Implementation of Temporal Data @ SD Worx in DB2

Before V10…

Luc Vermeiren, 6 December 2012

sdworx
Result driven HR

Implementation of Temporal Data @ SD Worx in DB2

1. What is SD Worx?
2. Temporal data @ SD Worx
3. How we model business time
4. How we model system time
5. Facts and Figures

sdworx
Result driven HR

What is SD Worx?

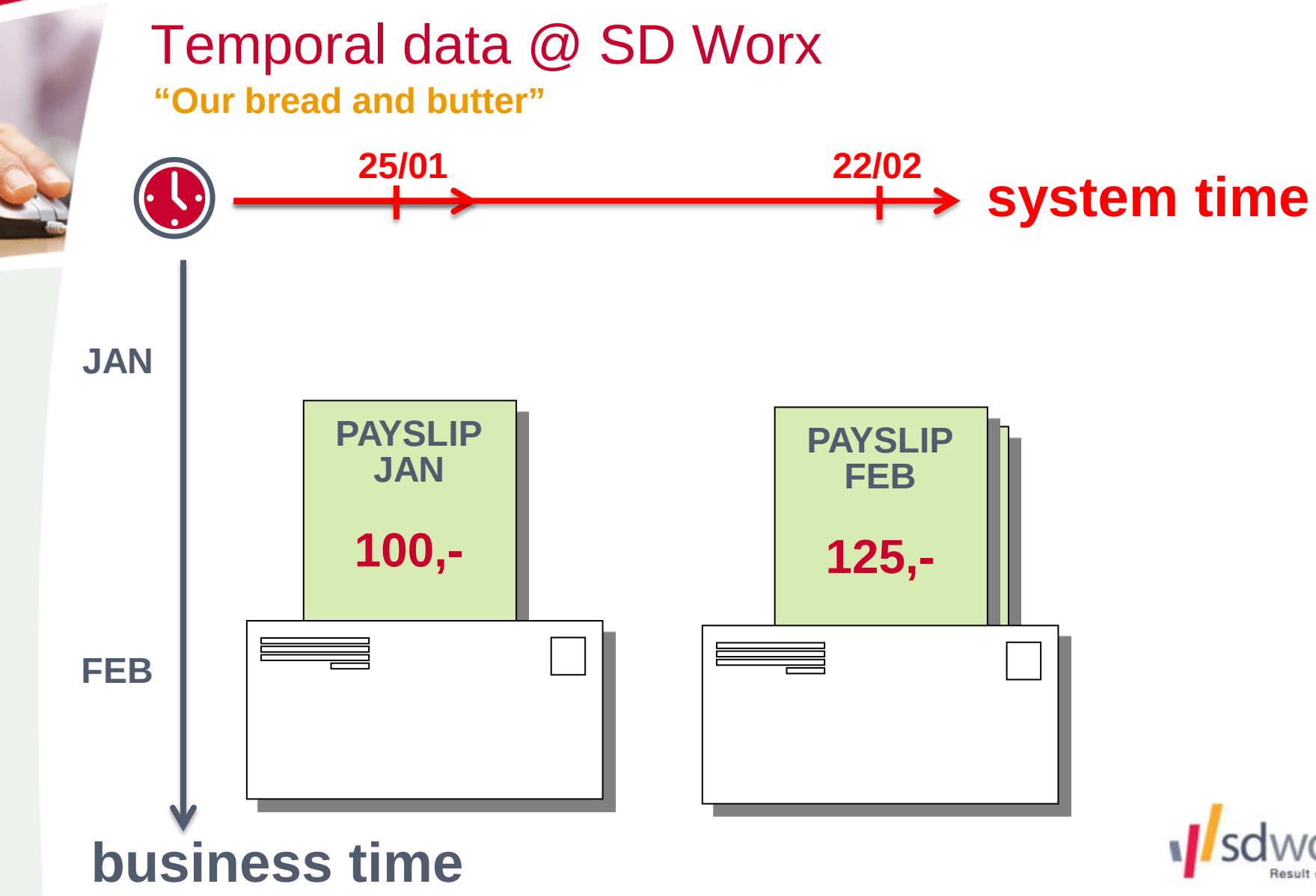
- Founded in 1945 as a company for “wage administration”
- Today: European HR Services Provider
  - 38 offices in 5 European countries
- Payroll still most important activity
  - In Belgium: 886.000 payslips
    For 45.500 clients (*)
    i.e. 29% market share (private sector)
- DB2: running V10 CM8 since October 2012

(*) December 2011

sdworx
Result driven HR

Temporal data @ SD Worx

"Our bread and butter"



Temporal data @ SD Worx

Business time versus System time

What is my salary?

	business time	system time	
→ Multiple answers	✓	✓	
→ Reference point in time is needed to select one answer	✓	✓	(1)
→ All answers are “currently” TRUE	✓	✗	
→ “Future”	✓	✗	

(1) for most processes this is understood to be CURRENT_TIMESTAMP

sdworx Result driven HR

Temporal data @ SD Worx

Functional types of business time

- Weakest: life span of an object
  - E.g.: “employee”: hire date → termination date
  - No evolution of values in business time
- “History with gaps”: there isn’t always a value
  - E.g.: benefits: the relationship between an employee and a benefit exists for (a) certain period(s) of time;
    or: “there are multiple relationships, each with its own life span”
- “Real history”: an attribute has a value at each point in time (during life span)
  - E.g.: salary

sdworx
Result driven HR

How we model **business time**

Table design: one table

- Take the PK of the object (e.g. EMPNO)
- Add 2 columns (granularity of business time = one day):
  - START_DATE DATE NOT NULL
  - END_DATE DATE
- We allowed END_DATE to be NULL (for +∞)
  - So we have to use “unique index” instead of PK
- END_DATE = next START_DATE: **[** t_1, t_2 **[**
  - Life span: **[** t_1, t_2 **]** ***Be aware of the issue…!***
- Convention:
  - Earliest START_DATE = ‘1001-01-01’ (“-∞“)
  - Greatest END_DATE = NULL (or ‘2999-12-31’)
  - Logical start/end = life span of object

sdworx
Result driven HR

How we model **business time**

Process support

- None whatsoever
  - No triggers, no automatic updates, no guards against overlaps, …
- All processes have to be “business-time aware”
- Reason: design focused on interactive applications
  - No “blind” updates
  - Time periods are presented to the user
  - User updates them “in memory”: difficult work is done here
  - Saving to the database: after the user clicks ‘Save’.

sdworx
Result driven HR

How we model **system time**

Table design: two tables

- Two tables
  - Only “current” values: for processes that don’t have to know system time
  - **All** values (including current): to avoid UNION
- “History table”:
  - Copy of base table
  - Add 2 columns to PK: START_TS and END_TS (**TIMESTAMP NOT NULL**) (in bi-temporal tables PK already contains START_DATE / END_DATE)
  - START_TS: based on MODIF_TS of base table
  - END_TS = next START_TS : **[** t_1, t_2 **[**
  - +∞ = ‘2999-12-31-00.00.00’

sdworx
Result driven HR

How we model **system time**

"History table" is maintained by triggers

- Base table has 4 triggers:
  - BEFORE UPDATE: ensure MODIF_TS goes up
  - AFTER INSERT: insert new row in "history table"
  - AFTER UPDATE: - update END_TS in "current row"
    - insert new row
  - AFTER DELETE: update last END_TS with current timestamp
- Bypass mechanism:
  - AFTER UPDATE trigger checks special register USER
  - When USER = …, trigger does nothing
  - Only for updates! (e.g. for mass updates of new columns)
- Approx. 600 "history tables" → 2400 triggers

sdworx
Result driven HR

How we model **system time**

Improved delete-processing: *history table* example

PK	START_TS	END_TS	MODIF_USER	MODIF_TS	COL_1
25	T1	T2	MARY	T1	A
		T3			

Extra row

1. At T1, user MARY created row ’25’ with value ‘A’.
2. At T2, user JOHN changed ‘A’ into ‘B’.
3. At T3, user MARY deleted the row.
   - table has to be read **[** T1,T2 **[** → **[** T3,T3 **[** is a “quantum row”
   - select “at T3”: row doesn’t exist

How we model **system time**

Improved delete-processing

- Why ? 2 delete issues: 1) END_TS is always current timestamp
  2) Who deleted?
- How? Programs delete through update-statement:

```
UPDATE BASE_TABLE
SET   CREATE_USER = '+DELETE+',   /* to indicate intention to delete */
      MODIF_TS = :WS_MODIF_TS,    /* timestamp of delete */
      MODIF_USER = :WS_USER       /* user who deletes */
WHERE …
```

- AFTER UPDATE trigger : (see next slide)

How we model **system time**

Improved delete-processing: trigger example

```
CREATE TRIGGER GDB2PSCA.TUSC3065
AFTER UPDATE ON GDB2PSCA.AUTR_ROL
REFERENCING OLD AS OLD NEW AS NEW
FOR EACH ROW MODE DB2SQL
WHEN (NEW.CRT_USER = '+DELETE+')
BEGIN ATOMIC
UPDATE GDB2PSCA.AUTR_ROL_TGV SET END_DT= NEW.LTS_WZG_DT
WHERE AUTR_ID = OLD.AUTR_ID AND BGN_DT = OLD.LTS_WZG_DT
AND END_DT = '2999-12-31-00.00.00.000000' ;
INSERT INTO GDB2PSCA.AUTR_ROL_TGV (AUTR_ID, BGN_DT, END_DT,
ROL_STTS, CRT_USER, CRT_DT, USRLW, LTS_WZG_DT, EIG_SD_PRS_ID,
ROL_NR) VALUES(NEW.AUTR_ID, NEW.LTS_WZG_DT, NEW.LTS_WZG_DT,
NEW. ROL_STTS, NEW.CRT_USER, NEW.CRT_DT, NEW.USRLW, NEW.
LTS_WZG_DT, NEW.EIG_SD_PRS_ID, NEW.ROL_NR) ;
DELETE FROM GDB2PSCA.AUTR_ROL WHERE AUTR_ID = NEW.AUTR_ID ;
END
```

sdworx
Result driven HR

Facts and Figures

Two important tables

Name	Owner	T	DB Name	TS Name	Cols	Rows
CONTRACT_DAG	GDB2PIVA	T	DBIV0001	TSIV0001	13	1.584.172.308
CONTRACT_DAG_TGV	GDB2PIVA	T	DBIV0001	TSIV0002	14	2.807.607.232
CONTRACT_HST	GDB2PMUA	T	DBMU0005	TSMU0469	59	40.187.099
CONTRACT_HST_TGV	GDB2PMUA	T	DBMU0005	TSMU0470	68	157.109.624

sdworx
Result driven HR

Facts and Figures

Overall number of tables

	DB A	DB B	DB C	TOTAL
non temp	1032	733	713	2478
with system time	186	163	160	509
with business time	23	21	20	64
bi-temporal	32	29	29	90
"history" for sys.time	186	163	160	509
"history" for bi-temp	32	29	29	90
TOTAL	1491	1138	1111	3740

sdworx
Result driven HR

Facts and Figures

Overall number of tables - percentage

	DB A	DB B	DB C	TOTAL
non temp	69%	64%	65%	66%
with system time	12%	14%	14%	14%
with business time	2%	2%	2%	2%
bi-temporal	2%	3%	3%	2%
"history" for sys.time	12%	14%	14%	14%
"history" for bi-temp	2%	3%	3%	2%
TOTAL	100%	100%	100%	100%

sdworx
Result driven HR

Facts and Figures

Overall number of rows

	DB A	DB B	DB C	TOTAL
bus.time / bi-temp	250.047.991	733.546	13.829.640	268.149.094
non temp / sys.time	13.684.584.029	30.082.624	357.610.220	14.068.738.956
"history" for sys.time	4.430.724.127	7.994.737	228.003.373	4.666.722.237
"history" for bi-temp	458.211.669	575.742	38.101.840	496.889.251
TOTAL	18.823.567.816	39.386.649	637.545.073	19.500.499.538

sdworx
Result driven HR

Facts and Figures

Overall number of rows - percentages

	DB A	DB B	DB C	TOTAL
base bustime/bitemp	1%	2%	2%	1%
non temp / systime	73%	76%	56%	72%
"history" for systime	24%	20%	36%	24%
"history" for bitemp	2%	1%	6%	3%
TOTAL	100%	100%	100%	100%

sdworx
Result driven HR

Facts and Figures

CPU usage (e.g. 30 min interval)

	PROGRAM	TYPE	CONTOKEN	PLCNT	SQL	TIMEPCT	CPUPCT	INDB2_TIME
_	WNA05310	PKGE	19504E581FB0684A	1	3887660	55.89%	71.98%	13:50.431462
_	**TIIV0001**	**TRIG**	**1935B88C06698026**	**1**	**3283541**	**21.31%**	**15.46%**	**05:16.686134**
_	TIIV0021	TRIG	1935B88B1D28E832	1	108272	.98%	1.66%	00:14.584488
_	SA$CDB	PKGE	18B42D2509E6CEC8	1	33167	.05%	.22%	00:00.804622
_	WNA26010	PKGE	194AA0181DD88148	1	33084	.83%	1.09%	00:12.440054
_	WNB31910	PKGE	194BE5AF1F1233E6	1	19175	1.12%	1.94%	00:16.644486
_	ADB2REM	PKGE	19069AA5020B581E	1	14607	.17%	.07%	00:02.534951
_	TIMU0884	TRIG	1935B89006075CD0	1	14568	.20%	.25%	00:03.055186
_	ALATRA10	PROC	18A66BFF032B702A	1	10336	.00%	.02%	00:00.139040
_	TUMU1627	TRIG	1935B89108AD3934	2	9534	.49%	.66%	00:07.412054
_	TUMU0627	TRIG	1935B891084E4C34	2	4767	.09%	.17%	00:01.403958
_	SYSSH200	PKGE	5359534C564C3031	1	4759	2.77%	2.77%	00:41.262012

sdworx
Result driven HR

Thank you!

- Luc.Vermeiren@sdworx.com

sdworx
Result driven HR