Asynchronous post commit triggers – Replication to SPL routine – Streaming analytics

HCL PRODUCTS & PLATFORMS

Agenda

- Use cases
- How it works
- Prerequisites
- Demo

Use cases

- Realtime Streaming analytics on OLTP data
  - MIN,MAX,AVG,SUM for a group of records.
    - Example: Per store sales reports
  - Realtime leaderboard calculation for an online game
  - Build materialized views
- Data transformations
  - Add additional fields ---like store id– while replicating data to central server.
- Update external systems like graph database, Hadoop, Spark, Queuing services …
- “No Key” data replication support
  - Replicate data for tables that do not have primary key, unique index or ER key.

How it works? (1)

- Solution is based on Enterprise Replication
- As part of transaction replay, instead of applying data to target table, user defined stored procedure gets fired for insert, update and delete operations.
- Works with loopback replication
  - Source and target participant can be defined on the same table-- acts likes a post commit asynchronous trigger
  - Source and target tables can be on the same server instance either in the same database or in different database
- Target table can be on a different Enterprise Replication server instance.
- Target table is used for parsing replicated row and extracting column values– data will not be applied to target table
- User can specify where clause filter to fire SPL on specific dataset!

How it works?(2)

- Data is staged in ER queues for asynchronous processing
- Source server id, transaction id, transaction commit time and operation type(I/U/D) are passed in as argument values to the SPL routine along with user data.
- For update operation, both before and after image of the column values are passed in as arguments to SPL routine.
- SPL routine execution can be configured to be invoked as user informix or table owner.

New options to ‘cdr define replicate’ command

- --splname=<spl routine name>
  - Stored procedure routine name to apply data to. SPL routine must exist at all participants
  - Input arguments: operation type, source id, txnid, before image of the row column list, after image of the row column list
- --jsonsplname=<spl routine name>
  - Stored procedure routine name to apply data to. SPL routine must exist at all participants
  - Input arguments: json document
  - --jsonsplname option expects input arguments to splname routine to be a json datatype. With json document as input to SPL routine, same SPL routine can be used for registering ‘replication to SPL’ replicate definition on multiple tables. For certain use cases --like queueing data to message queues -- this makes developer job a lot easier.
  - --jsonsplname option is mutually exclusive to –splname option.
- --cascaderepl=y|n enable cascade replication
  - Required if replication to SPL needs to be executed for the data applied through Enterprise Replication

--splname option stored procedure argument list

- Optype char(1) – operation type. Values include
  - I – Insert
  - U – Update
  - D – Delete
- Soucre_id integer – Source server id. Same as group id.
- Committime integer – Transaction commit time.
- Txnid bigint – Transaction id.
- Before value column list.
- After value column list.
  - Note: Column list for SPL routine extracted from select statement projection list

--jsonsplname option SPL routine json argument

Attribute name	Description
operation	Operation type: Insert/Delete/Update
table	Table name
owner	Table owner
database	Database name
txnid	8 byte unique id. Higher order 4 bytes: commit work log id, lower order 4 bytes: commit work log position.
commit_time	Transaction commit time for the event data.
rowdata	Row data in JSON document format. Data is returned in column name as key and column data as value.
before_rowdata	Before row data for “update” operation.

Example document format :

{“operation”:"insert",”table”:"creditcardtxns",”owner”:"informix",”database”:"creditdb",”txnid”:2250573177224,”commit_time”:1488243530,rowdata”:{“uid”:22,”car did”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T10:35:10.000Z } }}

{“opertion”:"update",table:"creditcardtxns",”owner”:"informix",”database”:"creditdb”,”txnid”:2250573308360,”commit_time”:1488243832,”rowdata”:{uid:21,cardid: "7777-7777-7777-7777",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":"25-Jan-2017 16:15"} },”before_rowdata”:{“uid”:21,”cardid”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T10:35:10.000Z } }}

{“opertion”:"delete",”table”:"creditcardtxns",”owner”:"informix",”database”:"creditdb",”txnid”:2250573287760,”commit_time”:1488243797,rowdata”:{“uid”:22,”car did”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T13:35:06.000Z } }}

Asynchronous post commit trigger support

- Define loopback replication server
- Create ‘replication to SPL’ type replicate with same “database and table” information for both source and target participants. Loopback server group name shall be specified with target participant definition:
- Example:
  - cdr define repl rep1 -C always -S row -M g_cdr_utm_nag_1 -A -R --splname=logger4repl2spl "test@g_mygroup:informix.t1" "select * from t1" "test@g_loopback:informix.t1" "select * from t1”
    - Note: g_mygroup is the local server ER group, and g_loopback is the pseudo ER server group.

Prerequisites

- Logging must be enabled for the tables
  - Does not work for RAW tables, and non-logged databases
- Cannot mix participant definitions that include table as the target, and SPL routine as the target.
  - Replicate definition marked with ”repl2spl” attribute only support SPL routine as the target to apply data.
- Even though data is applied to stored procedure routine, target table definition must exist.
- Out-of-row data datatypes like text,byte,blob,clob aren’t supported.

Use case 1 – Build staging table for data changes (--splname example)

```
create database test with log;

create table t1 (c1 int , c2 int);

create table t1log (optype int, srcdir int, committime int, txnid bigint, c1bef int, c2bef int, c1 int, c2 int);

create procedure logger4repl2spl (opType char(1), srcid integer, committime integer, txnid bigint, c1bef integer, c2bef int, c1 integer,
c2 int)

   insert into t1log values (opType, srcid, committime, txnid, c1bef, c2bef, c1, c2);

end procedure;

$ cdr define repl rep1 -C always -S row -M g_cdr_utm_nag_1 -A -R --splname=logger4repl2spl "test@g_mygroup:informix.t1" "select
* from t1" "test@g_loopback:informix.t1" "select * from t1”

$ cdr start repl rep1
```

Use case 1 – Build staging table for data changes (--jsonsplname example)

```
create database test with log;

create table t1 (c1 int , c2 int);

create table t2 (col1 int , col2 float);

create table staging (data json);

create procedure logger4repl2spl (data json)

   insert into staging values (data);

end procedure;
```

```
$ cdr define repl rep1 -C always -S row -M g_cdr_utm_nag_1 -A -R --jsonsplname=logger4repl2spl "test@g_mygroup:informix.t1" "select * from t1" "test@g_loopback:informix.t1" "select * from t1”

$ cdr define repl rep2 -C always -S row -M g_cdr_utm_nag_1 -A -R --jsonsplname=logger4repl2spl "test@g_mygroup:informix.t2" "select * from t2" "test@g_loopback:informix.t2" "select * from t2”

$ cdr start repl rep1

$ cdr start repl rep2
```

Use case 2 – Realtime aggregation framework

drop database retaildb;

create database retaildb with log;

create table sales (customerid int, storeid int , bill_amount float);

create table sales_summary(storeid int , s_count int, s_sum float, s_avg float, s_min float, s_max float);

CREATE PROCEDURE store_agg(opType char(1), srcid integer, committime integer, txnid bigint, customerid_bef integer, storeid_bef int, bill_amount_bef float, customerid int, storeid_aft int , bill_amount float)

---- -----

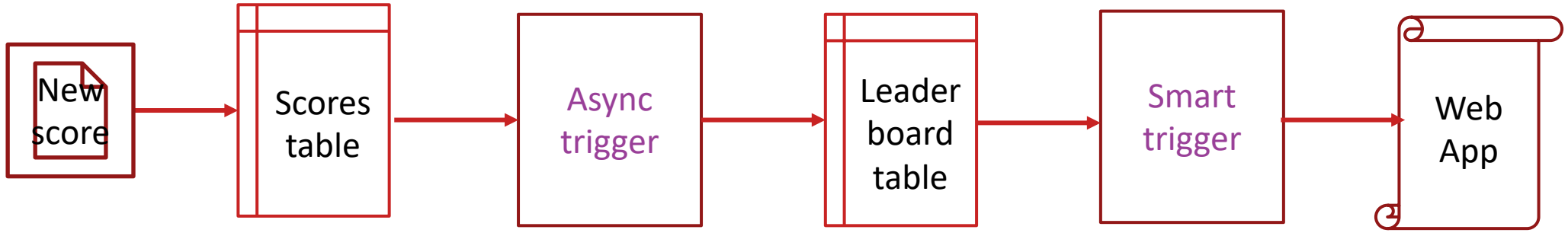
---- ----

END PROCEDURE;

$ cdr define repl rep1 -C always -S row -M utm_group_1 -A -R --serial --splname=store_agg "retaildb@g_mygroup:informix.sales" "select * from sales" "retaildb@g_loopback:informix.sales" "select * from sales"

$ cdr start repl rep1

Use case 3: Leader board calculation



Use case 4 – Publish data to MQTT using Java or C UDR

```
create database mqtt with log;

create table customer (name char(128), id int);

execute procedure sqlj.install_jar('file:$INFORMIXDIR/jars/mqtt_trigger.jar', 'mqtt_trigger_jar',1);

$cdr define repl mqrepl -C always -S row -M g_informix -A -R --serial --jsonsplname=mqtt_put
"mqtt@g_informix:informix.customer" "select * from customer" "mqtt@g_lb:informix.customer" "select * from customer"

$ dbaccess mqtt -

  > insert into customer values(”Bill", 1);

$ mosquitto_sub -t 'test/topic' -v

test/topic
{"operation":"insert","table":"customer","owner":"informix","database":"mqtt","txnid":21475983636,"commit_time":1550879224,"rowda
ta":{"name":”Bill","id":1 }}
```

‘cdr list repl’ sample output

```
$cdr list repl
CURRENTLY DEFINED REPLICATES
--------------------------------
REPLICATE:      rep2
STATE:          Active ON:utm_group_1
CONFLICT:       Always Apply
FREQUENCY:      immediate
QUEUE SIZE:     0
PARTICIPANT:    retaildb:informix.sales
OPTIONS:        row,ris,ats,fullrow,loopback,repl2spl
REPLID:         65539 / 0x10003
REPLMODE:       PRIMARY  ON:utm_group_1
APPLY-AS:       INFORMIX ON:utm_group_1
REPLTYPE:       Master
```

Demo