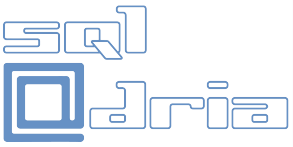
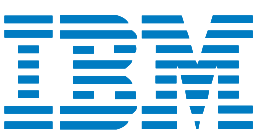


SQLADRIA SEMINAR

Opatija, 24 - 25 November 2011

Sponsored by



SQLAdria Seminar
SEMINAR: DB2 for z/OS V10
Speaker: Mike Bracey

DAY 01 - Thursday, November 24th 2011

Time	Topic	
10:00 – 10:15	Introduction	
10:15 – 11:00	New Core Features and Enhancements	Presentation
11:00 – 11:15	Break	
11:15 – 12:30	New Core Features and Enhancements	Hands on Lab
12:30 – 13:30	Launch	
13:30 – 14:00	New Core Features and Enhancements - continued	Hands on Lab
14:00 – 14:45	Improved Performance and Reduced CPU	Presentation
14:45 – 15:00	Tea break	
15:00 – 17:00	Improved Performance and Reduced CPU	Hands on Lab
17:00 – 18:00	Hand on Lab continuation - optional	Hands on Lab

DAY 02 – Friday, November 25th 2011

Time	Topic	
09:00 – 09:45	Improve Development Productivity	Presentation
09:45 – 11:00	Improve Development Productivity	Hands on Lab
11:00 – 11:15	Break	
11:15 – 12:00	Improve Development Productivity - continued	Hands on Lab
12:00 – 13:00	Open Forum and Close	

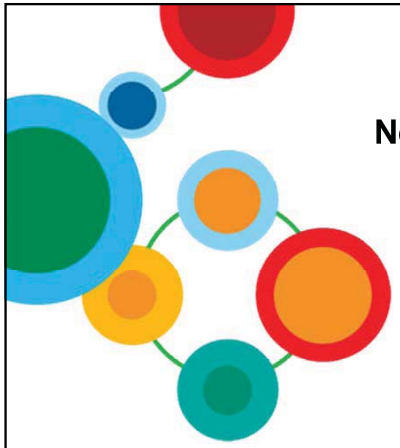
Sponsored by:

IBM

Opatija, 24th and 25th November, 2011
Hotel Ambasador

Participants of the SQLAdria Seminar
Opatija 24th - 25th November 2011

	Name	Surname	Company	E-mail
1	Jana	Perko	Informatika d.d.	jana.perko@informatika.si
2	Damir	Orlić	Informatika d.d.	damir.orlic@informatika.si
3	Borut	Komac	Informatika d.d.	borut.komac@informatika.si
4	Matija	Plavša	Informatika d.d.	matija.plavsa@informatika.si
5	Uroš	Lorbek	Informatika d.d.	uros.lorbek@informatika.si
6	Barbara	Rečnik	Informatika d.d.	barbara.recnik@informatika.si
7	Mojca	Muhič	Informatika d.d.	mojca.muhic@informatika.si
8	Karmen	Kamenečki	Informatika d.d.	karmen.kristan-kamenecki@informatika.si
9	Mateja	Janković	Informatika d.d.	mateja.jankovic@informatika.si
10	Sonja	Klančnik	ZZZS Slovenija	sonja.klancnik@zzzs.si
11	Mateja	Ošina	ZZZS Slovenija	mateja.osina@zzzs.si
12	Tatjana	Šumec	ZZZS Slovenija	tatjana.sumec@zzzs.si
13	Aljaž	Šepec	ZPIZ	aljaz.sepec@zpiz.si
14	Jure	Listar	ZPIZ	jure.listar@zpiz.si
15	Uroš	Kovšca	ZPIZ	uros.kovsca@zpiz.si
16	Tomaž	Kodrič	ZPIZ	tomaz.kodric@zpiz.si
17	Julija	Bojc	ZPIZ	julijana.bojc@zpiz.si
18	Željko	Blagaić	HZMO	zeljko.blagaic@mirovinsko.hr
19	Goran	Petrinjac	HZMO	goran.petrinjac@mirovinsko.hr
20	Iva	Mesarić	HZMO	iva.mesaric@mirovinsko.hr
21	Maja	Adam	Zagrebačka banka d.d.	maja.adam@unicreditgroup.zaba.hr
22	Sven	Lovrenčić	Zagrebačka banka d.d.	sven.lovrencic@unicreditgroup.zaba.hr
23	Mirna	Kos	Zagrebačka banka d.d.	mirna.kos@unicreditgroup.zaba.hr
24	Stanislav	Berdajs	NLB	stanislav.berdajs@nlb.si
25	Roman	Bučar	NLB	roman.bucar@nlb.si
26	Gorazd	Jereb	NLB	gorazd.jereb@nlb.si
27	Simon	Sirc	NLB	simon.sirc@nlb.si
28	Franci	Drnovšek	NLB	franci.drnovsek@nlb.si
29	Goran	Bavčar	NLB	goran.bavcar@nlb.si
30	Hana	Marić	APIS IT	hana.maric@apis-it.hr
31	Neven	Brezac	APIS IT	neven.brezac@apis-it.hr
32	Mario	Tičinović	FINA	mario.ticinovic@fina.hr
33	Branko	Horvat	FINA	branko.horvat@fina.hr
34	Rok	Žigon	Elektro Ljubljana Slovenija	rok.zigon@elektro-ljubljana.si
35	Marko	Žagar	Elektro Ljubljana Slovenija	marko.zagar@elektro-ljubljana.si
36	Peter	Pavković	IBM	peter.pavkovic@si.ibm.com
37	Mike	Bracey	IBM	mike_bracey@uk.ibm.com
38	Ivana	Martać	Ministarstvo finansija, uprava carina Srbije	martaci@carina.rs
39	Aleksandra	Skoko-Despinic	Ministarstvo finansija, uprava carina Srbije	skokoa@carina.rs
40	Snežana	Naumović	Ministarstvo finansija, uprava carina Srbije	naumovics@carina.rs
41	Željen	Stanić	CA	zeljen.stanic@ca.com
42	Olivera	Stanić	CTK SQLAdria	ostanic@ctk-rijeka.hr



DB2 10 for z/OS New Features and Functions

SQL Adria
Opatija
Croatia
24th and 25th November 2011

Mike Bracey, DB2 for z/OS, IBM UK,
braceym@uk.ibm.com
Peter Pavkovic, DB2 for z/OS, IBM Slovenia,
peter.pavkovic@si.ibm.com

Objectives

- **Learn the new features and functions of DB2 10 for z/OS**
  - **XML**
  - **Bi-temporal**
  - **And many more**
- **Gain insight via practical exercises**
- **Understand the implications for your application developers**

Agenda

- New Core Features and Enhancements in IBM DB2 10 for z/OS
  - Time Travel Query (Bitemporal Support)
  - Row and Column Access Control
  - SQLPL Extensions For Scalar UDF and Table UDF
  - XML Type Modifier
  - Sub-document update on XML document
- Improved Performance and Reduced CPU
  - Sub-document update on XML document
  - Binary XML
  - INLINE LOB
  - Dynamic Statement Cache Enhancement
  - Hash Access
  - Additional Non-Key Column in an Index (INCLUDE)

Agenda

- Improve Development Productivity
  - XQuery
  - XML Date and Time support
  - Extended Implicit CAST support
  - New OLAP Functions
  - Greater TIMESTAMP Precision
  - TIMESTAMP with TIME ZONE
- Learning pureXML and New SQL Features in DB2 10 for z/OS Using SPUFI and CLP (optional)
  - Learn XPath
  - Learn SQL/XML operators: XMLQUERY, XMLEXISTS, XMLTABLE, XMLCAST
  - Create and Utilize an XML Index
  - Learn XML Schema

Timetable - Thursday

Start	End	Topic	
10:00	10:15	Introduction	
10:15	11:00	New Core Features and Enhancements	Presentation
11:00	11:15	Break	
11:15	12:30	New Core Features and Enhancements	Hands on Lab
12:30	13:30	Lunch	
13:30	14:00	New Core Features and Enhancements - cont	Hands on Lab
14:00	14:45	Improved Performance and Reduced CPU	Presentation
14:45	15:00	Tea break	
15:00	17:00	Improved Performance and Reduced CPU	Hands on Lab
17:00	18:00	Hands on Lab continuation - optional	Hands on Lab

Timetable - Friday

Start	End	Topic	
09:00	09:45	Improve Development Productivity	Presentation
09:45	11:00	Improve Development Productivity	Hands on Lab
11:00	11:15	Break	
11:15	12:00	Improve Development Productivity - continued	Hands on Lab
12:00	13:00	Open Forum and Close	

Lab Cross Reference – by number

No	Title	Session	When
1	Lab setup for Labs 2 to 6	IDUG	Optional
2	XMLQuery	IDUG	Optional
3	XMLExist and XML index	IDUG	Optional
4	XMLTable	IDUG	Optional
5	XML Schema	IDUG	Optional
6	XML Type Modifier	IOD 1256	Thursday AM
7	Sub-document update on XML documents	IOD 1259	Thursday
8	XML Date and Time Support	IOD 1267	Friday AM
9	Time Travel Query (Bitemporal Support)	IOD 1256	Thursday AM
10	Row and Column Access Control	IOD 1256	Thursday AM
11	SQLPL Extensions for Scalar UDF and Table UDF	IOD 1256	Thursday AM
12	Implicit CAST support	IOD 1267	Friday AM
13	New OLAP functions	IOD 1267	Friday AM
14	Greater TIMESTAMP Precision	IOD 1267	Friday AM
15	TIMESTAMP with TIME ZONE	IOD 1267	Friday AM
16	INLINE LOB	IOD 1259	Thursday PM
17	Dynamic Statement Cache Enhancement	IOD 1259	Thursday PM
18	Hash Access	IOD 1259	Thursday PM
19	Additional Non-Key column in an Index (INCLUDE)	IOD 1259	Thursday PM
20	Binary XML Support	IOD 1259	Thursday PM
21	XQuery	IOD 1267	Friday AM

Lab Cross Reference – by session

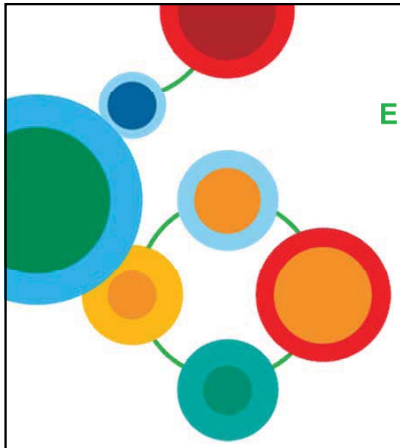
No	Title	Session	When
1	Lab setup for Labs 2 to 6	IDUG	Optional
2	XMLQuery	IDUG	Optional
3	XMLExist and XML index	IDUG	Optional
4	XMLTable	IDUG	Optional
5	XML Schema	IDUG	Optional
6	XML Type Modifier	IOD 1256	Thursday AM
9	Time Travel Query (Bitemporal Support)	IOD 1256	Thursday AM
10	Row and Column Access Control	IOD 1256	Thursday AM
11	SQLPL Extensions for Scalar UDF and Table UDF	IOD 1256	Thursday AM
7	Sub-document update on XML documents	IOD 1259	Thursday
16	INLINE LOB	IOD 1259	Thursday PM
17	Dynamic Statement Cache Enhancement	IOD 1259	Thursday PM
18	Hash Access	IOD 1259	Thursday PM
19	Additional Non-Key column in an Index (INCLUDE)	IOD 1259	Thursday PM
20	Binary XML Support	IOD 1259	Thursday PM
8	XML Date and Time Support	IOD 1267	Friday AM
12	Implicit CAST support	IOD 1267	Friday AM
13	New OLAP functions	IOD 1267	Friday AM
14	Greater TIMESTAMP Precision	IOD 1267	Friday AM
15	TIMESTAMP with TIME ZONE	IOD 1267	Friday AM
21	XQuery	IOD 1267	Friday AM

SQLADRIA SEMINAR

Opatija, 24 - 25 November 2011

New Core Features and Enhancements

Presentation



New Core Features and Enhancements in IBM DB2 10 for z/OS

Session Number 1256

Jane Man, IBM
Guogen Zhang, IBM
Steve Chen, IBM
Jerry Mukai, IBM
Mengchu Cai, IBM
Hao ZH Zhang, IBM

IBM Software
Information On Demand 2011

Please Note:

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Other DB2 V10 Labs..

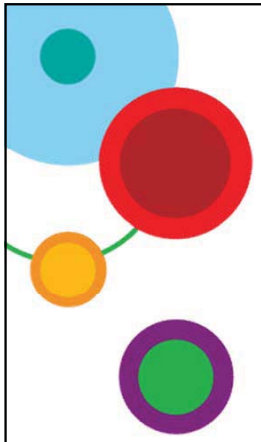
- **Session: IDZ-1259A New Features in IBM DB2 10 for z/OS II - Improved Performance and Reduced CPU**
  - Time: Mon, 24/Oct, 02:15 PM - 05:15 PM
  - Location: Mandalay Bay South Convention Center - Shorelines B Lab Room 1
- **Session: IDZ-1267A New Features in IBM DB2 10 for z/OS III - Improve Development Productivity**
  - Time: Tue, 25/Oct, 09:45 AM - 12:45 PM
  - Location: Mandalay Bay South Convention Center - Shorelines B Lab Room 9
- **Session: IDZ-1256A New Core Features and Enhancement in IBM DB2 10 for z/OS**
  Time: Wed, 26/Oct, 09:45 AM - 12:45 PM
  - Location: Mandalay Bay South Convention Center - Shorelines B Lab Room 9

Agenda

- Introduction (approx. 30 – 45 min)
  - Time Travel Query (Bitemporal Support)
  - Row and Column Access Control
  - SQLPL Extensions For Scalar UDF and Table UDF
  - XML Type Modifier
  - Sub-document update on XML document
- Lab (approx. 2 hrs)
  - Lab 0 – work together to ensure everyone has connectivity to z/OS machine
  - The rest of labs are on your own pace. There are no dependency among these labs and you can start with anyone you prefer.
  - Lab 9 – Time Travel Query (Bitemporal Support)

Agenda (cont'd)

- Lab 10 - Row and Column Access Control
- Lab 6 - XML Type Modifier
- Lab 7 - Sub-document update on XML documents
- Lab 11 - SQLPL Extensions for Scalar UDF and Table UDF



Time Travel Query (Bitemporal Support)

Bitemporal Support

- New concept of System_time and Business_time period
  - System_time captures DB2's creation and deletion of rows and automatically keeps historical versions of rows.
  - Business_time allows users to create their own validity period for a given row.
- Value to customers
  - meet compliance requirements : automatic propagation of old rows to a history table.
  - performs better than the home-grown solution.
  - easier to manage

Bitemporal Support – Example

```
CREATE TABLE  policy
(client              CHAR(4)          NOT NULL,
type                 CHAR(4)          NOT NULL,
copay                SMALLINT         NOT NULL,
eff_beg              DATE             NOT NULL,
eff_end              DATE             NOT NULL,
sys_start            TIMESTAMP(12) WITH TIME ZONE
                              NOT NULL IMPLICITLY HIDDEN
                              GENERATED ALWAYS AS ROW BEGIN,
sys_end              TIMESTAMP(12) WITH TIME ZONE
                              NOT NULL IMPLICITLY HIDDEN
                              GENERATED ALWAYS AS ROW END,
trans_id             TIMESTAMP(12) WITH TIME ZONE IMPLICITLY
                              HIDDEN
                     GENERATED  ALWAYS AS TRANSACTION START ID,
PERIOD BUSINESS_TIME(eff_beg, eff_end),
PERIOD SYSTEM_TIME(sys_start, sys_end));
```

Bitemporal Support – Example (cont)

```
CREATE TABLE  policy_hist LIKE policy;

ALTER TABLE policy
ADD VERSIONING USE HISTORY TABLE   policy_hist;

CREATE UNIQUE INDEX  ix_policy
ON policy (client, BUSINESS_TIME WITHOUT OVERLAPS);
```


Bitemporal Support – Example (cont)

Step	Actual Date	Activity
1	01/01/2004	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004 (future event).
2	09/01/2004	Customer called and changed to HMO as of today (present event)
3	03/01/2006	Copay increase to $15 starting 01/01/2007 (future event)
4	06/01/2008	Cancel policy as of today (present event)
5	09/01/2008	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007 (past event)

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
1	01/01/2004	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004 (future event).

```
INSERT INTO policy VALUES
 ('C882', 'PPO',10,'01/01/2004','12/31/9999');
```


Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	12/31/9999	2004-01-01…	9999-12-30…

Table: policy

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
2	09/01/2004	Customer called and changed to HMO as of today (present event)

```
UPDATE policy FOR PORTION OF BUSINESS_TIME
  FROM '09/01/2004' TO '12/31/9999'
  SET type = 'HMO'
    WHERE client = 'C882';
```


Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01…	9999-12-30…
C882	HMO	10	09/01/2004	12/31/9999	2004-09-01…	9999-12-30…

Table: policy

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
3	03/01/2006	Copay increase to $15 starting 01/01/2007 (future event)

```
UPDATE policy FOR PORTION OF BUSINESS_TIME
  FROM '01/01/2007' TO '12/31/9999'
  SET copay = 15
    WHERE client = 'C882';
```


Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01…	9999-12-30…
C882	HMO	10	09/01/2004	01/01/2007	2006-03-01…	9999-12-30…
C882	HMO	15	01/01/2007	12/31/9999	2006-03-01…	9999-12-30…

Table: policy

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
4	06/01/2008	Cancel policy as of today (present event)

```
UPDATE policy
  SET eff_end = '06/01/2008'
    WHERE client = 'C882'
     AND eff_end = '12/31/9999';
```

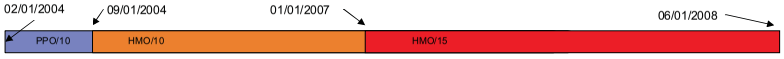

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
5	09/01/2008	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007 (past event)

```
UPDATE policy FOR PORTION OF BUSINESS_TIME
  FROM '05/01/2006' TO '10/01/2007'
  SET type = 'POS'
    WHERE client = 'C882';
```


Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007



client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01…	9999-12-30…
C882	HMO	10	09/01/2004	01/01/2007	2006-03-01…	9999-12-30…
C882	HMO	15	01/01/2007	06/01/2008	2008-06-01…	9999-12-30…

Table: policy

Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01…	9999-12-30…
C882	HMO	10	09/01/2004	05/01/2006	2008-09-01…	9999-12-30…
C882	POS	10	05/01/2006	01/01/2007	2008-09-01…	9999-12-30…
C882	POS	15	01/01/2007	10/01/2007	2008-09-01…	9999-12-30…
C882	HMO	15	10/01/2007	06/01/2008	2008-09-01…	9999-12-30…

Table: policy

Bitemporal Support – Example (cont)

Question: On 09/15/2008, client calls and complains. Client saw an out-of-network specialist on 07/01/2007.
Claims dept. denied client's claim due to HMO coverage for this visit on 07/15/2007. Client demands reimbursement.

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01…	9999-12-30…
C882	HMO	10	09/01/2004	05/01/2006	2008-09-01…	9999-12-30…
C882	POS	10	05/01/2006	01/01/2007	2008-09-01…	9999-12-30…
C882	POS	15	01/01/2007	10/01/2007	2008-09-01…	9999-12-30…
C882	HMO	15	10/01/2007	06/01/2008	2008-09-01…	9999-12-30…

Table: policy

Bitemporal Support – Example (cont)

Question: Did our claims department make an error denying the client's claim on 07/15/2007?

To answer this: Need historical data to see what claims department saw on 07/15/2007. Thus, the need for a bitemporal solution.

Bitemporal Support – Example (cont)

```
SELECT * FROM POLICY
  FOR BUSINESS_TIME AS OF '2007-07-01'
    WHERE CLIENT='C882';
```

Answer: Customer has "POS", so should be reimbursed.

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01…	9999-12-30…
C882	HMO	10	09/01/2004	05/01/2006	2008-09-01…	9999-12-30…
C882	POS	10	05/01/2006	01/01/2007	2008-09-01…	9999-12-30…
C882	POS	15	01/01/2007	10/01/2007	2008-09-01…	9999-12-30…
C882	HMO	15	10/01/2007	06/01/2008	2008-09-01…	9999-12-30…

Table: policy

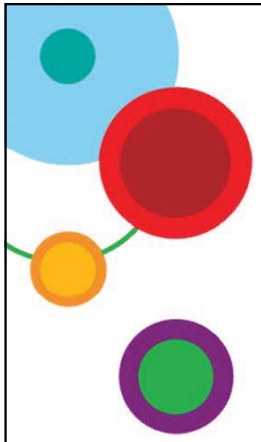
Bitemporal Support – Example (cont)

```
SELECT * FROM policy
  FOR BUSINESS_TIME AS OF '2007-07-01'
  FOR SYSTEM_TIME AS OF '2007-07-15'
    WHERE client='C882';
```

Answer: At 07/15/2007, claims saw 'HMO'.

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	12/31/9999	2004-01-01…	2004-09-01…
C882	HMO	10	09/01/2004	12/31/9999	2004-09-01…	2006-03-01…
C882	HMO	15	01/01/2007	12/31/9999	2006-03-01…	2008-06-01…
C882	HMO	10	09/01/2004	01/01/2007	2008-06-01…	2008-09-01…
C882	HMO	15	01/01/2007	06/01/2008	2008-06-01…	2008-09-01…

Table: policy_hist



Row and Column Access Control

Concerns about Database Security

- Separation of duties
  - Database administrators such as DBADM can access sensitive data
  - No designated authority such as SECADM to manage security policies
- Granularity of privilege model
  - Privileges are granted at database object level
  - Difficult to protect personal and sensitive information within the object
  - Cannot easily comply with data protection laws such as HIPPA, GLBA
- Overloading applications with security logic
  - Can be bypassed by malicious users
  - Hampers the ability to use ad-hoc query tools, report generation tools
- Alternative views for each group of users
  - Can be bypassed by malicious users
  - View's updatability may not correctly reflect security policies
- Evolution of security policies
  - Difficult to manage and maintain

Solution : Row and Column Access Control

- Tighter security
  - Data-centric within database
  - No backdoor to bypass views or applications
  - More granularity via row permissions and column masks
  - Separation of duties
  - Designated SECADM authority
  - No authority including DBADM is exempted from the control
  - Relief for the evolution of security policies
- Easy to implement
  - More flexibility via SQL
  - Separation of security logic and application logic

Row and Column Access Control – new terminology

- Row Permission
  - a database object that expresses a row access control rule for a table
  - contains a rule in the form of an SQL search condition that describes to which rows the users have access
  - applied by DB2 after the checking of table privileges (e.g. SELECT, INSERT privilege, etc.)

Row and Column Access Control – new terminology (cont'd)

- Column Mask
  - a database object that expresses a column access control rule for a specific column in a table
  - contains a rule in the form of an SQL CASE expression that describes to what masked value returned for a column value the users have access
  - applied by DB2 after the checking of table privileges (e.g. SELECT, UPDATE privilege, etc.)

Row and Column Access Control – Concept

Think a Decomposed View

```
CREATE VIEW EMPOLYEE_VIEW AS
    SELECT (CASE ... END) SSN, (CASE ... END) SALARY
        FROM EMPLOYEE
        WHERE STATE = 'CA' AND
              LASTNAME = 'SMITH' AND
              BDATE > '1970-01-01'
```

- Row Permission
  - The WHERE clause of the EMPLOYEE_VIEW
- Column Mask
  - The outermost SELECT clause in the EMPLOYEE_VIEW definition

Row and Column Access Control – Examples

- **Row Permission**

```
CREATE PERMISSION EMPLOYEE_PERMISSION ON EMPLOYEE
    FOR ROWS WHERE STATE = 'CA' AND
                   LASTNAME = 'SMITH' AND
                   BDATE > '1970-01-01'
ENFORCED FOR ALL ACCESS ENABLE ;
```

- **Column Mask**

```
CREATE MASK SSN_MASK ON EMPLOYEE
    FOR COLUMN SSN RETURN
        CASE WHEN SESSION_USER = 'SMITH'
             THEN SSN
             ELSE CHAR('XXX-XX-') || SUBSTR(SSN,8,4)
        END
ENABLE;
```

- **SELECT SSN FROM EMPLOYEE;**

Who can see what??

- New Built-in Functions
  - VERIFY_GROUP_FOR_USER
    - Verify primary and secondary authorization IDs

```
WHERE
   VERIFY_GROUP_FOR_USER (SESSION_USER, 'MGR', 'PAYROLL') = 1
```

  - VERIFY_TRUSTED_CONTEXT_ROLE_FOR_USER
    - Verify primary authorization ID's role

```
WHERE
   VERIFY_TRUSTED_CONTEXT_ROLE_FOR_USER (SESSION_USER,
                                         'MGR', 'PAYROLL') = 1
```

Activate Row and Column Access Control

- Activated by SECADM authority only
  - Job card … ,USER=SECADM, …
- Invalidate packages and cached statements
- Row permissions and column masks become effective in DML
  - All row permissions are merged to filter out rows
    - Multiple row permissions are connected with 'OR'
  - All column masks are applied to mask output columns
- Generate default row permission 1 = 0 if activated for row

```
ALTER TABLE table-name
   ACTIVATE ROW       ACCESS CONTROL
   ACTIVATE COLUMN  ACCESS CONTROL;

ALTER TABLE table-name
   ACTIVATE ROW       ACCESS CONTROL;
```


Deactivate Row and Column Access Control

- Deactivated by SECADM authority only
  - Job card … ,USER=SECADM, …
- Invalidate packages and cached statements
- Row permissions and column masks become ineffective in DML
  - Remove default row permission 1 = 0 if deactivated for row
  - Open all access to the table

```
ALTER TABLE table-name
   DEACTIVATE ROW       ACCESS CONTROL
   DEACTIVATE COLUMN  ACCESS CONTROL;

ALTER TABLE table-name
   DEACTIVATE ROW       ACCESS CONTROL;
```


Another use case scenario

- CUSTOMER table contains information about the bank customers
- INTERNAL_INFO table contains information about bank employees
- The bank has a security policy for access to customer information
- Table: CUSTOMER

Account	Name	Income	Branch
1111-2222-3333-4444	Alice	22,000	A
2222-3333-4444-5555	Bob	71,000	B
3333-4444-5555-6666	Carl	123,000	B
4444-5555-6666-7777	David	172,000	C

Another Use Case Scenario (Cont'd)

ROW PERMISSION FOR TELLERS

```
CREATE PERMISSION TELLER_ROW_ACCESS
ON CUSTOMER FOR ROWS WHERE
   VERIFY_ROLE_FOR_USER (USER, 'TELLER') = 1 AND
   BRANCH = (SELECT HOME_BRANCH FROM INTERNAL_INFO WHERE EMP_ID = USER)
ENFORCED FOR ALL ACCESS ENABLE
```

ROW PERMISSION FOR CSR and TELEMARKETER

```
CREATE PERMISSION CSR_ROW_ACCESS
ON CUSTOMER FOR ROWS WHERE
   VERIFY_ROLE_FOR_USER (USER, 'CSR') = 1 OR
   VERIFY_ROLE_FOR_USER (USER, 'TELEMARKETER') = 1
ENFORCED FOR ALL ACCESS ENABLE
```

Another Use Case Scenario (Cont'd)

COLUMN MASK FOR ACCOUNT NUMBER

```
CREATE  MASK  ACCOUNT_COL_MASK
ON  CUSTOMER FOR  COLUMN ACCOUNT  RETURN
   CASE WHEN (VERIFY_ROLE_FOR_USER (USER, 'TELEMARKETER')  =  1)
                THEN  'xxxx-xxxx-xxxx-' || SUBSTR(ACCOUNT,13,4)
         ELSE ACCOUNT
    END ENABLE
```

ACTIVATE ROW AND COLUMN LEVEL ACCESS CONTROL

```
ALTER TABLE CUSTOMER
  ACTIVATE ROW LEVEL ACCESS CONTROL
  ACTIVATE COLUMN LEVEL ACCESS CONTROL
```


Another Use Case Scenario Cont'd

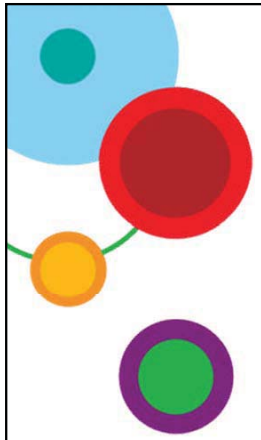
- Teller Amy issues the following query

```
SELECT ACCOUNT, NAME, BRANCH FROM CUSTOMER
```

Account	Name	Income	Branch
2222-3333-4444-5555	Bob	71,000	B
3333-4444-5555-6666	Carl	123,000	B

- Telemarketer Pat issues the following same query

Account	Name	Income	Branch
XXXX-XXXX-XXXX-4444	Alice	22,000	A
XXXX-XXXX-XXXX-5555	Bob	71,000	B
XXXX-XXXX-XXXX-6666	Carl	123,000	B
XXXX-XXXX-XXXX-7777	David	172,000	C



SQLPL Extensions for Scalar UDF and Table UDF

SQL PL Extensions for Scalar UDF and Table UDF

SQL PL: SQL procedural language

- Set of control statements defined in the SQL Standard
  - **ISO/IEC FCD 9075-4:2003**, *Information technology - Database languages - SQL - Part 4: Persistent Stored Modules (SQL/PSM)*
- Native SQL procedures (V9)

➢ **Simplifies** the task of writing database applications

SQL PL Extensions for Scalar UDF and Table UDF

- DB2 9 for z/OS
  - No support for SQL table functions; only external table functions are supported
  - Scalar function support limited to single RETURN statement
- Extended in V10 to allow for use within:
  - SQL scalar functions
  - SQL table functions (minimal subset)

Table UDF

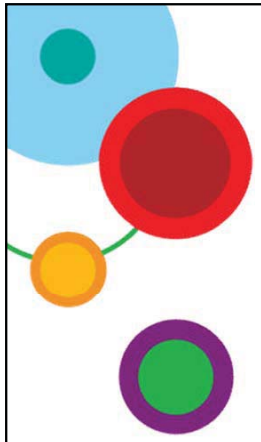
```
CREATE  FUNCTION  JTABLE (COLD_VALUE CHAR(9), T2_FLAG
    CHAR(1))
RETURNS TABLE (COLA INT, COLB INT, COLC INT)
LANGUAGE SQL
SPECIFIC DEPTINFO
NOT DETERMINISTIC
READS SQL DATA
RETURN
SELECT A.COLA, B.COLB, B.COLC
  FROM TABLE1 AS A
    LEFT OUTER JOIN
        TABLE2 AS B
      ON A.COL1 = B.COL1 AND  T2_FLAG = 'Y'
WHERE A.COLD = COLD_VALUE;
```

- function body specifies an SQL query that returns a result table
- result table is returned to the invoking statement

SQL Scalar Function

```
CREATE   FUNCTION   REVERSE  (INSTR   VARCHAR(4000))
    RETURNS VARCHAR (4000)
    DETERMINISTIC
    NO EXTERNAL ACTION
    CONTAINS SQL
BEGIN
  DECLARE REVSTR, RESTSTR VARCHAR(4000)   DEFAULT '';
  DECLARE LEN INT;
  IF INSTR IS NULL THEN
    RETURN NULL;
  END IF;
  SET RESTSTR = INSTR;
  SET LEN = LENGTH(INSTR);
  WHILE LEN > 0 DO
    SET REVSTR = SUBSTR(RESTSTR, 1, 1)
        CONCAT REVSTR;
    SET RESTSTR = SUBSTR(RESTSTR, 2, LEN - 1);
    SET LEN = LEN - 1;
  END WHILE;
  RETURN   REVSTR;
END
```

- Function body contains logic.
- If the input data is null the function simply returns null.
- Otherwise, the function reverses the order of the characters in the input string and returns the modified string to the invoking statement.



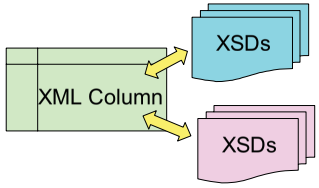
XML Type Modifier

Key XML Features in DB2 10

- **To ensure the quality & consistency of XML data inserted into a column**
  ➔ XML schema association with XML columns (a.k.a. XML column type modifier) and automatic schema validation for insert/update
- XML schema validation is CPU intensive, and with 50MB size limit.
  ➔ Using z/OS XML System Services, 100% zIIP/zAAP redirectable
- Native XML Date and Time important in business processing
  ➔ xs:date, xs:dateTime, and xs:dateTime support and XML index support
- **Update a piece of information in an XML document is common**
  ➔ Basic XML sub-document update
- Business logic standardization/performance with native SQL PL stored procedures and user defined functions
  ➔ XML support in SQL PL STP/UDF
- Performance enhancements, including binary XML between client and server.

XML Schemas as XML column type modifier

- Add XML schemas as column type modifier, DB2 enforces the conformance.
- Two ways to identify an XML schema (existing)
  - Schema name, or target namespace + optional schema location



Example

- CREATE TABLE T1 (X1 XML(XMLSCHEMA ID SYSXSR.schema1), X2 XML);
- ALTER TABLE T1 ALTER X2 SET DATA TYPE XML(XMLSCHEMA URI 'http://www.example.com/po' ELEMENT "purchaseorder")

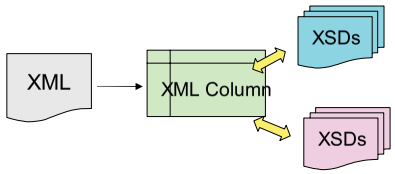
Automatic Validation

- Insert

```
INSERT INTO purchase_orders
 VALUES (:hv);
```

- Update

```
UPDATE purchase_orders
   SET content =:hv;
```



- Load performing validation
- If the source is already validated according to the same XML schema in the type modifier, it won't be revalidated again.
- Schemas evolve, multiple versions coexist
  - At insert/update time, choose one of them based on information from instance doc (target namespace, schema location).
  - If no schema location, with multiple schema matches, the latest will be chosen.

ALTER XML Type Modifier

- Always use ALTER TABLE T1 ALTER X2 SET DATA TYPE XML(…): whole set of schemas – DB2 figures out the delta
- CHECK DATA – validate docs or put invalid ones in exception table

Change from Type Modifier	To Type Modifier	XML Table Space Impact
No modifier	PO	Check pending
PO1	PO1, PO2	No
PO1	PO2	Check pending
PO1, PO2	PO1	Check pending
PO1	No modifier	No

Sub-document Update of XML documents

Sub-document Update

- No easy way to update parts of a document in V9
- Sub-document update with multi-versioning in DB2 10
- Only simple update: insert, replace, delete from XQuery update facility

Example

Increase premium by 10%

```
UPDATE POLICYTAB SET POLICY =
XMLMODIFY
( 'replace value of node /policy/premium with /policy/premium * 1.1')
WHERE POLICYID = '12345';
```

Add a new payment into payment record

```
UPDATE POLICYTAB SET PAYMENTS = XMLMODIFY
('insert node $n as last into /payments', :newpayment as "n")
WHERE POLICYID = :policyid;
```


Sub-document Update (cont'd)

- XMLMODIFY can only be used in RHS of UPDATE SET.
- One updater at a time for a document, concurrency control by the base table – row level locking, page level locking etc.
  - Document level lock to prevent UR reader
- Only changed rows in XML table are updated

Sub-document Update – Insert Expression (1 of 3)

Sample XML document:

```
<person>
  <firstName>Joe</firstName>
  <lastName>Smith</lastName>
  <nickName>Joey</nickName>
</person>
```

Sample insert statement:

```
update personinfo set info = xmlmodify('
  insert node $ins/ename
  after  /person/nickName', xmlparse(document '
      <ename>Joe.Smith@de.ibm.com</ename>') as  "ins") ;
```

Insert Operation	Resulting XML document
insert node $ins/ename into /person , XMLPARSE(document ' <ename>Joe.Smith@de.ibm.com</ename>') as "ins") (nondeterministic position)	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> <ename>Joe.Smith@de.ibm.com</ename> </person>
insert node $ins/ename as last into /person, XMLPARSE(document ' <ename>Joe.Smith@de.ibm.com</ename>') as "ins")	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> <ename>Joe.Smith@de.ibm.com</ename> </person>

Sub-document Update – Insert Expression (2 of 3)

Sample XML document:

```
<person>
  <firstName>Joe</firstName>
  <lastName>Smith</lastName>
  <nickName>Joey</nickName>
</person>
```

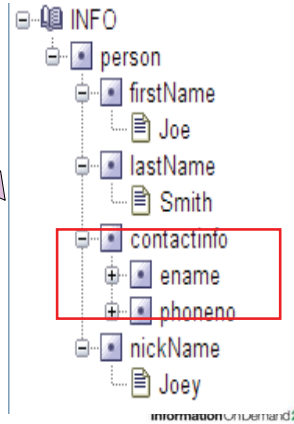
Insert Operation	Resulting XML document
Insert node $ins/ename **as first into** /person, XMLPARSE(document <ename>Joe.Smith@de.ibm.com</ename>') as "ins")	<person> **<ename>Joe.Smith@de.ibm.com</ename>** <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> </person>
Insert node $ins/ename **after** /person/nickName , XMLPARSE(document '<ename>Joe.Smith@de.ibm.com</ename>') as "ins")	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> **<ename>Joe.Smith@de.ibm.com</ename>** </person>
Insert node $ins/ename **before** /person/nickName , XMLPARSE(document ' <ename>Joe.Smith@de.ibm.com</ename>') as "ins")	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> **<ename>Joe.Smith@de.ibm.com</ename>** <nickName>Joey</nickName> </person>

Sub-document Update – Insert Expression (3 of 3)

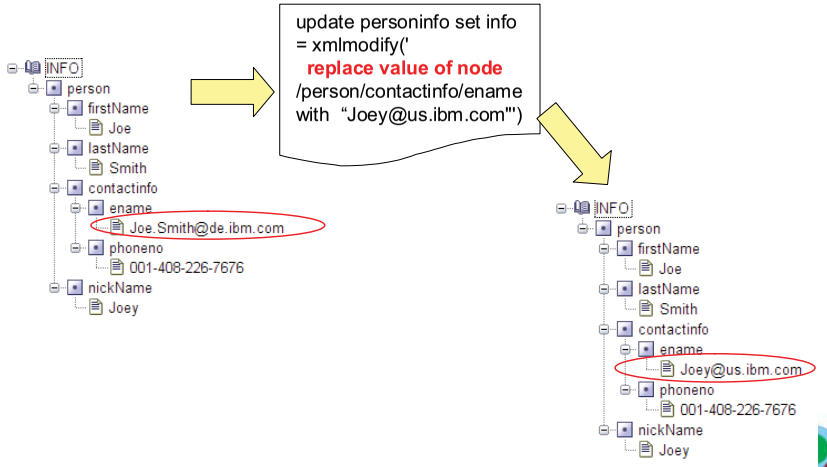
- Insertion of sequence of elements also possible
- Use same keywords as introduced before

```
<person>
  <firstName>Joe</firstName>
  <lastName>Smith</lastName>
  <nickName>Joey</nickName>
</person>
```

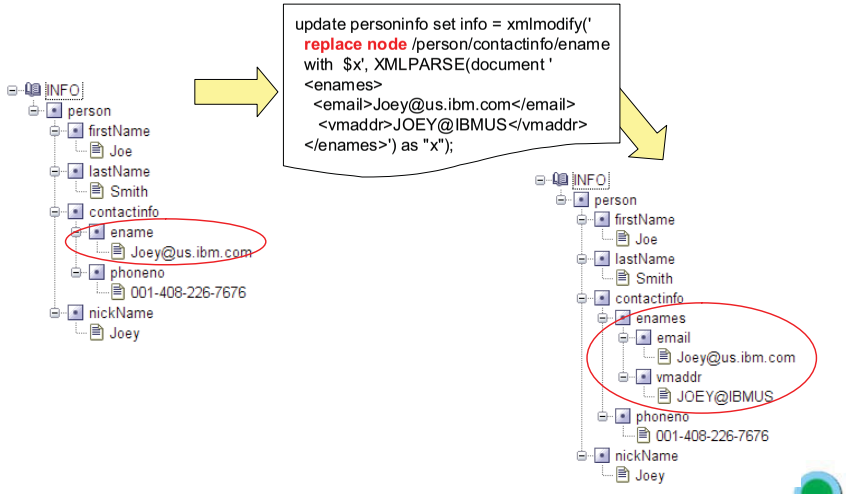
```
update personinfo set info = xmlmodify('
  insert node $ins/contactinfo
before /person/nickName',
xmlparse(document '
<contactinfo>
<ename>Joe.Smith@de.ibm.com</ename>
<phoneno>001-408-226-7676</phoneno>
</contactinfo>
') as "ins") ;
```



Sub-document Update – Replace Expression (1 of 2)

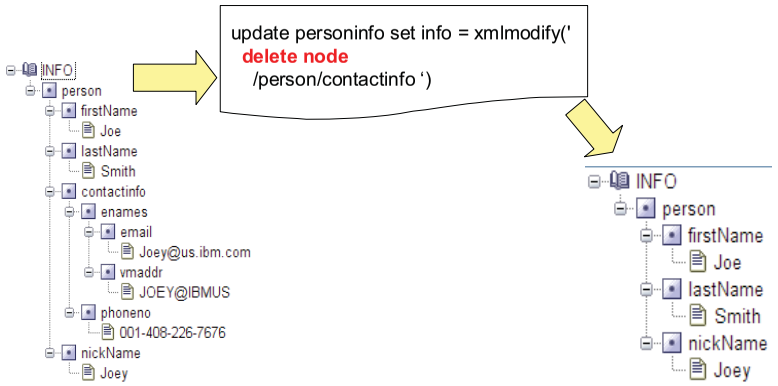


Sub-document Update – Replace Expression (2 of 2)



Sub-document Update – Delete Expression

- Delete can be used to delete nodes from a node sequence



An Exclusive Invitation for System z Attendees

ROCK THE MAINFRAME

at the

HOUSE OF BLUES

Music Hall

Wednesday, October 26th 7:00 pm - 10:00 pm

Enjoy a night of southern hospitality with cocktails and cajun hors d'oeuvres.
Keep the party rockin' by taking a turn on the Rock Band video game.
Join your colleagues, conference speakers and key members from your IBM System z team.

The House of Blues Music Hall is next door to the restaurant on the casino level across from the Mandalay Bay Hotel.

Wear your IOD badge and Z pin to get in

Thank You… Leveraging the Best of z!

"The benchmarks and analysis of functionality and performance have exceeded our expectations. So far our upgrades have gone smoothly and we are looking forward to completing our successful rollout of DB2 10"

–Verizon

"We had migrated five sub-systems to DB2 10 and have had no reported application issues running on this release to date."
–LabCorp

"The 'overall performance' in DB2 10 is better compared to DB2 9."

–HUK Coberg

[The Temporal Data] feature will drastically save developer time, test time … and improve business efficiency and effectiveness …
–Bankdata

Our regression tests showed performance improvements just by running the workload on a DB2 10 CM member…
–Dillards

Acknowledgements and Disclaimers:

Availability. References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS-IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

Communities

- **On-line communities, User Groups, Technical Forums, Blogs, Social networks, and more**
  - Find the community that interests you…
    - **Information Management** **ibm.com**/software/data/community
    - **Business Analytics** **ibm.com**/software/analytics/community
    - **Enterprise Content Management** **ibm.com**/software/data/content-management/usernet.html
- **IBM Champions**
  - Recognizing individuals who have made the most outstanding contributions to Information Management, Business Analytics, and Enterprise Content Management communities
    - **ibm.com**/champion

Thank You!
Your Feedback is Important to Us

- Access your personal session survey list and complete via SmartSite
  - Your smart phone or web browser at: iodsmartsite.com
  - Any SmartSite kiosk onsite
  - Each completed session survey increases your chance to win an Apple iPod Touch with daily drawing sponsored by Alliance Tech

SQLADRIA SEMINAR

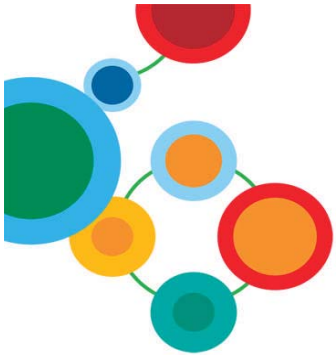
Opatija, 24 - 25 November 2011

New Core Features and Enhancements

Hands on Lab

Sponsored by

IBM



New Features in IBM DB2 10 for z/OS I – Core Features

Session Number 1256

Jane Man, IBM
Guogen Zhang, IBM
Steve Chen, IBM
Jerry Mukai, IBM
Mengchu Cai, IBM
Hao ZH Zhang, IBM

Contents

1. Introduction

1.1 Who are we?

We are from the DB2 development organization in IBM Silicon Valley Lab.

Jane Man
Advisory Software Engineer
janeman@us.ibm.com

Guogen Zhang
Distinguished Engineer
gzhang@us.ibm.com

Steve Chen
STSM
yschen@us.ibm.com

Mengchu Cai
Senior Software Engineer
mcai@us.ibm.com

Jerry Mukai
Senior Software Engineer
mukai@us.ibm.com

Hao ZH Zhang
Advisory Software Engineer
zhhao@cn.ibm.com

1.2 Objectives:

- Learn sub-document update on XML documents
- Learn XML type modifier
- Learn Bitemporal support
- Learn row permission and column masks
- Learn SQL PL extensions for scalar UDF and table UDFs

1.3 Suggested reading

- Introduction to pureXML in DB2 9 for z/OS

  ftp://ftp.software.ibm.com/software/data/db2zos/presentations/2007/misc/purexml.pdf

- Leveraging DB2 9 for z/OS pureXML Technology

  http://www.geocities.com/zhanggene/pub/Leveraging_DB29_for_zOS_whitepaper_v2.pdf

- DB2 Version 9.1 for z/OS XML Guide (SC18-9858)

  http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.xml/dsnxgk13.pdf?noframes=true

- DB2 Version 10 for z/OS SQL Reference (SC19-2983-02)

  http://publib.boulder.ibm.com/epubs/pdf/dsnsqm02.pdf

1.4 Acknowledgments

Many thanks for the following people to make this HOL possible:

Irene Liu

Ken Taylor

Xiaohong Fu

Yumi Tsuji

2. Lab 0 : Getting Started

In this hands-on-lab(HOL), we will use the DB2 Command Line Processor(CLP) that shipped with DB2 for z/OS to learn about the pureXML and new SQL features in DB2 10 for z/OS.

2.1 Dependency of the Labs

Below is the dependency of the labs.

Lab 0 must be done first to ensure the Command Line Processor (CLP) environment is set up properly and there is database connection to the DB2 server.

Lab 6 to Lab 7 are XML related topics while the rest are new SQL features. Below is a listing of labs.

Lab 6: XML Type Modifier

Lab 7: Sub-document update on XML document

Lab 9: Bitemporal Support

Lab 10: Row Permissions and Column Masks

Lab 11: SQL PL extensions for scalar UDF and table UDFs

2.2 LAB0E1 – Command Line Processor (CLP) setup

In this exercise, we will use the DB2 command line processor that ships with DB2 for z/OS. We will configure CLP and use it to execute a script to test the connectivity to the database.

1. Open a Command Prompt window by double clicking the Command prompt icon in the desktop.
2. Change directory to the location of the hands-on-lab materials.

   In the command prompt window, type: `CD C:\HOL`

3. Start notepad to edit the `clpsetup.bat` file.
   type: `notepad clpsetup.bat`

4. Replace the "XXXXXXXX" and "YYYYYYYY" with the username and password provided to you by the lab instructor. Please use UPPERCASE for all userid and password.

   Tip: You can quickly do find/replace in notepad by choosing "replace" from the Edit menu, or typing Ctrl-H at any time.

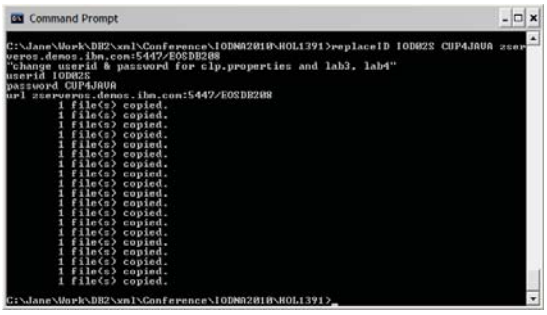
   Note that the alias "mydb2" has been created for you to direct CLP to connect to the server.

   Save the file (from the File menu) and quit notepad (from the File

menu).

5. Execute the clpsetup.bat batch file to define aliases and set environment variables to allow CLP to run. In the command prompt window, type: `clpsetup.bat`

   For a successful execution, you should see something similar to this:



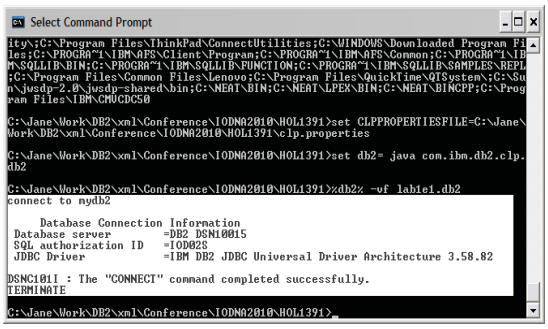
6. Browse the "lab0e1.db2" script. You do not have to change anything in this script (unless you want to).
   type: `notepad lab0e1.db2`

   Exit notepad when you are finished.

7. Execute the "lab0e1.db2" script. This script will connect to the database.
   type: `%db2% -vf lab0e1.db2`

   %db2% invokes the "db2" alias that we created in `clpsetup.bat`. The options "v" tells CLP to use verbose mode so it prints the commands and results to the screen, and "f" tells CLP to execute a file instead of running in interactive mode.
   "lab0e1.db2" is the script that we will run.

Verify that the "CONNECT" command completed successfully. If the screen looks as it does below(may see different `Database Server` and `SQL authorization ID`), then the script ran successfully and this exercise is complete.



CLP will be used again in the following labs. Leave the command prompt window open so we can easily invoke CLP again.

You can now start with any lab you prefer.

Please note we use userid IOD02S for illustration in this lab manual. You should use your own userid assigned by the instructor.

End of lab 0.

3. Lab 6 – Learning XML Type Modifier

In this set of exercises, we will learn how to enforce XML schema conformance automatically for an XML column. This is a DB2 10 features.

3.1 LAB6E1 - Register an XML schema

Before we can associate an XML column with one or more XML schemas, we need to register an XML schema first. In this exercise, we are going to register an XML schema called userid_POSCHEMA.

1. Open POSchema.xsd using notepad or any text editor. In the command prompt we have opened earlier, type

   ```
   notepad POSchema.xsd
   ```

   This schema requires 4 elements: `customer`, `shipping`, `items`, and `billing`. No change is required here.

2. In the same command prompt, type

   ```
   %db2% -vf lab6e1.db2
   ```

Below is the SQL statement (for userid IOD02S) for your reference

```
-- drop objects

DROP TABLE CUSTOMER;

REMOVE XMLSCHEMA SYSXSR.IOD02S_POSCHEMA;

-- register XML schema POSchema

REGISTER XMLSCHEMA
http://www.purchaseOrder.com/purchaseOrder/POSchema.xsd

FROM file://POSchema.xsd

AS SYSXSR.IOD02S_POSCHEMA

COMPLETE
```

For successful execution, you will see something like this:

```
DSNC101I : The "REGISTER XMLSCHEMA" command completed
successfully.
```

Your XML schema has successfully been registered. A schema with the name youruserid_POSCHEMA has been successfully created on the server and is available for use for XML Schema validation.

3.2 LAB6E2 - Create a table with an XML column associated with a registered XML schema

In this exercise, we are going to create a table with an XML column reference a registered an XML schema that we have registered.

1. In the command prompt that we have opened earlier, type

   ```
   %db2% -vf lab6e2.db2
   ```

Below is the SQL statement (for userid IOD02S) for your reference

```
CREATE TABLE CUSTOMER

(ID INTEGER,

  CUSTXML XML(XMLSCHEMA ID SYSXSR.IOD02S_POSCHEMA));
```

After a successful execution, CUSTXML column in CUSTOMER table is associated with XML schema `SYSXSR.IOD02S_POSCHEMA`. DB2 will enforce all the XML documents in this column are valid against this schema during INSERT, UPDATE, and LOAD.

3.3 LAB6E3 - Insert a valid XML document

In this exercise, we are going to insert an XML document that is valid against the schema that associated with the XML column.

In the command prompt that we have opened earlier, type

```
%db2% -vf lab6e3.db2
```

There should not be any error. And you should see the following at the end of the job output:

```
DSNC101I : The "SQL" command completed successfully.

SELECT ID FROM CUSTOMER

ID

1
```

3.4 LAB6E4 - Insert an invalid XML document

In this exercise, we are going to insert an XML document that is NOT valid against the schema that associated with the XML column.

1. Open lab6e4.db2 using notepad or any text editor. In the command prompt that we have opened earlier, type

   ```
   notepad lab6e4.db2
   ```

   Note that XML document that we plan to insert is missing the `items` element. No change is required in this file.

2. In the command prompt, type

   ```
   %db2% -vf lab6e4.db2
   ```

You should see an validation error message like this:

```
sqlcode : -20399 sqlstate: 2201R

sqlerr Message: ERROR ENCOUNTERED DURING XML PARSING OR
VALIDATION AT LOCATION 633 RC=0018,RSN=8604, FOR XML PARSING
OR VALIDATION AGAINST XML SCHEMA XSRID 63.. SQLCODE=-20399,
SQLSTATE=2201R, DRIVER=3.53.113
```

3.5 LAB6E5 - Find the schema name that an XML document is validated against with.

In this exercise, we are going to find out the schema name that an XML document is validated against with.

In LAB6E3, we have inserted a valid XML document with ID=1. We are going to find out what schema we used to validate this XML document. This is useful when the XML column is associated with more than 1 XML schemas.

In the command prompt that we have opened earlier, type

```
%db2% -vf lab6e5.db2
```

Below is the SQL statement for reference:

```
-- for ID=1, find the schema name that the XML document is

SELECT s.XSROBJECTNAME as schema_name

FROM CUSTOMER c, SYSIBM.XSROBJECTS s

WHERE XMLXSROBJECTID(custxml) = s.XSROBJECTID

      AND ID=1
```

You should see userid_POSCHEMA as output.

End of Lab 6.

4. Lab 7 – Learning sub-document update on an XML column

In DB2 9 for z/OS, an update on an XML document is a delete of whole document following by a new insert. In DB2 10 for z/OS, we are able to do sub-document update. In this set of lab exercises, we will explore this new feature.

4.1 LAB7E1 – Create and Populate CUSTOMER table

In this exercise, we will use CLP to create a CUSTOMER table and populate them with 10 same XML documents with different IDs in the ID column. Each XML document is only used once in the following exercises.

Below is the ddl of CUSTOMER table:

```
CREATE TABLE CUSTOMER (ID INTEGER, CUSTXML XML)
```

Below is the content of the XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<demopo:purchaseOrder id="0" orderDate="2001-01-01"
shipDate="2001-01-01" status=""
xmlns:demopo="http://www.purchaseOrder.com/purchaseO
rder" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://www.purchaseOrder.com/pur
chaseOrder POSchema_unq3.xsd ">
  <customer id="0">
    <CustomerName>Jane Man</CustomerName>
  </customer>
  <shipping shippingName="Jane Man">
    <address>
      <Address1>555 Bailey Avenue</Address1>
      <City>San Jose</City>
      <State>CA</State>
      <Zip>95141</Zip>
```

```
    </address>
  </shipping>
  <items>
    <item itemName="Toyota Camry" pid="1000"
price="20000" quantity="1"/>
  </items>
  <billing billingName="Guogen Zhang">
    <address>
      <Address1>123 Sesame Street</Address1>
      <City>San Jose</City>
      <State>CA</State>
      <Zip>95141</Zip>
    </address>
  </billing>
</demopo:purchaseOrder>
```

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e1.db2
```

For successful execution, you will see 10 records as output for

```
SELECT ID FROM CUSTOMER
```

statement, each with a different ID value.

4.2 LAB7E2 – Insert Before Exercise

In this exercise, we will learn about "insert before" sub-document update.

We want to insert

```
<shippingRemark>Little baby, please be
quiet</shippingRemark>
```

before the `shipping` element for ID=2.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e2.db2
```

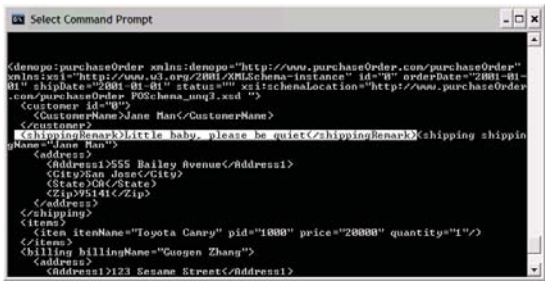

Below is the SQL statement:

```
UPDATE CUSTOMER

SET CUSTXML= XMLMODIFY(

 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";

  insert nodes $remark

  before /demopo:purchaseOrder/shipping',

  XMLPARSE(DOCUMENT

   '<shippingRemark>Little baby, please be
quiet</shippingRemark> '

   preserve whitespace)

    AS "remark")

WHERE ID =2
```

Note:

1. namespace need to be declared (`declare namespace demopo=http://www.purchaseOrder.com/purchaseOrder;`) here, otherwise, DB2 will issue error as it cannot find any match to update.
2. the parameter passed to XMLPARSE (`'<shippingRemark>Little baby, please be quiet</shippingRemark> '`) must be a well-formed document, otherwise, DB2 will issues a non well-formed error.

After a successful execution, you should see the `shippingRemark` element is added before `shipping` element.



4.3 LAB7E3 – Insert After Exercise

In this exercise, we will learn about "insert after" sub-document update.

We want to insert

```
<survey>Excellent</survey>
```

after the `billing` element for ID=3.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e3.db2
```

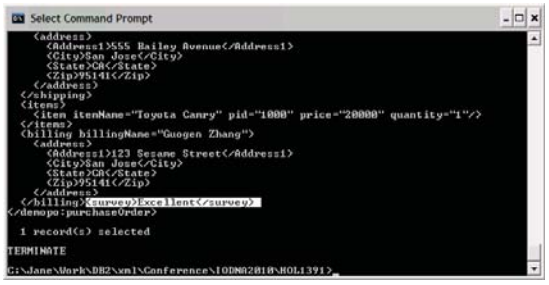
Below is the SQL statement:

```
UPDATE CUSTOMER

SET CUSTXML= XMLMODIFY(

 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";

  insert nodes $survey

  after /demopo:purchaseOrder/billing',

  XMLPARSE(DOCUMENT

   '<survey>Excellent</survey>' preserve whitespace)
```



```
    AS "survey")

WHERE ID =3;
```

After a successful execution, you should see the `survey` element is added after `billing` element.



4.4 LAB7E4 – Insert As First Exercise

In this exercise, we will learn about "insert as first" sub-document update.

We want to insert a comment

```
This is a urgent order!
```

as first child in the `demopo:purchaseOrder` element for ID=4.

1. In the command prompt that we have opened earlier, type

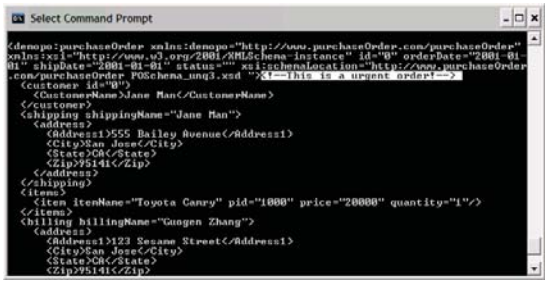
```
%db2% -vf lab7e4.db2
```

Below is the SQL statement:

```
UPDATE CUSTOMER

SET CUSTXML= XMLMODIFY(
```

```
 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";
  insert nodes $remark
  as first into /demopo:purchaseOrder',
  (SELECT XMLCOMMENT('This is a urgent order!') FROM
SYSIBM.SYSDUMMY1)
  AS "remark")
WHERE ID =4;
```

After a successful execution, you should see a comment is added as the first child of `demopo:purchaseOrder` element.



4.5 LAB7E5 – Insert As Last Exercise

In this exercise, we will learn about "insert as last" sub-document update.

We want to insert a new item

```
<item itemName="WII" pid="2" price="200"
quantity="1">
</item>
```

as last `item` in the `items` element for ID=5.

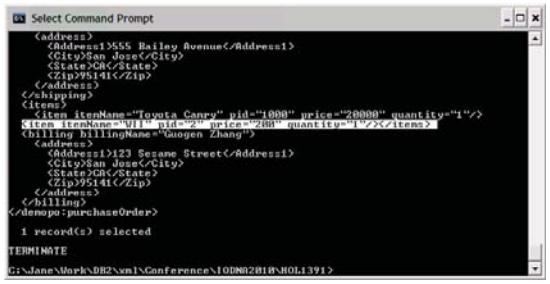
1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e5.db2
```

Below is the SQL statement:

```
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";
  insert nodes $newitem
  as last into /demopo:purchaseOrder/items',
  XMLPARSE(DOCUMENT
   '  <item itemName="WII" pid="2" price="200"
quantity="1"></item>' PRESERVE WHITESPACE)
  AS "newitem")
WHERE ID =5;
```

After a successful execution, you should see a new item(for WII) is added as the last child of `items` element.



4.6 LAB7E6 – Delete Nodes Exercise

In this exercise, we will learn about "delete nodes" sub-document update.

We want to delete the item where pid=1000 (i.e the following item)

```
<item itemName="Toyota Camry" pid="1000"
price="20000" quantity="1"/>
```

in the `items` element for ID=6.

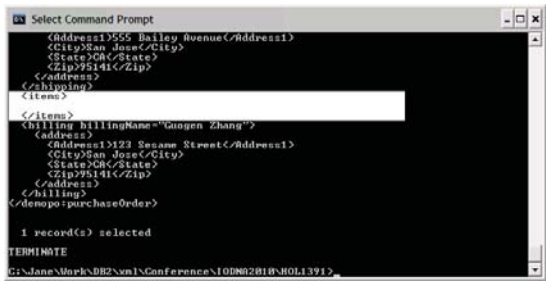
1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e6.db2
```

Below is the SQL statement:

```
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";
  delete nodes
   /demopo:purchaseOrder/items/item[@pid="1000"]'
  )
WHERE ID =6;
```

After a successful execution, you should not see any item inside the `items` element.



4.7 LAB7E7 – Replace Value of Node(attribute value) Exercise

In this exercise, we will learn about "replace value of node" sub-document update.

We want to replace the value of `status` (attribute in `/demopo:purchaseOrder` element) to `shipped` for ID=7.

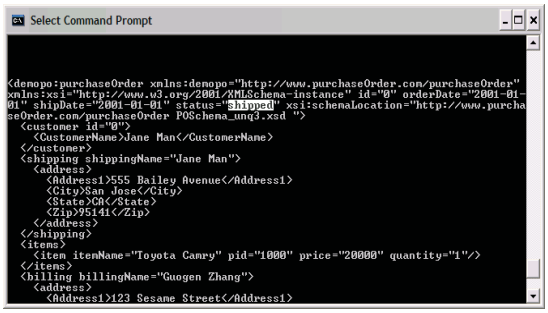
1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e7.db2
```

Below is the SQL statement:

```
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";
  replace value of node
   /demopo:purchaseOrder/@status with "shipped"'
  )
WHERE ID =7;
```

After a successful execution, you should see the attribute value of status has changed to `shipped`.



4.8 LAB7E8 – Replace Value of Node(sequence of nodes) Exercise

In this exercise, we want to replace the value of whole shipping address of `Guogen Zhang` to

```
<address>
   <Address1>999 Brown Street</Address1>
   <City>Fremont</City>
   <State>CA</State>
   <Zip>94560</Zip>
 </address>
```

for ID=8.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e8.db2
```

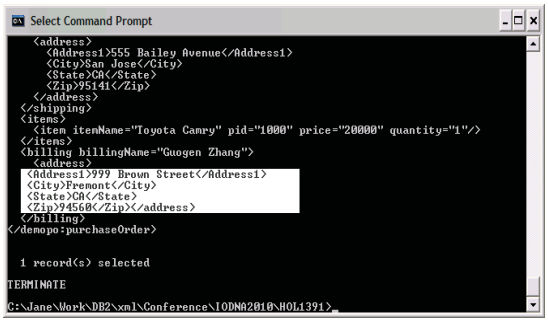
Below is the SQL statement:

```
UPDATE CUSTOMER
```



```
SET CUSTXML= XMLMODIFY(
 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";
  replace value of node
  /demopo:purchaseOrder/billing[@billingName="Guogen
Zhang"]/address
  with
  "
   <Address1>999 Brown Street</Address1>
   <City>Fremont</City>
   <State>CA</State>
   <Zip>94560</Zip>"
  '
  )
WHERE ID =8;
```

After a successful execution, you should see address of `Guogen Zhang` has been changed.



4.9 LAB7E9 – LAB7E10

The remaining lab exercises starting with member LAB7E9 and LAB7E10 are to be completed on your own.

The questions and the hints are described in the individual clp file The objective is to replace instances of "%%%%%%%%" with XPath or other expressions to complete the SQL statement.

Use CLP to execute these exercises.

The solutions are at the last chapter.

The exercises are listed here for your reference.

4.9.1 LAB7E9

```
-- For ID 9, replace the value of shipping Address1 to
-- "999 Union Street"

UPDATE CUSTOMER

SET CUSTXML= XMLMODIFY(

 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";

  %%%%%%%%

  /demopo:purchaseOrder/shipping/address/Address1

  with "999 Union Street"'

  )

WHERE ID =9;
```

4.9.2 LAB7E10

```
-- For ID 10, replace the value the quantity of pid=1000
-- (attribute of "item") to 2

UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
  'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";
  replace value of node
/demopo:purchaseOrder/items/item[@pid="1000"]/@quantity with
"%%%%%%%%")
WHERE ID=10;
```

End of Lab 7.

5. Lab 9 – Learning Bitemporal Support

In this set of lab exercises, we will learn about Bitemporal support using a health insurance policy example.
Each of these exercises is provided as a CLP file. They are named 'LAB9EX' where X is the lab number.

For example, to execute lab9e1 using CLP, in the command prompt that we have opened earlier, type

```
%db2% -vf lab9e1.db2
```

5.1 LAB9E1: Create POLICY and POLICY_HIST table

In Lab9e1, we are going to create the POLICY and POLICY_HIST table.
In the command prompt that we have opened earlier, type

```
%db2% -vf lab9e1.db2
```

Below is the SQL statement for your reference:

```
-- create POLICY table
CREATE TABLE POLICY (
   CLIENT CHAR(4) NOT NULL,
   TYPE   CHAR(4) NOT NULL,
   COPAY  SMALLINT NOT NULL,
   EFF_BEG DATE NOT NULL,
   EFF_END DATE NOT NULL,
   SYS_BEG TIMESTAMP(12) WITH TIME ZONE NOT NULL
   IMPLICITLY HIDDEN GENERATED AS ROW BEGIN,
   SYS_END TIMESTAMP(12) WITH TIME ZONE NOT NULL
   IMPLICITLY HIDDEN GENERATED AS ROW END,
   TRANS_ID TIMESTAMP(12) WITH TIME ZONE
   IMPLICITLY HIDDEN GENERATED AS TRANSACTION START ID,
   PERIOD SYSTEM_TIME(SYS_BEG,SYS_END),
   PERIOD BUSINESS_TIME(EFF_BEG,EFF_END));

-- create POLICY_HIST table
CREATE TABLE POLICY_HIST LIKE POLICY;
```

5.2 LAB9E2: Add versioning to POLICY table

In lab9e2, we ALTER the POLICY table to add versioning and insert a row to the POLICY table:

issues PPO policy to customer C882 with copay amount $10 starting from 01/01/2010.

In the command prompt that we have opened earlier, type

```
%db2% -vf lab9e2.db2
```

Below is the SQL statement for your reference

```
-- alter POLICY table
ALTER TABLE POLICY ADD VERSIONING
 USE HISTORY TABLE POLICY_HIST;

-- Insert to POLICY table :
-- issue PPO policy to customer C882 with copay amount $10
starting
-- from 01/01/2010
INSERT INTO POLICY
  VALUES('C882','PPO',10,'01/01/2010','12/31/9999');
```

```
    +-----------------------------------------------------+
    | CLIENT | TYPE |   COPAY   |  EFF_BEG   |  EFF_END   |
    +-----------------------------------------------------+
  1_| C882   | PPO  |        10 | 2010-01-01 | 9999-12-31 |
    +-----------------------------------------------------+
```

5.3 LAB9E3: Change to HMO

In lab9e3, customer C882 called and requested to change the type to HMO.

In the command prompt that we have opened earlier, type

```
%db2% -vf lab9e3.db2
```

Below is the SQL statement for your reference

```
-- alter POLICY table :
-- customer C882 called and changed to HMO
UPDATE POLICY FOR PORTION OF BUSINESS_TIME
  FROM '06/01/2010' TO '12/31/9999'
  SET TYPE = 'HMO'
    WHERE CLIENT = 'C882';

SELECT * FROM POLICY ORDER BY EFF_END;
```

```
    +-----------------------------------------------------+
    | CLIENT | TYPE |   COPAY   |  EFF_BEG   |  EFF_END   |
    +-----------------------------------------------------+
  1_| C882   | PPO  |        10 | 2010-01-01 | 2010-06-01 |
  2_| C882   | HMO  |        10 | 2010-06-01 | 9999-12-31 |  <-
    +-----------------------------------------------------+
```

Let's have a look on the history table. In the command prompt that we have opened earlier, type


```
%db2% -vf lab9SelectHistory.db2
```

```
SELECT * FROM POLICY_HIST;
```

```
    +-----------------------------------------------------+
    | CLIENT | TYPE |   COPAY   |  EFF_BEG   |  EFF_END   |
    +-----------------------------------------------------+
  1_| C882   | PPO  |        10 | 2010-01-01 | 9999-12-31 |
    +-----------------------------------------------------+
```

5.4 LAB9E4: Increase copay starting 10/01/2010

In lab9e4, the insurance company increases the copay to $15 starting 10/01/2010 for customer C882.

In the command prompt that we have opened earlier, type

```
%db2% -vf lab9e4.db2
```

Below is the SQL statement for your reference

```
-- alter POLICY table :
-- copay increase to $15 starting 10/01/2010
UPDATE POLICY FOR PORTION OF BUSINESS_TIME
  FROM '10/01/2010' TO '12/31/9999'
  SET COPAY = 15
    WHERE CLIENT = 'C882';

-- verify POLICY table
SELECT * FROM POLICY ORDER BY EFF_END;
```

```
    +-----------------------------------------------------+
    | CLIENT | TYPE |   COPAY   |  EFF_BEG   |  EFF_END   |
    +-----------------------------------------------------+
  1_| C882   | PPO  |        10 | 2010-01-01 | 2010-06-01 |
  2_| C882   | HMO  |        10 | 2010-06-01 | 2010-10-01 |
  3_| C882   | HMO  |        15 | 2010-10-01 | 9999-12-31 |  <-
    +-----------------------------------------------------+
```

Let's have a look on the history table. In the command prompt that we have opened earlier, type

```
%db2% -vf lab9SelectHistory.db2
```

```
SELECT * FROM POLICY_HIST;
```



```
    +-----------------------------------------------------+
    | CLIENT | TYPE |   COPAY   |  EFF_BEG   |  EFF_END   |
    +-----------------------------------------------------+
  1_| C882   | PPO  |        10 | 2010-01-01 | 2010-06-01 |
  2_| C882   | HMO  |        10 | 2010-06-01 | 9999-12-31 |  <-
    +-----------------------------------------------------+
```

5.5 LAB9E5: End policy on 12/15/2010

In lab9e5, we change the policy to end on 12/15/2010 for customer C882.

In the command prompt that we have opened earlier, type

```
%db2% -vf lab9e5.db2
```

Below is the SQL statement for your reference

```
-- change the policy to end on 12/15/2010
UPDATE POLICY
  SET EFF_END = '12/15/2010'
    WHERE CLIENT = 'C882'
     AND EFF_END = '12/31/9999';

-- verify POLICY table
SELECT * FROM POLICY ORDER BY EFF_END;
```

```
    +-----------------------------------------------------+
    | CLIENT | TYPE |   COPAY   |  EFF_BEG   |  EFF_END   |
    +-----------------------------------------------------+
  1_| C882   | PPO  |        10 | 2010-01-01 | 2010-06-01 |
  2_| C882   | HMO  |        10 | 2010-06-01 | 2010-10-01 |
  3_| C882   | HMO  |        15 | 2010-10-01 | 2010-12-15 |  <-
    +-----------------------------------------------------+
```

Let's have a look on the history table. In the command prompt that we have opened earlier, type

```
%db2% -vf lab9SelectHistory.db2
```

```
SELECT * FROM POLICY_HIST;
```

```
    +-----------------------------------------------------+
    | CLIENT | TYPE |   COPAY   |  EFF_BEG   |  EFF_END   |
    +-----------------------------------------------------+
  1_| C882   | PPO  |        10 | 2010-01-01 | 9999-12-31 |
  2_| C882   | HMO  |        10 | 2010-06-01 | 9999-12-31 |
  3_| C882   | HMO  |        15 | 2010-10-01 | 9999-12-31 |  <-
    +-----------------------------------------------------+
```


5.6 LAB9E6: Update policy to POS

In lab9e6, we correct error by retroactively updating policy to POS from 07/01/2010 to 10/15/2010 for customer C882.

In the command prompt that we have opened earlier, type

```
%db2% -vf lab9e6.db2
```

Below is the SQL statement for your reference

```
-- Correct error by retroactively updating policy to POS from
-- 07/01/2010 to 10/15/2010
UPDATE POLICY FOR PORTION OF BUSINESS_TIME
  FROM '07/01/2010' TO '10/15/2010'
  SET TYPE = 'POS'
    WHERE CLIENT = 'C882';


-- verify POLICY table
SELECT * FROM POLICY ORDER BY EFF_END;
```

```
    +-----------------------------------------------------+
    | CLIENT | TYPE |   COPAY   |  EFF_BEG   |  EFF_END   |
    +-----------------------------------------------------+
  1_| C882   | PPO  |        10 | 2010-01-01 | 2010-06-01 |
  2_| C882   | HMO  |        10 | 2010-06-01 | 2010-07-01 |
  3_| C882   | POS  |        10 | 2010-07-01 | 2010-10-01 |  <-
  4_| C882   | POS  |        15 | 2010-10-01 | 2010-10-15 |  <-
  5_| C882   | HMO  |        15 | 2010-10-15 | 2010-12-15 |
    +-----------------------------------------------------+
```

Let's have a look on the history table. In the command prompt that we have opened earlier, type

```
%db2% -vf lab9SelectHistory.db2
```

```
SELECT * FROM POLICY_HIST;
```

```
    +-----------------------------------------------------+
    | CLIENT | TYPE |   COPAY   |  EFF_BEG   |  EFF_END   |
    +-----------------------------------------------------+
  1_| C882   | PPO  |        10 | 2010-01-01 | 9999-12-31 |
  2_| C882   | HMO  |        10 | 2010-06-01 | 9999-12-31 |
  3_| C882   | HMO  |        15 | 2010-10-01 | 9999-12-31 |
  4_| C882   | HMO  |        15 | 2010-10-01 | 2010-12-15 |  <-
  5_| C882   | HMO  |        10 | 2010-06-01 | 2010-10-01 |  <-
    +-----------------------------------------------------+
```

5.7 LAB9E7: System time for policy end on 07/01/2010

In lab9e7, we find the system time for the policy that end on 07/01/2010 for customer C882.

In the command prompt that we have opened earlier, type

```
%db2% -vf lab9e7.db2
```

Below is the SQL statement for your reference

```
-- Find the system time for the policy that end at 07/01/2010
SELECT SYS_BEG AS TIME1 FROM POLICY
  WHERE EFF_END = '07/01/2010'

     +--------------------------------------------+
     |                TIME1                       |
     +--------------------------------------------+
   1_| 2011-07-28-17.00.36.721394697-07:00        |
     +--------------------------------------------+
Note: you may see a different TIME1 in your output.
```

5.8 LAB9E8: POS service denied for appt on 08/01/2010

Customer C882 complains and says claims dept denied POS service for appt on '08/01/2010'. Call was made before `time1` correction made in lab9e6.

We need historical data to see what claims department saw before correction time made in lab9e6.

1. In the command prompt that we have opened earlier, type

   ```
   notepad lab9e8.db2
   ```

   Replace `time1` with the actual timestamp output from previous exercise (lab9e7). Don't forget to put the single quote around the timestamp.

   For example: `TIMESTAMP_TZ('2011-07-28-17.00.36.721394697-07:00')`

   Save the change.

2. In the command prompt that we have opened earlier, type


```
%db2% -vf lab9e8.db2
```

Below is the SQL statement for your reference

```
SELECT TYPE FROM POLICY
  FOR BUSINESS_TIME AS OF '08/01/2010'
  WHERE CLIENT = 'C882';

     +------+
     | TYPE |
     +------+
   1_| POS  |
     +------+
```

As shown above, without historical data, the query will return 'POS'.

```
-- use historical data to see what claims department saw
before
-- correction time made in lab9e6

SELECT TYPE FROM POLICY
  FOR BUSINESS_TIME AS OF '08/01/2010'
  FOR SYSTEM_TIME AS OF (TIMESTAMP_TZ('2011-07-28-
17.00.36.721394697-07:00')  - 1 microsecond)
  WHERE CLIENT = 'C882';

     +------+
     | TYPE |
     +------+
   1_| HMO  |
     +------+
```

5.9 LAB9E9: Alter an existing application to use DB's solution.

Given an existing application that implements bitemporal with triggers, how would you ALTER to change to to use DB2's solution?

1. Create the existing application by executing `lab9e9.db2`

   In the command prompt that we have opened earlier, type

   ```
   %db2% -vf lab9e9.db2
   ```

2. Create your answer in `lab2e9Try.db2`

   In the command prompt that we have opened earlier, type

   ```
   notepad lab9e9Try.db2
   ```


   Put your answer in `lab9e9Try.db2` and save the file.

3. Execute your answer by entering the following in the same command prompt:

   ```
   %db2% -vf lab9e9Try.db2
   ```

Below is the SQL statement for the existing application for your reference:

```
DROP TABLE TEST;
DROP TABLE TEST_HISTORY;
COMMIT;
CREATE TABLE TEST (ID INTEGER NOT NULL, COUNT INTEGER NOT NULL,
                   BUS_START DATE NOT NULL,BUS_END DATE NOT NULL,
                   SYS_START TIMESTAMP NOT NULL                ,
                   SYS_END TIMESTAMP NOT NULL                  );
CREATE TABLE TEST_HISTORY(ID INTEGER NOT NULL, COUNT INTEGER NOT NULL,
                   BUS_START DATE NOT NULL,BUS_END DATE NOT NULL,
                   SYS_START TIMESTAMP NOT NULL                ,
                   SYS_END TIMESTAMP not null                  );
CREATE TRIGGER TR1 AFTER DELETE ON TEST
          REFERENCING OLD as X
FOR EACH ROW MODE DB2SQL
      INSERT INTO TEST_HISTORY VALUES( X.id,x.count,x.bus_start,
      x.bus_end,x.SYS_start,current timestamp);
CREATE TRIGGER TR2       NO CASCADE BEFORE INSERT ON TEST
          REFERENCING
                     NEW AS Y
FOR EACH ROW MODE DB2SQL
      SET Y.SYS_END    = '9999-12-31 24:00:00.000000';
CREATE TRIGGER TR3       NO CASCADE BEFORE INSERT ON TEST
          REFERENCING
                     NEW AS Y
FOR EACH ROW MODE DB2SQL
      SET Y.SYS_START = CURRENT TIMESTAMP;
CREATE TRIGGER TR4       NO CASCADE BEFORE UPDATE ON TEST
          REFERENCING
                     NEW AS Y
FOR EACH ROW MODE DB2SQL
      SET Y.SYS_START = CURRENT TIMESTAMP;
CREATE TRIGGER TR5      NO CASCADE BEFORE UPDATE ON TEST
          REFERENCING
                     NEW AS Y
FOR EACH ROW MODE DB2SQL
      SET Y.SYS_END    = '9999-12-31 24:00:00.000000';
CREATE TRIGGER TR6      AFTER UPDATE ON TEST
          REFERENCING OLD as X
                     NEW AS Y
FOR EACH ROW MODE DB2SQL
      INSERT INTO TEST_HISTORY VALUES( X.id,x.count,x.bus_start,
      x.bus_end,x.SYS_start,current timestamp);
```



```
COMMIT;
INSERT INTO TEST VALUES(1,1,'01/01/2010','12/31/9999',
   '1234-12-12-12:00:00.000000',
   '1234-12-12-12:00:00.000000');
COMMIT;
UPDATE TEST SET COUNT=10 WHERE ID = 1;
COMMIT;
SELECT * FROM TEST;
SELECT * FROM TEST_HISTORY;
DELETE FROM TEST where id=1;
COMMIT;
SELECT * FROM TEST;
SELECT * FROM TEST_HISTORY;
```

End of Lab 9.

6. Lab 10 – Row and Column Access Control

In this set of lab exercises, you will learn how to enhance security by controlling row and column access.

We will use a banking system as example here. There are 3 tables: CUSTOMER, CUSTOMER_CHOICE, and INTERNAL_INFO. You will use your primary id to create row permissions and column masks on table CUSTOMER such that a teller, or a customerserv, or a telemarketing person can only access part of the data from the CUSTOMER table.

For this lab, you will get a set of 4 userids: your primary id (the one assigned by the instructor) and 1 id for teller, 1 id for customerserv, and 1 id for telemarketing.

For example, if your primary userid is IOD02S, you will have

Userid: IOD02S**A** for teller

Userid: IOD02S**B** for customerserv

Userid: IOD02S**C** for telemarketing

If in the system panel DSNTIPP1, the field SEPARATE SECURITY has a value NO, the userid with the SYSADM authority has been given the SECADM authority (the security administrator authority). Remember that only the SECADM authority can create a row permission or a column mask.

If in the system panel DSNTIPP1, the field SEPARATE SECURITY has a value YES, the userid with the SYSADM authority does not have

the SECADM authority and the SYSADM authority alone cannot create a row permission or a column mask.

In Lab10, the field SEPARATE SECURITY has a value NO, that is, your primary userid has both the SYSADM authority and SECADM authority in order for you to create row permissions and column masks. It has been done this way because the SECADM authority cannot be granted to all students in the Lab (there are only two SECADM authorities allowed per DB2) but the SYSADM authority can be granted to all students.

In Lab10, your other userids such as IOD02SA, IOD02SB, and IOD02SC will be granted with the SELECT privilege on the CUSTOMER table.

6.1 LAB10E1: Create and populate required Tables.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab10e1.db2
```

This script will use your primary userid, i.e. the one with the SYSADM authority, to create and populate the following tables:

```
CREATE TABLE  CUSTOMER(ACCOUNT CHAR(19),
                       NAME    VARCHAR(60),
                       PHONE   CHAR(12),
                       INCOME  INT,
                       BRANCH  CHAR(1));
```

ACCOUNT	NAME	PHONE	INCOME	BRANCH
1111-2222-3333-4444	Alice Smith	408-111-1111	22000	A
2222-3333-4444-5555	Bob Mustafa	408-222-2222	71000	B
3333-4444-5555-6666	Louis Goodwi	408-333-3333	123000	B
4444-5555-6666-7777	David Parekh	408-444-4444	176000	C


```
CREATE TABLE  CUSTOMER_CHOICE(ACCOUNT CHAR(19),
                              PHONE_CHOICE CHAR(1));
```

	ACCOUNT	PHONE_CHOICE
1_	1111-2222-3333-4444	1
2_	2222-3333-4444-5555	1
3_	3333-4444-5555-6666	0
4_	4444-5555-6666-7777	1

PHONE_CHOICE value of 0 here indicate that account don't accept any marketing/sales call.

The INTERNAL_INFO table below is to store employees' info: emp_id, branch they are working, and the employee name.

```
CREATE TABLE  INTERNAL_INFO(EMP_ID      CHAR(8),
                            HOME_BRANCH CHAR(1),
                            NAME        VARCHAR(60));
```

	EMP_ID	HOME_BRANCH	NAME
1_	IOD02SA	B	Mary Carpentor
2_	IOD02SB	B	Peter Carpentor
3_	IOD02SC	B	Paul Carpentor

The script also GRANT the SELECT privilege on table CUSTOMER to teller, customerserv, and telemarketing person.

Below are GRANT statements issued by your primary id IOD02S

```
GRANT SELECT ON TABLE CUSTOMER TO IOD02SA
GRANT SELECT ON TABLE CUSTOMER TO IOD02SB
GRANT SELECT ON TABLE CUSTOMER TO IOD02SC
```

6.2 LAB10E2: Create Row Permissions

We are going to create 2 rows permissions to control row access:

- TELLER_ROW_ACCESS

We only allow teller from the same branch as the customer accounts to access the corresponding rows in the CUSTOMER table.

- CUSTOMERSERV_ROW_ACCESS

  We allow customerserv and telemarketing to access all rows in the CUSTOMER table

1. In the command prompt, that we have opened earlier, type

```
notepad lab10e2.db2
```

Inside `lab10e2.db2` , you can see the syntax for `TELLER_ROW_ACCESS` row permission:

```
CREATE PERMISSION TELLER_ROW_ACCESS ON CUSTOMER
  FOR ROWS WHERE
    VERIFY_GROUP_FOR_USER (SESSION_USER, 'IOD02SA') = 1
    AND
    BRANCH = (SELECT HOME_BRANCH FROM  INTERNAL_INFO
              WHERE EMP_ID = SESSION_USER)
    ENFORCED FOR ALL ACCESS
ENABLE;
```

Please note built-in function `VERIFY_GROUP_FOR_USER()` is used to verify the current session user has a particular id or not. In `TELLER_ROW_ACCESS` above, we check whether the session user is IOD02SA or not (note: IOD02SA is a teller).

Now you need to create a CUSTOMERSERV_ROW_ACCESS such that customerserv (your primariy id suffix with B) and telemarketing (your primariy id suffix with C) can access all rows in the CUSTOMER table.

2. Inside `lab10e2.db2` , find the section for

```
CREATE PERMISSION  CUSTOMERSERV_ROW_ACCESS ON  CUSTOMER
```



```
  FOR ROWS WHERE
    %%%%%%%%
    OR
    %%%%%%%%
  ENFORCED FOR ALL ACCESS
  ENABLE;
```

Replace %%%%%%%% with your answer. Save your change. (The solution is in the last chapter.)

3. Execute the script with your answer. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab10e2.db2
```

Check the job output and make sure all the CREATE PERMISSION statements are executed successfully.

6.3 LAB10E3: Create Column Masks

We are going to create 3 column masks to control columns access:

- INCOME_COLUMN_MASK

  Teller is not allowed to see the income values in the INCOME column in CUSTOMER table. Customerserv and telemarketing person can see the range of customer's income value.

- ACCOUNT_COLUMN_MASK

  We allow teller and customerserv to see the full account number. Telemarketing cannot see customer's full account number, only the last 4 digits.

- PHONE_COLUMN_MASK

  We allow teller and customerserv to see customer's phone number. Telemarketing cannot see customer's phone number unless the customer allow them to do so (PHONE_CHOICE column with value of 1 in CUSTOMER_CHOICE table)

1. In the command prompt, that we have opened earlier, type

```
notepad lab10e3.db2
```

2. Inside `lab10e3.db2`, you can see the syntax for `INCOME_COLUMN_MASK` and `PHONE_COLUMN_MASK` column masks.

```
-------------------------------------------------------------------
-- Create enabled column mask for column INCOME
--   - user IOD02SA TELLER
--       cannot see customer's income values
--   - user IOD02SB CUSTOMERSERV
--   - user IOD02SC TELEMARKETING
--       can see the ranges of the customer's income values
-------------------------------------------------------------------
CREATE MASK  INCOME_COLUMN_MASK ON  CUSTOMER
    FOR COLUMN INCOME RETURN
      CASE WHEN (VERIFY_GROUP_FOR_USER (SESSION_USER, 'IOD02SB') = 1 OR
                 VERIFY_GROUP_FOR_USER (SESSION_USER, 'IOD02SC') = 1)
             THEN CASE WHEN (INCOME > 150000) THEN 999999
                       WHEN (INCOME > 75000 ) THEN 150000
                       WHEN (INCOME > 25000 ) THEN 75000
                                              ELSE 25000
                  END
             ELSE NULL
      END
ENABLE;


-------------------------------------------------------------------
-- Create enabled column mask for column PHONE
--   - user IOD02SC TELEMARKETING
--       cannot see customer's phone number unless he or she opt'ed
--   - user IOD02SA TELLER
--   - user IOD02SB CUSTOMERSERV
--       can see customer's phone number
-------------------------------------------------------------------
  CREATE MASK  PHONE_COLUMN_MASK ON  CUSTOMER
    FOR COLUMN PHONE RETURN
      CASE WHEN (VERIFY_GROUP_FOR_USER (SESSION_USER, 'IOD02SC') = 1)
             THEN CASE WHEN(EXISTS
                               (SELECT 1 FROM  CUSTOMER_CHOICE C
                                 WHERE
                                 CUSTOMER.ACCOUNT = C.ACCOUNT AND
                                 C.PHONE_CHOICE = 1)
                           )
                       THEN PHONE
                       ELSE NULL
                  END
             ELSE PHONE
      END
```



```
ENABLE;
```

Now you need to create a ACCOUNT_COLUMN_MASK such that teller (your primary id suffix with A) and customerserv (your primary id suffix with B) to see the full account number. Telemarketing(your primary id suffix with C) cannot see customer's full account number, only the last 4 digits.

3. Inside `lab10e3.db2`, find the section for

```
-------------------------------------------------------------
-- Create enabled column mask for column ACCOUNT
--   - user IOD02SC TELEMARKETING
--       cannot see customer's full account number, only the
--       last 4 digits
--   - user IOD02SA TELLER
--   - user IOD02SB CUSTOMERSERV
--       can see customer's full account number
-------------------------------------------------------------
CREATE MASK ACCOUNT_COLUMN_MASK ON CUSTOMER

  FOR COLUMN ACCOUNT RETURN

    CASE WHEN (%%%%%%%%)

       THEN CHAR('xxxx-xxxx-xxxx-') ||  SUBSTR(ACCOUNT,16,4)

       ELSE  ACCOUNT

    END

ENABLE
```

Replace %%%%%%%% with your answer. Save your change. (The solution is in the last chapter.)

4. Execute the script with your answer. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab10e3.db2
```

Check the job output and make sure all the CREATE MASK statements are executed successfully.

6.4 LAB10E4: Logon as Teller

In this exercise, we logon as teller (your primary id suffix with A) and SELECT from CUSTOMER table.

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab10e4.db2
```

You should see something like this:

(Teller cannot see the customer's income and can see the account from his/her branch(B here))

```
+-------------------------------------------------------------------
| ACCOUNT             |     NAME     |     PHONE     | INCOME    | BRANCH
+-------------------------------------------------------------------
| 2222-3333-4444-5555 | Bob Mustafa  | 408-222-2222 | ?         | B     |
| 3333-4444-5555-6666 | Louis Goodwi | 408-333-3333 | ?         | B     |
+-------------------------------------------------------------------
```

6.5 LAB10E5: Logon as CustomerServ

In this exercise, we logon as customerserv (your primary id suffix with B) and SELECT from CUSTOMER table.

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab10e5.db2
```

You should see something like this:

(Customerserv can access all the accounts, but can only see range of customer's income)

```
+-------------------------------------------------------------------
|       ACCOUNT       |     NAME     |     PHONE     | INCOME    | BRANCH
+-------------------------------------------------------------------
| 1111-2222-3333-4444 | Alice Smith  | 408-111-1111 |      25000 | A    |
| 2222-3333-4444-5555 | Bob Mustafa  | 408-222-2222 |      75000 | B    |
| 3333-4444-5555-6666 | Louis Goodwi | 408-333-3333 |     150000 | B    |
```



```
| 4444-5555-6666-7777 | David Parekh | 408-444-4444 |     999999 | C    |
+-------------------------------------------------------------------
```

6.6 LAB10E6: Logon as TeleMarketing

In this exercise, we logon as telemarketing (your primary id suffix with C) and SELECT from CUSTOMER table.

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab10e6.db2
```

(TeleMarking can

- access all the accounts(rows),
- see range of customer's income
- see last 4-digits of the account number
- see customer's phone number if the customer allowed

)

You should see something like this:

```
+-------------------------------------------------------------------
|       ACCOUNT       |     NAME     |     PHONE     | INCOME    | BRANCH
+-------------------------------------------------------------------
| xxxx-xxxx-xxxx-4444 | Alice Smith  | 408-111-1111 |      25000 | A    |
| xxxx-xxxx-xxxx-5555 | Bob Mustafa  | 408-222-2222 |      75000 | B    |
| xxxx-xxxx-xxxx-6666 | Louis Goodwi | ?            |     150000 | B    |
| xxxx-xxxx-xxxx-7777 | David Parekh | 408-444-4444 |     999999 | C    |
+-------------------------------------------------------------------
```

End of Lab 10.

7. Lab 11 – SQL PL Extensions for scalar UDF and table UDFs

In DB2 9 for z/OS, there is no support for SQL table functions (only external table functions are supported) and SQL scalar function is limited to a single RETURN statement.

In DB2 10 for z/OS, we have extended the SQL PL support to cover these 2 areas. In this lab, we are going to learn about some of these extensions.

We are using CLP in this lab.

7.1 LAB11E1: Scalar Function

In this exercise, we are going to create a function called REVESE that returns a reversed order of a string input.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab11e1.db2
```

Below are the SQL statements for your reference:

```
CREATE FUNCTION REVERSE(INSTR VARCHAR(20))
RETURNS VARCHAR(20)
DETERMINISTIC
NO EXTERNAL ACTION
CONTAINS SQL
BEGIN
  DECLARE REVSTR, RESTSTR VARCHAR(20) DEFAULT '';
  DECLARE LEN INT;
  IF INSTR IS NULL THEN
    RETURN NULL;
  END IF;

  SET (RESTSTR, LEN) = (INSTR, LENGTH(INSTR));

  WHILE LEN > 0 DO
    SET (REVSTR, RESTSTR, LEN) = (SUBSTR(RESTSTR, 1, 1) CONCAT
    REVSTR, SUBSTR(RESTSTR, 2, LEN - 1), LEN - 1);
  END WHILE;

  RETURN REVSTR;
END;
```

To call the REVERSE function:


```
SELECT REVERSE('YUMI') FROM SYSIBM.SYSDUMMY1;

SELECT REVERSE('Jane') FROM SYSIBM.SYSDUMMY1;

SELECT REVERSE('This is fun!') FROM SYSIBM.SYSDUMMY1;
```

Result should be:

```
IMUY

enaJ

!nuf si sihT
```

7.2 LAB11E2: Table Function

In this exercise, we are going to create a table function called JTABLE that returns a table containing columns from joining 2 tables and the join condition is determined by the JTABLE parameter.

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab11e2.db2
```

Below are the SQL statements for your reference:

```
CREATE TABLE TABLE1 (COL1 INT, COLA INT, COLD CHAR(9));

INSERT INTO TABLE1 VALUES(2, 3, 'HELLO');

INSERT INTO TABLE1 VALUES(4, 5, 'WORLD');


CREATE TABLE TABLE2 (COL1 INT, COLB INT, COLC INT);

INSERT INTO TABLE2 VALUES(2, 5, 55     );

INSERT INTO TABLE2 VALUES(4, 6, 66     );


CREATE FUNCTION JTABLE (COLD_VALUE VARCHAR(9), T2_FLAG
VARCHAR(1))
RETURNS TABLE (COLA INT, COLB INT, COLC INT)
LANGUAGE SQL
SPECIFIC DEPTINFO
NOT DETERMINISTIC
READS SQL DATA
RETURN
  SELECT A.COLA, B.COLB, B.COLC
  FROM
    TABLE1 AS A LEFT OUTER JOIN TABLE2 AS B
    ON A.COL1 = B.COL1 AND T2_FLAG = 'Y'
  WHERE A.COLD = COLD_VALUE;
```


To execute the JTABLE we just created:

```
SELECT * FROM TABLE(JTABLE ('HELLO','Y')) X;
```

You should see:

```
COLA        COLB        COLC

3           5           55
```

Below is the SQL statement to verify this result (i.e. you should see the same result by the executing the following SQL statement):

```
SELECT A.COLA, B.COLB, B.COLC

FROM TABLE1 AS A

  LEFT OUTER JOIN

  TABLE2 AS B

  ON A.COL1 = B.COL1 AND 'Y'='Y'

WHERE A.COLD = 'HELLO';
```

Below is another example to execute the JTABLE we just created

```
SELECT *

 FROM TABLE(JTABLE ('HELLO','N')) X;
```

You should see:

```
COLA        COLB        COLC

3           <null>      <null>
```

Below is the SQL statement to verify this result (i.e. you should see the same result by the executing the following SQL statement):

```
SELECT A.COLA, B.COLB, B.COLC

FROM TABLE1 AS A

  LEFT OUTER JOIN

  TABLE2 AS B

  ON A.COL1 = B.COL1 AND 'N'='Y'

WHERE A.COLD = 'HELLO';
```


7.3 LAB11E3: Scalar Function that Encapsulate a Web Service

In this exercise, we are going to write a scalar function called CurrencyConvetor that returns the exchange rate from one currency to another currency (which are passed as parameters).

We will use DB2XML.SOAPHTTPNV function to send out a web service request to

http://www.webservicex.net/CurrencyConvertor.asmx

to ask for the exchange rate. Since the web services response is an XML document, we use XMLQuery() to get the exchange rate.

Please note the output of this exercise depend on how busy the web server is. So, you may get error when the web server is busy.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab11e3.db2
```

Below is the SQL statement for your reference:

```
CREATE FUNCTION CurrencyConvertor(FromCur VARCHAR(3), ToCur
VARCHAR(3) )
RETURNS XML
LANGUAGE SQL
NOT DETERMINISTIC
READS SQL DATA
RETURN
(SELECT XMLQUERY('declare namespace
soap="http://schemas.xmlsoap.org/soap/envelope/";
declare default element namespace
"http://www.webserviceX.NET/";
/soap:Envelope/soap:Body/ConversionRateResponse/ConversionRate
Result/text()'
passing XMLPARSE(DOCUMENT
DB2XML.SOAPHTTPNV(
  'http://www.webservicex.net/CurrencyConvertor.asmx',
  'http://www.webserviceX.NET/ConversionRate',
  '<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ConversionRate xmlns="http://www.webserviceX.NET/">
      <FromCurrency>'|| FromCur ||'</FromCurrency>
      <ToCurrency>'|| toCur ||'</ToCurrency>
    </ConversionRate>
  </soap:Body>
</soap:Envelope>')
```

```
)) FROM SYSIBM.SYSDUMMY1
);
```

To execute the CurrencyConvertor function we just created:

```
SELECT CurrencyConvertor('EUR', 'USD') as exchangeRate FROM
SYSIBM.SYSDUMMY1;
```

The expected output should be similar to the following:

```
1.3312
```

If you see error in executing CurrencyConvertor, you can check the status of the web services provider by going to the following site in a web browser like IE or FireFox:

http://www.webservicex.net/CurrencyConvertor.asmx

7.4 LAB11E4: Table Function that Encapsulate a Web Service

This exercise is very similar to the previous exercise, but we are going to create a table function, not scalar function.

We are going to create a function called StockQuote that take a stock symbol as input and return the stock quote and other details for that particular symbol.

We will use DB2XML.SOAPHTTPNV function to send out a web service request to

```
http://www.webserviceX.NET
```

to ask for the stock details. Since the web services response is an XML document, we use XMLTable() to format the response into a SQL table format.

Once again, the output of this exercise depends on how busy the web server is. So, you may get error when the web server is busy.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab11e4.db2
```

Below is the SQL statement for your reference:

```
CREATE FUNCTION STOCKQUOTE(INSYMBOL VARCHAR(4))
RETURNS TABLE (
```



```
 Symbol VARCHAR(4),
 LAST DECIMAL(6,2),
 Date VARCHAR(10),
 Time    VARCHAR(8),
 Change  VARCHAR(8),
 Open    VARCHAR(8),
 High    VARCHAR(8),
 Low     VARCHAR(8),
 Volume  VARCHAR(12))
LANGUAGE SQL
NOT DETERMINISTIC
RETURN
(
SELECT *
FROM XMLTABLE('/StockQuotes/Stock' PASSING
 XMLPARSE(DOCUMENT XMLCAST( XMLQUERY('declare namespace
soap="http://schemas.xmlsoap.org/soap/envelope/"; declare
default element namespace "http://www.webserviceX.NET/";
/soap:Envelope/soap:Body/GetQuoteResponse/GetQuoteResult'
passing XMLPARSE(DOCUMENT
DB2XML.SOAPHTTPNC('http://www.webservicex.net/stockquote.asmx'
,  'http://www.webserviceX.NET/GetQuote',
  '<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetQuote xmlns="http://www.webserviceX.NET/">
      <symbol>' || INSYMBOL ||'</symbol>
    </GetQuote>
  </soap:Body>
</soap:Envelope>') )  ) as VARCHAR(2000)) )
COLUMNS
 "Symbol"  VARCHAR(4),
 "Last"    DECIMAL(6,2),
 "Date"    VARCHAR(10),
 "Time"    VARCHAR(8),
 "Change"  VARCHAR(8),
 "Open"    VARCHAR(8),
 "High"    VARCHAR(8),
 "Low"     VARCHAR(8),
 "Volume"  VARCHAR(12) ) XT
);
```

To execute the StockQuote function we just created:

```
SELECT * FROM TABLE( STOCKQUOTE('IBM')) as X;
```

The expected output should be similar to the following:

```
SYMBOL LAST     DATE       TIME     CHANGE   OPEN     HIGH     LOW
VOLUME
```



```
IBM    131.67   9/23/2010  4:01pm   -0.90    131.42   132.78   131.22
3898904
```

If you see error in executing StockQuote, you can check the status of the web services provider by going to the following site in a web browser like IE or FireFox:

```
http://www.webserviceX.NET
```

End of lab 11.

8. Answers for Selected Exercises

The following is a set of answers for the self-study exercises.

8.1 LAB7E9:

```
-- For ID 9, replace the value of shipping Address1 to
-- "999 Union Street"

UPDATE CUSTOMER

SET CUSTXML= XMLMODIFY(

 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";

  replace value of node

  /demopo:purchaseOrder/shipping/address/Address1

  with "999 Union Street"'

  )

WHERE ID =9;
```

8.2 LAB7E10:

```
-- For ID 10, replace the value the quantity of pid=1000
-- (attribute of "item") to 2
```



```
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
  'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";
  replace value of node
/demopo:purchaseOrder/items/item[@pid="1000"]/@quantity with "2"')
WHERE ID=10;
```

8.3 LAB9E9:

```
-- Sample solution of converting an application that does
-- bitemporal into DB2's solution for bitemporal.
DROP TRIGGER TR1;
DROP TRIGGER TR2;
DROP TRIGGER TR3;
DROP TRIGGER TR4;
DROP TRIGGER TR5;
DROP TRIGGER TR6;
ALTER TABLE TEST ALTER SYS_START SET DATA TYPE TIMESTAMP(12);
ALTER TABLE TEST_HISTORY ALTER SYS_START SET DATA TYPE TIMESTAMP(12);
ALTER TABLE TEST ALTER SYS_START
  SET GENERATED ALWAYS AS ROW BEGIN;
ALTER TABLE TEST ALTER SYS_END SET DATA TYPE TIMESTAMP(12);
ALTER TABLE TEST_HISTORY ALTER SYS_END SET DATA TYPE TIMESTAMP(12);
ALTER TABLE TEST ALTER SYS_END
  SET GENERATED ALWAYS AS ROW END  ;
ALTER TABLE TEST ADD   TRANS_ID  TIMESTAMP(12)
      GENERATED ALWAYS AS TRANSACTION START ID;
ALTER TABLE TEST_HISTORY ADD   TRANS_ID  TIMESTAMP(12);
ALTER TABLE TEST ADD PERIOD SYSTEM_TIME(SYS_START,SYS_END);
ALTER TABLE TEST ADD PERIOD BUSINESS_TIME(BUS_START,BUS_END);
CREATE UNIQUE INDEX IX ON TEST(ID,BUSINESS_TIME WITHOUT OVERLAPS);
ALTER TABLE TEST ADD VERSIONING USE HISTORY TABLE TEST_HISTORY;
COMMIT;
INSERT INTO TEST VALUES(2,1,'01/01/2010','12/31/9999',
   DEFAULT,DEFAULT,DEFAULT);
INSERT INTO TEST VALUES(2,5,'05/01/2010','06/01/2010',
   DEFAULT,DEFAULT,DEFAULT);
COMMIT;
UPDATE TEST FOR PORTION OF BUSINESS_TIME FROM
  '05/01/2010' TO '06/01/2010'
SET COUNT=10 WHERE ID = 2;
COMMIT;
SELECT * FROM TEST;
SELECT * FROM TEST_HISTORY;
DELETE FROM TEST where id=2;
COMMIT;
SELECT * FROM TEST;
SELECT * FROM TEST_HISTORY;
```

Note: you will get -803 after the 2nd INSERT statement. **This error is expected** as this show how BUSINESS_TIME is now in effect by checking the overlapping index.

```
DSNT408I SQLCODE = -803, ERROR:  AN INSERTED OR UPDATED VALUE IS
INVALID BECAUSE INDEX IN INDEX SPACE IX CONSTRAINS COLUMNS OF THE TABLE
SO NO TWO ROWS CAN CONTAIN DUPLICATE VALUES IN THOSE COLUMNS.
```

8.4 LAB10E2:

```
-------------------------------------------------------------------------
-- Create enabled row permissions for  CUSTOMERSERV(userid IOD02SB)
-- and TELEMARKETING(userid IOD02SC)
-------------------------------------------------------------------------
  CREATE PERMISSION  CUSTOMERSERV_ROW_ACCESS ON  CUSTOMER
    FOR ROWS WHERE
      VERIFY_GROUP_FOR_USER (SESSION_USER, 'IOD02SB') = 1
      OR
      VERIFY_GROUP_FOR_USER (SESSION_USER, 'IOD02SC') = 1

  ENFORCED FOR ALL ACCESS
  ENABLE#
```

8.5 LAB10E3:

```
-------------------------------------------------------------------------
-- Create enabled column mask for column ACCOUNT
--   - user IOD02SC TELEMARKETING
--     cannot see customer's full account number, only the last 4
--     digits
--   - user IOD02SA TELLER
--   - user IOD02SB CUSTOMERSERV
--     can see customer's full account number
-------------------------------------------------------------------------
  CREATE MASK  ACCOUNT_COLUMN_MASK ON  CUSTOMER
    FOR COLUMN ACCOUNT RETURN
      CASE WHEN (VERIFY_GROUP_FOR_USER (SESSION_USER, 'IOD02SC') = 1)
             THEN CHAR('xxxx-xxxx-xxxx-') || SUBSTR(ACCOUNT,16,4)
             ELSE  ACCOUNT
      END
  ENABLE
```


Acknowledgements and Disclaimers:

Availability. References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS-IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

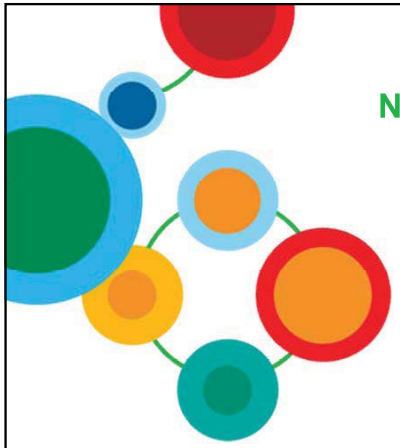
All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

SQLADRIA SEMINAR
Opatija, 24 - 25 November 2011

Improved Performance and Reduced CPU

Presentation

Sponsored by
IBM



New Features in IBM DB2 10 for z/OS II – Improved Performance and Reduced CPU

Session Number 1259

Jane Man, IBM
Guogen Zhang, IBM
Yonghua Ding, IBM
Lily Chen, IBM

IBM Software
Information On Demand 2011

Other DB2 V10 Labs..

- **Session: IDZ-1259A New Features in IBM DB2 10 for z/OS II - Improved Performance and Reduced CPU**
  Time: Mon, 24/Oct, 02:15 PM - 05:15 PM
  Location: Mandalay Bay South Convention Center
  - Shorelines B Lab Room 1
- **Session: IDZ-1267A New Features in IBM DB2 10 for z/OS III - Improve Development Productivity**
  Time: Tue, 25/Oct, 09:45 AM - 12:45 PM
  Location: Mandalay Bay South Convention Center - Shorelines B Lab Room 8
- **Session: IDZ-1256A New Core Features and Enhancement in IBM DB2 10 for z/OS**
  Time: Wed, 26/Oct, 09:45 AM - 12:45 PM
  Location: Mandalay Bay South Convention Center - Shorelines B Lab Room 8

Agenda

- Introduction (approx. 30 – 45 min)
  - Sub-document update on XML document
  - Binary XML
  - INLINE LOB
  - Dynamic Statement Cache Enhancement
  - Hash Access
  - Additional Non-Key Column in an Index (INCLUDE)
- Lab (approx. 2 hrs)
  - Lab 0 – work together to ensure everyone has connectivity to z/OS machine
  - The rest of labs are on your own pace. There are no dependency among these labs and you can start with anyone you prefer.

Agenda (cont'd)

- Lab 7 - Sub-document update on XML document
- Lab 16 - INLINE LOB
- Lab 17 - Dynamic Statement Cache Enhancement
- Lab 18 – Hash Access
- Lab 19 – Additional Non-Key column in an Index (INCLUDE)
- Lab 20 – Binary XML Support

Sub-document Update of XML documents

Sub-document Update

- No easy way to update parts of a document in V9
- Sub-document update with multi-versioning in DB2 10
- Only simple update: insert, replace, delete from XQuery update facility

Example

Increase premium by 10%

```
UPDATE POLICYTAB SET POLICY =
XMLMODIFY
( 'replace value of node /policy/premium with /policy/premium * 1.1')
WHERE POLICYID = '12345';
```

Add a new payment into payment record

```
UPDATE POLICYTAB SET PAYMENTS = XMLMODIFY
('insert node $n as last into /payments', :newpayment as "n")
WHERE POLICYID = :policyid;
```


Sub-document Update (cont'd)

- XMLMODIFY can only be used in RHS of UPDATE SET.
- One updater at a time for a document, concurrency control by the base table – row level locking, page level locking etc.
  - Document level lock to prevent UR reader
- Only changed rows in XML table are updated

Sub-document Update – Insert Expression (1 of 3)

Sample XML document:

```
<person>
  <firstName>Joe</firstName>
  <lastName>Smith</lastName>
  <nickName>Joey</nickName>
</person>
```

Sample insert statement:

```
update personinfo set info = xmlmodify('
  insert node $ins/ename
  after  /person/nickName', xmlparse(document '
      <ename>Joe.Smith@de.ibm.com</ename>') as  "ins") ;
```

Insert Operation	Resulting XML document
insert node $ins/ename into /person , XMLPARSE(document ' <ename>Joe.Smith@de.ibm.com</ename>') as "ins") (nondeterministic position)	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> **<ename>Joe.Smith@de.ibm.com</ename>** </person>
insert node $ins/ename as last into /person, XMLPARSE(document ' <ename>Joe.Smith@de.ibm.com</ename>') as "ins")	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> **<ename>Joe.Smith@de.ibm.com</ename>** </person>

Sub-document Update – Insert Expression (2 of 3)

Sample XML document:

```
<person>
  <firstName>Joe</firstName>
  <lastName>Smith</lastName>
  <nickName>Joey</nickName>
</person>
```

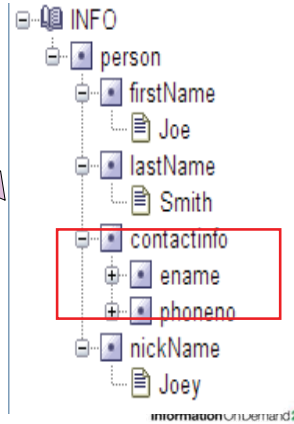
Insert Operation	Resulting XML document
Insert node $ins/ename **as first into** /person, XMLPARSE(document <ename>Joe.Smith@de.ibm.com</ename>') as "ins")	<person> **<ename>Joe.Smith@de.ibm.com</ename>** <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> </person>
Insert node $ins/ename **after** /person/nickName , XMLPARSE(document '<ename>Joe.Smith@de.ibm.com</ename>') as "ins")	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> **<ename>Joe.Smith@de.ibm.com</ename>** </person>
Insert node $ins/ename **before** /person/nickName , XMLPARSE(document ' <ename>Joe.Smith@de.ibm.com</ename>') as "ins")	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> **<ename>Joe.Smith@de.ibm.com</ename>** <nickName>Joey</nickName> </person>

Sub-document Update – Insert Expression (3 of 3)

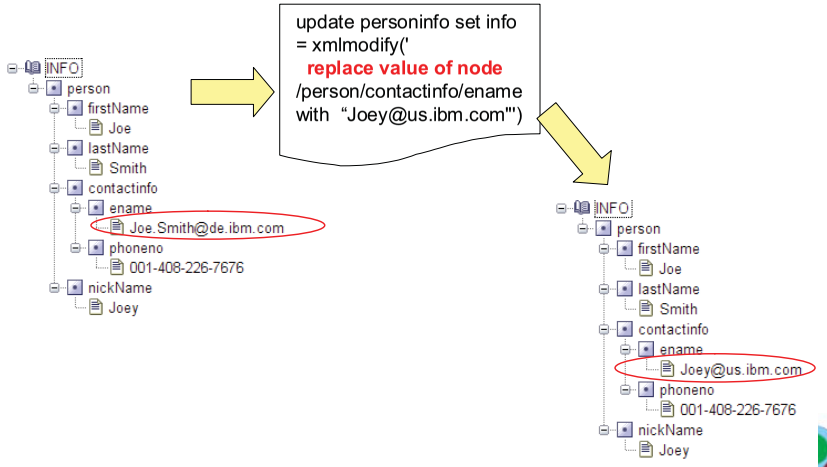
- Insertion of sequence of elements also possible
- Use same keywords as introduced before

```
<person>
  <firstName>Joe</firstName>
  <lastName>Smith</lastName>
  <nickName>Joey</nickName>
</person>
```

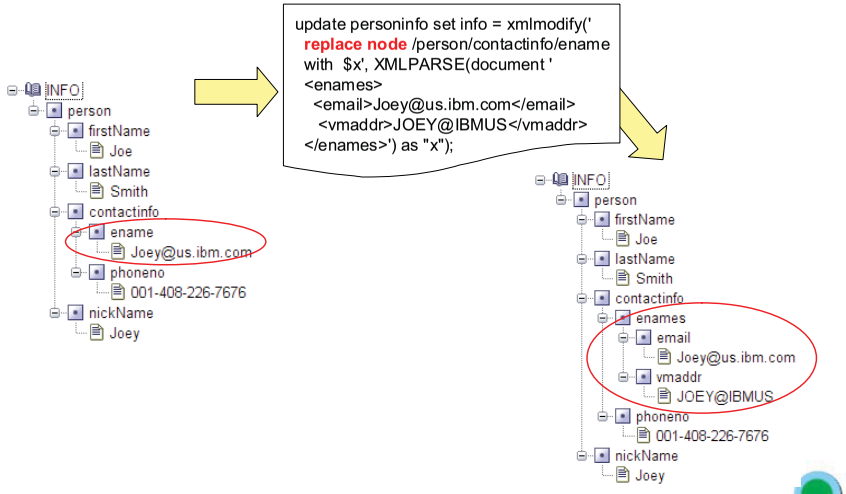
```
update personinfo set info = xmlmodify('
  insert node $ins/contactinfo
before /person/nickName',
xmlparse(document '
<contactinfo>
<ename>Joe.Smith@de.ibm.com</ename>
<phoneno>001-408-226-7676</phoneno>
</contactinfo>
') as "ins") ;
```



Sub-document Update – Replace Expression (1 of 2)

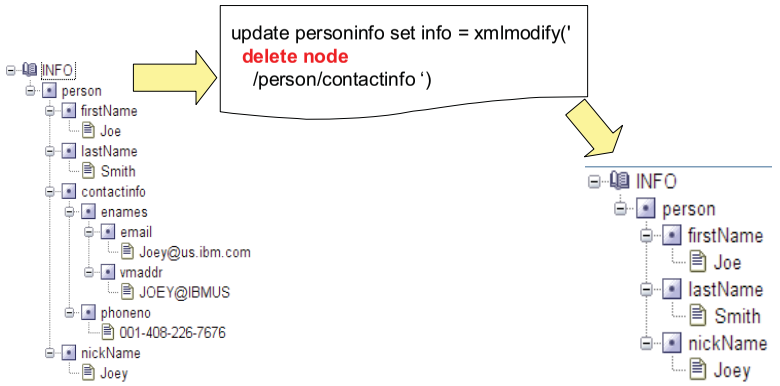


Sub-document Update – Replace Expression (2 of 2)



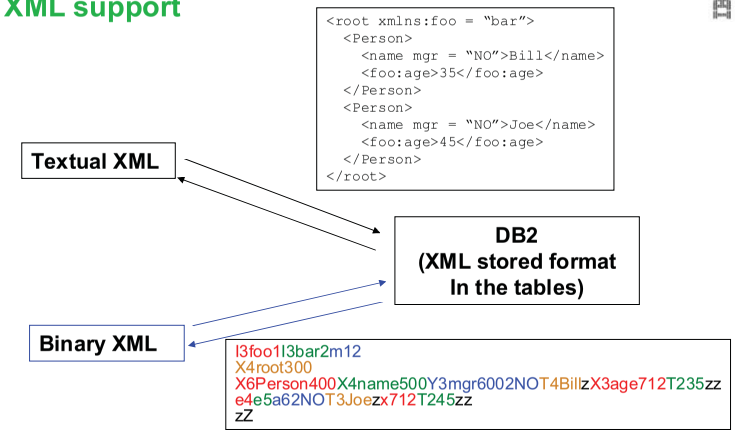
Sub-document Update – Delete Expression

– Delete can be used to delete nodes from a node sequence



Binary XML Support

Binary XML support



-- Binary XML is about 17%-46% smaller in size
-- Save DB2 over 9%-30% CPU during insert,
-- End to end time saving 8%-50% for insert

Usage: Application using JDBC 4.0 (JDK6.0)

Use DB2SimpleDataSource.setXmlFormat() to turn on and off binary XML support explicitly.

```
DB2SimpleDataSource ds = new DB2SimpleDataSource();

//to turn on binary XML
ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_BINARY);

//to turn on textual XML
ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_TEXTUAL);
```

Usage: Application using JDBC 4.0(JSR-221) -1/2

Fetch XML as SQLXML type:

```
String sql = "SELECT xml_col from T1";
PreparedStatement pstmt = con.prepareStatement(sql);
ResultSet resultSet = pstmt.executeQuery();

// get the result XML as SQLXML
SQLXML sqlxml = resultSet.getSQLXML(column);

// get a DOMSource from SQLXML object
DOMSource domSource = sqlxml.getSource(DOMSource.class);
Document document = (Document) domSource.getNode();

// or: get a SAXSource from SQLXML object
SAXSource saxSource = sqlxml.getSource(SAXSource.class);
XMLReader xmlReader = saxSource.getXMLReader();
xmlReader.setContentHandler(myHandler);
xmlReader.parse(saxSource.getInputSource());

// or: get binaryStream or string from SQLXML object
InputStream binaryStream = sqlxml.getBinaryStream(); // or:
String xmlString = sqlxml.getString();
```

New SQLXML type

Retrieve DOM tree

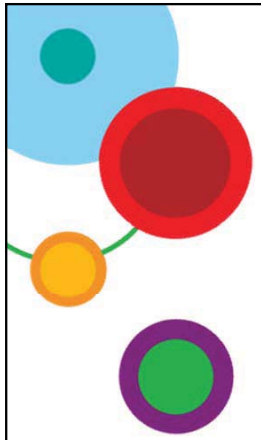
Or get SAX events

SQLXML object can only be read once

Usage: Application using JDBC 4.0(JSR-221) -2/2

Insert and update XML using SQLXML

```
String sql = "insert into T1 values(?)";
PreparedStatement pstmt = con.prepareStatement(sql);

SQLXML sqlxml = con.createSQLXML();

// create a SQLXML object from a string
sqlxml.setString(xmlString);

// set that xml doucment as the input to parameter marker 1
pstmt.setSQLXML(1, sqlxml);

pstmt.executeUpdate();

Sqlxml.free();
```

INLINE LOB

INLINE LOB

- A LOB resides
  - completely in the base table along with other non-LOB columns OR
  - partially in the base table along with other non-LOB columns and partially in the LOB table space. That is, a LOB is split between base table space and LOB table space.
- Conditions for enabling inline LOB attribute
  - DB2 needs to be in New Function Mode
  - Table space needs to be an Universal Table Space (PBG or PBR)
  - Table space is in RRF format (Reordered Row Format)

How to specify an inline LOB

- A new ZPARM(LOB_INLINE_LENGTH) is introduced for the DB2 subsystem level (default inline length) – default 0
- CREATE TABLE: a new clause INLINE LENGTH is introduced
- ALTER TABLE ADD column: a new clause INLINE LENGTH is introduced
- CREATE DISTINCT DATA TYPE: a new clause INLINE LENGTH is introduced

Examples: How to specify an inline LOB?

- CREATE DISTINCT DATA TYPE mylobtype1 CLOB(10K) INLINE LENGTH(20)
- CREATE TABLE mytab (col1 CLOB(10K) INLINE LENGTH 20)
- ALTER TABLE mytab ADD col1 CLOB(10K) INLINE LENGTH 50
- Note: zPARM serves as a default value for inline LOB length, statement level's attribute will override zparm's setting.

How to know the inline attribute for a LOB column?

```
CREATE TABLE TB01
 (COLVARCHAR VARCHAR(100),
  COLCLOB CLOB(1M) INLINE LENGTH 100);
```

- SELECT NAME, LENGTH FROM SYSIBM.SYSCOLUMNS WHERE TBNAME = 'TB01 AND NAME = 'COLCLOB'
- LENGTH – 4 = inline length

```
-----------------
| LENGTH   |
----------------
|     104     |
-----------------
```


Dynamic Statement Cache Enhancement

Background

Some customers and vendors choose not to use parameter marker coding ('?') in their applications SQL, but instead they use an actual literal value –

```
SELECT X, Y, Z FROM TABLE1 WHERE X < ?
```

vs.

```
SELECT X, Y, Z FROM TABLE1 WHERE X < 9
```


Background cont.

- With DB2 dynamic SQL statement caching, transactions that issue dynamic SQL containing literal constants instead of parameter markers
  - do not get the performance benefit of cached statement reuse, but instead incur dynamic prepare cost each time one or more transactions issue the same SQL statement but with different literal constants than were cached with the statement.
  - can eventually flush out of the cache those dynamic statements that both use parameter markers and have higher frequency of cache reuse.

Introduction

In DB2 V10, we allow a higher cache reuse from dynamic statements that reference literal constants.

This new behavior will be requested by specifying a new clause,

CONCENTRATE STATEMENTS WITH LITERALS

in the ATTRIBUTES string of the PREPARE . The default is CONCENTRATE STATEMENTS OFF.

Introduction – cont.

- When the new clause is specified, DB2 will replace certain literal constants in the statement text with a new special marker, '&', such that the literal constants are not cached with the SQL statement text.
- This modified statement text containing '&' for a literal will be inserted into the dynamic statement cache.
- A reoptimization re-prepare under **REOPT(AUTO)** behavior can override the literal replacement behavior – literals will not be replaced with '&'

Examples

```
EXEC SQL DECLARE C1 CURSOR FOR DYNSQL_WITH_LITCONST;

DYNSQL_SELECT =
       'SELECT X, Y, Z FROM TABLE1 WHERE X < 9';
attrstring = 'CONCENTRATE STATEMENTS WITH LITERALS';

EXEC SQL PREPARE DYNSQL_WITH_LITCONST ATTRIBUTES
  :attrstring  FROM :DYNSQL_SELECT;
```

New text after parse by Parser:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < &
```

Both cache insert and cache match search will use this new text.

JDBC Support

The JCC T2 and T4 Drivers for DB2 10 for z/OS provide a new datasource/connection property named **statementConcentrator.**

When user specifies

statementConcentrator=DB2BaseDataSource.STATEMENT_CONCENTRATOR_WITH_LITERALS

the JCC Drivers will specify **CONCENTRATE STATEMENTS WITH LITERALS** for dynamic SQL statements that JCC sends to DB2 10.

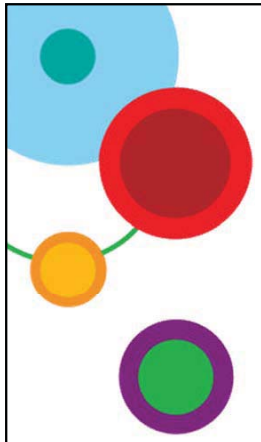
ODBC Support

The ODBC/CLI Driver for DB2 10 for z/OS provides a new keyword named **LITERALREPLACEMENT** in the DB2 ODBC initialization file.

When **LITERALREPLACEMENT = YES**,

ODBC will specify **CONCENTRATE STATEMENTS WITH LITERALS**

for dynamic SQL statements the driver sends to DB2 10.



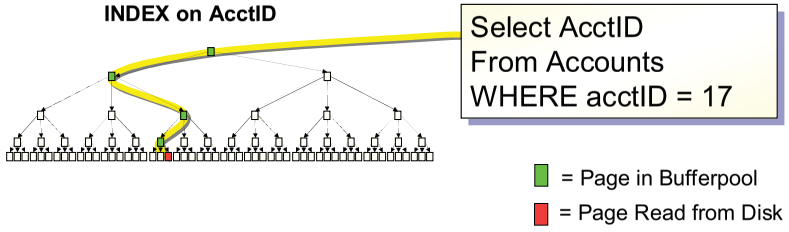
Hash Access

Objectives

- Faster access method for OLTP
  - Single row fetch using a fully qualified unique key
- Conversion
  - Easily convert existing tables for Hash Access

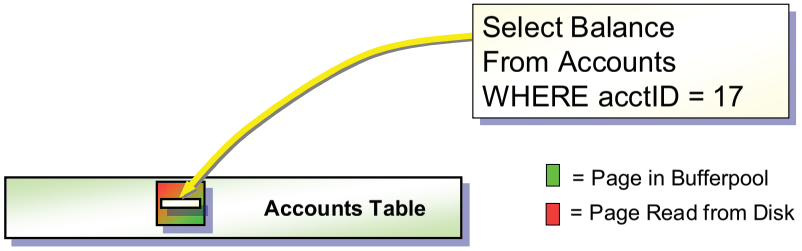
.

Index Only Access Path



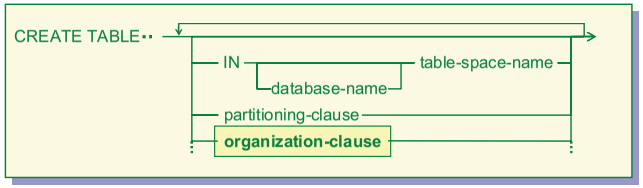
- Traverse down Index Tree
  - Typically non-leaf portion of tree in the bufferpool
  - **For Random Accesses**, Leaf Pages require I/O
  - Requires searching pages at each level of the index
- No Access to the Data Page
- For a 5 Level Index
  - 5 GETP/RELPs, 1 I/O's, and 5 index page searches

Hash Access



- Hash Access provides the ability to directly locate a row in a table without having to use an index
- Single GETP/RELP in most cases
- 1 Synch I/Os in common case
  - 0 If In Memory Table
- **Greatly reduced Search CPU expense**

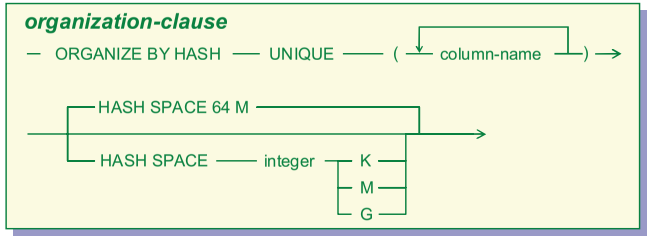
CREATE TABLE and the New *Organization-clause*



CREATE TABLE...
...
PARTITION BY RANGE...
PARTITION 1 ...
....
PARTITION 5 .....*HASH SPACE 1G*
...
ORGANIZE BY HASH UNIQUE
(lastname,firstname)
HASH SPACE 2G

- Hash organization requires UTS
- If table space not specified with IN clause, DB2 will create one implicitly

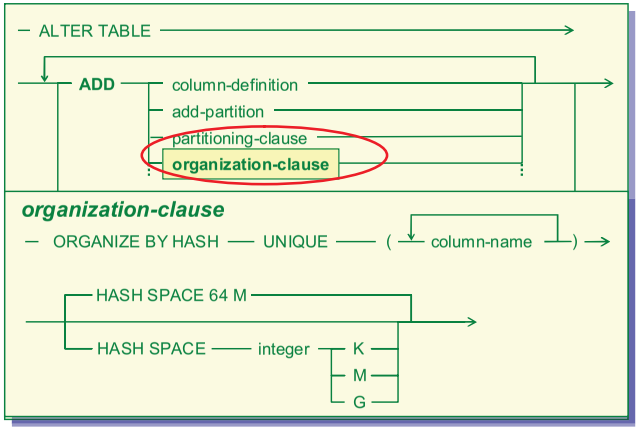
Organization-clause




```
CREATE TABLE…
  …
  PARTITION BY RANGE…
    PARTITION 1 …
    ….
    PARTITION 5 ….HASH SPACE 1G
  …
  ORGANIZE BY HASH UNIQUE
    (lastname,firstname)
    HASH SPACE 2G
```

- **Hash key**:
  - Enforces uniqueness
  - Is not updatable
    - Delete/Insert needed
  - Can enforce Referential Constraint
- **Hash Space**:
  - Specifies size of fixed hash space

ALTER TABLE ADD HASH



- Alter the table to make it organized by hash
  - Table is accessible
  - Changed at next Reorg

Examples Where Hash Access is Used

- Simple hash access
  - **SELECT * FROM T1 WHERE HASHCOL = :HV**
  - **SELECT * FROM T1 WHERE HASHCOL IN (SELECT C1 FROM T2)**
- List prefetch/multi-index access with hash access
  - **SELECT * FROM T1 WHERE (HASHCOL = :HV1 OR INDEXCOL = :HV2)**
- Referential Integrity
  - Parent key check on INSERT/UPDATE/LOAD/CHECKDATA uses hash access if available
  - Cascade Delete/Set Null do not use hash access even if available
- No parallelism with hash access
- No hash access with star join or hybrid join

Deciding to Use Hash Organization

CAUTION

Before choosing Hash Organization for your table consider:

- ✓ That the table has a unique key
- ✓ The table is primarily used for index probes
- ✓ The look ups are random
  - If existing table, verify using IFCID 199 that the look ups are truly random
- ✓ The table is relatively static in size
- ✓ The table is relatively less volatile
  - Increased insert/update/delete activity may result in more overflow rows, causing performance degradation
- ✓ The row sizes are not drastically different
- ✓ There are multiple rows per page (ideally 20 and above)
- ✓ More often than not, the requested row is found in the table
- ✓ Deeper indexes show more marked improvement with hash
- ✓ 20% more space is not a problem

Additional Non-Key Column In An Index (INCLUDE)

Background

In the previous version of DB2, primary key on a table require a unique index to be defined and that index could contain only the primary key columns.

```
CREATE TABLE CUSTTBL (ACCTID INT NOT NULL,
                      NAME VARCHAR(30) NOT NULL,
                      ADDRESS VARCHAR(50) NOT NULL);
CREATE UNIQUE INDEX INDEX1 ON CUSTTBL (ACCTID);
```

To support the following SELECT:

```
SELECT NAME FROM CUSTTBL WHERE ACCTID=10;
```

We need to create a separate additional index

```
CREATE UNIQUE INDEX INDEX2 ON CUSTTBL (ACCTID,
  NAME);
```

-> unnecessary overheads to INSERT/UPDATE/DELETE to maintain 2 indexes!

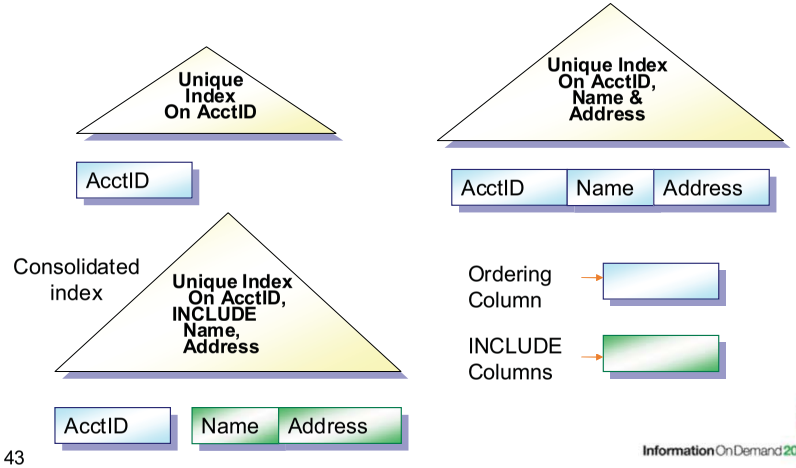
DB2 10 allows columns other than the unique key to be specified in the index definition.

New Syntax in CREATE INDEX & ALTER INDEX

```
CREATE TABLE CUSTTBL (ACCTID INT NOT NULL,
                      NAME VARCHAR(30) NOT NULL,
                      ADDRESS VARCHAR(50) NOT NULL);

CREATE UNIQUE INDEX CUST1 ON CUSTTBL (ACCTID)
  INCLUDE(NAME);

ALTER INDEX CUST1 ADD INCLUDE COLUMN (ADDRESS);
```


INCLUDE columns at a glance

DB2 10 allows columns other than the unique key to be specified in the index definition.



Overview

- What is an “ordering” column vs. an INCLUDE column?
  - Ordering columns control the “unique” index key sequencing
    - Must have at least 1 Ordering column
  - INCLUDE columns do not participate in the ordering/sequencing of the index keys
- How do INCLUDE columns fit into the basic index key?
  - An INCLUDE column must follow at least 1 unique column
  - Maximum number of total index key columns is still 64, so maximum number of INCLUDE columns is 63

How to determine if the index has INCLUDE columns?

Query the catalog SYSIBM.SYSINDEXES

- UNIQUE_COUNT
  - Value > 0, index has INCLUDE columns
  - Value = 0, index does not have include columns

Advice for customers

- **When should a customer consider consolidating indexes by using INCLUDE columns?**
  - When the customer determines they have multiple unique indexes with the same leading columns
- **How can a customer determine which indexes can be consolidated?**
  - A sample query is available that ***may*** help to identify indexes that could be consolidated by using INCLUDE columns. Please see Appendix in the lab instruction. Note: this is only a guide!

Sample output from the query:

	DBID	SIMPLEST_UNIQUE_IX	OBID	IX_WITH_CAND_INCLUDE_COLS	OBID	COLTYPE	COLNAME
1_	274	ADMF001.CUST1	4	ADMF001.CUST2	6	UNIQUE	ACCTID
2_						INCLUDE	NAME
3_						INCLUDE	ADDRESS

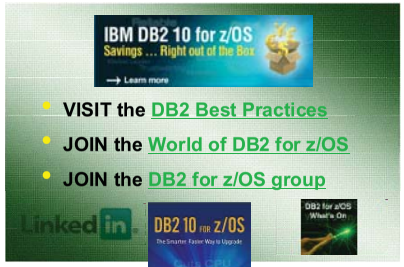
An Exclusive Invitation for System z Attendees

Thank You… Leveraging the Best of z!

"The benchmarks and analysis of functionality and performance have exceeded our expectations. So far our upgrades have gone smoothly and we are looking forward to completing our successful rollout of DB2 10"

–Verizon

"We had migrated five sub-systems to DB2 10 and have had no reported application issues running on this release to date."
–LabCorp



"The 'overall performance' in DB2 10 is better compared to DB2 9."

–HUK Coberg

[The Temporal Data] feature will drastically save developer time, test time … and improve business efficiency and effectiveness …
–Bankdata

Our regression tests showed performance improvements just by running the workload on a DB2 10 CM member…
–Dillards

Acknowledgements and Disclaimers:

Availability. References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS-IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

Communities

- **On-line communities, User Groups, Technical Forums, Blogs, Social networks, and more**
  - Find the community that interests you…
    - **Information Management** ibm.com/software/data/community
    - **Business Analytics** ibm.com/software/analytics/community
    - **Enterprise Content Management** ibm.com/software/data/content-management/usernet.html
- **IBM Champions**
  - Recognizing individuals who have made the most outstanding contributions to Information Management, Business Analytics, and Enterprise Content Management communities
    - ibm.com/champion

Thank You!
Your Feedback is Important to Us

- Access your personal session survey list and complete via SmartSite
  - Your smart phone or web browser at: iodsmartsite.com
  - Any SmartSite kiosk onsite
  - Each completed session survey increases your chance to win an Apple iPod Touch with daily drawing sponsored by Alliance Tech

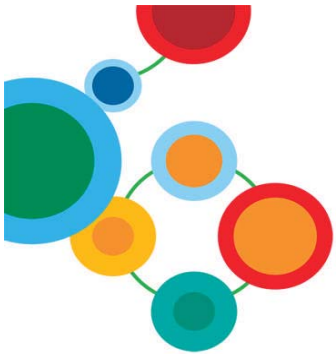
SQLADRIA SEMINAR
Opatija, 24 - 25 November 2011

Improved Performance and Reduced CPU

Hands on Lab

Sponsored by

IBM



New Features in IBM DB2 10 for z/OS II – Improved Performance and Reduced CPU

Session Number 1259

Jane Man, IBM
Guogen Zhang, IBM
Yonghua Ding, IBM
Lily Chen, IBM

Contents

1. Introduction

1.1 Who are we?

We are from the DB2 development organization in IBM Silicon Valley Lab.

Jane Man
Advisory Software Engineer
janeman@us.ibm.com

Guogen Zhang
Distinguished Engineer
gzhang@us.ibm.com

Yonghua Ding
Advisory Software Engineer
dingy@us.ibm.com

Lily Chen
Software Engineer
chelin@us.ibm.com

1.2 Objectives

- Learn the following new features in DB2 10 for z/OS:
  - Performing sub-document update on an XML column to reduce the amount of log data written for updates to large XML documents, improving performance and reducing CPU and elapsed time.
  - Using the new LOB-INLINE_LENGTH option to reduce the memory consumption and improve CPU for LOB operations.
  - Using binary XML to limit the size of XML which improve overall performance of XML objects.

- Using dynamic statement cache enhancement for dynamic SQL containing literal constants instead of parameter markers.
- Including additional columns in unique index to avoid unnecessary overheads to maintain multiple indexes.
- Using hash access to reduce data access to a single I/O for single row fetch (in common case), dramatically decreasing the CPU workload and speeding up application response time.

1.3 Suggested reading

- Introduction to pureXML in DB2 9 for z/OS
  ftp://ftp.software.ibm.com/software/data/db2zos/presentations/2007/misc/purexml.pdf
- Leveraging DB2 9 for z/OS pureXML Technology
  http://www.ibm.com/developerworks/wikis/download/attachments/1824/Leveraging_DB29_for_zOS_whitepaper_v2.pdf
- DB2 Version 10 for z/OS XML Guide (SC19-2981-02)
  http://publib.boulder.ibm.com/epubs/pdf/dsnxgm02.pdf
- DB2 Version 10 for z/OS SQL Reference (SC19-2983-02)
  http://publib.boulder.ibm.com/epubs/pdf/dsnsqm02.pdf

1.4 Acknowledgement

Many thanks for the following people for their valuable input to make this HOL possible:

Kalpana Shyam

Ken Taylor

Li-Mey Lee

Mike Shadduck

2. Lab 0 – Introduction and Getting Started

2.1 LAB0E1 – Command Line Processor (CLP) setup

In this exercise, we will use the DB2 command line processor that ships with DB2 for z/OS. We will configure CLP and use it to execute a script to test the connectivity to the database.

This exercise will be done together as a class, so follow along with the lab instructor.

1. Open a Command Prompt window by double clicking the Command prompt icon in the desktop.
2. Change directory to the location of the hands-on-lab materials.

   In the command prompt window, type: `CD C:\HOL`

3. Start notepad to edit the `clpsetup.bat` file.
   type: `notepad clpsetup.bat`

4. Replace the "XXXXXXXX" and "YYYYYYYY" with the username and password provided to you by the lab instructor. Please use UPPERCASE for all userid and password.

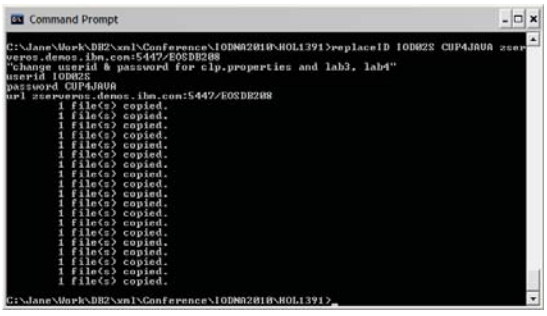
   Tip: You can quickly do find/replace in notepad by choosing "replace" from the Edit menu, or typing Ctrl-H at any time.

   Note that the alias "mydb2" has been created for you to direct CLP to connect to the server.

Save the file (from the File menu) and quit notepad (from the File menu).

5. Execute the clpsetup.bat batch file to define aliases and set environment variables to allow CLP to run. In the command prompt window, type:
   `clpsetup.bat`

   For a successful execution, you should see something similar to this:



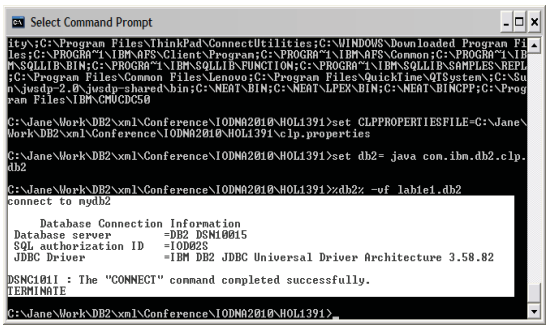
6. Browse the "lab0e1.db2" script. You do not have to change anything in this script (unless you want to).
   type: `notepad lab0e1.db2`

   Exit notepad when you are finished.

7. Execute the "lab0e1.db2" script. This script will connect to the database.
   type: `%db2% -vf lab0e1.db2`

   %db2% invokes the "db2" alias that we created in `clpsetup.bat`. The options "v" tells CLP to use verbose mode so it prints the commands and results to the screen, and "f" tells CLP to execute a file instead of running in interactive mode. "lab0e1.db2" is the script that we will run.

Verify that the "CONNECT" command completed successfully. If the screen looks as it does below (you may see different `Database Server` and `SQL authorization ID`), then the script ran successfully and this exercise is complete.



CLP will be used again in following labs. Leave the command prompt window open so we can easily invoke CLP again.

There is no dependency between these lab, so you can pick any lab to start with. Please note we use userid IOD02S for illustration. You should use your own userid assigned by the instructor.

End of lab 0.

3. Lab 7 – Learning sub-document update on an XML column

In DB2 9 for z/OS, an update on an XML document is a delete of whole document following by a new insert. In DB2 10 for z/OS, we are able to do sub-document update. In this set of lab exercises, we will explore this new feature.

3.1 LAB7E1 – Create and Populate CUSTOMER table

In this exercise, we will use CLP to create a CUSTOMER table and populate them with 10 XML documents with different IDs in the ID column. Each XML is the same and they will be used in the following exercises.

Below is the ddl of CUSTOMER table:

```
CREATE TABLE CUSTOMER (ID INTEGER, CUSTXML XML)
```

Below is the content of the XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<demopo:purchaseOrder id="0" orderDate="2001-01-01" shipDate="2001-01-01"
status="" xmlns:demopo="http://www.purchaseOrder.com/purchaseOrder"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.purchaseOrder.com/purchaseOrder
POSchema_unq3.xsd ">
  <customer id="0">
    <CustomerName>Jane Man</CustomerName>
  </customer>
  <shipping shippingName="Jane Man">
    <address>
      <Address1>555 Bailey Avenue</Address1>
      <City>San Jose</City>
      <State>CA</State>
      <Zip>95141</Zip>
    </address>
  </shipping>
  <items>
    <item itemName="Toyota Camry" pid="1000" price="20000" quantity="1"/>
  </items>
  <billing billingName="Guogen Zhang">
    <address>
      <Address1>123 Sesame Street</Address1>
      <City>San Jose</City>
      <State>CA</State>
      <Zip>95141</Zip>
    </address>
  </billing>
</demopo:purchaseOrder>
```

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e1.db2
```

For successful execution, you will see 10 records as output for

```
SELECT ID FROM CUSTOMER
```

statement, each with a different ID value.

3.2 LAB7E2 – Insert Before Exercise

In this exercise, we will learn about "insert before" sub-document update.

We want to insert

```
<shippingRemark>Little baby, please be quiet</shippingRemark>
```

before the `shipping` element for ID=2.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e2.db2
```

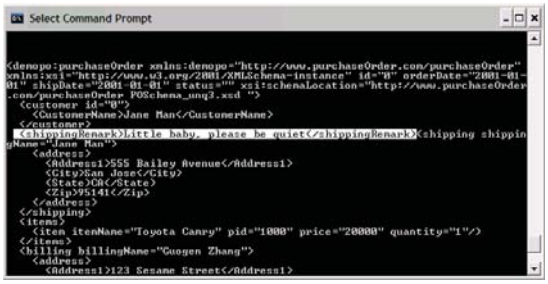
Below is the SQL statement:

```
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
 'declare namespace demopo="http://www.purchaseOrder.com/purchaseOrder";
  insert nodes $remark
  before /demopo:purchaseOrder/shipping',
  XMLPARSE(DOCUMENT
    '<shippingRemark>Little baby, please be quiet</shippingRemark> '
   preserve whitespace)
    AS "remark")
WHERE ID =2
```

Note:

1. namespace need to be declared (`declare namespace demopo=http://www.purchaseOrder.com/purchaseOrder;`) here, otherwise, DB2 will issue error as it cannot find any match to update.
2. the parameter passed to XMLPARSE (`'<shippingRemark>Little baby, please be quiet</shippingRemark> '`) must be a well-formed document, otherwise, DB2 will issues a non well-formed error.

After a successful execution, you should see the `shippingRemark` element is added before `shipping` element.



3.3 LAB7E3 – Insert After Exercise

In this exercise, we will learn about "insert after" sub-document update.

We want to insert

```
<survey>Excellent</survey>
```

after the `billing` element for ID=3.

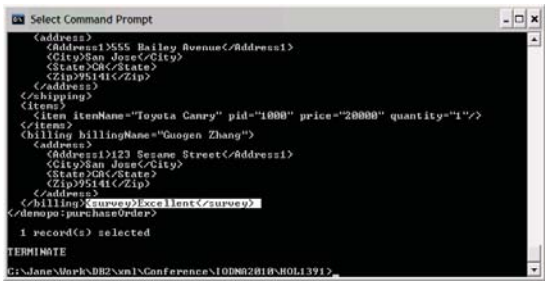
1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e3.db2
```

Below is the SQL statement:

```
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";
  insert nodes $survey
  after /demopo:purchaseOrder/billing',
  XMLPARSE(DOCUMENT
   '<survey>Excellent</survey>' preserve whitespace)
    AS "survey")
WHERE ID =3;
```

After a successful execution, you should see the `survey` element is added after `billing` element.



3.4 LAB7E4 – Insert As First Exercise

In this exercise, we will learn about “insert as first” sub-document update.

We want to insert a comment

`This is a urgent order!`

as first child in the `demopo:purchaseOrder` element for ID=4.

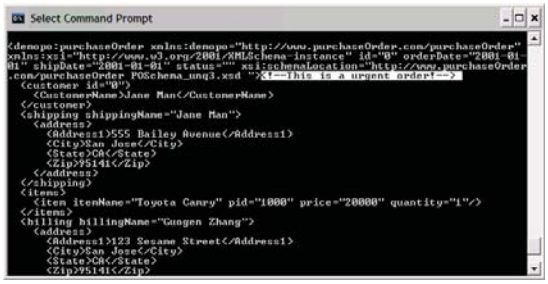
1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e4.db2
```

Below is the SQL statement:

```
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";
  insert nodes $remark
  as first into /demopo:purchaseOrder',
  (SELECT XMLCOMMENT('This is a urgent order!') FROM SYSIBM.SYSDUMMY1)
  AS "remark")
WHERE ID =4;
```

After a successful execution, you should see a comment is added as the first child of `demopo:purchaseOrder` element.



3.5 LAB7E5 – Insert As Last Exercise

In this exercise, we will learn about “insert as last” sub-document update.

We want to insert a new item

`<item itemName="WII" pid="2" price="200" quantity="1">`

`</item>`

as last `item` in the `items` element for ID=5.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e5.db2
```

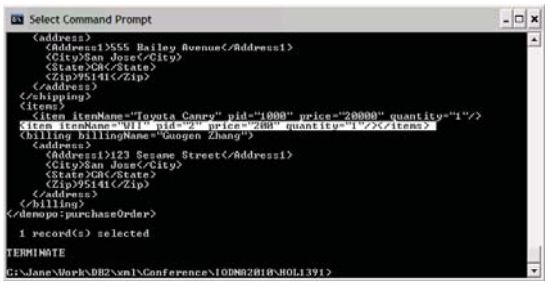
Below is the SQL statement:

```
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
 'declare namespace demopo="http://www.purchaseOrder.com/purchaseOrder";
  insert nodes $newitem
  as last into /demopo:purchaseOrder/items',
  XMLPARSE(DOCUMENT
```



```
  ' <item itemName="WII" pid="2" price="200" quantity="1"></item>'
PRESERVE WHITESPACE)
  AS "newitem")
WHERE ID =5;
```

After a successful execution, you should see a new item(for WII) is added as the last child of `items` element.



3.6 LAB7E6 – Delete Nodes Exercise

In this exercise, we will learn about “delete nodes” sub-document update.

We want to delete the item where pid=1000 (i.e the following item)

```
<item itemName="Toyota Camry" pid="1000" price="20000"
quantity="1"/>
```

in the `items` element for ID=6.

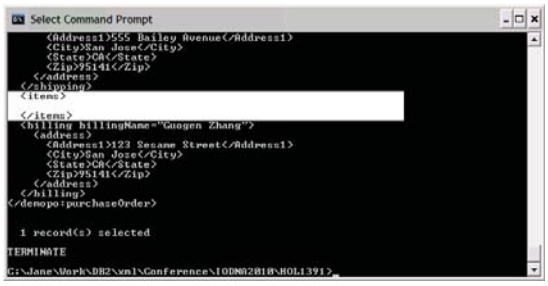
1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e6.db2
```


Below is the SQL statement:

```
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";
  delete nodes
   /demopo:purchaseOrder/items/item[@pid="1000"]'
  )
WHERE ID =6;
```

After a successful execution, you should not see any item inside the `items` element.



3.7 LAB7E7 – Replace Value of Node(attribute value) Exercise

In this exercise, we will learn about “replace value of node” sub-document update.

We want to replace the value of `status` (attribute in

/demopo:purchaseOrder element) to shipped for ID=7.

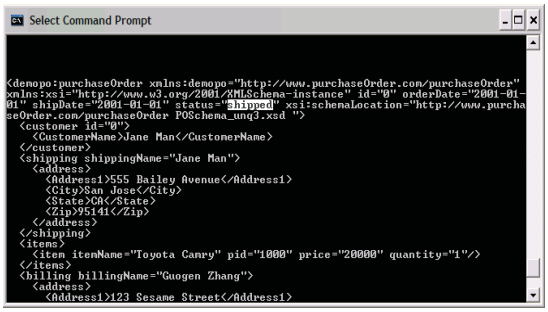
1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e7.db2
```

Below is the SQL statement:

```
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";
  replace value of node
   /demopo:purchaseOrder/@status with "shipped"'
  )
WHERE ID =7;
```

After a successful execution, you should see the attribute value of status has changed to shipped.



3.8 LAB7E8 – Replace Value of Node(sequence of nodes)

Exercise

In this exercise, we want to replace the value of whole shipping address of Guogen Zhang to

```
<address>
  <Address1>999 Brown Street</Address1>
  <City>Fremont</City>
  <State>CA</State>
  <Zip>94560</Zip>
</address>
```

for ID=8.

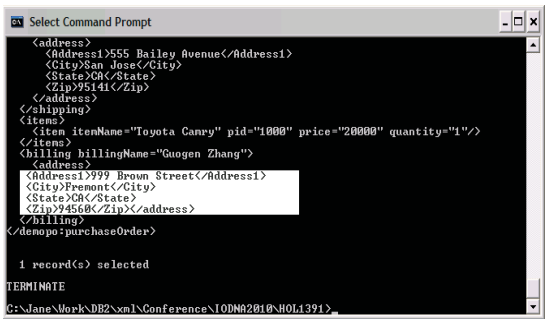
1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab7e8.db2
```

Below is the SQL statement:

```
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
 'declare namespace
demopo="http://www.purchaseOrder.com/purchaseOrder";
  replace value of node
  /demopo:purchaseOrder/billing[@billingName="Guogen Zhang"]/address
  with
  "
   <Address1>999 Brown Street</Address1>
   <City>Fremont</City>
   <State>CA</State>
   <Zip>94560</Zip>"
  '
  )
WHERE ID =8;
```


After a successful execution, you should see address of Guogen Zhang has been changed.



3.9 LAB7E9 – LAB7E10

The remaining lab exercises starting with member LAB7E9 and LAB7E10 are to be completed on your own.

The questions and the hints are described in the individual clp file The objective is to replace instances of "%%%%%%%%" with XPath or other expressions to complete the SQL statement.

Use CLP to execute these exercises.

The solutions are at the end of this document.

The exercises are listed here for your reference.

3.9.1 LAB7E9

```
-- For ID 9, replace the value of shipping Address1 to
-- "999 Union Street"
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
 'declare namespace demopo="http://www.purchaseOrder.com/purchaseOrder";
  %%%%%%%%
  /demopo:purchaseOrder/shipping/address/Address1
  with "999 Union Street"'
  )
WHERE ID =9;
```


3.9.2 LAB7E10

```
-- For ID 10, replace the value the quantity of pid=1000
-- (attribute of "item") to 2

UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
  'declare namespace demopo="http://www.purchaseOrder.com/purchaseOrder";
  replace value of node
/demopo:purchaseOrder/items/item[@pid="1000"]/@quantity with "%%%%%%%%"')
WHERE ID=10;
```

End of Lab 7.

4. LAB 16 - INLINE LOB

In DB2 V10, we have a new feature for handling LOB objects – inline LOB – a LOB resides

- completely in the base table along with other non-LOB columns OR
- partially in the base table along with other non-LOB columns and partially in the LOB tablespace. That is, a LOB is split between base table space and LOB table space.

Conditions for enabling inline LOB:

- DB2 needs to be in New Function Mode
- Table space needs to be an Universal Table Space(PBG or PBR)

- Table space is in Reordered Row Format(RRF)

4.1 LAB16E1 – Create objects using INLINE LENGTH

In this exercise, we will use CLP to create objects using the new INLINE LENGTH clause.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab16e1.db2
```

Below is the SQL statement for IOD02S

```
SET CURRENT RULES = 'STD';
DROP DATABASE IOD02SDB;
CREATE DATABASE IOD02SDB;
CREATE TABLESPACE MYTS MAXPARTITIONS 2 IN IOD02SDB DEFINE NO;
---------------------------------------------------------------
-- Create a LOB column with inline length = 100
---------------------------------------------------------------
CREATE TABLE MYTB(
  ROW_ID ROWID NOT NULL GENERATED ALWAYS,
  COLCLOB  CLOB(1G) INLINE LENGTH 100) IN IOD02SDB.MYTS;
```

We have created a COLCLOB column of inline length of 100. That is, when the size of LOB is less than or equal to 100, it will be stored inline with other columns in the base table. When the size of LOB is larger than 100, the first 100 bytes will be stored inline in base table while the rest will be stored in the LOB table space.

We also use `DEFINE NO` option so that the dataset is not created until the data is inserted.

4.2 LAB16E2 – Verify inline length in catalog

In this exercise, we are going to verify the inline length of the object we just created. LENGTH column in SYSIBM.SYSCOLUMNS table indicates whether

LOB column has inline attribute. If LENGTH > 4, it is an inline LOB column with defined inline length LENGTH - 4.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab16e2.db2
```

Below is the SQL statement for user IOD02S:

```
SELECT NAME, TBNAME, LENGTH FROM SYSIBM.SYSCOLUMNS WHERE
       TBNAME = 'MYTB' AND NAME = 'COLCLOB' AND
       TBCREATOR = 'IOD02S';
```

In the previous exercise, we have created table with inline length of 100. So the output should look like this:

```
NAME       TBNAME    LENGTH
COLCLOB    MYTB         104
  1 record(s) selected
```

Note: the extra 4 bytes are used for indicator/house-keeping.

4.3 LAB16E3 – Insert small objects that can be stored inline

In this exercise, we are going to insert some small objects (of size less than 100 bytes) that can be stored inline.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab16e3.db2
```

Below are the SQL statements:

```
INSERT INTO MYTB(COLCLOB) VALUES (REPEAT('A',99));
INSERT INTO MYTB(COLCLOB) VALUES ('ABCDE');
```


4.4 LAB16E4 – Check the space used

In this exercise, we check the space used for storing the small objects that we inserted in the previous exercise. The SPACE column in SYSIBM.SYSTABLEPART catalog table indicates the number of kilobytes of DASD storage allocated to the table space partition.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab16e4.db2
```

Below is the SQL statement for IOD02S:

```
SELECT TSNAME, SPACE FROM SYSIBM.SYSTABLEPART
WHERE DBNAME='IOD02SDB';
```

You should see something like this:

```
TSNAME               SPACE
LFBQ73XS              -1      <-
MYTS                  0
  2 record(s) selected
```

- `LFBQ73XS` is the LOB table space implicitly created by DB2. (Note: you may see different name for LOB table space.)

  SPACE value of -1 indicate the table space was defined with the DEFINE NO clause, which defers the physical creation of the data sets until data is first inserted into one of the partitions, and data has yet to be inserted. **In other words, this indicates whatever we have inserted in the previous exercise are NOT stored in the LOB table space, but they are physically stored inline in base table!**

- `MYTS` is the base table space we declared when we create the objects.

  SPACE value of 0 indicates the STOSPACE or RUNSTATS utility has not been run.

We need the table space names (both base and LOB) for the RUNTSTATS later. You can either write down the table space names or just leave the Command prompt window open.

4.5 LAB16E5 – Insert large LOB objects

In this exercise, we are going to insert some large LOB objects with size more than 100 bytes.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab16e5.db2
```

Below is the SQL statement:

```
INSERT INTO MYTB(COLCLOB) VALUES (REPEAT('ABCDE',200));
INSERT INTO MYTB(COLCLOB) VALUES (REPEAT('11111',100));
```

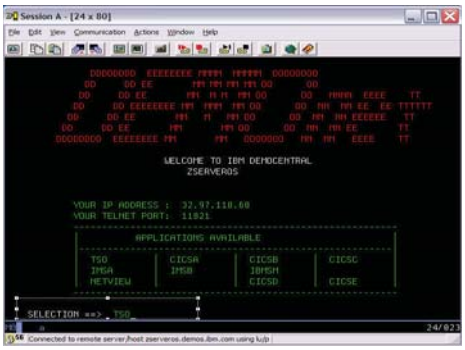
As you may recall, the inline length we declared is 100, so these 2 objects are too big to store inline in the base table.

4.6 LAB16E6 – RUNSTATS

In this exercise, we are going to run RUNSTATS which will update the catalog table. By doing so, we can tell how many storage are used to store these LOB objects.

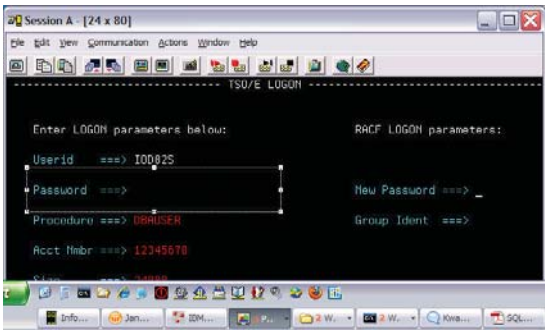
1. Double click the zserveros icon in the desktop. In case you prompt for password, just type `password`

2. Enter `TSO` under SELECTION and press <enter> key



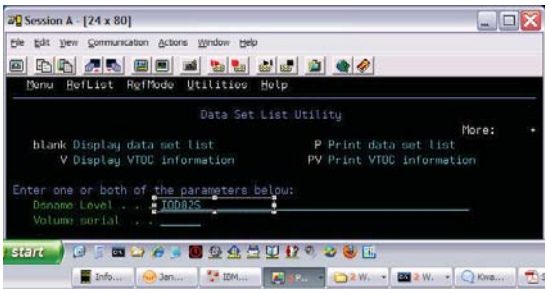
3. Enter your assigned userid and press <enter> key

4. IN the TSO/E LOGON console, enter your assigned password user Password field and then press <enter> key.

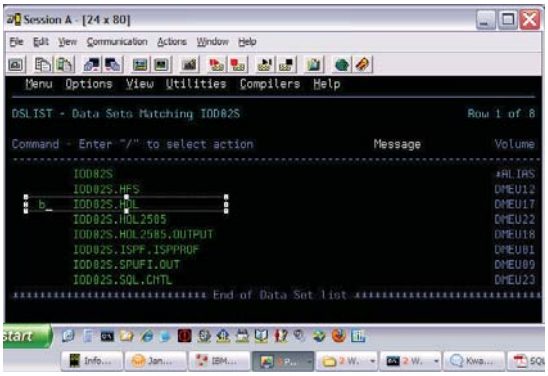


5. In the ISPF Option Menu, enter `3.4` under Option field and then press <enter> key.

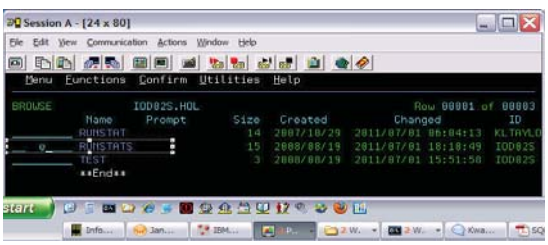
6. In the Data Set List Utility, enter your userid under Dsname Level field and then press <enter> key.



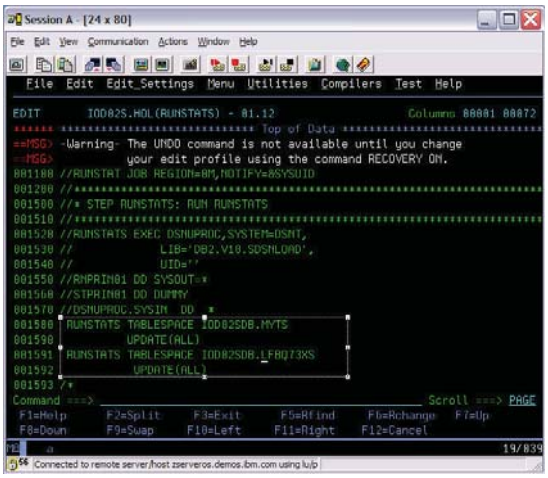
7. In DSLIST, type `b` under Command next to `userid.HOL` and then press <enter> key.



8. Enter `e` next to the RUNSTATS job and then press <enter> key.



9. Update the table space names to the output of LAB16E4. Make sure there are 2 RUNSTATS statements, 1 for base table space(MYTS) and 1 for LOB table space. Qualify the table space name with your assigned id. Below is what you will see for IOD02S.



10. Press F3 to exit

11. Enter `sub` next to RUNSTATS job and press <enter> key to submit the job

12. After you get a successful return code, go back to your CLP command window, type

```
%db2% -vf lab16e4.db2
```

You should see something like this:

```
TSNAME                SPACE
LFBQ73XS              7200  ←
MYTS                  720   ←
  2 record(s) selected
```

As we have inserted large LOB object, part of these objects are stored in LOB table space LFBQ73XS. This is indicated by non -1 value in SPACE column of SYSIBM.SYSTABLEPART.

End of Lab 16.

5. LAB 17 - Dynamic Statement Cache Enhancements

For dynamic SQL applications coded with parameter markers, DB2 allows dynamic SQL statements to be cached for performance purpose. However, transactions that issue dynamic SQL containing literal constants instead of parameter markers do not get the performance benefit of cached statement reuse, but instead incur dynamic prepare cost each time with one or more transactions issue the same SQL statement but with different literal constants.

In DB2 V10, we allow a higher cache reuse from dynamic statements that reference literal constants.

- In **host language applications**, this new behavior will be enabled by specifying a new clause,

  CONCENTRATE STATEMENTS WITH LITERALS

  in the ATTRIBUTES string of the PREPARE . The default is

  CONCENTRATE STATEMENTS OFF.

- In JAVA application, the JCC T2 and T4 Drivers for DB2 10 for z/OS provide a new datasource/connection property named **statementConcentrator .**

  When user specifies

  statementConcentrator=2

  where 2 stands for **DB2BaseDataSource.STATEMENT_CONCENTRATOR_WITH_LITERALS**

  the JCC Drivers will specify **CONCENTRATE STATEMENTS WITH LITERALS** for dynamic SQL statements that JCC sends to DB2 10.

- In ODBC application, the ODBC/CLI Driver for DB2 10 for z/OS provides a new keyword named **LITERALREPLACEMENT** in the DB2 ODBC initialization file. When **LITERALREPLACEMENT = YES**

  ODBC will specify **CONCENTRATE STATEMENTS WITH LITERALS** for dynamic SQL statements the driver sends to DB2 10.

In this lab, we will CLP (i.e. JAVA application) to demonstrate this enhancement.

5.1 LAB17CreateCacheTable – Create DSN_STATEMENT_CACHE_TABLE

In this exercise, we are going to create DSN_STATEMENT_CACHE_TABLE table. We will use this table to look at the statement text and other information about the statement cache.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab17CacheTable.db2
```

After a successful execution, DSN_STATEMENT_CACHE_TABLE will be created.

5.2 LAB17E1 – default behavior

In this exercise, we are going to create a table and insert 3 rows with literal constants without using this new enhancement. We then call EXPLAIN STMTCACHE and look at the DSN_STATEMENT_CACHE_TABLE.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab17e1.db2
```

Below are the SQL statements for IOD02S:

```
drop table lab17Table;
commit;
create table lab17table(num_col int,  char_col char,
  string_col varchar(30));
insert into lab17table values(1, '1', '111');
insert into lab17table values(2, '2', '222');
insert into lab17table values(3, '3', '333');


delete from DSN_STATEMENT_CACHE_TABLE;
EXPLAIN STMTCACHE ALL;
SELECT SUBSTR(CURSQLID,1,8) CURSQLID,
      SUBSTR(STMT_TEXT,1, 45) STMT_TEXT,
      LITERAL_REPL,
      STAT_EXECB
FROM DSN_STATEMENT_CACHE_TABLE
WHERE CURSQLID='IOD02S' AND STMT_TEXT LIKE '%lab17table%'
ORDER BY STMT_ID;
```

You may see the following as output from DSN_STATEMENT_CACHE_TABLE:

```
CURSQLID STMT_TEXT                        LITERAL_REPL STAT_EXECB
IOD02S   insert into lab17table values(1, '1', '111')            1
IOD02S   insert into lab17table values(2, '2', '222')            1
IOD02S   insert into lab17table values(3, '3', '333')            1
```


For the 3 insert statements, there are 3 rows in the DSN_STATEMENT_CACHE_TABLE, meaning they are considered as 3 different statements and prepared separately.

- **LITERAL_REPL** is a new column added to the

  DSN_STATEMENT_CACHE_TABLE to identify cached statement whose literals are replaced with '&'s. The '&'s are visible in the actual cached statement text. The new column LITERAL_REPL can have the following values:

  R The statement's literals were replaced with '&'

  D The same as value R, but is a duplicate statement instance with different literal reusability criteria

  blank The default. Literals not replaced with '&'.

- **STAT_EXECB** column in DSN_STATEMENT_CACHE_TABLE indicates the number of times this statement has been run. As seen in the output above, each INSERT statement is executed once.

5.3 LAB17E2 – Enable Dynamic Statement Cache Enhancement

In this exercise, we are going to enable dynamic statement cache enhancement using **statementConcentrator=2** in our connection to DB2. Then, we insert 3 more rows and finally look at the DSN_STATEMENT_CACHE_TABLE.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab17e2.db2
```

Below is the SQL statement for IOD02S:

```
connect to zserveros.demos.ibm.com:5447/EOSDB208:statementConcentrator=2;
user IOD02S using password;
insert into lab17table values(4, '4', '444');
insert into lab17table values(5, '5', '555');
```

```
insert into lab17table values(6, '6', '666');
```

You may see the following in the last part of the output (from DSN_STATEMENT_CACHE_TABLE) :

```
CURSQLID STMT_TEXT                          LITERAL_REPL STAT_EXECB
IOD02S   insert into lab17table values(1, '1', '111')            1
IOD02S   insert into lab17table values(2, '2', '222')            1
IOD02S   insert into lab17table values(3, '3', '333')            1
IOD02S   insert into lab17table values(&, &, &)   R              3
```

A new (last) row is added to DSN_STATEMENT_CACHE_TABLE with

- STMT_TEXT: all literal constants are replaced by &
- LITERAL_REPL: R – indicate the statement's literals were replaced with '&'
- STAT_EXECB: 3 – indicate this statement is executed for 3 times.

This illustrate we have reused the cache.

5.4 LAB17E3 – Disable Dynamic Statement Cache Enhancement

In this exercise, we are going to disable this enhancement, then insert 3 more rows and finally look at DSN_STATEMENT_CACHE_TABLE.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab17e3.db2
```

Below is the SQL statements:

```
insert into lab17table values(7, '7', '777');
insert into lab17table values(8, '8', '888');
insert into lab17table values(9, '9', '999');
```

You may see the following in the last part of the output (from DSN_STATEMENT_CACHE_TABLE) :

```
CURSQLID STMT_TEXT                          LITERAL_REPL STAT_EXECB
```



```
IOD02S   insert into lab17table values(1, '1', '111')            1
IOD02S   insert into lab17table values(2, '2', '222')            1
IOD02S   insert into lab17table values(3, '3', '333')            1
IOD02S   insert into lab17table values(&, &, &)   R              3
IOD02S   insert into lab17table values(7, '7', '777')            1
IOD02S   insert into lab17table values(8, '8', '888')            1
IOD02S   insert into lab17table values(9, '9', '999')            1
```

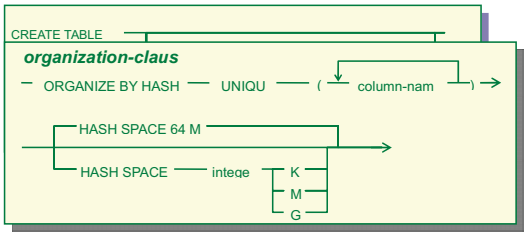
From this output, we can tell the last 3 INSERTS are no longer using the new enhancement.

End of lab 17.

6. LAB 18 – Hash Access

Hash access provides faster access for single row fetch using a fully qualified unique key. It provides the ability to directly locate a row in a table without having to use an index. In common case, we just need 1 synch I/Os and this greatly reduces search CPU expense.

We can enable hash access in CREATE TABLE or ALTER TABLE using the new *organization-clause*.



In this lab, we are going to learn how to use hash access in CREATE TABLE.

6.1 LAB18E1 – Create table with single col. hash key

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab18e1.db2
```

Below are the SQL statements (for IOD02S):

```
DROP DATABASE IOD02SDB;
CREATE DATABASE IOD02SDB;
CREATE TABLE TEST01TB (
            C1 CHAR(4) NOT NULL,
            C2 VARCHAR(8),
            C3 CHAR(4))
  ORGANIZE BY HASH UNIQUE (C1) HASH SPACE 1 M
  IN DATABASE IOD02SDB;
COMMIT;
```

We have created a table called TEST01TB with a single column (C1) hash key with hash space of 1M. The hash key will enforce uniqueness and is not updatable.

6.2 LAB18CAT1 – Select from Catalog

In this exercise, we are going to look into DB2 catalog to see some hash access related info for the table we have just created.

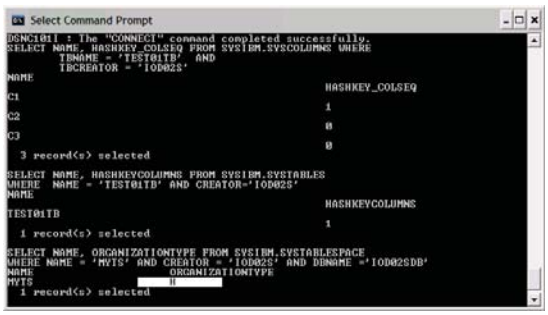
1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab18cat1.db2
```

Below are the SQL statements (for IOD02S) :

```
SELECT NAME, HASHKEY_COLSEQ FROM SYSIBM.SYSCOLUMNS
  WHERE TBNAME = 'TEST01TB'  AND TBCREATOR = 'IOD02S';
SELECT NAME, HASHKEYCOLUMNS FROM SYSIBM.SYSTABLES
  WHERE NAME = 'TEST01TB' AND CREATOR='IOD02S';
SELECT NAME, ORGANIZATIONTYPE FROM SYSIBM.SYSTABLESPACE
  WHERE NAME = 'MYTS' AND CREATOR = 'IOD02S' AND DBNAME ='IOD02SDB';
```


You may see the following output:



- For the output for the first SELECT statement:

  HASHKEY_COLSEQ indicates C1's position within the hash key. Since C2 and C3 are not used in the hash key, so their values are 0.

- For the output for the second SELECT statement:

  HASHKEYCOLUMNS indicates there is only 1 column in hash key in TEST01TB table.

- For the output for the third SELECT statement:

  ORGANIZATIONTYPE's value in SYSIBM.SYSTABLESPACE table is 'H', which indicates MYTS is using hash organization.

6.3 LAB18E2 – INSERT and SELECT from table with hash access

We are going to insert some rows into the table we just created and then select 2 rows from it.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab18e2.db2
```

Below are the SQL statements :

```
INSERT INTO TEST01TB VALUES('BBBB','FOUR',NULL);
INSERT INTO TEST01TB VALUES('CCCC','SEVEN',NULL);
INSERT INTO TEST01TB VALUES('CCCD','SEVEN',NULL);
…..
```

Then we SELECT from the table:

```
SELECT * FROM TEST01TB WHERE C1 = 'BBBB'#
SELECT * FROM TEST01TB WHERE C1 = 'XXXX'#
```

From the output, only first SELECT return 1 row:

6.4 RUNSTATS

Before we look further into the catalog, let's run RUNSTAT to update the catalog data.

Please refer to the instructions in LAB16E6 for running RUNSTATS. In step #9, as there is no LOB table space created in this exercise, you can either ignore or delete the second RUNSTATS statement. After you submit the job, if you don't delete the 2nd RUNSTATS statement, you may see RC8 since DB2 cannot find the LOB table space (this is expected and will not affect the rest of the lab).

6.5 LAB18CAT2 – Select from Catalog to see when hash access used

In this exercise, we are going to look into the following fields in the SYSIBM.SYSTABLESPACESTATS table in the catalog.

REORGHASHACCESS – indicates number of times data is accessed during hash access.

HASHLASTUSED – indicates date when hash access used.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab18cat2.db2
```

Below are the SQL statements(for IOD02S) and possible output

```
SELECT HASHLASTUSED, REORGHASHACCESS
FROM SYSIBM.SYSTABLESPACESTATS
WHERE NAME = 'MYTS' AND DBNAME ='IOD02SDB':
```

The output indicates we have used hash access 2 times (1 for each SELECT) today.

6.6 LAB18E3 – Create table with 2 col hash keys

In this exercise, we are going to create a table with 2 column hash keys.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab18e3.db2
```

Below is the SQL statements (for IOD02S):

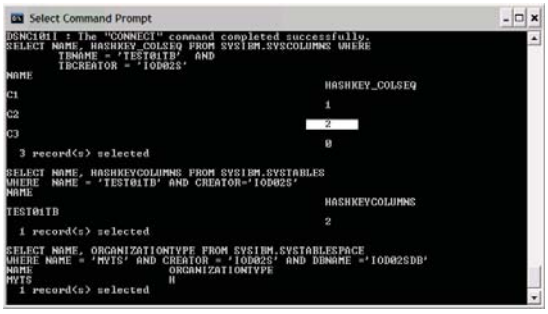

```
DROP DATABASE IOD02SDB;
CREATE DATABASE IOD02SDB;
CREATE TABLESPACE MYTS MAXPARTITIONS 2 IN IOD02SDB;
CREATE TABLE TEST01TB (
            C1 CHAR(4) NOT NULL,
            C2 VARCHAR(8) NOT NULL,
            C3 CHAR(4))
  ORGANIZE BY HASH UNIQUE (C1, C2) HASH SPACE 1 M
  IN IOD02SDB.MYTS;
```

6.7 LAB18CAT1 – Check Catalog

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab18cat1.db2
```

You may see the following output:



The output indicates both C1 and C2 are hash keys; C1 is in 1st position and C2 is in 2nd position.

HASHKEYCOLUMNS indicates there are 2 columns in the hash key.

ORGANIZATIONTYPE value of 'H' indicates hash organization is used.

6.8 LAB18E4- INSERT and SELECT

In this exercise, we are going to insert some data to the table we just created in the previous job and then select 3 rows.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab18e4.db2
```

Below are the SQL statements:

```
INSERT INTO TEST01TB VALUES('BBBB','FOUR',NULL);
INSERT INTO TEST01TB VALUES('CCCC','SEVEN',NULL);
INSERT INTO TEST01TB VALUES('CCCD','SEVEN',NULL);
…
```

Then we SELECT 3 rows:

```
SELECT * FROM TEST01TB WHERE C1 = 'BBBB' AND C2 = 'FOUR';
SELECT * FROM TEST01TB WHERE C1 = 'CCCC' AND C2 = 'SEVEN';
SELECT * FROM TEST01TB WHERE C1 = 'XXXX' AND C2 = 'FOUR';
```

6.9 RUNSTATS

Similarly, we need to run RUNSTATS before we look further into catalog. Please refer to RUNSTATS instruction above(in section 6.4).

6.10 LAB18CAT2 – Select from Catalog to see when hash access used

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab18cat2.db2
```

You may see the following output:

REORGHASHACCESS is having a value of 3 since we have 3 SELECT statements in the previous job.

End of lab 18.

7. LAB 19 – Include Additional Non-Key Columns in Unique Index

When a primary key is defined on a table, DB2 requires a unique index to be defined. In previous versions of DB2, that index could contain only the primary key columns. If additional columns were required to support specific SQL statements (such as the SELECT statement below), it was necessary to create a separate additional index.

For examples,

```
CREATE TABLE CUSTTBL (ACCTID INT NOT NULL,
                      NAME VARCHAR(30) NOT NULL,
                      ADDRESS VARCHAR(50) NOT NULL);
CREATE UNIQUE INDEX INDEX1 ON CUSTTBL (ACCTID);
CREATE UNIQUE INDEX INDEX2 ON CUSTTBL (ACCTID, NAME);
SELECT NAME FROM CUSTTBL WHERE ACCTID=10;
```

In this example, DB2 can use index-only access on INDEX2 to satisfy the SELECT query.

This approach allowed the SQL statement to be executed efficiently, but added unnecessary overhead to the INSERT, UPDATE, and DELETE operations as two indexes needed to be maintained.

DB2 10 allows columns other than the unique key to be specified in the index definition. This allows the second index (INDEX2 above) to be dropped, removing the DASD, CPU, and I/O overhead associated with that index. Initial lab tests have shown up to 30% CPU reduction in INSERT, with identical query performance in one example where two indexes were replaced with a single one using INCLUDE columns.

In this lab, we are going to learn how to use this new feature in CREATE INDEX and ALTER INDEX using INCLUDE.

7.1 LAB19E1 – Create index using INCLUDE

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab19e1.db2
```

Below is the SQL statements:

```
CREATE TABLE CUSTTBL (ACCTID INT NOT NULL,
                      NAME VARCHAR(30) NOT NULL,
                      ADDRESS VARCHAR(50) NOT NULL);
CREATE UNIQUE INDEX CUST1 ON CUSTTBL (ACCTID) INCLUDE(NAME);
```

After a successful execution, NAME is an INCLUDE column that follows the unique column, ACCTID. The INCLUDE column does not participate in the ordering/sequencing of the index keys.

7.2 LAB19CAT1 – Select from Catalog

In this exercise, we are going to check the UNIQUE_COUNT in SYSIBM.SYSINDEXES to see the number of columns that makes up the unique constraint of an index, when other non-constraint enforcing columns or key-targets exist. This value (UNIQUE_COUNT) should be greater than 0 if the index has INCLUDE columns, otherwise, the value is 0.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab19cat1.db2
```

Below is the SQL statement (for IOD02S):

```
SELECT UNIQUE_COUNT FROM SYSIBM.SYSINDEXES
WHERE NAME='CUST1' AND CREATOR='IOD02S';
```

The output may look like this:

```
UNIQUE_COUNT
1
  1 record(s) selected
```

In our lab, there is only ACCTID (1 column) that makes up the unique constraint of index CUST1, when other non-constraint enforcing columns or key-targets exist. Therefore, the value of UNIQUE_COUNT is 1.

7.3 LAB19E2 – Create Index without using INCLUDE columns

In this exercise, we are going to create 2 indexes without using the INCLUDE columns. Then, we will ALTER INDEX and ADD INCLUDE COLUMN in the subsequent exercise.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab19e2.db2
```

Below are the SQL statements:

```
CREATE TABLE CUSTTBL (ACCTID INT NOT NULL,
                      NAME VARCHAR(30) NOT NULL,
                      ADDRESS VARCHAR(50) NOT NULL);
CREATE UNIQUE INDEX CUST1 ON CUSTTBL (ACCTID);
CREATE UNIQUE INDEX CUST2 ON CUSTTBL (ACCTID, NAME);
```

2 Unique indexes are created.

7.4 LAB19CAT2 – Select from Catalog

In this exercise, we are going to check the UNIQUE_COUNT in SYSIBM.SYSINDEXES for the indexes we have created.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab19cat2.db2
```

You may see the following output:

In this case, since both indexes, CUST1 and CUST2, do not have INCLUDE columns, therefore, the value of UNIQUE_COUNT is 0.

7.5 LAB19E3 – ALTER INDEX ADD INCLUDE COLUMN

In this exercise, we are going to ALTER INDEX and ADD INCLUDE COLUMN for the index, CUST1 that we have created in the previous exercise.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab19e3.db2
```

Below are the SQL statements:

```
DROP INDEX CUST2;
ALTER INDEX CUST1 ADD INCLUDE COLUMN (NAME);
ALTER INDEX CUST1 ADD INCLUDE COLUMN (ADDRESS);
```

After a successful execution of these SQL statements, NAME and ADDRESS columns are included in the index, CUST1.

7.6 LAB19CAT1 – Select from Catalog to verify UNIQUE_COUNT

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab19cat1.db2
```

You may see the following output:

Since the index, CUST1 has INCLUDE columns (NAME and ADDRESS), and there is only 1 column (ACCTID) that makes up unique constraint of an index, therefore, the value of UNIQUE_COUNT is 1.

7.7 FYI..how to determine which indexes can be consolidated?

See Appendix A for sample SQL query that may identify indexes that could be consolidated by using INCLUDE columns. Please note that this is only a guide.

End of lab 19.

8. LAB20 – Binary XML Support

DB2 10 support a new binary XML format(formally called Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format) that can be used for more efficiently passing XML documents between a client application and DB2. This can dramatically reduce the size of XML documents and has reduced in CPU and elapsed time reductions ranging from 10 to 30 percent in internal IBM tests.

Retrieval of data as binary XML is supported only in JDBC, SQLJ, or ODBC applications. Binary XML is also supported by the LOAD and UNLOAD utilities.

In this lab, we will learn about binary XML support using JDBC application. To use binary XML via JDBC application, we need JCC 4.11 or later and JDK1.6.

The following exercises can be combined in a single JDBC application, but for illustration purpose, we break it to several steps.

8.1 LAB20E1 – create table

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab20e1.db2
```

This job will create a table like this:

```
CREATE TABLE LAB20TBL (ID INT NOT NULL,
                       XMLCOL XML);
```

8.2 LAB20E2 – Insert using binary XML

1. In the command prompt that we have opened earlier, type

```
notepad Lab20Insert.java
```

From the main() in the source, you can use `ds.setXmlFormat()` to explicitly turn on and off binary XML. The code below turns on binary XML support.

```
DB2SimpleDataSource ds = new DB2SimpleDataSource();
….
ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_BINARY);
```



```
//ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_TEXTUAL);
```

2. In the command prompt that we have opened earlier, type

```
javac Lab20Insert.java
```

to compile the JAVA application. Please ignore the following warnings:

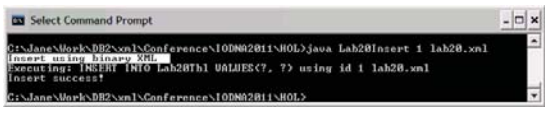
```
Note: Lab20Insert.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

3. In the command prompt that we have opened earlier, type

```
java Lab20Insert 1 lab20.xml
```

to execute the JAVA application to insert id of 1 using lab20.xml as XML data in XMLCOL.

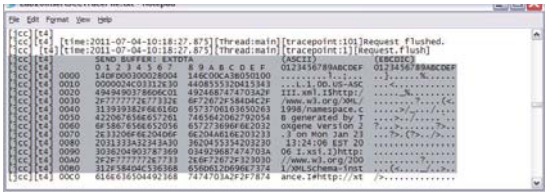
For a successful execution, you should see "`Insert success!`" at the end.



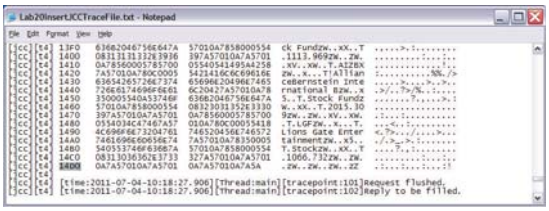
4. Now, let's look at the JCC trace for the INSERT job. In the command prompt that we have opened earlier, type

```
notepad Lab20InsertJCCTraceFile.txt
```

Search for "insert into", then scroll a little down, you should see the binary XML data sent by JCC to DB2:



Scroll down until we see the end of the binary XML data. The length is 14D0 (highlighted below) (Note: you may see a different number. Occasionally, the data may be sent using more than 1 buffer, in that case, the total length is the sum of all buffers sent)



8.3 LAB20E3 – Insert using textual XML

Now, we are going to use textual XML to insert.

1. In the command prompt that we have opened earlier, type

```
notepad Lab20Insert.java
```

From the main() in the source, locate the following 2 lines. Comment out(add // in the front) the binary XML setting and uncomment(remove //) the textual XML setting like this:

```
//ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_BINARY);

ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_TEXTUAL);
```

2. press Ctrl-S to save the change

3. Compile and execute the application in the command prompt

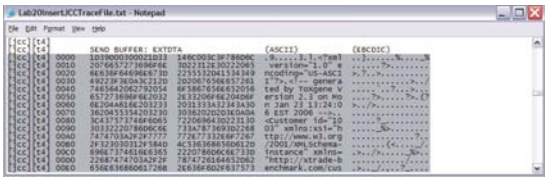
```
javac Lab20Insert.java

java Lab20Insert 2 lab20.xml
```

Make sure there is no compilation and execution error.

4. Open JCC trace for the INSERT job. In the command prompt that we have opened earlier, type

```
notepad Lab20InsertJCCTraceFile.txt
```

Search for “insert into”, then scroll a little down, you should see the textual XML data sent from JCC to DB2:



Scroll down until we see the end of the binary XML data. The length is 1D30’x (highlighted below) (Note: you may see a different number).

As you may recall, the length of sending the same file using binary XML is 14D0’x and it is 860’x SHORTER than sending by textual XML!

8.4 LAB20E4 – Select using binary XML

We are going to learn about how to select using binary XML.

Same as INSERT, the following turn on/off binary XML explicitly

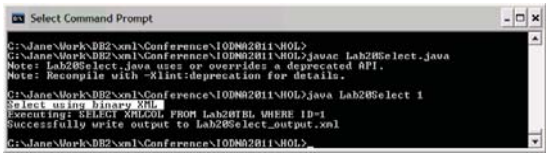
```
ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_BINARY);

//ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_TEXTUAL);
```

Let’s SELECT using binary XML first.

1. In the command prompt that we have opened earlier, type

```
javac Lab20Select.java

java Lab20Select 1
```

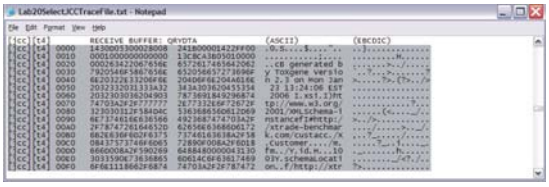
You can ignore the compilation warnings. The 2nd command will SELECT XMLCOL from table for id=1 and write the output to a file.



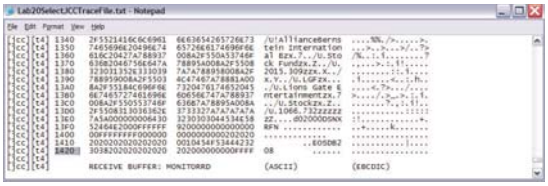
2. Open JCC trace for the SELECT job. In the command prompt that we have opened earlier, type

```
notepad Lab20SelectJCCTraceFile.txt
```

Search for “select XMLCOL”, then scroll a little down, you should see the binary XML data sent from DB2 to JCC:



Scroll down until we see the end of the binary XML data. The length is 1420’x (highlighted below) (Note: you may see a different number).



8.5 LAB20E5 – Select using textual XML

Now, let’s SELECT using textual XML.

1. In the command prompt that we have opened earlier, type

```
notepad Lab20Select.java
```

From the main() in the source, locate the following 2 lines. Comment out(add // in the front) the binary XML setting and uncomment(remove //) the textual XML setting like this:

```
//ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_BINARY);

ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_TEXTUAL);
```

2. press Ctrl-S to save the change

3. Compile and execute the application in the command prompt

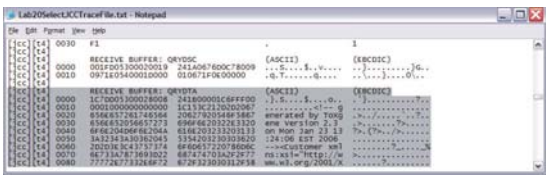
```
javac Lab20Select.java

java Lab20Select 1
```

You can ignore the compilation warnings. The 2nd command will SELECT XMLCOL from table for id=1 and write the output to a file.

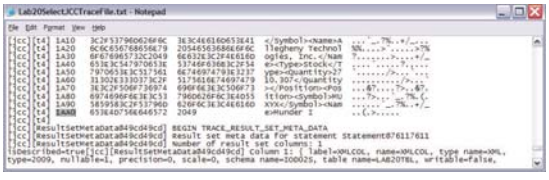
4. Open JCC trace for the SELECT job. In the command prompt that we have opened earlier, type

```
notepad Lab20SelectJCCTraceFile.txt
```

Search for “select XMLCOL”, then scroll a little down, you should see the textual XML data sent by DB2 to JCC:



Scroll down until we see the end of the textual XML data. The length is 1AA0x (highlighted below) (Note: you may see a different number).



As you may recall, the length of selecting the same file using binary XML is 1420'x and it is 680'x SHORTER than selecting using textual XML!

9. Answers for Selected Exercises

The following is a set of answers for the self-study exercises.

9.1 LAB7E9:

```
-- For ID 9, replace the value of shipping Address1 to
-- "999 Union Street"
UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
 'declare namespace demopo="http://www.purchaseOrder.com/purchaseOrder";
  replace value of node
  /demopo:purchaseOrder/shipping/address/Address1
  with "999 Union Street"'
  )
WHERE ID =9;
```

9.2 LAB7E10:

```
-- For ID 10, replace the value the quantity of pid=1000
-- (attribute of "item") to 2

UPDATE CUSTOMER
SET CUSTXML= XMLMODIFY(
  'declare namespace demopo="http://www.purchaseOrder.com/purchaseOrder";
  replace value of node
/demopo:purchaseOrder/items/item[@pid="1000"]/@quantity with "2"')
WHERE ID=10;
```


10. Appendix A

Sample SQL query that may identify indexes that could be consolidated by using INCLUDE columns.

Note : this is only a guide!

```
//*************************************************************
//*
//*  DISCLAIMER:
//*    - THIS QUERY IDENTIFIES INDEXES THAT HAVE THE SAME LEADING
//*      COLUMNS IN THE SAME ORDER AS A SUBSET INDEX THAT IS UNIQUE.
//*      WHILE IT DOES MATCH INDEXES WITH COLUMNS IN THE SAME
//*      SEQUENCE, IT:
//*      - DOES NOT CONSIDER THE ORDERING ATTRIBUTE OF EACH
//*        COLUMN (ASC, DESC OR RANDOM)
//*      - DOES NOT CONSIDER IF A SUBSET/SUPERSET INDEX IS
//*        UNIQUE WHERE NOT NULL
//*      - DOES NOT CONSIDER WHETHER THE INDEXES ARE PARTITIONED OR
//*        NOT
//*    - THEREFORE, CUSTOMERS SHOULD USE THIS AS A GUIDE FOR INDEXES
//*      THAT SHOULD BE CONSIDERED, BUT NOT GUARANTEED, TO BE
//*      CONSOLIDATED INTO A UNIQUE INDEX WITH INCLUDE COLUMNS.
//*
//*
//*  DB2 VERSION 10 FOR Z/OS
//*  SQL REFERENCE MARCH 12,2010
//*  UNIQUERULE  WHETHER THE INDEX IS UNIQUE:
//*
//*  VALUES      C  YES, AND IT IS USED TO ENFORCE THE UNIQUENESS OF
//*                 HASH KEY COLUMNS.
//*             D  NO (DUPLICATES ARE ALLOWED)
//*             U  YES
//*             P  YES, AND IT IS A PRIMARY INDEX (AS IN PRIOR
//*                RELEASES OF DB2, A VALUE OF P IS USED FOR PRIMARY
//*                KEYS THAT ARE USED TO ENFORCE A REFERENTIAL
//*                CONSTRAINT.)
//*             C  YES, AND IT IS AN INDEX USED TO ENFORCE UNIQUE
//*                CONSTRAINT
//*             N  YES, AND IT IS DEFINED WITH UNIQUE WHERE
//*                NOT NULL
//*             R  YES, AND IT IS AN INDEX USED TO ENFORCE THE
//*                UNIQUENESS OF A NON-PRIMARY PARENT KEY
//*             G  YES, AND IT IS AN INDEX USED TO ENFORCE THE
//*                UNIQUENESS OF VALUES IN A COLUMN DEFINED AS
//*                ROWID GENERATED BY DEFAULT
//*             X  YES, AND IT IS AN INDEX USED TO ENFORCE THE
//*                UNIQUENESS OF VALUES IN A COLUMN THAT IS USED TO
//*                IDENTIFY OR FIND XML VALUES ASSOCIATED WITH A
//*                SPECIFIC ROW
//*
//*************************************************************

SELECT CASE WHEN SK.COLSEQ = 1 THEN
            CAST(A.DBID AS CHAR(5))
          ELSE
            ' '
       END AS DBID,
       CASE WHEN SK.COLSEQ = 1 THEN
             SUBSTR(STRIP(A.CREATOR, TRAILING) CONCAT
                    '.' CONCAT STRIP(A.NAME,TRAILING),1,27)
```



```
          ELSE ' '
       END AS SIMPLEST_UNIQUE_IX,
       CASE WHEN SK.COLSEQ = 1 THEN
             CAST(A.OBID AS CHAR(5))
          ELSE
            ' '
       END AS OBID,
       CASE WHEN SK.COLSEQ = 1 THEN
             SUBSTR(STRIP(B.CREATOR, TRAILING) CONCAT
                    '.' CONCAT STRIP(B.NAME,TRAILING),1,27)
          ELSE
            ' '
       END AS IX_WITH_CAND_INCLUDE_COLS,
       CASE WHEN SK.COLSEQ = 1 THEN
             CAST(B.OBID AS CHAR(5))
          ELSE
            ' '
       END AS OBID,
       CASE WHEN SK.COLSEQ <= A.COLCOUNT
              AND (SK.COLSEQ <= A.UNIQUE_COUNT OR
                   A.UNIQUE_COUNT = 0) THEN
         'UNIQUE'
       ELSE
         'INCLUDE'
       END AS COLTYPE,
       SUBSTR(SK.COLNAME,1,27) AS COLNAME
FROM SYSIBM.SYSINDEXES A,
     SYSIBM.SYSINDEXES B,
     SYSIBM.SYSKEYS    SK
WHERE A.UNIQUERULE IN ('U', 'N', 'P', 'R')
  AND B.UNIQUERULE IN ('U', 'N', 'P', 'R', 'D')
  AND (A.UNIQUERULE <> 'N' OR
      (A.UNIQUERULE = 'N' AND B.UNIQUERULE IN ('D', 'N')))
  AND NOT (A.CREATOR = B.CREATOR
       AND A.NAME = B.NAME)
  AND A.TBCREATOR = B.TBCREATOR
  AND A.TBNAME = B.TBNAME
  AND SK.IXNAME = B.NAME
  AND SK.IXCREATOR = B.CREATOR
  AND A.COLCOUNT <= B.COLCOUNT
  AND A.COLCOUNT =
                (SELECT COUNT(*)
                 FROM SYSIBM.SYSKEYS X,
                      SYSIBM.SYSKEYS Y
                 WHERE X.IXCREATOR = A.CREATOR
                  AND  X.IXNAME = A.NAME
                  AND  Y.IXCREATOR = B.CREATOR
                  AND  Y.IXNAME = B.NAME
                  AND  X.COLNAME = Y.COLNAME
                  AND  X.COLSEQ = Y.COLSEQ)
ORDER BY A.CREATOR, A.NAME, B.CREATOR, B.NAME, SK.COLSEQ
WITH UR;
/*
//
```

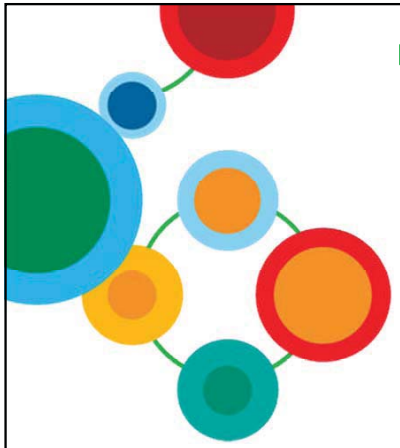
SQLADRIA SEMINAR

Opatija, 24 - 25 November 2011

Improve Development Productivity

Presentation

Sponsored by



New Features in IBM DB2 10 for z/OS III – Improve Development Productivity

Session Number 1267

Jane Man, IBM
Guogen Zhang, IBM
Yonghua Ding, IBM
Li Chen, IBM

IBM Software
Information On Demand 2011

Other DB2 V10 Labs..

- **Session: IDZ-1259A New Features in IBM DB2 10 for z/OS II - Improved Performance and Reduced CPU**
  - Time: Mon, 24/Oct, 02:15 PM - 05:15 PM
  - Location: Mandalay Bay South Convention Center - Shorelines B Lab Room 1
- **Session: IDZ-1267A New Features in IBM DB2 10 for z/OS III - Improve Development Productivity**
  - Time: Tue, 25/Oct, 09:45 AM - 12:45 PM
  - Location: Mandalay Bay South Convention Center - Shorelines B Lab Room 9
- **Session: IDZ-1256A New Core Features and Enhancement in IBM DB2 10 for z/OS**
  Time: Wed, 26/Oct, 09:45 AM - 12:45 PM
  - Location: Mandalay Bay South Convention Center - Shorelines B Lab Room 9

Please Note:

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

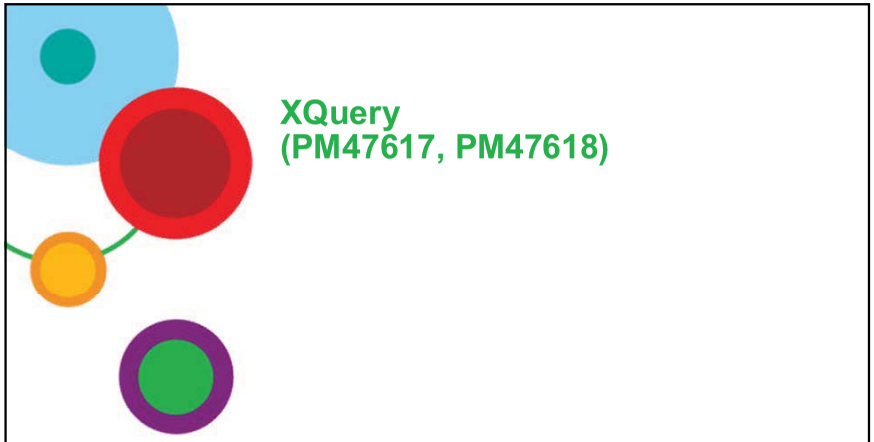
Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Agenda

- Introduction (approx. 30 – 45 min)
  - XQuery
  - XML Date and Time support
  - Extended Implicit CAST support
  - New OLAP Functions
  - Greater TIMESTAMP Precision
  - TIMESTAMP with TIME ZONE
- Lab (approx. 2 hrs)
  - Lab 0 – work together to ensure everyone has connectivity to z/OS machine
  - The rest of labs are on your own pace. There are no dependency among these labs and you can start with anyone you prefer.
  - Lab 8 – XML Date and Time Support

Agenda (cont'd)

- Lab 12 – Implicit CAST support
- Lab 13 – New OLAP functions
- Lab 14 – Greater TIMESTAPM Precision
- Lab 15 – TIMESTAMP with TIME ZONE
- Lab 21 - XQuery



XQuery (PM47617, PM47618)

XQuery Showcases – Basic FLWOR Constructs

- FLWOR - ***For, Let, Where, Order By, Return***
  - Query all items purchased in a purchase order, and order the items ascendingly on the total cost

```
SELECT XMLQUERY(
  'for $i at $pos in $po/purchaseOrder/items/item
   let $p := $i/USPrice,
       $q := $i/quantity,
       $r := $i/productName
   where xs:decimal($p) > 0 and xs:integer($q) > 0
   order by $p * $q
   return fn:concat("Item ", $pos, " productName: ", $r,
                    " price: US$", $p, " quantity: ", $q)
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

xquery

```
<purchaseOrder orderDate="2011-05-18">
 <shipTo country="US">
  <name>Alice Smith</name>
  <street>..</street><city>..</city><state>..</state><zip>..</zip>
 </shipTo>
 <billTo country="US"> ..</billTo>
 <items>
  <item partNum="872-AA">
   <productName>Lawnmower</productName>
   <quantity>1</quantity>
   <USPrice>149.99</USPrice>
   <shipDate>2011-05-20</shipDate>
  </item>
  <item partNum="926-AA">..</item>
  <item partNum="945-ZG">..</item>
 </items>
</purchaseOrder>
```

Input xml doc

```
<?xml version="1.0" encoding="IBM037"?>
Item 2 productName: Baby Monitor price: US$39.98 quantity: 2
Item 1 productName: Lawnmower price: US$149.99 quantity: 1
Item 3 productName: Sapphire Bracelet price: US$178.99 quantity: 2
```

Output xml doc

Conditional expression

- ***If*** (<TestExpression>) ***Then*** (<Expression>) ***Else*** (<Expression>)
  - Query all items purchased in a purchase order, and calculate the shipping cost for each item

```
SELECT XMLQUERY(
  'let $s := $po/purchaseOrder/shipTo
   let $b := $s/@country="US" and $s/state="California"
   for $i in $po/purchaseOrder/items/item
   return (
     if ($b)
     then fn:concat($i/productName, " : shipping cost US$", 8)
     else fn:concat($i/productName, " : shipping cost US$", 12)
       )
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

XQuery

```
<purchaseOrder orderDate="2011-05-18">
 <shipTo country="US">
  <name>Alice Smith</name>
  <street>..</street>...<state>California</state><zip>..</zip>
 </shipTo>
 <billTo country="US"> ..</billTo>
 <items>
  <item partNum="872-AA">
   <productName>Lawnmower</productName> ...
  </item>
  <item partNum=..><productName>Baby Monitor</productName>..
  </item>
  <item partNum=..><productName>Sapphire Bracelet</productName>..
  </item>
 </items>
</purchaseOrder>
```

Input xml doc

```
<?xml version="1.0" encoding="IBM037"?>
Lawnmower : shipping cost US$8
Baby Monitor : shipping cost US$8
Sapphire Bracelet : shipping cost US$8
```

Output xml doc

Value Comparison and Node Comparison

- Value comparisons compare two single values - ***eq, ne, lt, le, gt, ge***
- Node comparisons compare two nodes - ***is, <<, >>***
  - Check if the product with partNum 872-AA matches the product lawnmower

```
SELECT XMLQUERY(
  'if ($po/purchaseOrder/items/item[@partNum eq "872-AA"]
      is
      $po/purchaseOrder/items/item[productName eq "Lawnmower"])
   then "partNum 872-AA matches the product Lawnmower"
   else "partNum 872-AA DOES NOT match the product Lawnmower"
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

XQuery

```
<purchaseOrder orderDate="2011-05-18">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>..</street>...<state>California</state><zip>..</zip>
  </shipTo>
  <billTo country="US"> ..</billTo>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName> ...
    </item>
    ..
  </items>
</purchaseOrder>
```

Input xml doc

```
<?xml version="1.0" encoding="IBM037"?>
partNum 872-AA matches the product Lawnmower
```

Output xml doc

Castable and fn:avg()

- *Castable* expressions test whether a value can be cast to a specific data type – Expression ***castable as*** TargetType?
- ***fn:avg***(sequence-expression)
  - Calculate the average cost of each item

```
SELECT XMLQUERY(
  'let $cost := (
    for $i in $po/purchaseOrder/items/item
    let $p := if ($i/USPrice castable as xs:decimal)
              then xs:decimal($i/USPrice)
              else 0.0
    let $q := if ($i/quantity castable as xs:integer)
              then xs:integer($i/quantity)
              else 0
    return $p * $q)
   return fn:round(fn:avg($cost))
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

XQuery

```
<purchaseOrder orderDate="2011-05-18">
  <shipTo country="US"> .. </shipTo>
  <billTo country="US"> ..</billTo>
  <items>
    <item partNum="872-AA"> ..<quantity>1</quantity>
      <USPrice>149.99</USPrice>
    </item>
    <item partNum="926-AA"> ..<quantity>2</quantity>
      <USPrice>39.98</USPrice>
    </item>
    <item partNum="945-ZG"> ..<quantity>2</quantity>
      <USPrice>178.99</USPrice>
    </item>
  </items>
</purchaseOrder>
```

Input xml doc

(149.99*1 + 39.98*2 + 178.99*2)/3 = 195.97666667

```
<?xml version="1.0" encoding="IBM037"?>196
```

Output xml doc

Constructor

- Constructors create XML structure within a query
  - Reconstruct all items NOT shipped yet in the purchase order

```
declare boundary-space strip;
declare copy-namespaces no-preserve, inherit;
<shipping orderDate="{$po/purchaseOrder/@orderDate}">
 <shipTo>{ let $s := $po/purchaseOrder/shipTo
           return fn:concat($s/state, " ", $s/zip, " ", $s/@country)
         }
 </shipTo>
 {for $i in $po/purchaseOrder/items/item
  where fn:not($i/shipDate castable as xs:date) or
        xs:date($i/shipDate) > fn:current-date()
  return <item partNum="{$i/@partNum}">
           <partName> {$i/productName/text()} </partName>
           { $i/quantity , $i/USPrice }
         </item>
 }
</shipping>
```

XQuery

```
<purchaseOrder orderDate="2011-05-18">
 <shipTo country="US"> ..
   <street>..</street><city>..</city><state>..</state><zip>..</zip>
 </shipTo>
 <billTo country="US"> ..</billTo>
 <items>
  <item partNum="872-AA"> ..
    <shipDate>2011-05-20</shipDate>
  </item>
  <item partNum="926-AA">..
    <shipDate>2011-05-22</shipDate>
  </item>
  <item partNum="945-ZG">
    <productName>Sapphire Bracelet</productName>
    <quantity>2</quantity>
    <USPrice>178.99</USPrice>
    <comment>Not shipped</comment>
  </item>
 </items>
</purchaseOrder>
```

Input xml doc

```
<shipping orderDate="2011-05-18">
  <shipTo>California 95123 US</shipTo>
  <item partNum="945-ZG">
    <partName>Sapphire Bracelet</partName>
    <quantity>2</quantity>
    <USPrice>178.99</USPrice>
  </item>
</shipping>
```

Output xml doc

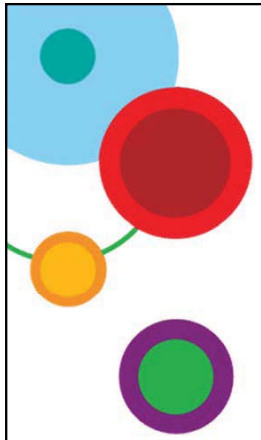
XML Date and Time Support

XML Date and Time Support

- Date and time are not supported in XPath in V9
  - Workaround: XMLTABLE and SQL date and time
- Add support xs:date, xs:time, and xs:dateTime and indexes for DATE and TIMESTAMP
  - Implicit timezone will be UTC if a date or time does not have a timezone
  - You get back the original or use fn:adjust-…-to-timezone() for a local timezone
- Supported types
  - xs:dateTime
  - xs:time
  - xs:date
  - xs:duration
  - xs:yearMonthDuration
  - xs:dayTimeDuration
- And a bunch of functions and operators: with time zone support, surpass SQL.

Examples

- XMLQUERY('/purchaseorder/items/item[shipdate > xs:date("2007-01-31")]' PASSING X1)
- CREATE INDEX XIX1 ON T1(X1) GENERATE KEYS USING XMLPATTERN '/purchaseorder/items/item/shipdate' as SQL DATE
- XMLQUERY('/purchaseorder [ordertime > xs:dateTime("2007-01-31T10:20:30-08:00")]' PASSING X1)
  - Note: xs:dateTime("2007-01-31T10:20:30-8:00") does not work
- CREATE INDEX XIX1 ON T1(X1) GENERATE KEYS USING XMLPATTERN '/purchaseorder/ordertime' as SQL TIMESTAMP
- XMLQUERY('fn:adjust-dateTime-to-timezone(xs:dateTime("2007-01-31T10:20:30-05:00"), xs:dayTimeDuration("-PT8H"))')
  - Result: 2007-01-31T07:20:30-08:00



Extended Implicit CAST Support

Overview

- Extended support for implicit casts: implicit cast between string and numeric data types
- Indexable/sargable predicates
- new semantics of assignments & comparisons & unions & expressions & function resolution

Examples

```
CREATE TABLE employee (
      empno  INTEGER,
      level  CHAR(3),
      salary DECIMAL(15,2));

INSERT INTO EMPLOYEE VALUES( '1001', 3, 89000.39);

UPDATE employee
    SET level = level + 1,
    salary = salary * '1.1'
    WHERE empno = '1001';
```

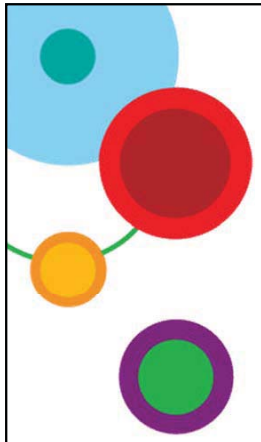

From Numeric to String

Numbers ⟷ Strings: character string (no CLOB, no FOR BIT DATA subtype) or graphic string (no DBCLOB, UNICODE encoding scheme)

Source Data Type	Target Data Type
SMALLINT	VARCHAR(6)
INTEGER	VARCHAR(11)
BIGINT	VARCHAR(20)
NUMERIC/DECIMAL	VARCHAR(precision+2)
FLOAT (REAL, DOUBLE)	VARCHAR(24)
DECFLOAT	VARCHAR(42)

From String to Numeric

Source Data Type	Target Data Type
CHAR	DECFLOAT(34)
VARCHAR	DECFLOAT(34)
GRAPHIC	DECFLOAT(34)
VARGRAPHIC	DECFLOAT(34)
CHAR/VARCHAR **FOR BIT DATA**	N/A
BINARY/VARBINARY	N/A
CLOB/BLOB/DBCLOB	N/A



New OLAP functions (Moving Average, Running Total, etc.)

Overview

- DB2 9 for z/OS has already supported 2 classes of OLAP specifications
  - Ranking Specifications – RANK(), DENSE_RANK()
  - Numbering Specifications – ROW_NUMBER().
- DB2 10 for z/OS will introduce the last class of OLAP specifications
  - Aggregation Specifications – SUM(), AVG() and other aggregate function etc.

Moving Sums and Moving Averages

- compute a single value for the current row based on some or all of the rows in a defined group.
- support cumulative sums and moving averages by using a window.
- can be used in a select-list, or in the ORDER BY clause of a select-statement.
- Limitation: cannot use with XMLQUERY function or an XMLEXISTS predicate,

Example Data

```
create table Sales_history (
  Territory  VARCHAR(10), -- Business Territory
  Month      INTEGER,    -- Six-digit in YYYYMM format
  Sales      INTEGER      -- Total sales for Territory/Month
 )
```

Data in the table:

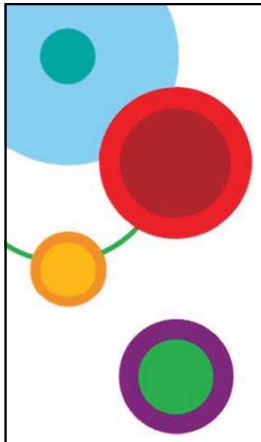
```
Territory  Month       Sales
---------- ---------- -----------
East       199810              10
East       199811               4
East       199812              10
East       199901               7
East       199902              10
West       199810               8
West       199811              12
West       199812               7
West       199901              11
West       199902               6
```


EXAMPLE for Moving SUM, Moving AVG, etc

Display the business territory, month, total sales for each territory/month and the sale in the territory averaged over the current month and the preceding two months.

```
SELECT Sh.Territory, Sh.Month, Sh.Sales,
       AVG(Sh.Sales) OVER (PARTITION BY Sh.Territory
                           ORDER BY Sh.Month
                         ROWS 2 PRECEDING) as Moving _average
  FROM Sales_history as Sh;
```

```
Territory  Month       Sales       Moving_average
---------- ---------- ----------- ----------------
East       199810              10               10
East       199811               4                7
East       199812              10                8
East       199901               7                7
East       199902              10                9
West       199810               8                8
West       199811              12               10
West       199812               7                9
West       199901              11               10
West       199902               6                8
```



Greater Timestamp Precision,

TIMESTAMP WITH TIME ZONE

Background

Problem:

- The existing TIMESTAMP data type does not capture an associated time zone, the timestamp is ambiguous
- Some customers store time zone in another column

Solution:

- New data type

  TIMESTAMP WITH TIME ZONE

New Terminology

- **fractional second** - A portion of a second that is greater than 0 but less than 1.
- **timestamp precision** - The maximum number of digits that can be included in a fractional second.

Timestamp

- A timestamp is a **six or** seven-part value (year, month, day, hour, minute, second, and **optional fractional second**) **with an optional time zone specification** that represents a date and time. The time could include specification of a fraction of a second.
- The number of digits in the fractional second is specified using an attribute in the range from 0 to 12 with a default of 6.
- Example: TIMESTAMP(12)

  '2000-01-15-08.30.00.123456789012'

TIMESTAMP WITH TIME ZONE

- TIMESTAMP WITH TIME ZONE format:
  - <timestamp> TZH:TZM
  - TZH (time zone hour) – '*xhh*' -24 to 24
  - TZM (time zone minute) – 'mm' 00 to 59 (has to be 00 if TZH is 24)
- Example:
  - New York is 4 hours behind UTC, so New York time "11:42" on 2009-06-09 can be represented as '2009-06-09-11.42.00.000000-04:00'.
  - Same UTC representation:
    - '2009-06-09-11.42.00.000000-04:00'
    - '2009-06-09-08.42.00.000000-07:00'
    - '2009-06-09-15.42.00.000000-00:00'

More Examples:

```
CREATE TABLE LAB15_Table
 (ID    INTEGER      NOT NULL WITH DEFAULT,
TSTZ12  TIMESTAMP(12) WITH TIME ZONE WITH DEFAULT
        '9999-12-31 23.59.59.123456789012 +14:00');

INSERT INTO LAB15_Table(ID, TSTZ12)
 VALUES (4,
  TIMESTAMP '2012-12-31 02:02:02.123456789012+08:00' AT TIMEZONE '-08:00'    );

INSERT INTO LAB15_Table (ID, TSTZ12)
 VALUES (5,
TIMESTAMP '2012-12-31 02:02:02.123456789012+08:00'
        AT TIMEZONE SESSION TIME ZONE    )
```


An Exclusive Invitation for System z Attendees

Thank You… Leveraging the Best of z!

"The benchmarks and analysis of functionality and performance have exceeded our expectations. So far our upgrades have gone smoothly and we are looking forward to completing our successful rollout of DB2 10"

–Verizon

"We had migrated five sub-systems to DB2 10 and have had no reported application issues running on this release to date."

–LabCorp

"The 'overall performance' in DB2 10 is better compared to DB2 9."

–HUK Coberg

[The Temporal Data] feature will drastically save developer time, test time … and improve business efficiency and effectiveness …

–Bankdata

Our regression tests showed performance improvements just by running the workload on a DB2 10 CM member…

–Dillards

Acknowledgements and Disclaimers:

Availability. References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS-IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

Communities

- **On-line communities, User Groups, Technical Forums, Blogs, Social networks, and more**
  - Find the community that interests you…
    - **Information Management** **ibm.com**/software/data/community
    - **Business Analytics** **ibm.com**/software/analytics/community
    - **Enterprise Content Management** **ibm.com**/software/data/content-management/usernet.html
- **IBM Champions**
  - Recognizing individuals who have made the most outstanding contributions to Information Management, Business Analytics, and Enterprise Content Management communities
    - **ibm.com**/champion

Thank You!
Your Feedback is Important to Us

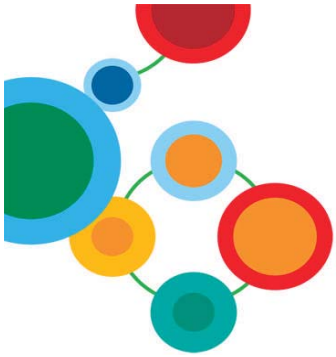
- Access your personal session survey list and complete via SmartSite
  - Your smart phone or web browser at: iodsmartsite.com
  - Any SmartSite kiosk onsite
  - Each completed session survey increases your chance to win an Apple iPod Touch with daily drawing sponsored by Alliance Tech

SQLADRIA SEMINAR
Opatija, 24 - 25 November 2011

Improve Development Productivity

Hands on Lab

Sponsored by



New Features in IBM DB2 10 for z/OS III – Improve Development Productivity

Session Number 1267

Jane Man, IBM
Guogen Zhang, IBM
Yonghua Ding, IBM
Lily Chen, IBM

DISCLAIMERS

Please Note:

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Contents

1. Introduction

1.1 Who are we?

We are from the DB2 development organization in IBM Silicon Valley Lab.


```
Jane Man
 Advisory Software Engineer
 janeman@us.ibm.com

Guogen Zhang
 Distinguished Engineer
 gzhang@us.ibm.com

Yonghua Ding
 Advisory Software Engineer
 dingy@us.ibm.com

Lily Chen
 Software Engineer
 chelin@us.ibm.com
```


1.2 Objectives:

- Learn new OLAP functions(moving average)
- Learn about greater precision timestamp
- Learn about timestamp with time zone
- Learn about extended implicit CAST support
- Learn about XQuery

1.3 Suggested reading

- Introduction to pureXML in DB2 9 for z/OS

  ftp://ftp.software.ibm.com/software/data/db2zos/presentations/2007/misc/pure

  xml.pdf

- Leveraging DB2 9 for z/OS pureXML Technology

  http://www.geocities.com/zhanggene/pub/Leveraging_DB29_for_zOS_whitepaper_v2.pdf

- DB2 Version 9.1 for z/OS XML Guide (SC18-9858)

  http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.xml/dsnxgk13.pdf?noframes=true

- DB2 Version 10 for z/OS XML Guide (SC19-2981-02)

  http://publib.boulder.ibm.com/epubs/pdf/dsnxgm02.pdf

- DB2 Version 10 for z/OS SQL Reference (SC19-2983-02)

  http://publib.boulder.ibm.com/epubs/pdf/dsnsqm02.pdf

1.4 Acknowledgments

Many thanks for the following people to make this HOL possible:

Fung Lee

Ken Taylor

Rick Chang

Ruiping Li

Xiaohong Fu

Ying Zeng

2. Lab0 : Getting Started

In this hands-on-lab(HOL), we will use the DB2 Command Line Processor(CLP) that shipped with DB2 for z/OS and SPUFI to learn about the pureXML and new SQL features in DB2 10 for z/OS.

2.1 Dependency of the Labs

Lab 0 must be done first to ensure the Command Line Processor (CLP) environment is set up properly and there is database connection to the DB2 server.

After Lab 0 is done, you can work on the following labs in any order.

Lab 8: XML date and time support

Lab 12: Extended Implicit CAST support

Lab 13: New OLAP functions (moving average)

Lab 14: Greater Precision Timestamp

Lab 15: Timestamp with Time Zone

Lab 21: XQuery

2.2 LAB0E1 – Command Line Processor (CLP) setup

In this exercise, we will use the DB2 command line processor that ships with DB2 for z/OS. We will configure CLP and use it to execute a script to test the connectivity to the database.

1. Open a Command Prompt window by double clicking the Command prompt icon in the desktop.
2. Change directory to the location of the hands-on-lab materials.

   In the command prompt window, type: `CD C:\HOL`

3. Start notepad to edit the `clpsetup.bat` file.
   type: `notepad clpsetup.bat`

4. Replace the "XXXXXXXX" and "YYYYYYYY" with the username and password provided to you by the lab instructor. Please use UPPERCASE for all userid and password.

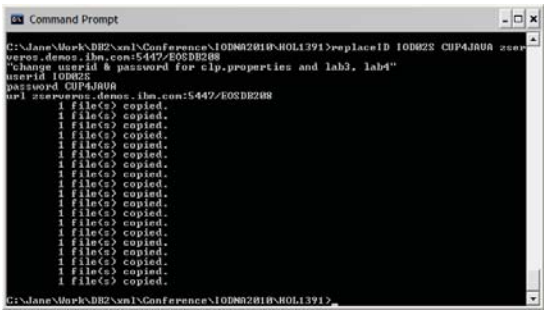
   Tip: You can quickly do find/replace in notepad by choosing "replace" from the Edit menu, or typing Ctrl-H at any time.

   Note that the alias "mydb2" has been created for you to direct CLP to connect to the server.

   Save the file (from the File menu) and quit notepad (from the File menu).

5. Execute the clpsetup.bat batch file to define aliases and set environment variables to allow CLP to run. In the command prompt window, type: `clpsetup.bat`

   For a successful execution, you should see something similar to this:



6. Browse the "lab0e1.db2" script. You do not have to change anything in this script (unless you want to).
   type: `notepad lab0e1.db2`

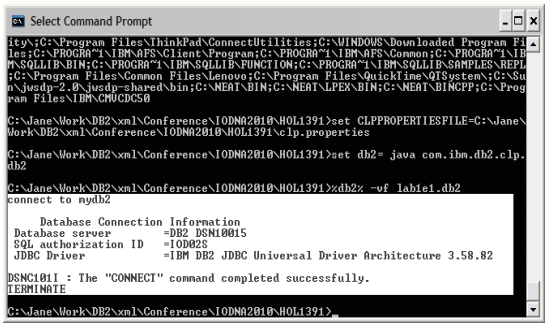
   Exit notepad when you are finished.

7. Execute the "lab0e1.db2" script. This script will connect to the database.
   type: `%db2% -vf lab0e1.db2`

   %db2% invokes the "db2" alias that we created in `clpsetup.bat`. The options "v" tells CLP to use verbose mode so it prints the commands and results to the screen, and "f" tells CLP to execute a

file instead of running in interactive mode.
"lab0e1.db2" is the script that we will run.

Verify that the "CONNECT" command completed successfully. If the screen looks as it does below(may see different `Database Server` and `SQL authorization ID`), then the script ran successfully and this exercise is complete.



CLP will be used again in the following labs. Leave the command prompt window open so we can easily invoke CLP again.

You can now start with any lab you prefer.

Please note we use userid IOD02S for illustration. You should use your own userid assigned by the instructor.

End of lab 0.

3. Lab 8 – Learning Native XML Date and Time Support

Date and time are not supported in XPATH in DB2 9 for z/OS. In DB2 10 for z/OS, we add support xs:date, xs:time, and xs:dataTime and XML indexes for DATE and TIMESTAMP. In this lab, we are going to learn some of these features using a meeting schedule as example.

Our example is a meeting schedule of 3 different meetings in 3 different countries:

- The San Jose meeting (meeting 1) lasts for 3 hours 30 min, start at `2010-10-11T14:00:00-07:00`
- The Germany meeting (meeting 2) lasts for 2 hours, start at `2010-10-11T10:00:00+01:00`
- The Beijing meeting (meeting 3) lasts for 2 hours, start at `2010-10-12T08:00:00+08:00`

We will use CLP for this lab.

3.1 LAB8E1: Create Table, Insert Data, and Create XML indexes using TIMESTAMP

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab8e1.db2
```

Below is the SQL statement:

```
CREATE TABLE TAB1 (I1 INT, X1 XML);
```

We create an XML index using XPATH /meeting/schedule/startTime. Below is the SQL statement:

```
-- Create index on startTime
CREATE INDEX IDX1 ON TAB1(X1)
GENERATE KEYS USING XMLPATTERN
```



```
'/meeting/schedule/startTime' AS SQL TIMESTAMP;
```

Then, we insert an XML document.

Below is the SQL statement:

```
-- Insert into TAB1
INSERT INTO TAB1 (I1,X1) VALUES(10,
'<?xml version="1.0" encoding="UTF-8"?>
 <meeting>
   <name>MCDM</name>
   <description>Multi Country Development Meeting
</description>
   <schedule>
     <name>Meeting 1</name>
     <startTime>2010-10-11T14:00:00-07:00</startTime>
     <duration>PT3H30M</duration>
     <lab>
       <city>San Jose</city>
       <country>US</country>
     </lab>
   </schedule>
   <schedule>
     <name>Meeting 2</name>
     <startTime>2010-10-11T10:00:00+01:00</startTime>
     <duration>PT2H</duration>
     <lab>
       <city>Boeblingen</city>
       <country>Germany</country>
     </lab>
  </schedule>
  <schedule>
    <name>Meeting 3</name>
    <startTime>2010-10-12T08:00:00+08:00</startTime>
    <duration>PT2H</duration>
    <lab>
      <city>Beijing</city>
      <country>China</country>
    </lab>
  </schedule>
</meeting>');
```

3.2 LAB8E2: Basic Date and Time functions in XPATH

In this exercise, we are going to learn about the following date and time functions in XPATH:

fn:current-dateTime()

fn:current-date()

fn:current-time()

fn:implicit-timezone()

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab8e2.db2
```

Below is the SQL statements:

```
SELECT XMLQUERY('fn:current-dateTime()')
 AS CURRENT_DATE_TIME
 FROM SYSIBM.SYSDUMMY1
```

You may see result : `2010-09-24T10:59:50.2896166875-07:00`

(Note the actual output depends on the actual time and date, -07:00 at the end mean Pacific Daylight Time(PDT) which is GMT-07:00))

```
SELECT XMLSERIALIZE(XMLQUERY('fn:current-dateTime()') AS CLOB
 EXCLUDING XMLDECLARATION)
 AS CURRENT_DATE_TIME
 FROM SYSIBM.SYSDUMMY1
```

You may see result : `2010-09-24T10:59:50.387254585937-07:00`

```
SELECT XMLQUERY('fn:current-date()')
 AS CURRENT_DATE
 FROM SYSIBM.SYSDUMMY1;
```

You may see result : `2010-09-24-07:00`

```
SELECT XMLQUERY('fn:current-time()')
 AS CURRENT_TIME
 FROM SYSIBM.SYSDUMMY1;
```

You may see result : `10:59:50.4330395625-07:00`

```
SELECT XMLQUERY('fn:implicit-timezone()')
 AS CURRENT_TIMEZONE
 FROM SYSIBM.SYSDUMMY1
```

You may see result : `PT0S`

The fn:implicit-timezone function returns the time zone that is used when a date, time, or dateTime value that does not have a time zone is used in a comparison or arithmetic operation. The value PT0S indicates that UTC is the implicit time zone.

3.3 LAB8E3: fn:adjust-dateTime-to-timezone()

In this exercise, we will show some examples on fn:adjust-dateTime-to-timezone() by adjusting the start time of all the meetings to PDT.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab8e3.db2
```

Below is the SQL statements:

-- 1st meeting in San Jose

```
SELECT XMLQUERY('fn:adjust-dateTime-to-timezone(
  /meeting/schedule[lab/city="San Jose"]/startTime,
  xs:dayTimeDuration("-PT7H"))'
 PASSING X1)
 as Adjust_Time FROM TAB1;
```

Result: `2010-10-11T14:00:00-07:00`

Before adjust: `2010-10-11T14:00:00-07:00`

-- 2nd meeting in Germany

```
SELECT XMLQUERY('fn:adjust-dateTime-to-timezone(
  /meeting/schedule[lab/city="Boeblingen"]/startTime,
  xs:dayTimeDuration("-PT7H"))'
```



```
 PASSING X1)
 as Adjust_Time FROM TAB1;
```

Result: `2010-10-11T02:00:00-07:00`

Before adjust: `2010-10-11T10:00:00+01:00`

-- 3rd meeting in Beijing

```
SELECT XMLQUERY('fn:adjust-dateTime-to-timezone(
  /meeting/schedule[lab/city="Beijing"]/startTime,
  xs:dayTimeDuration("-PT7H"))'
 PASSING X1)
 Adjust_Time FROM TAB1;
```

Result: `2010-10-11T17:00:00-07:00`

Before adjust: `2010-10-12T08:00:00+08:00`

3.4 LAB8E4: xs:dateTime() and xs:dayTimeDuration

In this exercise, we show how to find the meeting end time by using xs:dateTime() and xs:dayTimeDuration().

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab8e4.db2
```

Below is the SQL statements:

-- 1st meeting in San Jose

```
SELECT XMLQUERY('
/meeting/schedule[lab/city="San Jose"]/xs:dateTime(startTime)
+
/meeting/schedule[lab/city="San
Jose"]/xs:dayTimeDuration(duration)'
 PASSING X1)
as meeting_end_time FROM TAB1;
```

Result(Meeting end time): `2010-10-11T17:30:00-07:00`

Meeting start time: `2010-10-11T14:00:00-07:00, duration 3hr 30min`

-- 2nd meeting in Germany

```
SELECT XMLQUERY('
/meeting/schedule[lab/city="Boeblingen"]/xs:dateTime(startTime
) +
/meeting/schedule[lab/city="Boeblingen"]/xs:dayTimeDuration(du
ration)'
 PASSING X1)
as meeting_end_time FROM TAB1;
```

Result(Meeting end time): `2010-10-11T12:00:00+01:00`

Meeting start time: `2010-10-11T10:00:00+01:00, duration 2hr`

-- 3rd meeting in Beijing

```
SELECT XMLQUERY('
/meeting/schedule[lab/city="Beijing"]/xs:dateTime(startTime) +
/meeting/schedule[lab/city="Beijing"]/xs:dayTimeDuration(durat
ion)'
 PASSING X1)
as meeting_end_time FROM TAB1;
```

Result(Meeting end time): `2010-10-12T10:00:00+08:00`

Meeting start time: `2010-10-12T08:00:00+08:00, duration 2 hrs`

3.5 LAB8E5: XML Index for TIMESTAMP

In this exericise, we are going to show the XML index(IDX1) that we created in lab8e1 is picked by DB2.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab8CreatePlanTable.db2
```

This job will create the PLAN_TABLE.

2. In the same command prompt, type


```
%db2% -vf lab8e5.db2
```

Below is the SQL statement inside lab8e5.db2:

```
EXPLAIN ALL SET QUERYNO=1 FOR
SELECT I1
FROM TAB1
WHERE XMLEXISTS('
  /meeting/schedule[startTime =
    xs:dateTime("2010-10-11T14:00:00-07:00")]'
  PASSING X1);

SELECT QUERYNO,QBLOCKNO,PLANNO,METHOD,
       CAST(CREATOR AS CHAR(8)),CAST(TNAME AS CHAR(18)),
       ACCESSTYPE,JOIN_TYPE,TABLE_TYPE,
       ACCESS_DEGREE, CAST(ACCESSNAME AS
CHAR(18)),PARALLELISM_MODE
  FROM PLAN_TABLE WHERE QUERYNO = 1
ORDER BY MIXOPSEQ;
```

The output should like this: DX as ACCESSTYPE and IDX1(as index name, the one we created in lab8e1). DX stands for DOCID list access.

Please note: you may see a different result, depend on the statistics:

```
QUERYNO     QBLOCKNO PLANNO METHOD 5         6
ACCESSTYPE JOIN_TYPE TABLE_TYPE ACCESS_DEGREE 11
PARALLELISM_MODE

1           1        1      0      IOD02S   TAB1
DX     T           <null>        IDX1                <null>
```

End of lab 8.

4. Lab 12 – Extended Implicit Cast Support

In DB2 10 for z/OS, we support implicit CAST between string and numeric data types.

Below is the conversion details between these 2 datatypes:

Source Data Type	Target Data Type
SMALLINT	VARCHAR(6)
INTEGER	VARCHAR(10)
BIGINT	VARCHAR(20)
NUMERIC/DECIMAL	VARCHAR(precision+2)
FLOAT (REAL, DOUBLE)	VARCHAR(24)
DECFLOAT	VARCHAR(42)

Source Data Type	Target Data Type
CHAR	DECFLOAT(34)
VARCHAR	DECFLOAT(34)
GRAPHIC	DECFLOAT(34)
VARGRAPHIC	DECFLOAT(34)
CHAR/VARCHAR FOR BIT DATA	N/A
BINARY/VARBINARY	N/A
CLOB/BLOB/DBCLOB	N/A

In this lab, we are going to learn about this new support through some examples using CLP.

4.1 LAB12E1: Tables with Same Column names but different data types

In this exercise, we are going to create 2 tables:

- EMPLOYEE
- EMPLOYEE_HISTORY

Both tables contain 3 columns: empno, level, and salary

However, the empno is of different datatypes in these 2 tables.

```
CREATE TABLE employee (
       empno  INTEGER,                 <-
       level  CHAR(3),
       salary DECIMAL(15,2));
CREATE TABLE employee_history (
       empno  VARCHAR(10),             <-
       level  CHAR(3),
       salary DECIMAL(15,2));
```

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab12e1.db2
```

This job will create and populate the tables:

```
select * from employee

EMPNO       LEVEL SALARY
1000        008   100000.39
1002        06    7000.42
1001        7     88000.39
  3 record(s) selected


select * from employee_history

EMPNO      LEVEL SALARY
800         10   180000.39
1002       06    7200.42
```



```
101         7     89000.39
  3 record(s) selected
```

4.2 LAB12E2: Insert an INTEGER to a VARCHAR column

In the command prompt that we have opened earlier, type

```
%db2% -vf lab12e2.db2
```

This job will insert an INTEGER to a VARCHAR column:

```
INSERT INTO employee_history
      SELECT * FROM employee
      WHERE empno = '1001';
```

Note here:

Empno in EMPLOYEE table is of datatype INTEGER, but it is of VARCHAR datatype in EMPLOYEE_HISTORY table.

Therefore, this INSERT statement insert empno(INTEGER) from EMPLOYEE table to empno(VARCHAR(10)) in EMPLOYEE_HISTORY table!

After the INSERT, inside EMPLOYEE_HISTORY table:

```
EMPNO       LEVEL SALARY
800          10   180000.39
1002        06    7200.42
101         7     89000.39
1001        7     88000.39            <-
  4 record(s) selected
```

4.3 LAB12E3: Do Math with CHAR column or CHAR input

In the command prompt that we have opened earlier, type


```
%db2% -vf lab12e3.db2
```

This job will execute the following SQL statements

```
UPDATE employee
     SET level = level + 1,
     salary = salary * '1.1'
     WHERE empno = '1001';
```

Note here: `level` column in EMPLOYEE table is of datatype CHAR(3) and `salary` of datatype DECIMAL(15, 2).

Now, we are doing addition on a CHAR(3) column!

After the UPDATE, inside EMPLOYEE table, both the level and salary for empno 1001 get updated!

```
SELECT * FROM employee
EMPNO       LEVEL SALARY
1000        008   100000.39
1002        06    7000.42
1001        8     96800.43    <-
  3 record(s) selected
```

4.4 LAB12E4: Watch out for Overflow or Underflow

In the command prompt that we have opened earlier, type

```
%db2% -vf lab12e4.db2
```

This job will execute the following SQL statements

```
UPDATE employee
      SET level = level + 1234567890,
      salary = salary * '1.1'
      WHERE empno = '1001';
```

```
UPDATE employee
      SET level = level + 1,
      salary = salary * '123456789123456789'
      WHERE empno = '1001';
```

The output of 1st UPDATE statement:

```
sqlcode : -433 sqlstate: 22001

sqlerr Message: VALUE 1234567898 IS TOO LONG. SQLCODE=-433,
SQLSTATE=22001, DRIVER=3.58.82
```

Note here that `level` column is of datatype CHAR(3)

The output of 2nd UPDATE statement:

```
sqlcode : -413 sqlstate: 22003

sqlerr Message: OVERFLOW OR UNDERFLOW OCCURRED DURING NUMERIC
DATA TYPE CONVERSION. SQLCODE=-413, SQLSTATE=22003,
DRIVER=3.58.82
```

Note here that `salary` column is of datatype DECIMAL(15,2)

4.5 LAB12E5: Function Overloading

In this exercise, we are going to create 2 functions with same function name(F2), but with different parameter datatype.

In the command prompt that we have opened earlier, type

```
%db2% -vf lab12e5.db2
```

This job will create 2 user-defined functions, both called F2, 1 with INTEGER as parameter while the other has CLOB(20) as parameter:

```
CREATE FUNCTION F2 (C1 INTEGER)
       RETURNS BIGINT
       RETURN BIGINT(C1) + 1;
```



```
CREATE FUNCTION F2 (C1 CLOB(20))
       RETURNS BIGINT
       RETURN BIGINT(CHAR(C1));
```

The job also executes the following SELECT statements:

```
SELECT F2 ('1') as F2_output FROM SYSIBM.SYSDUMMY1

F2_OUTPUT

1

  1 record(s) selected
```

Note: F2 with CLOB(20) parameter is used here.

```
SELECT F2 (1) as F2_output FROM SYSIBM.SYSDUMMY1

F2_OUTPUT

2

  1 record(s) selected
```

Note: F2 with INTEGER parameter is used here.

```
SELECT F2 (1.0) as F2_output FROM SYSIBM.SYSDUMMY1:

sqlcode : -245 sqlstate: 428F5

sqlerr Message: THE INVOCATION OF FUNCTION F2 IS AMBIGUOUS.
SQLCODE=-245, SQLSTATE=428F5, DRIVER=3.58.82
```

Note: DB2 doesn't know which F2 should use here.

```
SELECT F2 (1E0) as F2_output FROM SYSIBM.SYSDUMMY1

sqlcode : -245 sqlstate: 428F5

sqlerr Message: THE INVOCATION OF FUNCTION F2 IS AMBIGUOUS.
SQLCODE=-245, SQLSTA

TE=428F5, DRIVER=3.58.82
```


Note: DB2 doesn't know which F2 should use here.

End of lab 12.

5. Lab 13 – New OLAP functions: Moving Average

In DB2 10 for z/OS, we have added some new OLAP (On-Line Analytical Processing) functions and we will learn some of these functions in this lab: Moving Average, MAX(), and MIN().

We are going to use CLP in this lab.

5.1 LAB13E1: Sales History Example

We will first start with a simple example, Sales History. We are going to find the total sales for each territory/month and the sales in the territory averaged over the current month and the preceding two months.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab13e1.db2
```

This job will create a Sales_History table, populate the tables and find the sales in the territory averaged over the current month and the preceding two months.

Below are some of the SQL statements for your reference:

```
CREATE TABLE SALES_HISTORY (
 Territory VARCHAR(10),    <-- Business Territory
 Month     INTEGER,        <-- Six-digit in YYYYMM format
 Sales     INTEGER);       <-- Total sales for Territory/Month

SELECT Sh.Territory, Sh.Month, Sh.Sales,
       AVG(Sh.Sales) OVER (PARTITION BY Sh.Territory
                           ORDER BY Sh.Month
                           ROWS 2 PRECEDING) as Moving_average
```



```
  FROM Sales_history as Sh;
```

Below is the output:

```
TERRITORY  MONTH       SALES       MOVING_AVERAGE
East       199810      10          10
East       199811      4           7
East       199812      10          8  <-
East       199901      7           7
East       199902      10          9
West       199810      8           8
West       199811      12          10
West       199812      7           9
West       199901      11          10
West       199902      6           8
  10 record(s) selected
```

Take the row for East(Territory) of month 199812 as example, the moving average for the preceding 2 months and the current month is

(10 + 4 + 10)/3 = 8

5.2 LAB13E2: Stock Example

In the following exercises, we will use a Stock example to look further into the moving average features.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab13e2.db2
```

This job will create and populate a stock table.

Below are some of the SQL statements for your reference:

```
Create table stock (
  date date,
  symbol char(3),
  close_price dec(9,3));
```

Inside the stock table:

```
SELECT *  FROM stock

DATE       SYMBOL CLOSE_PRICE
2007-04-23 XYZ    110.125
2007-04-24 XYZ    109.500
2007-04-25 XYZ    110.000
```

```
2007-04-26 XYZ    119.750
2007-04-27 XYZ    110.625
2007-04-30 XYZ    111.125
2007-05-01 XYZ    113.750
2007-05-02 XYZ    114.000
2007-05-03 XYZ    113.750
2007-05-04 XYZ    112.125
2007-05-07 XYZ    109.750
2007-05-08 XYZ    111.000
2007-05-09 XYZ    110.750
  13 record(s) selected
```

5.3 LAB13E3: Specify Window by ROWS

In this exercise, we are going to find the seven day centered average of XYZ stock for each day the stock traded. The window is specified by the row clause.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab13e3.db2
```

This job will execute the following SQL statement:

```
SELECT date,symbol, close_price,

  decimal(avg(close_price) over (order by date rows between 3
preceding and 3 following),6,3)

  as smooth_cp

  FROM stock;
```

The result should look like this:

```
DATE       SYMBOL CLOSE_PRICE SMOOTH_CP
2007-04-23 XYZ    110.125     112.343
2007-04-24 XYZ    109.500     112.000
2007-04-25 XYZ    110.000     111.854
2007-04-26 XYZ    119.750     112.125        <-
2007-04-27 XYZ    110.625     112.678
2007-04-30 XYZ    111.125     113.285
2007-05-01 XYZ    113.750     113.589
2007-05-02 XYZ    114.000     112.160
2007-05-03 XYZ    113.750     112.214
2007-05-04 XYZ    112.125     112.160
2007-05-07 XYZ    109.750     111.895
2007-05-08 XYZ    111.000     111.475
2007-05-09 XYZ    110.750     110.906
  13 record(s) selected
```

Take 2007-04-26 as example, the moving average is :


```
   2007-04-23 XYZ    110.125
   2007-04-24 XYZ    109.500
   2007-04-25 XYZ    110.000
   2007-04-26 XYZ    119.750        <-
   2007-04-27 XYZ    110.625
   2007-04-30 XYZ    111.125
+  2007-05-01 XYZ    113.750
----------------------------------------
                     784.875/7 = 112.125
```

5.4 LAB13E4: Specify Window by RANGE

Stock tables have the weekends missing. RANGE can be used to overcome gaps as illustrated in this exercise.

In this exercise, we want to find the 7 days historical average for each day the stock traded.

Attempting to use ROWS in setting the window for seven calendar days actually returns 7 preceding rows. These seven rows span more than one calendar week.

RANGE fixes this problem by recognizing the weekend gap. Therefore RANGE is appropriate when there are gaps in the input data.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab13e4.db2
```

This job will execute the following SQL statement:

```
SELECT date,
       CASE DAYOFWEEK(DATE)
                WHEN 2 THEN 'MONDAY'
                WHEN 3 THEN 'TUESDAY'
                WHEN 4 THEN 'WEDNESDAY'
                WHEN 5 THEN 'THURSDAY'
                WHEN 6 THEN 'FRIDAY'
            END AS DAY,
       close_price,
       decimal(avg(close_price) over (order by date range
         00000006. preceding),7,2) as avg_7_range,
       count(close_price) over (order by date range
         00000006. preceding) as count_7_range
  FROM stock;
```

Note: 00000006 in time interval, compatible with date data type (YYYYMMDD). “Range 00000006 preceding” means from six days before current date to current date so it is 7 days in total.

You should see the following output:

```
DATE       DAY       CLOSE_PRICE AVG_7_RANGE COUNT_7_RANGE
2007-04-23 MONDAY    110.125     110.12      1
```



```
2007-04-24 TUESDAY   109.500     109.81      2
2007-04-25 WEDNESDAY 110.000     109.87      3
2007-04-26 THURSDAY  119.750     112.34      4
2007-04-27 FRIDAY    110.625     112.00      5
2007-04-30 MONDAY    111.125     112.20      5
2007-05-01 TUESDAY   113.750     113.05      5   <-
2007-05-02 WEDNESDAY 114.000     113.85      5
2007-05-03 THURSDAY  113.750     112.65      5
2007-05-04 FRIDAY    112.125     112.95      5
2007-05-07 MONDAY    109.750     112.67      5
2007-05-08 TUESDAY   111.000     112.12      5
2007-05-09 WEDNESDAY 110.750     111.47      5
  13 record(s) selected
```

Take 2007-05-01 as example, the moving average is :

```
   2007-04-25 WEDNESDAY 110.000
   2007-04-26 THURSDAY  119.750
   2007-04-27 FRIDAY    110.625
   2007-04-30 MONDAY    111.125
+  2007-05-01 TUESDAY   113.750
----------------------------------------
                     565.25/5 = 113.05
```

5.5 LAB13E5: MAX() and MIN()

In this exercise, we are going to use MAX() and MIN() column functions. We want to find the seven day centered MAX and MIN of XYZ stock for each day the stock traded.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab13e5.db2
```

This job will execute the following SQL statement:

```
SELECT DATE,SYMBOL,CLOSE_PRICE,

       DECIMAL(MAX(CLOSE_PRICE)

         OVER (ORDER BY DATE

               ROWS BETWEEN 3 PRECEDING AND 3 FOLLOWING),

         6,3) AS MAX_CP,

       DECIMAL(MIN(CLOSE_PRICE)

         OVER (ORDER BY DATE

               ROWS BETWEEN 3 PRECEDING AND 3 FOLLOWING),
```



```
         6,3) AS MIN_CP

FROM STOCK;
```

You should see the following output:

```
DATE       SYMBOL CLOSE_PRICE MAX_CP   MIN_CP
2007-04-23 XYZ    110.125     119.750  109.500
2007-04-24 XYZ    109.500     119.750  109.500
2007-04-25 XYZ    110.000     119.750  109.500
2007-04-26 XYZ    119.750     119.750  109.500   <-
2007-04-27 XYZ    110.625     119.750  109.500
2007-04-30 XYZ    111.125     119.750  110.000
2007-05-01 XYZ    113.750     119.750  110.625
2007-05-02 XYZ    114.000     114.000  109.750
2007-05-03 XYZ    113.750     114.000  109.750
2007-05-04 XYZ    112.125     114.000  109.750
2007-05-07 XYZ    109.750     114.000  109.750
2007-05-08 XYZ    111.000     113.750  109.750
2007-05-09 XYZ    110.750     112.125  109.750
  13 record(s) selected
```

Take 2007-04-26 as example:

```
2007-04-23 XYZ    110.125     119.750  109.500
2007-04-24 XYZ    109.500     119.750  109.500
2007-04-25 XYZ    110.000     119.750  109.500
2007-04-26 XYZ    119.750     119.750  109.500   <-
2007-04-27 XYZ    110.625     119.750  109.500
2007-04-30 XYZ    111.125     119.750  110.000
2007-05-01 XYZ    113.750     119.750  110.625
```

The max in this window is 119.750 and the min in this window is 109.500.

End of lab 13.

6. Lab 14 – Greater Timestamp Precision

DB2 10 for z/OS supports greater timestamp precision for applications that are written in host languages, such as Java, .NET, and more.

For the TIMESTAMP data type, the number of digits of fractional seconds (a portion of a second that is greater than 0 but less than 1) has been extended to support a range from 0 to 12. The default is 6 digits. This enhancement enables TIMESTAMP data with varying timestamp precision (the maximum number of digits that can be

included in a fractional second) to be stored or loaded into DB2® tables, and allows manipulation of that data.

In other words, a TIMESTAMP is a **six or** seven-part value (year, month, day, hour, minute, second, and **optional fractional second**) with an optional time zone specification(will discuss more in lab 15), that represents a date and time. For example:

'2000-01-15-08.30.00.123456789012'

We are going to learn about this new feature using CLP.

6.1 LAB14E1: TIMESTAMP Columns with Different TIMESTAMP Precision

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab14e1.db2
```

This job will create and populate a table with TIMESTAMP columns of different timestamp precisions. Below are the SQL statements for your reference:

```
CREATE TABLE LAB14_TB1    ( COL1 CHAR(6),
                            C1  TIMESTAMP,
                            C2  TIMESTAMP(0),
                            C3  TIMESTAMP(1),
                            C4  TIMESTAMP(6),
                            C5  TIMESTAMP(12)
                          );

INSERT INTO LAB14_TB1 VALUES (
   'ABC'
  ,'2000-01-15-08.30.00.888'
  ,'2000-01-15-08.30.00'
  ,'2000-01-15-08.30.00.3'
  ,'2000-01-15-08.30.00.1'
  ,'2000-01-15-08.30.00.123456789012'
  );
```

You will see the following output:

```
SELECT * FROM LAB14_TB1

COL1   C1                          C2                    C3
C4
                          C5
```



```
ABC    2000-01-15-08.30.00.888    2000-01-15-08.30.00.0 2000-
01-15-08.30.00.3 2000-01-15-08.30.00.1      2000-01-15-
08.30.00.123456789012

  1 record(s) selected
```

The actual length of the timestamp in C5 can be found as follows:

```
SELECT LENGTH(VARCHAR(C5)) AS TS12_LENGTH from LAB14_TB1

TS12_LENGTH

32

  1 record(s) selected
```

6.2 LAB14E2: Impact on Built-in-Functions

Any SQL statements, built-in-functions, etc that allow TIMESTAMP data type is enhanced to support this new features. We are going to show some examples on built-in-functions.

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab14e2.db2
```

This job will execute the following SQL statements:

```
SELECT DATE(C5) as DATE,
       DAY(C5) as DAY,
       DAYS(C5) as DAYS,
       MONTH(C5) as MONTH,
       YEAR(C5) as YEAR,
       HOUR(C5) as HOUR,
       MINUTE(C5) as MINUTE,
       SECOND(C5,0) as SECOND,
       MICROSECOND(C5) as MICROSECOND
FROM LAB14_TB1;
```

You should the following output:

```
DATE       DAY         DAYS        MONTH       YEAR
HOUR        MINUTE

   SECOND MICROSECOND

2000-01-15 15          730134      1           2000        8
30

   0      123456
```



```
  1 record(s) selected
```

Note: For `MICROSECOND()`, if the argument is a timestamp or string representation of a timestamp, the result is the microsecond part of the value, which is an integer between 0 and 999999. If the precision of the timestamp exceeds 6, the value is truncated.

```
--------------------------------------------------------------

-- add 1 microsecond to C5: 2000-01-15-08.30.00.123456789012

--------------------------------------------------------------

SELECT TIMESTAMPADD(1, 1,

    (SELECT C5 FROM LAB14_TB1)

   ) as nextTS

FROM SYSIBM.SYSDUMMY1
```

For `TIMESTAMPADD`, the 1st parameter is the interval: value of 1 means microseconds; the 2nd parameter is the actual number of microsecond to add to the 3rd parameter.

You should see the following result:

```
NEXTTS

2000-01-15 08:30:00.123457789

  1 record(s) selected
```

We will see more examples of greater precision timestamp in the lab of **Timestamp With Time Zone**.

End of Lab 14.

7. Lab 15 – Timestamp With Time Zone

The existing TIMESTAMP data type does not capture an associated time zone, and this make the timestamp ambiguous. To solve this problem, some customers stored time zone in another column.

DB2 10 for z/OS introduces a new data type: TIMESTAMP WITH TIME ZONE to resolve this problem. The time zone is the difference in hours and minutes between local time and UTC. The range of the hour offset is -12 to 14, and the minute offset is 00 to 59. The optional time

zone is specified in the format ±th.tm, with values ranging from -12.59 to +14.00.

The external representation of a TIMESTAMP WITH TIME ZONE value is the local timestamp followed by the time zone offset. For example, New York is 5 hours behind Greenwich during standard time, so New York time "8:15" on 2010-02-10 can be represented as '2010-02-10.08.15.00-5:00'. This timestamp with time zone value represents a UTC value '2010-02-10.13.15.00', which is derived by subtracting the time zone offset from local timestamp.

We are going to learn more about this new support using CLP in the following exercises.

7.1 LAB15E1: TIMESTAMP columns with TIME ZONE

1. In the command prompt that we have opened earlier, type

```
%db2% -vf lab15e1.db2
```

This job will create and populate a table with TIMESTAMP columns of different timestamp precisions with WITH TIME ZONE clause. Below are the SQL statements for your reference:

```
CREATE TABLE LAB15_Table

  (ID    INTEGER      NOT NULL WITH DEFAULT,

   TSTZ0 TIMESTAMP(0) WITH TIME ZONE NOT NULL WITH DEFAULT

           '0001-01-01-00.00.01 -12:59',

   TSTZ5   TIMESTAMP(5) WITH TIME ZONE WITH DEFAULT NULL,

   TSTZ12  TIMESTAMP(12) WITH TIME ZONE WITH DEFAULT

           '9999-12-31-23.59.59.123456789012 +14:00',

   PRIMARY KEY (ID)

  );


INSERT INTO LAB15_Table    (ID, TSTZ0, TSTZ5, TSTZ12)

 VALUES (1, '2000-01-01-02.02.02. -12:59',

        '2005-1-1 2:02:02.12345 +00:50',

        '2012-01-01 02:02:02.123456789012 +14:00');
```

```
INSERT INTO LAB15_Table    (ID, TSTZ0, TSTZ5, TSTZ12)
 VALUES (2, DEFAULT,
        DEFAULT,
        DEFAULT);


INSERT INTO LAB15_Table    (ID) VALUES (3);


SELECT ID, SUBSTR(CHAR(TSTZ0),1,25) AS TSTZ0,
           SUBSTR(CHAR(TSTZ5),1,31) AS TSTZ5,
           SUBSTR(CHAR(TSTZ12),1,38) AS TSTZ12
FROM LAB15_Table ORDER BY ID

ID          TSTZ0                     TSTZ5
TSTZ12

1           2000-01-01-02.02.02-12:59 2005-01-01-
02.02.02.12345+00:50 2012-01-01-02.02.02.123456789012+14:00

2           0001-01-01-00.00.01-12:59 <null>
9999-12-31-23.59.59.123456789012+14:00

3           0001-01-01-00.00.01-12:59 <null>
9999-12-31-23.59.59.123456789012+14:00

  3 record(s) selected
```

7.2 LAB15E2: Insert Timestamp with different TIME ZONE

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab15e2.db2
```

This job will insert the following data to the table created in lab15e1:

```
INSERT INTO LAB15_Table(ID, TSTZ0,  TSTZ5, TSTZ12)
 VALUES (4,
   TIMESTAMP '2000-12-31 02:02:02-08:00' AT TIME ZONE
'+08:00',
   TIMESTAMP '2005-12-31 02:02:02.12345+00:00' AT TIMEZONE '-
08:00',
```



```
   TIMESTAMP '2012-12-31 02:02:02.123456789012+08:00' AT
TIMEZONE '-08:00'
     );
```

Below is SELECT output for ID=4:

```
ID          TSTZ0                     TSTZ5
TSTZ12

4           2000-12-31-18.02.02+08:00 2005-12-30-
18.02.02.12345-08:00 2012-12-30-10.02.02.123456789012-08:00

  1 record(s) selected
```

DB2 adjust the time according to the time zone info given in the INSERT statement and store the time zone value from 'AT TIMEZONE'

7.3 LAB15E3: Insert Timestamp with AT LOCAL, SESSION TIME ZONE

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab15e3.db2
```

This job will find out the CURRENT(local) and SESSION time zone info and then insert the following timestamp with AT LOCAL and AT TIMEZONE SESSION TIME ZONE to the table created in lab15e1.

Below is the system info about time zone:

```
SESSION_TIME_ZONE    CURRENT_TIMESTAMP

CURRENT_TIME_ZONE    CURRENT_TIMESTAMP_WITH_TIME_ZONE

-05:00               2010-09-27-16.52.24.350904

-50000               2010-09-27-14.52.24.350904-07:00
```

Below is the SQL statement for INSERT:

```
INSERT INTO LAB15_Table (ID, TSTZ0,  TSTZ5,  TSTZ12)
 VALUES (5,
        '2000-12-31-02.02.02-08:00' AT LOCAL,
```



```
        '2005-12-31 02:02:02.12345+08:00' AT TIMEZONE SESSION
            TIME ZONE,
         TIMESTAMP '2012-12-31 02:02:02.123456789012+08:00'
           AT TIMEZONE SESSION TIME ZONE     )
```

Below is SELECT output for ID=5:

```
ID          TSTZ0                     TSTZ5
TSTZ12

5           2000-12-31-05.02.02-05:00 2005-12-30-
13.02.02.12345-05:00 2012-12-30-13.02.02.123456789012-05:00

  1 record(s) selected
```

Both LOCAL(CURRENT_TIME_ZONE) and SESSION_TIME_ZONE is -05:00. DB2 adjust the time according to the time zone info given in the INSERT statement and store the time zone value -05:00

7.4 LAB15E4: Insert Timestamp with CURRENT TIMESTAMP and WITH/WITHOUT TIME ZONE clause

In this exercise, we are going to insert timestamp with CURRENT TIMESTAMP and WITH/WITHOUT TIME ZONE clause.

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab15e4.db2
```

This job will insert the following timestamp with WITHOUT TIME ZONE clause to the table created in lab15e1.

Below is the SQL statement:

```
INSERT INTO LAB15_Table (ID, TSTZ0,  TSTZ5,  TSTZ12)
 VALUES (6,
         CURRENT TIMESTAMP(5) WITHOUT TIME ZONE,
         CURRENT TIMESTAMP(1) WITH TIME ZONE,
         SYSTIMESTAMP
```



```
     );
```

Below is the output for ID=6:

```
ID          TSTZ0                     TSTZ5
TSTZ12

6           2010-09-27-17.13.58-05:00 2010-09-27-
17.13.58.00000-05:00 2010-09-27-17.13.58.007986699707-05:00

  1 record(s) selected
```

End of lab 15.

8. LAB 21 – XQuery

XQuery is delivered as a V10 post-GA features.

In this lab, we are going to learn about the following features in XQuery:

- FLWOR expression (For, Let, Where, Order by, Return)
- Conditional expression
- Node comparison and Value comparison
- Castable expression and fn:avg built-in function
- XQuery constructor

Part of this lab is designed for you to write some XQuery. For those exercises, you are supposed to replace %%%%%%% with your answer and execute them. Answers can be found at the end of this lab instruction.

8.1 LAB21E1: Create table and Insert

In this exercise, we create a table and then insert an XML document there.

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab21e1.db2
```

Below is the SQL statement:

```
CREATE TABLE PURCHASE_ORDER (POID INTEGER, PODOC XML);

INSERT INTO PURCHASE_ORDER (POID, PODOC) VALUES (1,
 '<purchaseOrder orderDate="2011-05-18">
    <shipTo country="US">
      <name>Alice Smith</name>
      <street>123 Maple Street</street>
      <city>San Jose</city>
      <state>California</state>
      <zip>95123</zip>
    </shipTo>
    <billTo country="US">
      <name>Robert Smith</name>
      <street>8 Oak Avenue</street>
      <city>San Jose</city>
      <state>California</state>
      <zip>95123</zip>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <items>
      <item partNum="872-AA">
        <productName>Lawnmower</productName>
        <quantity>1</quantity>
        <USPrice>149.99</USPrice>
        <comment>Confirm this is gas powered</comment>
        <shipDate>2011-05-20</shipDate>
      </item>
      <item partNum="926-AA">
        <productName>Baby Monitor</productName>
        <quantity>2</quantity>
        <USPrice>39.98</USPrice>
        <shipDate>2011-05-22</shipDate>
      </item>
      <item partNum="945-ZG">
        <productName>Sapphire Bracelet</productName>
        <quantity>2</quantity>
        <USPrice>178.99</USPrice>
        <comment>Not shipped</comment>
      </item>
    </items>
  </purchaseOrder>');
```

8.2 LAB21E2: FLWOR Expression

In this exercise, we are going to learn FLWOR expression. We are going to write a query that find all items purchased in a purchase order, and list the items in ascending order based on their total cost.

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab21e2.db2
```


Below is the SQL statement:

```
SELECT XMLQUERY(
  'for $i at $pos in $po/purchaseOrder/items/item
   let $p := $i/USPrice,
       $q := $i/quantity,
       $r := $i/productName
   where xs:decimal($p) > 0 and xs:integer($q) > 0
   order by $p * $q
   return fn:concat("Item ", $pos, " productName: ", $r,
                    " price: US$", $p, " quantity: ", $q)
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

For each item under /purchaseOrder/items, we assign

```
$p := $i/USPrice,
$q := $i/quantity,
$r := $i/productName
```

only for those item with price > 0 and quantity > 0

Order by price * quantity

Then, concatenate the productName, their position, the price, and quantity.

You should see the following output (reformat for display purpose):

```
Item 2 productName: Baby Monitor price: US$39.98 quantity: 2
Item 1 productName: Lawnmower price: US$149.99 quantity: 1
Item 3 productName: Sapphire Bracelet price: US$178.99
quantity: 2
```

8.3 LAB21E3: Exercise on FLWOR Expression

In this exercise, you are going to write a query to find out all **shipped** items in a purchase order with POID=1.

1. In the command prompt, that we have opened earlier, type

```
notepad lab21e3.db2
```

Below is the SQL statement:

```
SELECT XMLQUERY(
  'for $i at $pos in $po/purchaseOrder/items/item
```



```
   let $d := $i/shipDate,
       $r := $i/productName
   where %%%%%%%%
   return fn:concat("Item ", $pos, " productName: ", $r,
                    " shipDate: ", $d)
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

Hints:

- all shipped items will have shipDate earlier than today
- fn:current-date() return current date

2. Replace %%%%%%%% with your answer.

3. Press Ctrl-S to save your change.

4. To execute, in the command prompt, that we have opened earlier, type

```
%db2% -vf lab21e3.db2
```

You should see the following output:

```
Item 1 productName: Lawnmower shipDate: 2011-05-20 Item 2
productName: Baby Monitor shipDate: 2011-05-22
```

Note: Item 3 has NOT shipped yet.

8.4 LAB21E4: Conditional Expression

In this exercise, we are going to use conditional expression to find out all items purchased in a purchase order, and calculate the shipping cost for each item.

The shipping cost for any item with unit price > $100 is $10 and the shipping cost for any item below $100 is $5.

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab21e4.db2
```

Below is the SQL statement:

```
SELECT XMLQUERY(
  'for $i in $po/purchaseOrder/items/item
   return (
```



```
    if (xs:decimal($i/USPrice) < 100)
    then fn:concat($i/productName, " : shipping cost US$", 5)
    else fn:concat($i/productName, " : shipping cost US$", 10)
  )
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

In this query, we use conditional expression (IF, THEN, ELSE) to check the price and decide the shipping cost.

You should see the following output:

```
Lawnmower : shipping cost US$10 Baby Monitor : shipping cost
US$5 Sapphire Bracelet : shipping cost US$10
```

8.5 LAB21E5: Exercise on Conditional Expression

In this exercise, you are going to write a query to find out the shipping cost based on the shipTo address. If the item is shipped inside California, the shipping cost is $8, otherwise the shipping cost is $12.

1. In the command prompt, that we have opened earlier, type

```
notepad lab21e5.db2
```

Below is the SQL statement:

```
SELECT XMLQUERY(
  'let $s := $po/purchaseOrder/shipTo
   let $b := $s/@country = "US" and %%%%%%%%
   for $i in $po/purchaseOrder/items/item
   return (
     if ($b)
     then fn:concat($i/productName, " : shipping cost US$", 8)
     else fn:concat($i/productName, " : shipping cost US$", 12)
  )
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

2. Replace %%%%%%%% with your answer.

3. Press Ctrl-S to save your change.

4. To execute, in the command prompt, that we have opened earlier, type

```
%db2% -vf lab21e5.db2
```

For a successful execution, you should see the following output:

```
Lawnmower : shipping cost US$8 Baby Monitor : shipping cost
US$8 Sapphire Bracelet : shipping cost US$8
```

8.6 LAB21E6: Castable Expression and fn:avg()

In this exercise, you are going to use castable expression and fn:avg() to find out the average cost of each item in the purchase order.

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab21e6.db2
```

Below is the SQL statement:

```
SELECT XMLQUERY(
  'let $cost := (
     for $i in $po/purchaseOrder/items/item
     let $p := if ($i/USPrice castable as xs:decimal)
               then xs:decimal($i/USPrice)
               else 0.0
     let $q := if ($i/quantity castable as xs:integer)
               then xs:integer($i/quantity)
               else 0
     return $p * $q)
   return fn:round(fn:avg($cost))
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

In this query, we

- Use castable expression to check whether USPrice can be casted as xs:decimal; similarly, we check whether quantity can be casted as xs:integer
- find the total cost for each item
- call fn:avg() to find out the average cost

The result should be 196.

8.7 LAB21E7: Exercise on Castable Expression and fn:avg()

In this exercise, you are going to write a query to find out the average duration between orderDate and shipDate(If not shipped yet, use the date “2011-06-29”)

1. In the command prompt, that we have opened earlier, type

```
notepad lab21e7.db2
```

Below is the SQL statement

```
SELECT XMLQUERY(
  'if ($po/purchaseOrder/@orderDate castable as xs:date) then
     let $d := (
       let $od := xs:date($po/purchaseOrder/@orderDate)
       for $i in $po/purchaseOrder/items/item
       let $sd := if ($i/shipDate castable as xs:date)
                  then xs:date($i/shipDate)
                  else xs:date("2011-06-29")
       return %%%%%%%%
       )
     return fn:avg($d)
   else "Invalid orderDate!"
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1
```

2. Replace %%%%%%%% with your answer.

3. Press Ctrl-S to save your change.

4. To execute, in the command prompt, that we have opened earlier, type

```
%db2% -vf lab21e7.db2
```

You should see the following output: P16D

8.8 LAB21E8: XQuery Constructor 1

In this exercise, we are going to use XQuery constructor to construct an XML document. We will start with a simple one first.

1. In the command prompt, that we have opened earlier, type


```
%db2% -vf lab21e8.db2
```

Below is the SQL statement:

```
SELECT XMLQUERY('<a>Hello</a>')

FROM SYSIBM.SYSDUMMY1;
```

The output is:

```
<a>Hello</a>
```

8.9 LAB21E9: XQuery Constructor 2

We are going to use XQuery constructor to list all items that are NOT shipped yet.

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab21e9.db2
```

Below is the SQL statement:

```
SELECT XMLQUERY(
  'declare boundary-space strip;
   declare copy-namespaces no-preserve, inherit;
   <shipping orderDate="{$po/purchaseOrder/@orderDate}">
     <shipTo>{ let $s := $po/purchaseOrder/shipTo
               return fn:concat($s/state, " ", $s/zip, " ",
                   $s/@country)
             }
     </shipTo>
     {for $i in $po/purchaseOrder/items/item
      where fn:not($i/shipDate castable as xs:date) or
            xs:date($i/shipDate) > fn:current-date()
      return <item partNum="{$i/@partNum}">
               <partName> {$i/productName/text()} </partName>
               { $i/quantity , $i/USPrice }
             </item>
     }
   </shipping>
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

Note:

```
where fn:not($i/shipDate castable as xs:date) or
      xs:date($i/shipDate) > fn:current-date()
```

For item not shipped yet, there is no <shipDate>

Below is the output (re-format for display purpose):

```
<shipping orderDate="2011-05-18">
  <shipTo>California 95123 US</shipTo>
  <item partNum="945-ZG">
    <partName>Sapphire Bracelet</partName>
    <quantity>2</quantity>
    <USPrice>178.99</USPrice>
  </item>
</shipping>
```

8.10 LAB21E10: XQuery Constructor 3 (used in XMLModify)

We can also use XQuery constructor inside XMLModify. This provides an easy way to construct a source document for update.

Suppose we are going to add the following item to the purchase order that we have inserted.

```
<item partNum="833-AA">
  <productName>Lapis necklace</productName>
  <quantity>1</quantity>
  <USPrice>99.95</USPrice>
  <shipDate>2011-11-20</shipDate>
</item>
```

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab21e10.db2
```

Below are the SQL statements:

```
UPDATE PURCHASE_ORDER
  SET PODOC = XMLMODIFY(
    'insert node <item partNum="833-AA">
                   <productName>Lapis necklace</productName>
                   <quantity>1</quantity>
                   <USPrice>99.95</USPrice>
                   <shipDate>2011-11-20</shipDate>
                 </item>
     into /purchaseOrder/items
    ')
WHERE POID = 1;
```

In the same script, we also verify the inserted item,

```
SELECT XMLQUERY(
  '$po/purchaseOrder/items/item[@partNum = "833-AA"]'
  PASSING PODOC AS "po")
```

```
FROM PURCHASE_ORDER
WHERE POID = 1;
```

The output should look like this:

```
<item partNum="833-AA"><productName>Lapis
necklace</productName><quantity>1</qua
ntity><USPrice>99.95</USPrice><shipDate>2011-11-
20</shipDate></item>
```

At the end, the script deletes the item we just inserted:

```
UPDATE PURCHASE_ORDER
  SET PODOC = XMLMODIFY(
    'delete node /purchaseOrder/items/item[@partNum = "833-
AA"]
    ')
WHERE POID = 1;
```

8.11 LAB21E11: Exercise on XQuery Constructor

In this exercise, you are going to write a SQL statement to insert a new purchase order with

- POID=2,
- orderDate is today,
- both the shipTo and billTo information is same as the purchase order with POID=1 (the one we inserted),
- only 1 item is ordered.

```
<item partNum="833-AA">
<productName>Lapis necklace</productName>
<quantity>1</quantity>
<USPrice>99.95</USPrice>
</item>
```

- The shipDate is one week after the orderDate.

1. In the command prompt, that we have opened earlier, type

```
notepad lab21e11.db2
```

Below is the SQL statement

```
INSERT INTO PURCHASE_ORDER (POID, PODOC) VALUES (2,
  (SELECT XMLDOCUMENT(
```



```
XMLQUERY(
  'declare boundary-space strip;
   <purchaseOrder orderDate="{fn:current-date()}">
     {%%%%%%%%%}
     <items>
       <item partNum="833-AA">
         <productName>Lapis necklace</productName>
         <quantity>1</quantity>
         <USPrice>99.95</USPrice>
         <shipDate>
            {fn:current-date() + xs:dayTimeDuration("P7D")}
         </shipDate>
       </item>
     </items>
   </purchaseOrder>
  '
  PASSING PODOC AS "po"))
 FROM PURCHASE_ORDER
 WHERE POID = 1
));
```

2. Replace %%%%%%%% with your answer.

3. Press Ctrl-S to save your change.

4. To execute, in the command prompt, that we have opened earlier, type

```
%db2% -vf lab21e11.db2
```

For a successful execution, you should see the following output(reformat for display purpose):

```
<purchaseOrder orderDate="2011-07-04-07:00">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>San Jose</city>
    <state>California</state>
    <zip>95123</zip>
  </shipTo>
  <billTo country="US">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>San Jose</city>
    <state>California</state>
    <zip>95123</zip>
  </billTo><items>
  <item partNum="833-AA">
    <productName>Lapis necklace</productName>
    <quantity>1</quantity>
    <USPrice>99.95</USPrice>
    <shipDate>2011-07-11-07:00</shipDate>
```



```
  </item>
</items></purchaseOrder>
```

8.12 LAB21E12: Node Comparison and Value Comparison

In this exercise, we are going to learn about node comparison and value comparison.

Suppose we have an XML document like:

```
<instruction>
       <step>Do something 1</step>
       <step>Do something 2</step>
       <step>Do something 3</step>
       <step>Do something 4</step>
       <step>Do something 5</step>
       <step>Do something 6</step>
       <step>Do something 7</step>
       <step>Do something 8</step>
       <step>Do something 9</step>
 </instruction>
```

And we want to find all the steps before the 5th step.

1. In the command prompt, that we have opened earlier, type

```
%db2% -vf lab21e12.db2
```

Below are the SQL statements:

```
SELECT XMLQUERY(
  'let $step5 := $ins/instruction/step[5]
  for $s in $ins/instruction/step[. << $step5]
  return $s'
  PASSING
   XMLPARSE(document
     '<instruction>
       <step>Do something 1</step>
       <step>Do something 2</step>
       <step>Do something 3</step>
       <step>Do something 4</step>
       <step>Do something 5</step>
       <step>Do something 6</step>
       <step>Do something 7</step>
       <step>Do something 8</step>
       <step>Do something 9</step>
     </instruction>') as "ins"
)
FROM SYSIBM.SYSDUMMY1;
```


Below is the output (re-format for display purpose):

```
<step>Do something 1</step>
<step>Do something 2</step>
<step>Do something 3</step>
<step>Do something 4</step>
```

9. Answers for Selected Exercises

The following is a set of answers for the self-study exercises.

9.1 LAB21E3

```
-------------------------------------------------------------------------
-
-- Exercise on FLWOR expression:
-- Query all shipped items in a purchase order with POID = 1
-- Note: all shipped items will have shipDate earlier than today
-- replace %%%%%%%% with your answer
-- hint: fn:current-date() return current date
-------------------------------------------------------------------------
-

SELECT XMLQUERY(
  'for $i at $pos in $po/purchaseOrder/items/item
   let $d := $i/shipDate,
       $r := $i/productName
   where xs:date($d) < fn:current-date()
   return fn:concat("Item ", $pos, " productName: ", $r,
                    " shipDate: ", $d)
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1
```

9.2 LAB21E5

```
-------------------------------------------------------------------------
-- Exercise on conditional expression:
-- Calculate the shipping cost based on the shipTo address.
-- If it's shipped inside the state California, the shipping cost is
-- $8,
-- otherwise, the shipping cost is $12.
-- Replace %%%%%%%% with your answer
-------------------------------------------------------------------------

SELECT XMLQUERY(
  'let $s := $po/purchaseOrder/shipTo
   let $b := $s/@country = "US" and $s/state = "California"
   for $i in $po/purchaseOrder/items/item
```

```
   return (
     if ($b)
     then fn:concat($i/productName, " : shipping cost US$", 8)
     else fn:concat($i/productName, " : shipping cost US$", 12)
   )
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1
```

9.3 LAB21E7

```
-------------------------------------------------------------------------
-- Exercise on castable expression and fn:avg
-- Calculate the average duration between orderDate
-- and shipDate (If not shipped yet, use the date "2011-06-29").
--
--
-- Replace %%%%%%%% with your answers
-------------------------------------------------------------------------

SELECT XMLQUERY(
  'if ($po/purchaseOrder/@orderDate castable as xs:date) then
     let $d := (
       let $od := xs:date($po/purchaseOrder/@orderDate)
       for $i in $po/purchaseOrder/items/item
       let $sd := if ($i/shipDate castable as xs:date)
                  then xs:date($i/shipDate)
                  else xs:date("2011-06-29")
       return ($sd - $od)
       )
     return fn:avg($d)
   else "Invalid orderDate!"
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1
```

9.4 LAB21E11

```
-------------------------------------------------------------------------
-- Exercise on XQuery constructor
-- Insert a new purchase order with POID = 2. orderDate is today, and
-- both the shipTo and billTo information is the same as the purchase
-- order with POID = 1, but only one item is ordered,
--    <item partNum="833-AA">
--       <productName>Lapis necklace</productName>
--       <quantity>1</quantity>
--       <USPrice>99.95</USPrice>
--    </item>
-- The shipDate is one week after the orderDate.
-- Hint: use XMLQUERY with XQuery constructor inside a subquery.
--
-- Replace %%%%%%%% with your answer
```



```
-------------------------------------------------------------------------

INSERT INTO PURCHASE_ORDER (POID, PODOC) VALUES (2,
  (SELECT XMLDOCUMENT(
    XMLQUERY(
    'declare boundary-space strip;
     <purchaseOrder orderDate="{fn:current-date()}">
       {$po/purchaseOrder/shipTo, $po/purchaseOrder/billTo}
       <items>
         <item partNum="833-AA">
           <productName>Lapis necklace</productName>
           <quantity>1</quantity>
           <USPrice>99.95</USPrice>
           <shipDate>
              {fn:current-date() + xs:dayTimeDuration("P7D")}
           </shipDate>
         </item>
       </items>
     </purchaseOrder>
    '
    PASSING PODOC AS "po") )
   FROM PURCHASE_ORDER
   WHERE POID = 1
  ))
```

SQLADRIA SEMINAR
Opatija, 24 - 25 November 2011

Learning pureXML and New SQL Features inDB2 10 for z/OS Using SPUFI and CLP

Presentation

Sponsored by

Gougen Zhang – Distinguish Engineer
Jane Man – Advisory Engineer

11/10/2010

Learning pureXML and New SQL Features in DB2 10 for z/OS Using SPUFI and CLP

IDUG EMEA 2010

Agenda

- Introduction
- Lab 1 : setup for Lab 2, Lab 3, Lab 4, and Lab 5
- Lab 2 : XMLQuery
- Lab 3 : XMLExist and XML index
- Lab 4 : XMLTable
- Lab 5: XML Schema
- Lab 6: XML Type Modifier
- Lab 7: Sub-document update on XML document
- Lab 8 : XML date and time support
- Lab 9: Bitemporal Support
- Lab 10: Row Permissions and Column Masks
- Lab 11: SQL PL extensions for scalar UDF and table UDFs
- Lab 12: Extended Implicit CAST support
- Lab 13: New OLAP functions(moving average)
- Lab 14: Greater Precision Timestamp
- Lab 15: Timestamp with Time Zone

- ***Disclaimer:*** *Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.*

Introduction - Objectives

- Learn XPath
- Learn SQL/XML operators
  - XMLQUERY
  - XMLEXISTS
  - XMLTABLE
  - XMLCAST
- Create and Utilize an XML Index
- Learn XML Schema
  - Register XML Schema
  - Perform validation of an XML documents
- Learn sub-document update on XML documents
- Learn XML type modifier
- Learn XML data and time support

Introduction – Objectives – cont'd

- Learn Bitemporal support
- Learn row permission and column masks
- Learn SQL PL extensions for scalar UDF and table UDFs
- Learn new OLAP functions(moving average)
- Learn about greater precision timestamp
- Learn about timestamp with time zone
- Learn about extended implicit case support

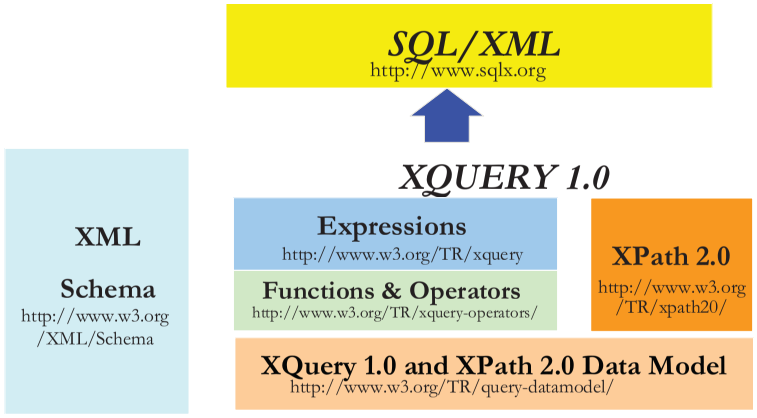
XPath

- Xpath is an expression language for locating parts of a document
- XPath is used inside of another language (XQuery, SQL/XML, XSLT)
- Path expressions denote levels of an XML document

```
/Customer/Name/FirstName
```

- XPath supports complex expressions and functions/operators beyond the scope of today's lab.

Relevant XML Query Standards



SQL/XML

- Extensions to SQL for XML processing
  - XMLQUERY – Evaluate an XPath expression (against an XML document)

```
SELECT
  XMLQUERY('/Customer/Name/FirstName'
           PASSING XMLCOLUMN)
```

  - XMLEXISTS – A predicate that returns true when the XPath expression evaluates to non-empty

```
WHERE
  XMLEXISTS('/Customer/Name[FirstName='Fred']'
        PASSING XMLCOLUMN)
```

SQL/XML

- XMLTABLE – a table expression to return XML data as a relational table

```
SELECT ... FROM
XMLTABLE('/Customer/Name'
         PASSING XMLCOLUMN
         COLUMNS FIRSTNAME PATH 'firstname',
                 LASTNAME  PATH 'lastname') X
```

- XMLCAST – cast data from the XML datatype to other datatypes

```
XMLCAST(XMLQUERY(...) AS INTEGER)
```


XML Schema

- An XML Schema adds constraints to XML documents.
- XML Schema are registered before use in DB2.
- The validation is performed via DSN_XMLVALIDATE.

```
INSERT INTO CUSTOMER(custxml) VALUES
(XMLPARSE(DSN_XMLVALIDATE(:xmlhv,
'SYSXSR.CUSTOMERSCHEMA')));
```


Creating an XML Index

- Indexes may be created on commonly used XPath expressions.

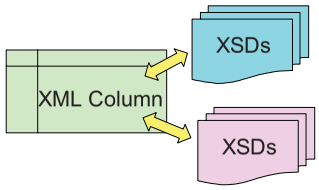
```
CREATE INDEX myidx ON CUSTOMER(CUSTXML)
GENERATE KEY USING XMLPATTERN
'/Customer/Name/FirstName'
AS SQL VARCHAR(50);
```

- Explain is used to show whether the index is used for a given query

XML Type Modifier

XML Schemas as XML column type modifier

- Add XML schemas as column type modifier, DB2 enforces the conformance.
- Two ways to identify an XML schema (existing)
  - Schema name, or target namespace + optional schema location



Example

- CREATE TABLE T1 (X1 XML(XMLSCHEMA ID SYSXSR.schema1), X2 XML);
- ALTER TABLE T1 ALTER X2 SET DATA TYPE XML(XMLSCHEMA URI 'http://www.example.com/po' ELEMENT "purchaseorder")

ALTER XML Type Modifier

- Always use ALTER TABLE T1 ALTER X2 SET DATA TYPE XML(…): whole set of schemas – DB2 figures out the delta
- CHECK DATA – validate docs or put invalid ones in exception table

Change from Type Modifier	To Type Modifier	XML Table Space Impact
No modifier	PO	Check pending
PO1	PO1, PO2	No
PO1	PO2	Check pending
PO1, PO2	PO1	Check pending
PO1	No modifier	No

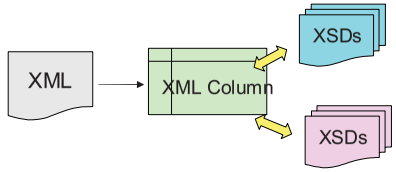
Automatic Validation

- Insert

```
INSERT INTO purchase_orders
VALUES (:hv);
```

- Update

```
UPDATE purchase_orders
   SET content =:hv;
```



- Load performing validation
- If the source is already validated according to the same XML schema in the type modifier, it won't be revalidated again.
- Schemas evolve, multiple versions coexist
  - At insert/update time, choose one of them based on information from instance doc (target namespace, schema location).
  - If no schema location, with multiple schema matches, the latest will be chosen.

Sub-document Update of XML documents

Sub-document Update

- No easy way to update parts of a document in V9
- Sub-document update with multi-versioning in DB2 10
- Only simple update: insert, replace, delete from XQuery update facility

Example

Increase premium by 10%

```
UPDATE POLICYTAB SET POLICY =
XMLMODIFY
( 'replace value of node /policy/premium with /policy/premium * 1.1')
WHERE POLICYID = '12345';
```

Add a new payment into payment record

```
UPDATE POLICYTAB SET PAYMENTS = XMLMODIFY
('insert node $n as last into /payments', :newpayment as "n")
WHERE POLICYID = :policyid;
```


Sub-document Update (cont'd)

- XMLMODIFY can only be used in RHS of UPDATE SET.
- One updater at a time for a document, concurrency control by the base table – row level locking, page level locking etc.
  - Document level lock to prevent UR reader
- Only changed rows in XML table are updated
- If there is a schema type modifier on the updated XML column, partial revalidation occurs.
  - Global constraints are not validated.

Sub-document Update – Insert Expression (1 of 3)

Sample XML document:

```
<person>
  <firstName>Joe</firstName>
  <lastName>Smith</lastName>
  <nickName>Joey</nickName>
</person>
```

Sample insert statement:

```
update personinfo set info = xmlmodify('
  insert node $ins/ename
  after  /person/nickName', xmlparse(document '
      <ename>Joe.Smith@de.ibm.com</ename>') as  "ins") ;
```

Insert Operation	Resulting XML document
insert node $ins/ename into /person , XMLPARSE(document '<ename>Joe.Smith@de.ibm.com</ename>') as "ins") (nondeterministic position)	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> <ename>Joe.Smith@de.ibm.com</ename> </person>
insert node $ins/ename as last into /person, XMLPARSE(document '<ename>Joe.Smith@de.ibm.com</ename>') as "ins")	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> <ename>Joe.Smith@de.ibm.com</ename> </person>

Sub-document Update – Insert Expression (2 of 3)

Sample XML document:

```
<person>
  <firstName>Joe</firstName>
  <lastName>Smith</lastName>
  <nickName>Joey</nickName>
</person>
```

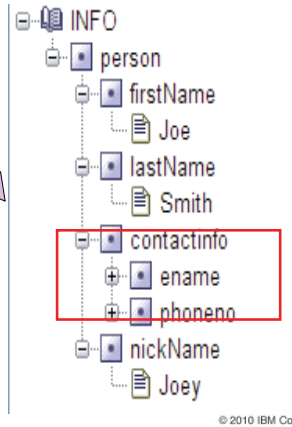
Insert Operation	Resulting XML document
Insert node $ins/ename as first into /person, XMLPARSE(document '<ename>Joe.Smith@de.ibm.com</ename>') as "ins")	<person> <ename>Joe.Smith@de.ibm.com</ename> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> </person>
Insert node $ins/ename after /person/nickName , XMLPARSE(document '<ename>Joe.Smith@de.ibm.com</ename>') as "ins")	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> <ename>Joe.Smith@de.ibm.com</ename> </person>
Insert node $ins/ename before /person/nickName , XMLPARSE(document '<ename>Joe.Smith@de.ibm.com</ename>') as "ins")	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <ename>Joe.Smith@de.ibm.com</ename> <nickName>Joey</nickName> </person>

Sub-document Update – Insert Expression (3 of 3)

- Insertion of sequence of elements also possible
- Use same keywords as introduced before

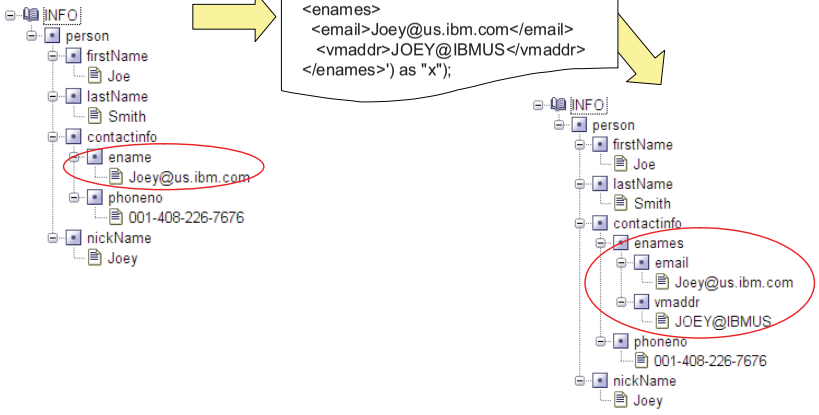
```
<person>
   <firstName>Joe</firstName>
   <lastName>Smith</lastName>
   <nickName>Joey</nickName>
</person>
```

```
update personinfo set info = xmlmodify('
  insert node $ins/contactinfo
before /person/nickName',
xmlparse(document '
<contactinfo>
<ename>Joe.Smith@de.ibm.com</ename>
<phoneno>001-408-226-7676</phoneno>
</contactinfo>
') as "ins") ;
```



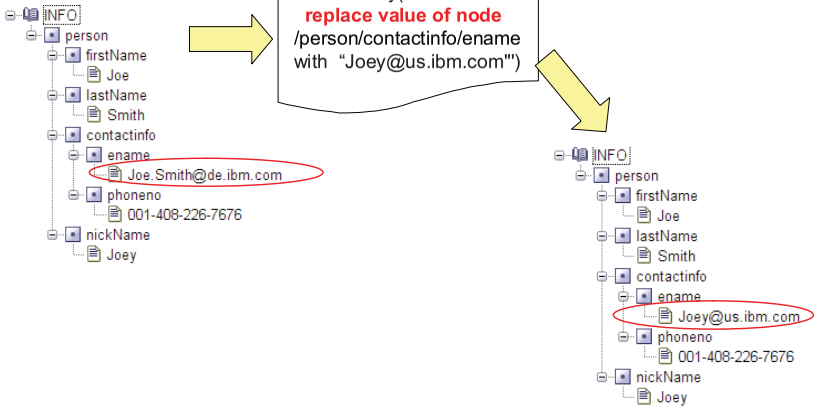
Sub-document Update – Replace Expression (2 of 2)

```
update personinfo set info = xmlmodify('
  replace node /person/contactinfo/ename
  with  $x', XMLPARSE(document '
  <enames>
    <email>Joey@us.ibm.com</email>
    <vmaddr>JOEY@IBMUS</vmaddr>
  </enames>') as "x");
```



Sub-document Update – Replace Expression (1 of 2)

```
update personinfo set info
= xmlmodify('
  replace value of node
/person/contactinfo/ename
with  "Joey@us.ibm.com"')
```



Sub-document Update – Delete Expression

– Delete can be used to delete nodes from a node sequence

```
update personinfo set info = xmlmodify('
  delete node
  /person/contactinfo ')
```

XML Date and Time Support

- Date and time are not supported in XPath in V9
  - Workaround: XMLTABLE and SQL date and time
- Add support xs:date, xs:time, and xs:dateTime and indexes for DATE and TIMESTAMP
  - Implicit timezone will be UTC if a date or time does not have a timezone
  - You get back the original or use fn:adjust-…-to-timezone() for a local timezone
- Supported types
  - xs:dateTime
  - xs:time
  - xs:date
  - xs:duration
  - xs:yearMonthDuration
  - xs:dayTimeDuration
- And a bunch of functions and operators: with time zone support, surpass SQL.

Bitemporal Support

Examples

- XMLQUERY('/purchaseorder/items/item[shipdate > xs:date("2007-01-31")]' PASSING X1)
- CREATE INDEX XIX1 ON T1(X1) GENERATE KEYS USING XMLPATTERN '/purchaseorder/items/item/shipdate' as SQL DATE
- XMLQUERY('/purchaseorder [ordertime > xs:dateTime("2007-01-31T10:20:30-08:00")]' PASSING X1)
  - Note: xs:dateTime("2007-01-31T10:20:30-8:00") does not work
- CREATE INDEX XIX1 ON T1(X1) GENERATE KEYS USING XMLPATTERN '/purchaseorder/ordertime' as SQL TIMESTAMP
- XMLQUERY('fn:adjust-dateTime-to-timezone(xs:dateTime("2007-01-31T10:20:30-05:00"), xs:dayTimeDuration("-PT8H"))')
  - Result: 2007-01-31T07:20:30-08:00

Bitemporal Support

- New concept of System_time and Business_time period
  - System_time captures DB2's creation and deletion of rows and automatically keeps historical versions of rows.
  - Business_time allows users to create their own validity period for a given row.
- Value to customers
  - meet compliance requirements : automatic propagation of old rows to a history table.
  - performs better than the home-grown solution.
  - easier to manage

Bitemporal Support – Example

```
CREATE TABLE  policy
(client              CHAR(4)          NOT NULL,
type                 CHAR(4)          NOT NULL,
copay                SMALLINT         NOT NULL,
eff_beg              DATE             NOT NULL,
eff_end              DATE             NOT NULL,
sys_start            TIMESTAMP(12) NOT NULL IMPLICITLY HIDDEN
                                 GENERATED ALWAYS AS ROW BEGIN,
sys_end              TIMESTAMP(12) NOT NULL IMPLICITLY HIDDEN
                                 GENERATED ALWAYS AS ROW END,
trans_id             TIMESTAMP(12) IMPLICITLY HIDDEN
                     GENERATED  ALWAYS AS TRANSACTION START ID,
PERIOD BUSINESS_TIME(eff_beg, eff_end),
PERIOD SYSTEM_TIME(sys_start, sys_end));
```


Bitemporal Support – Example (cont)

Step	Actual Date	Activity
1	01/01/2004	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004 (future event).
2	09/01/2004	Customer called and changed to HMO as of today (present event)
3	03/01/2006	Copay increase to $15 starting 01/01/2007 (future event)
4	06/01/2008	Cancel policy as of today (present event)
5	09/01/2008	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007 (past event)

Bitemporal Support – Example (cont)

```
CREATE TABLE  policy_hist LIKE policy;

ALTER TABLE policy
ADD VERSIONING USE HISTORY TABLE   policy_hist;

CREATE UNIQUE INDEX  ix_policy
ON policy (client, BUSINESS_TIME WITHOUT OVERLAPS);
```


Bitemporal Support – Example (cont)

Step	Actual Date	Activity
1	01/01/2004	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004 (future event).

```
INSERT INTO policy VALUES
 ('C882', 'PPO',10,'01/01/2004','12/31/9999');
```

Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	12/31/9999	2004-01-01...	9999-12-31...

Table: policy

Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01...	9999-12-31...
C882	HMO	10	09/01/2004	12/31/9999	2004-09-01...	9999-12-31...

Table: policy

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
2	09/01/2004	Customer called and changed to HMO as of today (present event)

```
UPDATE policy FOR PORTION OF BUSINESS_TIME
  FROM '09/01/2004' TO '12/31/9999'
  SET type = 'HMO'
    WHERE client = 'C882';
```


Bitemporal Support – Example (cont)

Step	Actual Date	Activity
3	03/01/2006	Copay increase to $15 starting 01/01/2007 (future event)

```
UPDATE policy FOR PORTION OF BUSINESS_TIME
  FROM '01/01/2007' TO '12/31/9999'
  SET copay = 15
    WHERE client = 'C882';
```

Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

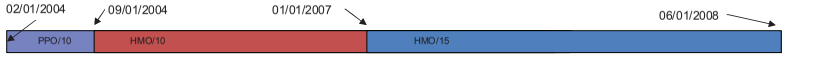
client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01...	9999-12-31...
C882	HMO	10	09/01/2004	01/01/2007	2006-03-01...	9999-12-31...
C882	HMO	15	01/01/2007	12/31/9999	2006-03-01...	9999-12-31...

Table: policy

IDUG EMEA 2010

Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007



client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01...	9999-12-31...
C882	HMO	10	09/01/2004	01/01/2007	2006-03-01...	9999-12-31...
C882	HMO	15	01/01/2007	06/01/2008	2008-06-01...	9999-12-31...

Table: policy

IDUG EMEA 2010

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
4	06/01/2008	Cancel policy as of today (present event)

```
UPDATE policy
  SET eff_end = '06/01/2008'
    WHERE client = 'C882'
      AND eff_end = '12/31/9999';
```


IDUG EMEA 2010

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
5	09/01/2008	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007 (past event)

```
UPDATE policy FOR PORTION OF BUSINESS_TIME
  FROM '05/01/2006' TO '10/01/2007'
  SET type = 'POS'
    WHERE client = 'C882';
```

Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01...	9999-12-31...
C882	HMO	10	09/01/2004	05/01/2006	2008-09-01...	9999-12-31...
C882	POS	10	05/01/2006	01/01/2007	2008-09-01...	9999-12-31...
C882	POS	15	01/01/2007	10/01/2007	2008-09-01...	9999-12-31...
C882	HMO	15	10/01/2007	06/01/2008	2008-09-01...	9999-12-31...

Table: policy

Bitemporal Support – Example (cont)

```
SELECT * FROM POLICY
  FOR BUSINESS_TIME AS OF '2007-07-01'
    WHERE CLIENT='C882';
```

Answer: Customer has "POS", so should be reimbursed.

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01...	9999-12-31...
C882	HMO	10	09/01/2004	05/01/2006	2008-09-01...	9999-12-31...
C882	POS	10	05/01/2006	01/01/2007	2008-09-01...	9999-12-31...
C882	POS	15	01/01/2007	10/01/2007	2008-09-01...	9999-12-31...
C882	HMO	15	10/01/2007	06/01/2008	2008-09-01...	9999-12-31...

Table: policy

Bitemporal Support – Example (cont)

Question: On 09/15/2008, client calls and complains. Client saw an out-of-network specialist on 07/01/2007.
Claims dept. denied client's claim due to HMO coverage for this visit on 07/15/2007. Client demands reimbursement.

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01...	9999-12-31...
C882	HMO	10	09/01/2004	05/01/2006	2008-09-01...	9999-12-31...
C882	POS	10	05/01/2006	01/01/2007	2008-09-01...	9999-12-31...
C882	POS	15	01/01/2007	10/01/2007	2008-09-01...	9999-12-31...
C882	HMO	15	10/01/2007	06/01/2008	2008-09-01...	9999-12-31...

Table: policy

Bitemporal Support – Example (cont)

Question: Did our claims department make an error denying the client's claim on 07/15/2007?

To answer this: Need historical data to see what claims department saw on 07/15/2007. Thus, the need for a bitemporal solution.

Bitemporal Support – Example (cont)

```
SELECT * FROM policy
  FOR BUSINESS_TIME AS OF '2007-07-01'
  FOR SYSTEM_TIME AS OF '2007-07-15'
   WHERE client='C882';
```

Answer: At 07/15/2007, claims saw 'HMO'.

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	12/31/9999	2004-01-01...	2004-09-01...
C882	HMO	10	09/01/2004	12/31/9999	2004-09-01...	2006-03-01...
C882	HMO	15	01/01/2007	12/31/9999	2006-03-01...	2008-06-01...
C882	HMO	10	09/01/2004	01/01/2007	2008-06-01...	2008-09-01...
C882	HMO	15	01/01/2007	06/01/2008	2008-06-01...	2008-09-01...

Table: policy_hist

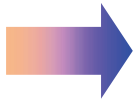
Concerns about Database Security

- Separation of duties
  - Database administrators such as DBADM can access sensitive data
  - No designated authority such as SECADM to manage security policies
- Granularity of privilege model
  - Privileges are granted at database object level
  - Difficult to protect personal and sensitive information within the object
  - Cannot easily comply with data protection laws such as HIPPA, GLBA
- Overloading applications with security logic
  - Can be bypassed by malicious users
  - Hampers the ability to use ad-hoc query tools, report generation tools
- Alternative views for each group of users
  - Can be bypassed by malicious users
  - View's updatability may not correctly reflect security policies
- Evolution of security policies
  - Difficult to manage and maintain

Row and Column Access Control

Solution : Row and Column Access Control

- Tighter security
  - Data-centric within database
  - No backdoor to bypass views or applications
  - More granularity via row permissions and column masks
  - Separation of duties
  - Designated SECADM authority
  - No authority including DBADM is exempted from the control
  - Relief for the evolution of security policies



- Easy to implement
  - More flexibility via SQL
  - Separation of security logic and application logic

Row and Column Access Control – new terminology

- Row Permission
  - a database object that expresses a row access control rule for a table
  - contains a rule in the form of an SQL search condition that describes to which rows the users have access
  - applied by DB2 after the checking of table privileges (e.g. SELECT, INSERT privilege, etc.)

Row and Column Access Control – Concept

Think a Decomposed View

```
CREATE VIEW EMPOLYEE_VIEW AS
   SELECT (CASE ... END) SSN, (CASE ... END) SALARY
      FROM EMPLOYEE
      WHERE STATE = 'CA' AND
            LASTNAME = 'SMITH' AND
            BDATE > '1970-01-01'
```

- Row Permission
  - The WHERE clause of the EMPLOYEE_VIEW
- Column Mask
  - The outermost SELECT clause in the EMPLOYEE_VIEW definition

Row and Column Access Control – new terminology (cont'd)

- Column Mask
  - a database object that expresses a column access control rule for a specific column in a table
  - contains a rule in the form of an SQL CASE expression that describes to what masked value returned for a column value the users have access
  - applied by DB2 after the checking of table privileges (e.g. SELECT, UPDATE privilege, etc.)

Row and Column Access Control – Examples

- **Row Permission**

```
CREATE PERMISSION EMPLOYEE_PERMISSION ON EMPLOYEE
   FOR ROWS WHERE STATE = 'CA' AND
                  LASTNAME = 'SMITH' AND
                  BDATE > '1970-01-01'
ENFORCED FOR ALL ACCESS ENABLE ;
```

- **Column Mask**

```
CREATE MASK SSN_MASK ON EMPLOYEE
    FOR COLUMN SSN RETURN
      CASE WHEN SESSION_USER = 'SMITH'
             THEN SSN
             ELSE CHAR('XXX-XX-') || SUBSTR(SSN,8,4)
      END
  ENABLE;
```

- **SELECT SSN FROM EMPLOYEE;**

Who can see what??

- New Built-in Functions
  - VERIFY_GROUP_FOR_USER
    - Verify primary and secondary authorization IDs

```
WHERE
   VERIFY_GROUP_FOR_USER (SESSION_USER, 'MGR', 'PAYROLL') = 1
```

  - VERIFY_TRUSTED_CONTEXT_ROLE_FOR_USER
    - Verify primary authorization ID's role

```
WHERE
   VERIFY_TRUSTED_CONTEXT_ROLE_FOR_USER (SESSION_USER,
                                         'MGR', 'PAYROLL') = 1
```


Deactivate Row and Column Access Control

- Deactivated by SECADM authority only
  - Job card … ,USER=SECADM, …
- Invalidate packages and cached statements
- Row permissions and column masks become ineffective in DML
  - Remove default row permission 1 = 0 if deactivated for row
  - Open all access to the table

```
ALTER TABLE table-name
   DEACTIVATE ROW       ACCESS CONTROL
   DEACTIVATE COLUMN  ACCESS CONTROL;

ALTER TABLE table-name
   DEACTIVATE ROW       ACCESS CONTROL;
```


Activate Row and Column Access Control

- Activated by SECADM authority only
  - Job card … ,USER=SECADM, …
- Invalidate packages and cached statements
- Row permissions and column masks become effective in DML
  - All row permissions are merged to filter out rows
    - Multiple row permissions are connected with 'OR'
  - All column masks are applied to mask output columns
- Generate default row permission 1 = 0 if activated for row

```
ALTER TABLE table-name
   ACTIVATE ROW       ACCESS CONTROL
   ACTIVATE COLUMN  ACCESS CONTROL;

ALTER TABLE table-name
   ACTIVATE ROW       ACCESS CONTROL;
```


SQL PL Extensions for Scalar UDF and Table UDF

SQL PL Extensions for Scalar UDF and Table UDF

SQL PL: SQL procedural language

- Set of control statements defined in the SQL Standard
  - **ISO/IEC FCD 9075-4:2003**, *Information technology - Database languages - SQL - Part 4: Persistent Stored Modules (SQL/PSM)*
- Native SQL procedures (V9)

➢ **Simplifies** the task of writing database applications

Table UDF

```
CREATE  FUNCTION  JTABLE (COLD_VALUE CHAR(9), T2_FLAG CHAR(1))
RETURNS TABLE (COLA INT, COLB INT, COLC INT)
LANGUAGE SQL
SPECIFIC DEPTINFO
NOT DETERMINISTIC
READS SQL DATA
RETURN
SELECT A.COLA, B.COLB, B.COLC
  FROM TABLE1 AS A
    LEFT OUTER JOIN
        TABLE2 AS B
      ON A.COL1 = B.COL1 AND  T2_FLAG = 'Y'
WHERE A.COLD = COLD_VALUE;
```

- function body specifies an SQL query that returns a result table
- result table is returned to the invoking statement

SQL PL Extensions for Scalar UDF and Table UDF

- DB2 9 for z/OS
  - No support for SQL table functions; only external table functions are supported
  - Scalar function support limited to single RETURN statement
- Extended in V10 to allow for use within:
  - SQL scalar functions
  - SQL table functions (minimal subset)

SQL Scalar Function

```
CREATE   FUNCTION   REVERSE  (INSTR
    VARCHAR(4000))
   RETURNS VARCHAR (4000)
   DETERMINISTIC
   NO EXTERNAL ACTION
   CONTAINS SQL
BEGIN
 DECLARE REVSTR, RESTSTR VARCHAR(4000)
    DEFAULT '';
 DECLARE LEN INT;
 IF INSTR IS NULL THEN
   RETURN NULL;
 END IF;
 SET RESTSTR = INSTR;
 SET LEN = LENGTH(INSTR);
 WHILE LEN > 0 DO
   SET REVSTR = SUBSTR(RESTSTR, 1, 1)
       CONCAT REVSTR;
   SET RESTSTR = SUBSTR(RESTSTR, 2, LEN - 1);
   SET LEN = LEN - 1;
 END WHILE;
 RETURN   REVSTR;
END
```

- Function body contains logic.
- If the input data is null the function simply returns null.
- Otherwise, the function reverses the order of the characters in the input string and returns the modified string to the invoking statement.

Extended Implicit CAST Support

From Numeric to String

Numbers ⟷ Strings: character string (no CLOB, no FOR BIT DATA subtype) or graphic string (no DBCLOB, UNICODE encoding scheme)

Source Data Type	Target Data Type
SMALLINT	VARCHAR(6)
INTEGER	VARCHAR(11)
BIGINT	VARCHAR(20)
NUMERIC/DECIMAL	VARCHAR(precision+2)
FLOAT (REAL, DOUBLE)	VARCHAR(24)
DECFLOAT	VARCHAR(42)

Overview

- Extended support for implicit casts: implicit cast between string and numeric data types
- Indexable/sargable predicates
- NO: installation change/setup for implicit cast
- NO: catalog change
- NO: new SQL statements & functions
- YES: new semantics of assignments & comparisons & unions & expressions & function resolution

From String to Numeric

Source Data Type	Target Data Type
CHAR	DECFLOAT(34)
VARCHAR	DECFLOAT(34)
GRAPHIC	DECFLOAT(34)
VARGRAPHIC	DECFLOAT(34)
CHAR/VARCHAR **FOR BIT DATA**	N/A
BINARY/VARBINARY	N/A
CLOB/BLOB/DBCLOB	N/A

New OLAP functions (Moving Average Sum, Moving Average, etc.)

Moving Sums and Moving Averages

- compute a single value for the current row based on some or all of the rows in a defined group.
- support cumulative sums and moving averages by using a window.
- can be used in a select-list, or in the ORDER BY clause of a select-statement.

Overview

- DB2 9 for z/OS has already supported 2 classes of OLAP specifications
  - Ranking Specifications – RANK(), DENSE_RANK()
  - Numbering Specifications – ROW_NUMBER().
- DB2 10 for z/OS will introduce the last class of OLAP specifications
  - Aggregation Specifications – SUM(), AVG() and other column function etc.

Moving Averages Example

Find the seven day centered average of XYZ stock for each day the stock traded.

```
SELECT date,symbol, close_price,
  decimal(avg(close_price) over
   (order by date rows between 3 preceding
    and 3 following),6,3)
  as smooth_cp
  FROM stock;
```

Greater Timestamp Precision,

Timestamp WITH TIME ZONE

New Terminology

- **fractional second** - A portion of a second that is greater than 0 but less than 1.
- **timestamp precision** - The maximum number of digits that can be included in a fractional second.

Background

Problem:

- The existing TIMESTAMP data type does not capture an associated time zone, the timestamp is ambiguous
- Some customers store time zone in another column

Solution:

- New data type

  TIMESTAMP WITH TIME ZONE

Timestamp

- A timestamp is a **six or** seven-part value (year, month, day, hour, minute, second, and **optional fractional second**) **with an optional time zone specification** that represents a date and time. The time could include specification of a fraction of a second.
- The number of digits in the fractional second is specified using an attribute in the range from 0 to 12 with a default of 6.

- TIMESTAMP WITH TIME ZONE format:
  - <timestamp> TZH:TZM
  - TZH (time zone hour) – '*xhh*' -24 to 24
  - TZM (time zone minute) – 'mm' 00 to 59 (has to be 00 if TZH is 24)
- Example:
  - New York is 4 hours behind UTC, so New York time "11:42" on 2009-06-09 can be represented as '2009-06-09-11.42.00.000000-04:00'.
  - Same UTC representation:
    - '2009-06-09-11.42.00.000000-04:00'
    - '2009-06-09-08.42.00.000000-07:00'
    - '2009-06-09-15.42.00.000000-00:00'

More Examples:

```
CREATE TABLE LAB15_Table
 (ID    INTEGER     NOT NULL WITH DEFAULT,
TSTZ12  TIMESTAMP(12) WITH TIME ZONE WITH DEFAULT
        '9999-12-31-23.59.59.123456789012 +14:00');

INSERT INTO LAB15_Table(ID, TSTZ12)
 VALUES (4,
  TIMESTAMP '2012-12-31 02:02:02.123456789012+08:00' AT TIMEZONE '-
  08:00'    );

INSERT INTO LAB15_Table (ID, TSTZ12)
 VALUES (5,
TIMESTAMP '2012-12-31 02:02:02.123456789012+08:00'
        AT TIMEZONE SESSION TIME ZONE     )
```


Disclaimer

SQLADRIA SEMINAR

Opatija, 24 - 25 November 2011

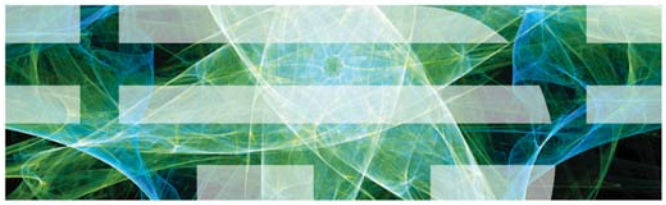
Learning pureXML and New SQL Features inDB2 10 for z/OS Using SPUFI and CLP

Hands on Lab

Sponsored by

IBM

IDUG EMEA 2010



HOL
Learning pureXML and new SQL Features in DB2 10 for z/OS Using SPUFI and CLP

Jane Man (janeman@us.ibm.com)
Guogen Zhang (gzhang@us.ibm.com)

Contents

1. Introduction

1.1 Who are we?

We are from the DB2 development organization in IBM Silicon Valley Lab.

Jane Man
Advisory Software Engineer
janeman@us.ibm.com

Guogen Zhang
Distinguished Engineer
gzhang@us.ibm.com

1.2 Objectives:

- Learn XPath
- Learn SQL/XML operators
  - XMLQUERY
  - XMLEXISTS
  - XMLTABLE
  - XMLCAST
- Create and Utilize an XML Index
- Learn XML Schema
  - Register XML Schema
  - Perform validation of an XML documents
- Learn sub-document update on XML documents
- Learn XML type modifier
- Learn XML data and time support
- Learn Bitemporal support
- Learn row permission and column masks

- Learn SQL PL extensions for scalar UDF and table UDFs
- Learn new OLAP functions(moving average)
- Learn about greater precision timestamp
- Learn about timestamp with time zone
- Learn about extended implicit case support

1.3 Suggested reading

- Introduction to pureXML in DB2 9 for z/OS

  ftp://ftp.software.ibm.com/software/data/db2zos/presentations/2007/misc/purexml.pdf

- Leveraging DB2 9 for z/OS pureXML Technology

  http://www.geocities.com/zhanggene/pub/Leveraging_DB29_for_zOS_whitepaper_v2.pdf

- DB2 Version 9.1 for z/OS XML Guide (SC18-9858)

  http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.xml/dsnxgk13.pdf?noframes=true

1.4 Acknowledgments

Many thanks for the following people to make this HOL possible:

Fung Lee

Irene Liu

Jerry Mukai

Ken Taylor

Rick Chang

Ruiping Li

Xiaohong Fu

Ying Zeng

Yumi Tsuji

2. Lab0 : Getting Started

In this hands-on-lab(HOL), we will use the DB2 Command Line Processor(CLP) that shipped with DB2 for z/OS and SPUFI to learn about the pureXML and new SQL features in DB2 10 for z/OS.

2.1 Dependency of the Labs

Below is the dependency of the labs.

Lab 0 must be done first to ensure the Command Line Processor (CLP) environment is set up properly and there is database connection to the DB2 server.

Lab 1 to Lab 8 are XML related topics while the rest are new SQL features. Below is a listing of labs.

Lab 1 : setup for Lab 2, Lab 3, Lab 4, and Lab 5

Lab 2 : XMLQuery

Lab 3 : XMLExist and XML index

Lab 4 : XMLTable

Lab 5: XML Schema

Lab 6: XML Type Modifier

Lab 7: Sub-document update on XML document

Lab 8 : XML date and time support

Lab 9: Bitemporal Support

Lab 10: Row Permissions and Column Masks

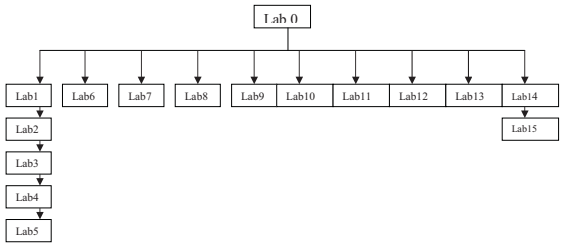
Lab 11: SQL PL extensions for scalar UDF and table UDFs

Lab 12: Extended Implicit CAST support

Lab 13: New OLAP functions(moving average)

Lab 14: Greater Precision Timestamp

Lab 15: Timestamp with Time Zone



2.2 LAB0E1 – Command Line Processor (CLP) setup

In this exercise, we will use the DB2 command line processor that ships with DB2 for z/OS. We will configure CLP and use it to execute a script to test the connectivity to the database.

1. Open a Command Prompt window by double clicking the Command prompt icon in the desktop.
2. Change directory to the location of the hands-on-lab materials.

   In the command prompt window, type: `CD C:\IDUGHOL`

3. Start notepad to edit the `clpsetup.bat` file.
   type: `notepad clpsetup.bat`

4. Replace the "XXXXXXXX" and "YYYYYYYY" with the username and password provided to you by the lab instructor. Please use UPPERCASE for all userid and password.

   Tip: You can quickly do find/replace in notepad by choosing "replace" from the Edit menu, or typing Ctrl-H at any time.

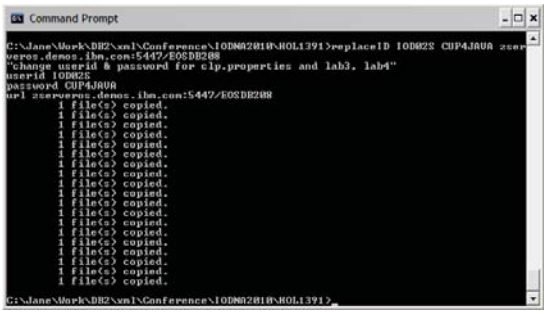
   Note that the alias "mydb2" has been created for you to direct CLP

to connect to the server.

Save the file (from the File menu) and quit notepad (from the File menu).

5. Execute the clpsetup.bat batch file to define aliases and set environment variables to allow CLP to run. In the command prompt window, type: `clpsetup.bat`

   For a successful execution, you should see something similar to this:



6. Browse the "lab0e1.db2" script. You do not have to change anything in this script (unless you want to).
   type: `notepad lab0e1.db2`

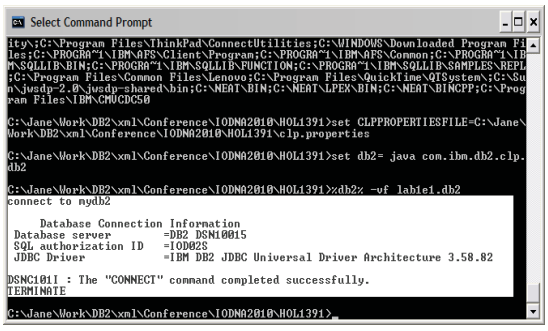
   Exit notepad when you are finished.

7. Execute the "lab0e1.db2" script. This script will connect to the database.
   type: `%db2% -vf lab0e1.db2`

   %db2% invokes the "db2" alias that we created in `clpsetup.bat`. The options "v" tells CLP to use verbose mode so it prints the

commands and results to the screen, and "f" tells CLP to execute a file instead of running in interactive mode.
"lab0e1.db2" is the script that we will run.

Verify that the "CONNECT" command completed successfully. If the screen looks as it does below(may see different `Database Server` and `SQL authorization ID`), then the script ran successfully and this exercise is complete.



CLP will be used again in the following labs. Leave the command prompt window open so we can easily invoke CLP again.

You can now start with any lab you prefer (pls see section 2.1 for dependency).

Please note we use userid IOD02S for illustration. You should use your own userid assigned by the instructor.

End of lab 0.

3. Lab1 : Set up for Lab1 to Lab5

3.1 LAB1E1 – CLP setup: creating the CUSTOMER table

In this exercise, we will use the DB2 command line processor that ships with DB2 for z/OS. We will configure CLP and use it to execute a script containing SQL to create a database with a table, and populate the table with some sample data.

1. Browse the "lab1e1.db2" script. We will use notepad to browse the file. You do not have to change anything in this script (unless you want to). In the command line window we have opened,
   type: `notepad lable1.db2`

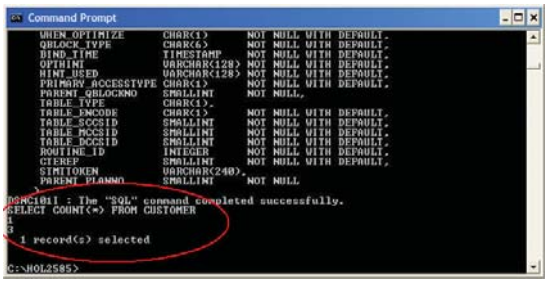
   Scroll up and down the file and see the different SQL statements that will be executed. Exit notepad when you are finished.

2. Execute the "lab1e1.db2" script. This script will create a table youruserid.CUSTOMER. The script will populate the table with three rows.

   type: `%db2% -vf lable1.db2`

   At the end of the script, there is a SELECT statement which counts the number of rows in the CUSTOMER table.

   Verify that the result of the SELECT COUNT returned three rows. If the screen looks as it does below, then the script ran successfully and this exercise is complete.



3.2 LAB1E2 – SPUFI setup: testing the CUSTOMER table

In this exercise, we will connect to the z/OS server using TSO. We will configure SPUFI through DB2I so it can be used for many of the lab exercises.

1. Open a Personal Communications (PCOM) window:

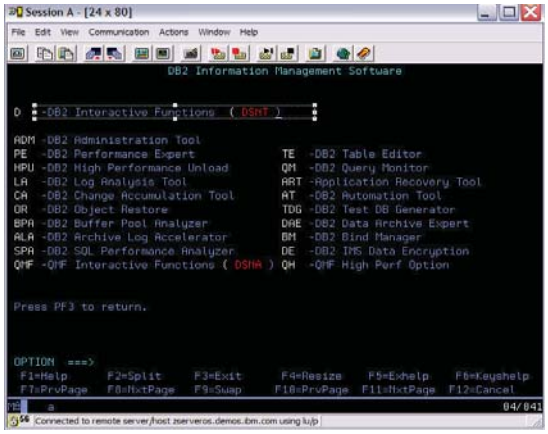
   On the desktop, click on the ZSERVEROS icon to start PCOM.

2. Start TSO:
   at the SELECTION ==> prompt, type: `TSO`

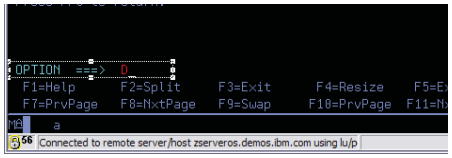
3. Enter your userid and password provided by the lab instructor.

4. Now we will set the defaults for SPUFI.
   At the ISPF option prompt, type: =P.I to bring up the DB2I options panel.

5. Enter the DB2 subsystem name `DSNT`.
   Where the database subsystem name is specified, type `DSNT`.



6. At the DB2I Option prompt, type `D`
   and press enter.



3.3 LAB1E3 – Using SPUFI

In this exercise, we will use SPUFI as we configured it in exercise LAB1E2. Because we will be using SPUFI in so many exercises, these steps will be very useful as a reference.

1. Start SPUFI.

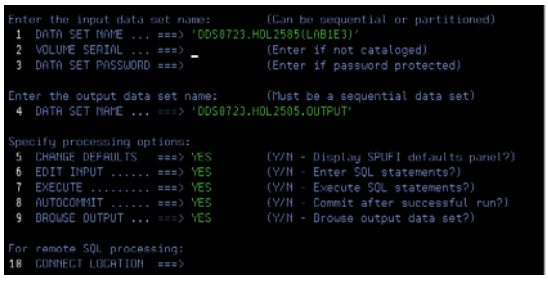
   At the DB2I primary option menu Command prompt, type `1`
   and press enter to start SPUFI.

2. Enter the input and output dataset names.

   At the Data Set Name (#1) prompt, type
   '`youruserid.HOL2585(LAB1E3)`'
   At the Data Set Name (#2) prompt, type
   '`youruserid.HOL2585.OUTPUT`'

   Ensure the value from #5 to #9 are `YES`

   Below is an example for userid DDS0723:



3. In the next panel, change #15 (MAX CHAR FIELD from 80 to 240)

4. Press Enter until the Edit panel shows up.

   Browse the contents of the dataset to examine which queries will be run by SPUFI.

5. Press enter twice again. SPUFI will execute the SQL in the dataset and display the contents of the output dataset that you named in step 2. At the bottom of the output dataset is the same SELECT COUNT(*) statement that was in LAB1E1 step 7. If you see that there are 3 rows in the table, then this exercise is complete.

   Leave the PCOM window open so SPUFI is available for use later. The steps described in LAB1E3 will be used several times throughout the day.

End of lab 1.

4. Lab 2 – Learning XPath with XMLQUERY

In this set of lab exercises, we will use the SQL/XML operator XMLQUERY in an SQL select statement to embed XPath expressions that we want to operate against our XML data.

4.1 LAB2E1-LAB2E7

Each of these exercises is provided as a SPUFI PDS member. They are named '`youruserid.HOL2585(LAB2EX)`' where youruserid is your user id and X is the lab number. Please refer to the instructions in LAB1E3 to use SPUFI to evaluate the SQL in these dataset members.

The queries are listed here for your reference.

Please note XMLQUERY will return XML data. Unfortunately, SPUFI cannot handle XML datatype very well, we need to call

XMLSERIALIZE to convert the XML datatype to CLOB. Therefore, you will see XMLSERIALIZE in the queries in the actual dataset members.

Please see Appendix for the pictorial view of XML data.

LAB2E1:

```
-- get Fred Flintstone's account balance
-- use a simple XPath expression with child axis
SELECT
 XMLQUERY('declare default element namespace
          "http://tpox-benchmark.com/custacc";
          /Customer/Accounts/Account/Balance/OnlineActualBal'
          PASSING custxml)
FROM CUSTOMER
WHERE id=1;
```

LAB2E2:

```
-- lab2e2
-- get Fred Flintstone's account balance
-- use the same XPath expression as lab2e1 with XMLCAST
SELECT XMLCAST
       (XMLQUERY('declare default element namespace
                 "http://tpox-benchmark.com/custacc";
                 /Customer/Accounts/Account/Balance/OnlineActualBal'
                 PASSING custxml)
        AS DECIMAL(21,6))
FROM CUSTOMER
WHERE id=1;
```

LAB2E3:

```
-- get Fred Flintstone's account balance
-- use the same XPath expression with descendant axis
SELECT XMLQUERY('declare default element namespace
                "http://tpox-benchmark.com/custacc";
                //OnlineActualBal'
                PASSING custxml)
FROM CUSTOMER
WHERE id=1;
```

LAB2E4:

```
-- get the name of the customer with id=1
SELECT XMLQUERY('declare default element namespace
                "http://tpox-benchmark.com/custacc";
                /Customer/Name'
                PASSING custxml)
FROM CUSTOMER
WHERE id=1;

-- tricky: get all "Name" elements of the customer with id=1
```



```
SELECT XMLQUERY('declare default element namespace
                "http://tpox-benchmark.com/custacc";
                //Name'
                PASSING custxml)
FROM CUSTOMER
WHERE id=1;
```

LAB2E5:

```
-- find the names of positions of which Fred
-- holds more than 10000 shares
SELECT XMLCAST
       (XMLQUERY('declare default element namespace
                  "http://tpox-benchmark.com/custacc";
                  //Holdings/Position[Quantity>10000]/Name'
                 PASSING custxml)
        AS VARCHAR(50))
FROM CUSTOMER
WHERE id=1;

-- host variable/parameter marker example
--SELECT XMLCAST
--       (XMLQUERY('declare default element namespace
--                  "http://tpox-benchmark.com/custacc";
--                  //Holdings/Position[Quantity>$qty]/Name'
--                 PASSING custxml, :QUANTITY AS "qty")
--        AS VARCHAR(50))
--FROM CUSTOMER
--WHERE id=1;
```

LAB2E6:

```
-- what is Fred's primary phone number?
-- predicate and attribute axis
SELECT XMLQUERY('declare default element namespace
                "http://tpox-benchmark.com/custacc";

/Customer/Addresses/Address/Phones/Phone[@primary="Yes"]'
                PASSING custxml)
FROM CUSTOMER
WHERE id=1;
```

LAB2E7:

```
-- Find out how many shares of anything that Fred owns
-- using functions and operators
SELECT XMLQUERY('declare default element namespace
                 "http://tpox-benchmark.com/custacc";
                 fn:sum(//Quantity)'
                PASSING custxml)
FROM CUSTOMER
WHERE id=1;
```


4.2 LAB2E8-LAB2E13

The remaining lab exercises starting with member LAB2E8 through LAB2E13 are to be completed on your own.

The questions and the hints are described in the individual PDS members. The objective is to replace instances of "%%%%%%%%" with XPath or other expressions to complete the queries.

Use SPUFI to execute these exercises.

You may want to refer to the solutions to exercises LAB2E1 – LAB2E7 for examples of how to approach these. Note that the exercises get progressively more difficult, and it is not uncommon to run out of time before being able to complete every one of them.

The exercises are listed here for your reference. The solutions are in the last chapter.

LAB2E8:

```
-- write a query to find Fred's email addresses
-- fill in the XPath expression
-- hint: use an absolute path from the root to the "Email" element
SELECT XMLQUERY('declare default element namespace
                "http://tpox-benchmark.com/custacc";
                %%%%%%%%'
                PASSING custxml)
FROM CUSTOMER
WHERE id=1;
```

LAB2E9:

```
-- what state does Fred live in?
-- write an XPath expression to retrieve the "State" element
-- use the // axis
-- extra credit: use XMLCAST to convert the result to VARCHAR(20)
SELECT XMLCAST
       (XMLQUERY('declare default element namespace
                 "http://tpox-benchmark.com/custacc";
                 %%%%%%%%'
                 PASSING custxml)
        AS VARCHAR(50))
FROM CUSTOMER
WHERE id=1;
```


LAB2E10:

```
-- Fred owns shares of a company with ticker symbol RCL
-- what is the name of that company?
-- write an XPath expression
-- hint: find the "Position" element
--       write a predicate on the "Symbol"
--       and then get the "Name" element
SELECT XMLQUERY('declare default element namespace
                 "http://tpox-benchmark.com/custacc";
                 %%%%%%%%'
                PASSING custxml)
FROM CUSTOMER
WHERE id=1;
```

LAB2E11:

```
-- how long has Fred been our customer?
-- choose the right query
-- hint: We need to retrieve the "CustomerSince" element
--       and use XMLCAST to convert it to a date.
--       Then we will use date arithmetic in SQL to
--       calculate how long Fred has been a customer.
SELECT %%%%%%%%
       (XMLQUERY('declare default element namespace
                 "http://tpox-benchmark.com/custacc";
                  %%%%%%%%'
                  PASSING custxml)
       %%%%%%%%
FROM CUSTOMER
WHERE id=1;
```

LAB2E12:

```
-- how many email addresses does Fred have?
-- hint: use fn:count and write a path expression
--       to retrieve Fred's email addresses
SELECT XMLQUERY('declare default element namespace
                "http://tpox-benchmark.com/custacc";
                 %%%%%%%%'
                PASSING custxml)
FROM CUSTOMER
WHERE id=1;
```

LAB2E13:

```
-- what is the difference between Fred's online
-- cleared balance and online actual balance?
-- hint: retrieve each value, and subtract one from the other
SELECT XMLQUERY('declare default element namespace
                "http://tpox-benchmark.com/custacc";
                %%%%%%%%
                PASSING custxml)
FROM CUSTOMER
WHERE id=1;
```

End of Lab 2.

5. Lab 3 – Filtering Documents with XMLEXISTS and Creating an XML Index

In this set of lab exercises, we will use the SQL/XML predicate operator XMLEXISTS to filter documents. XMLEXISTS evaluates an XPath expression and returns TRUE for those documents for which the XPath expression is satisfied.

5.1 LAB3E1-LAB3E3

Each of these exercises is provided as a SPUFI PDS member. They are named '`youruserid.HOL2585(LAB3EX)`' where youruserid is your user id and X is the lab number. Please refer to the instructions in LAB1E3 to use SPUFI to evaluate the SQL in these dataset members.

The queries in the dataset members are listed here for your reference:

LAB3E1:

```
-- use XMLEXISTS with an XPath expression
-- for document filtering
-- find those customers who owns more than
-- 10000 shares of any holding
SELECT id
FROM CUSTOMER
```



```
WHERE XMLEXISTS(
'declare default element namespace
"http://tpox-benchmark.com/custacc";
/Customer/Accounts/Account/Holdings/
 Position[Quantity>10000]'
PASSING custxml);

--the same query using a hostvariable
--SELECT id
--FROM CUSTOMER
--WHERE XMLEXISTS(
--'declare default element namespace
--"http://tpox-benchmark.com/custacc";
--/Customer/Accounts/Account/Holdings/
--  Position[Quantity>$qty]'
-- PASSING custxml, :quantity AS "qty");
```

LAB3E2:

```
-- create an index on Quantity
CREATE INDEX QUANTITYIDX ON CUSTOMER(custxml)
GENERATE KEY USING XMLPATTERN
'declare default element namespace
 "http://tpox-benchmark.com/custacc";
/Customer/Accounts/Account/Holdings/Position/Quantity'
AS SQL DECFLOAT(34);

-- run explain
DELETE FROM PLAN_TABLE;
EXPLAIN ALL SET QUERYNO = 1 FOR
SELECT id
FROM CUSTOMER
WHERE XMLEXISTS(
'declare default element namespace
 "http://tpox-benchmark.com/custacc";
 /Customer/Accounts/Account/Holdings/
  Position[Quantity>10000]'
PASSING custxml);


-- select from plan_table
SELECT QUERYNO,QBLOCKNO,PLANNO,METHOD,
       SUBSTR(TNAME,1,8) AS TNAME,
       TABNO,ACCESSTYPE,
       SUBSTR(ACCESSNAME,1,15) AS ACCESSNAME
FROM PLAN_TABLE
WHERE QUERYNO = 1
ORDER BY 1,2,3;
```


LAB3E3:

```
-- find a customer who owns more than 10000 shares of SGP
-- create indexes to make the query indexable

-- create another index on Symbol
CREATE INDEX SYMBOLIDX ON CUSTOMER(custxml)
GENERATE KEY USING XMLPATTERN
'declare default element namespace
"http://tpox-benchmark.com/custacc";
/Customer/Accounts/Account/Holdings/Position/Symbol'
AS SQL VARCHAR(7);

-- run explain
EXPLAIN ALL SET QUERYNO = 2 FOR
SELECT id
FROM CUSTOMER
WHERE XMLEXISTS(
'declare default element namespace
 "http://tpox-benchmark.com/custacc";
 /Customer/Accounts/Account/Holdings/Position
  [Quantity>10000 and Symbol="SGP"]'
PASSING custxml);

-- select from plan_table
SELECT QUERYNO,QBLOCKNO,PLANNO,METHOD,
       SUBSTR(TNAME,1,8) AS TNAME,
       TABNO,ACCESSTYPE,
       SUBSTR(ACCESSNAME,1,15) AS ACCESSNAME
FROM PLAN_TABLE
WHERE QUERYNO = 1
ORDER BY 1,2,3;

-- run the actual query
SELECT id
FROM CUSTOMER
WHERE XMLEXISTS(
'declare default element namespace
 "http://tpox-benchmark.com/custacc";
 /Customer/Accounts/Account/Holdings/Position
  [Quantity>10000 and Symbol="SGP"]'
PASSING custxml);
```

5.2 LAB3E4-LAB3E6

The remaining lab exercises starting with member LAB3E4 through LAB3E6 are to be completed on your own.

The questions and the hints are described in the individual PDS members. The objective is to replace instances of "%%%%%%%%" with XPath or other expressions to complete the queries.

Use SPUFI to execute these exercises.

The exercises are listed here for your reference. The solutions are at the last chapter.

LAB3E4:

```
-- find customers who have not been contacted since 2002-01-01
-- hint: write a query that checks if LastContactDate
--       is <= "2002-01-01"
-- what index might you create to make this query indexable?
SELECT id
FROM CUSTOMER
WHERE XMLEXISTS('declare default element namespace
                "http://tpox-benchmark.com/custacc";
                %%%%%%%%'
                PASSING custxml);


CREATE INDEX CONTACTIDX ON CUSTOMER(custxml)
GENERATE KEY USING XMLPATTERN
'declare default element namespace
"http://tpox-benchmark.com/custacc";
%%%%%%%%'
AS SQL VARCHAR(10);
```

LAB3E5:

```
-- find the names of customers that live in zipcode 30123
-- hint: use XMLEXISTS for document filtering,
--       and XMLQUERY to retrieve the customer's Name element
SELECT XMLQUERY('declare default element namespace
                "http://tpox-benchmark.com/custacc";
                %%%%%%%%'
                PASSING custxml)
FROM CUSTOMER
WHERE XMLEXISTS('declare default element namespace
```

```
                "http://tpox-benchmark.com/custacc";
                %%%%%%%% '
                PASSING custxml);
```

LAB3E6:

```
-- find the id of the customer that only has one phone number
-- hint: use XMLEXISTS and write a query that
--       counts the number of phone numbers and compares it to 1

SELECT id
FROM CUSTOMER
WHERE XMLEXISTS('declare default element namespace
                "http://tpox-benchmark.com/custacc";
                %%%%%%%%'
                PASSING custxml);
```

End of Lab 3.

6. Lab 4 – Using XMLTABLE

In this set of exercises we will use the SQL/XML operator XMLTABLE to query XML data and return results as if they were relational data. We will explore simple "shredding" of XML data, and use XMLTABLE for iteration through repeating elements or through multiple documents.

6.1 LAB4E1-LAB4E4

Each of these exercises is provided as a SPUFI PDS member. They are named '`youruserid.HOL2585(LAB3EX)`' where youruserid is your user id and X is the lab number. Please refer to the instructions in LAB1E3 to use SPUFI to evaluate the SQL in these dataset members.

The queries in the dataset members are listed here for your reference:

LAB4E1:

```
-- use XMLTABLE to get information on all customers
SELECT X.*
FROM CUSTOMER C,
```



```
     XMLTABLE(XMLNAMESPACES
                (DEFAULT 'http://tpox-benchmark.com/custacc'),
              '/Customer'
              PASSING C.custxml
              COLUMNS ID      INTEGER       PATH '@id',
                      FNAME   VARCHAR(20)   PATH 'Name/FirstName',
                      LNAME   VARCHAR(20)   PATH 'Name/LastName',
                      BALANCE DECIMAL(15,2) PATH
'Accounts/Account/Balance/WorkingBalance') X;
```

LAB4E2:

```
-- use XMLQUERY to list the languages that Fred speaks
SELECT XMLQUERY('declare default element namespace
                 "http://tpox-benchmark.com/custacc";
                 /Customer/Languages/Language'
                 PASSING custxml)
FROM CUSTOMER C
WHERE ID=1;

-- use XMLTABLE to list the languages that Fred speaks
SELECT X.LANG
FROM CUSTOMER C,
     XMLTABLE(XMLNAMESPACES
                (DEFAULT 'http://tpox-benchmark.com/custacc'),
              '/Customer/Languages/Language'
              PASSING C.custxml
              COLUMNS LANG VARCHAR(15) PATH '.') X
WHERE ID=1;
```

LAB4E3:

```
-- use XMLTABLE to iterate through rows and items within a row
SELECT X.*
FROM CUSTOMER C,
     XMLTABLE(XMLNAMESPACES
                (DEFAULT 'http://tpox-benchmark.com/custacc'),
              '/Customer/Accounts/Account/Holdings/Position'
              PASSING C.custxml
              COLUMNS POSITION VARCHAR(15)   PATH 'Name',
                      SYMBOL   VARCHAR(4)    PATH 'Symbol',
                      QTY      DECIMAL(15,5) PATH 'Quantity',
                      ID       INTEGER       PATH '../../../../@id') X;
```

LAB4E4:

```
-- use XMLTABLE to iterate through rows and items within a row
-- use SQL functions on top to analyze the data
SELECT SUM(QTY)
FROM CUSTOMER C,
     XMLTABLE(XMLNAMESPACES
                (DEFAULT 'http://tpox-benchmark.com/custacc'),
              '/Customer/Accounts/Account/Holdings/Position'
              PASSING C.custxml
              COLUMNS QTY      DECIMAL(15,5) PATH 'Quantity',
                      ID       INTEGER       PATH '../../../../@id') X
          GROUP BY X.ID;
```


6.2 LAB4E5-LAB4E8

The remaining lab exercises starting with member LAB4E5 through LAB4E8 are to be completed on your own.

The questions and the hints are described in the individual PDS members. The objective is to replace instances of "%%%%%%%%" with XPath or other expressions to complete the queries.

Use SPUFI to execute these exercises.

The exercises are listed here for your reference. The solutions are at the last chapter.

LAB4E5:

```
-- create a table of customer address information
-- hint: use XMLTABLE to create a table with 5 columns
--       write a row-expression to lead to the Address element
--       and produce the following columns:
--         Street VARCHAR(50)
--         City   VARCHAR(20)
--         PostalCode DECIMAL(5,0)
--         State      VARCHAR(2)
--         Country    VARCHAR(20)
SELECT ID, X.*
FROM CUSTOMER C,
     XMLTABLE(XMLNAMESPACES
                (DEFAULT 'http://tpox-benchmark.com/custacc'),
              '%%%%%%%%'
              PASSING C.custxml
              COLUMNS STREET  VARCHAR(50)   PATH '%%%%%%%%',
                      CITY    VARCHAR(20)   PATH '%%%%%%%%',
                      ZIP     DECIMAL(5,0)  PATH '%%%%%%%%',
                      STATE   VARCHAR(20)   PATH '%%%%%%%%',
                      COUNTRY VARCHAR(20)   PATH '%%%%%%%%') X;
```

LAB4E6:

```
-- write a query to identify all languages spoken by any customer
-- hint: use XMLTABLE to retrieve all languages and
--       return them as a varchar
SELECT DISTINCT X.LANGUAGE
FROM CUSTOMER C,
     XMLTABLE(XMLNAMESPACES
                (DEFAULT 'http://tpox-benchmark.com/custacc'),
              %%%%%%%%) X(LANGUAGE);
```


LAB4E7:

```
-- create a table of ShortNames and email addresses
-- remember that each customer has one ShortName and multiple email
addresses,
-- so each customer should have one row per email addresses
-- (with the ShortName repeated)
SELECT X.SHORTNAME, X.EMAIL
FROM CUSTOMER C,
     XMLTABLE(XMLNAMESPACES
                (DEFAULT 'http://tpox-benchmark.com/custacc'),
              '%%%%%%%%'
              PASSING C.custxml
              COLUMNS SHORTNAME VARCHAR(20) PATH '%%%%%%%%',
                      EMAIL     VARCHAR(50) PATH '%%%%%%%%') X;
```

LAB4E8:

```
-- this one is hard
-- find the holdings of a customer who has more than
-- 5000 shares of any stock and has a working balance
-- of more than 170000
-- return the customer's id, working balance,
-- and holding's symbol and quantity
SELECT C.id, X.*
FROM CUSTOMER C,
     XMLTABLE(XMLNAMESPACES
                (DEFAULT 'http://tpox-benchmark.com/custacc'),
              '%%%%%%%%'
              PASSING C.custxml
              COLUMNS BALANCE DECIMAL(15,2) PATH '%%%%%%%%',
                      SYMBOL  VARCHAR(4)    PATH '%%%%%%%%',
                      QTY     DECIMAL(15,5) PATH '%%%%%%%%') X;
```

End of Lab 4.

7. Lab 5 – Using XML Schema

In this set of lab exercises, we will explore XML Schema by registering several XML Schemas and use them to validate documents.

7.1 LAB5E1 – A Simple XML Schema

In this exercise, we will use CLP to execute a script to register an XML Schema. The XML Schema documents and the CLP script are provided, the script will create a unique name based on your userid to provide to DB2.

1. Switch the active window back to your command window still should still be open.

2. Run the schema registration job using CLP.

type: `%db2% -vf lab5e1.db2`

If the registration was successful, you should see two DSNC101I messages saying that the REGISTER and COMPLETE commands completed successfully.

Your XML schema has successfully been registered. A schema with the name youruserid_CUST has been successfully created on the server and is available for use for XML Schema validation.

7.2 LAB5E2 – Validate XML Documents

In this exercise, we will use SPUFI to execute an INSERT SQL statement to insert data and invoke the validation function DSN_XMLVALIDATE.

1. As per the instructions specified in LAB1E3, specify the following datasets:

   For the input dataset, specify `youruserid.HOL2585(LAB5E2)`
   For the output dataset, specify `youruserid.HOL2585.OUTPUT`

2. Edit the LAB5E2 member. Scroll to the bottom of the dataset member and locate the XXXXXXXX. Replace XXXXXXXX with your userid.

   Tip: you can use the CHANGE command in the ISPF editor to change all instances of XXXXXXXX with your userid.
   At the command prompt, type: CHANGE `XXXXXXXX youruserid ALL`

   Save the file and exit the editor.

3. Let SPUFI execute the SQL. When browsing the output dataset, scroll to the bottom of the file to see the error message returned from the validating XML parser. You may see an error message like this(note: the message text may be different):

```
SQLCODE = -20399, ERROR:  XML PARSING OR
VALIDATION ERROR  ENCOUNTERED DURING XML SCHEMA
VALIDATION AT LOCATION 42 5013 cvc-complex-
type.4: Attribute "id" must appear on element …
```

   Note that due to a 70-byte limit for the SQLCA in SPUFI, the error message may be truncated. The SQL statement GET_DIAGNOSTICS can be used to retrieve the entire error message.

4. Edit the LAB5E2 member. Add an attribute `id="1001"` to element Customer. Save the file and exit the editor

5. Let SPUFI execute the SQL. When browsing the output dataset, scroll to the bottom of the file to see the error message returned from the validating XML parser. You may see an error message like this(note: the message text may be different):

```
SQLCODE = -20399, ERROR:  XML PARSING OR
VALIDATION ERROR  ENCOUNTERED DURING XML SCHEMA
VALIDATION AT LOCATION 238 6012 cvc-enumeration-
valid = cvc-enumeration-valid: Value "male"

SQLSTATE   = 2201R
```

6. Edit the LAB5E2 member. Change "`male`" to "`Male`". Save the file and exit the editor.

   Let SPUFI execute the SQL. When browsing the output dataset, scroll to the bottom of the file to see that the INSERT statement was successfully executed.

7.3 LAB5E3 – Register the simple_cust2 schema

In this exercise, we will use CLP to execute a script to register an XML Schema with multiple schema documents. The XML Schema documents and the CLP script are provided, the script will create a unique name based on your userid to provide to DB2.

1. Switch the active window back to your command window still should still be open.

2. Run the schema registration job using CLP.

   type: `%db2% -vf lab5e3.db2`

   If the registration was successful, you should see two DSNC101I messages saying that the REGISTER and COMPLETE commands completed successfully.

3. Browse the simple_cust2.xsd file.

   type: `notepad simple_cust2.xsd`

   There are two `<xs:include>` elements, each of which include another XML schema document: **simple-name.xsd and simple-addr.xsd.** Instead of xs:string, the type of the Name element is defined as custacc:NameType, the type of the Address element is defined as custacc:AddressType. So they have their own nested structures.

   Exit notepad and take a look at the included files.

4. type: `notepad simple_name.xsd`

   Take a look at how the **custacc:NameType** is defined. Exit notepad.

5. type: `notepad simple_addr.xsd`

   This file defines custacc:AddressType. Exit notepad.

7.4 LAB5E4 – Validate an XML document using SPUFI

In this exercise, we will use SPUFI to execute an INSERT SQL statement to insert data and invoke the validation function

DSN_XMLVALIDATE.

1. As per the instructions specified in LAB1E3, specify the following datasets:

   For the input dataset, specify `youruserid.HOL2585(LAB5E4)`
   For the output dataset, specify `youruserid.HOL2585.OUTPUT`

2. Edit the LAB5E4 member. Scroll to the bottom of the dataset member and locate the XXXXXXXX. Replace XXXXXXXX with your userid.

   Save the file and exit the editor.

   Let SPUFI execute the SQL. When browsing the output dataset, scroll to the bottom of the file to see that the INSERT statement was successfully executed.

7.5 LAB5E5 – Register the CUSTACC schema

In this exercise, we will use CLP to execute a script to register a complex XML Schema. The XML Schema documents and the CLP script are provided, the script will create a unique name based on your userid to provide to DB2.

1. Switch the active window back to your command window that should still be open.
   Run the schema registration job using CLP.

   type: `%db2% -vf lab5e5.db2`

   If the registration was successful, you should see two DSNC101I messages saying that the REGISTER and COMPLETE commands completed successfully.

   Your XML schema has successfully been registered. A schema with the name youruserid_CUSTOMER has been successfully created on the server and is available for use for XML Schema validation.

7.6 LAB5E6 – Validate an XML document using CLP

We will use CLP for this exercise.

1. Switch the active window back to your command window that should still be open. Run the schema validation job using CLP.

   type: `%db2% -vf lab5e6.db2`

Below is part of the SQL statement(for userid IOD02S):

```
INSERT INTO CUSTOMER (id, custxml) VALUES

(6, XMLPARSE(DOCUMENT DSN_XMLVALIDATE(

'<?xml version="1.0" encoding="US-ASCII"?>

<Customer id="6" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://tpox-benchmark.com/custacc"
xsi:schemaLocation="http://tpox-benchmark.com/custacc
custacc.xsd">

…

</Customer>',

'SYSXSR.IOD02S_CUSTOMER')));
```

You should see the INSERT run successfully.

7.7 LAB5E7 – Free Play

At this point, you are free to play with the validation capability provided by DSN_XMLVALIDATE. The following files are provided for you to try:

- A SPUFI dataset member `youruserid.HOL2585(LAB5E7)` is an exact copy of the LAB5E6 member.

A CLP file `cleanup.db2` contains statements to DROP the XML schema and database objects. This may be done when you are complete to clean up the objects in the database.

End of Lab 5.