Tramite la funzione `%nullind` si può verificare se una variabile contiene null oppure impostare il suo valore a null.

```
Esempio controllo del contenuto:
       if %nullind(DataConsegna);
         // gestione data nulla
       else;
         // gestione data valorizzata
       endif;

Esempio impostazione campo data a nullo
```

```
eval %nullind(DataConsegna) = *on;
```

Una variabile data/ora settata a null viene stampata o visualizzata a video come 0001-01-01 (valore di default); è necessario valorizzare il campo nel printer o display file solo se l'indicatore del valore nullo è *off. Cfr. anche il par. display e printer files per l'utilizzo della keyword MAPVAL.

Fuso orario ▲

Parlando di time o timestamp può essere importante conoscere il fuso orario a cui si riferiscono. Oppure può essere necessario convertire un orario da un fuso orario ad un altro.

Esistono vari metodi per conoscere la differenza rispetto a UTC (Universal Time Coordinated, conosciuta anche come GMT).
Lo special register SQL CURRENT TIMEZONE restituisce la differenza tra UTC e l'orario locale del server. Il risultato è espresso come una durata ovvero un numero nel quale i primi due digits sono il numero di ore e i successivi due digits il numero di minuti e gli ultimi due sono il numero dei secondi. Il numero di ore è compreso tra -24 e +24. Sottraendo CURRENT TIMEZONE all'orario locale si ottiene l'orario UTC.

Globalizzazione ▲

Le **costanti descrittive** di nomi di giorni, mesi o altro relativo a date/ore sono memorizzate nei seguenti messaggi del file QCPFMSG:

CPX9035: meridian indicator AM/PM

CPX9034: nome del giorno

CPX9039: nome del giorno abbreviato

CPX3BC0: nome del mese

CPX8601: nome del mese abbreviato

Bibliografia

* **ILE RPG Reference 7.3** SC09-2508-10, 2016,
https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/rzasd/sc092508.pdf?view=kc
* **Database DB2 for i SQL reference 7.3**,
https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/db2/rbafzpdf.pdf?view=kc
* Manuale corso OE85IT "Da RPG/400 a RPG IV", 2005
* Redbook SG24-7214-01 "Getting Started with DB2 Web Query for i",
http://www.redbooks.ibm.com/redbooks/pdfs/sg248378.pdf
* RPGIV for the RPG/400 Programmer Built-in functions, di Charles Massoglia, 1-giu-2004,

www.massoglia.com
* RPGIV for the RPG/400 Programmer Date, time and timestamp support, di Charles Massoglia, 1-giu-2004, www.massoglia.com
* Con la V5R2 aumenta il numero delle BIF RPG IV, di Bryan Meyers, articolo di iSeries News mag-2003
* RPG Academy: BIF Up your code! More on moving MOVE and MOVEL out of your code, di Rafael Victoria Pereira, articolo di MCPressOnline 4-mar-2015, https://www.mcpressonline.com/programming/rpg/rpg-academy-bif-up-your-code-more-on-moving-move-and-movel-out-of-your-code
* Date arithmetic functions added to SQL, di Simon Hutchinson, 20-lug-2023, https://www.rpgpgm.com/2023/07/date-arithmetic-functions-added-to-sql.html
* Date con valori nulli, di David Robertson, articolo di News/400 mag-1999
* Determining the last Friday/first Monday in the month, di Simon Hutchinson, 7-ott-2015, https://www.rpgpgm.com/2015/10/determining-last-fridayfirst-monday-in.html
* Guru: Easy date difference, di Ted Holt, IT Jungle 14-gen-2019, https://www.itjungle.com/2019/01/14/guru-easy-date-difference/
* Easy way to convert date to words using SQL, di Simon Hutchinson, 21-dic-2016, https://www.rpgpgm.com/2016/12/easy-way-to-convert-date-to-words-using.html
* Everything you wanted to know about dates but never dared to ask, part 1, di Simon Hutchinson, 20-ott-2015, https://www.rpgpgm.com/2015/10/everything-you-wanted-to-know-about.html
* Everything you wanted to know about dates but never dared to ask, part 2, di Simon Hutchinson, 21-ott-2015, https://www.rpgpgm.com/2015/10/everything-you-wanted-to-know-about_21.html
* Everything you wanted to know about dates, FAQ, di Simon Hutchinson, 22-ott-2015
* La V5 potenzia l'RPG IV, di Gary Guthrie, mar-2002, iSeries News
* Semantics of UDATE and *DATE in IBM RPG/400 and ILE RPG, IBM Technical document n. 642099, 17-giu-2018, https://www.ibm.com/support/pages/semantics-udate-and-date-ibm-rpg400-and-ile-rpg
* Tipi dati standard in RPG IV, di Paul Conte, mag-2003, iSeries News
* Use RPG to find the day of the week, di Nick Litten, 30-nov-2017, https://www.nicklitten.com/use-rpg-to-find-the-day-of-the-week/
* Validate dates in RPG, di Simon Hutchinson, 20-set-2013, https://www.rpgpgm.com/2013/09/validating-dates-in-rpgle.html
* Whats New and Exciting in RPG, di Scott Klement, 2016, https://www.scottklement.com/presentations/Whats%20New%20and%20Exciting%20in%20RPG.pd
* What's new for RPG in 7.2, di Barbara Morris, 2015, https://www.oceanusergroup.org/assets/whats_new_in_rpg_for_7.2.pdf
* Conversione delle date con le API, di Peter Levy, mag-2011
* Guru: More Date And Time Conversions Using SQL, di Ted Holt, 26-mar-2018, ITJungle, https://www.itjungle.com/2018/03/26/guru-more-date-and-time-conversions-using-sql/
* Guru: Odds And Ends, di Ted Holt, 14-mag-2018,

ITJungle, https://www.itjungle.com/2018/05/14/guru-odds-and-ends/
* Extracting parts of date and time using a SQL function, di Simon Hutchinson, 24-ott-2018, https://www.rpgpgm.com/2018/10/extracting-parts-of-date-and-time-using.html
* More SQL and dates part II, di Simon Hutchinson, 18-set-2013, https://www.rpgpgm.com/2013/09/more-sql-and-dates-part-ii.html
* SQL NOW and playing with timestamps, di Simon Hutchinson, 17-ott-2018, https://www.rpgpgm.com/2018/10/sql-now-and-playing-with-timestamps.html