Three ways to capture data changes when they happen!

By Art Kagel

New Informix Tech Talks by the IIUG

We are launching a new channel on YouTube for Informix Users! This will be a place for Informix how-to videos. More information will be coming soon.

Capturing Data Changes

Art S. Kagel, ASK Database Management

art@askdbmgt.com
art.kagel@gmail.com

There are three ways to capture data changes from your databases that are built into Informix

- Change Data Capture API
- Push Data Triggers (aka Smart Triggers the Java API for Push Data Triggers)
- Asynchronous Post Commit Triggers

Change Data Capture API

Change Data Capture

The Change Data Capture API provides functions to capture transactional data. You can use a variety of clients to run these functions, such as, JDBC, ODBC, ESQL/C, and DB-Access. The data is returned as CDC records by standard IBM Informix smart large object read functions. How the captured data is processed depends on your application. The following types of operations are captured:

INSERT, DELETE, UPDATE, TRUNCATE

The Change Data Capture API starts capturing transactions from the current logical log and processes all transactions sequentially. The first time you start capturing data for a particular table, data capture starts at the current log position. If you later stop capture and the restart it, you can restart at the point in the logical logs where data capture was stopped. You cannot go backwards in time through the logical logs to capture the history of the table or perform random seeking in the logical logs.

At the beginning of data capture for a table, the Change Data Capture API provides the table schema information that you can use in your application to create a target table. However, any changes to the table schema after data capture begins are not captured by the Change Data Capture API.

The Change Data Capture API does not capture changes to table schemas or any other database changes. If you attempt to alter a table while data capture is active, the alter process fails.

The Change Data Capture API can capture data only from databases that have logging enabled.

(From the Informix v14.10 online manual pages.)

Push Data Triggers
aka
Smart Triggers

Pushing Data Triggers

Push data feature lets clients register for changes in a dataset using simple SELECT statements and WHERE clauses. Once the server captures data for push data event conditions which evaluates to true for WHERE clause condition, the server pushes committed data to the client, based on registered events. Scaling is achieved by clients not having to poll for data, and not having to parse, prepare, and execute SQL queries. Database servers with parallel architecture – Enterprise Replication log snooper and grouper -- feed the data to all clients by asynchronously reading logical log file changes. This design lets client applications scale linearly without adding significant overhead to the database server or any OLTP applications making changes to the database. Data that is returned to the client is in a developer-friendly JSON format.

* From Informix v14.10 manual:
https://www.ibm.com/support/knowledgecenter/en/SSGU8G_14.1.0/com.ibm.erep.doc/ids_erp_pushdata.htm

Asynchronous Post Commit Data Triggers

Asynchronous Post Commit Data Triggers

Uses the ability of Informix v14.10 to optionally create an Enterprise Replicant in a database on the same physical server instance as the source table combined with the new Replicate to Stored Procedure feature of ER to create a way to capture data changes.

At target participant, ‘replication to SPL routine’ type replicate definition causes SPL routine to be executed instead of applying data to target table. Target participant for “replication to SPL routine” replicate definition can be configured to be same as source database, different database on the same server, or remote peer Enterprise Replication server. “Replication to SPL routine” replicate definition does not enforce the requirement to have primary key, unique index or ER key on the replicated table.

The target stored procedure routine can save the data locally, or transmit it to other applications such as MQTT or home grown applications.

Configuring
Change Data Capture API

Configuring Change Data Capture

1. Open a CDC session:
   EXECUTE FUNCTION cdc_opensess() INTO :ses_id;
2. Set full row logging on:
   EXECUTE FUNCTION cdc_set_fullrowlogging()
3. Define the table(s) to be captured:
   EXECUTE FUNCTION cdc_startcapture()
4. Begin data capture:
   EXECUTE FUNCTION cdc_activatesess(ses_id, log_pos)
5. End data capture:
   EXECUTE FUNCTION cdc_endcapture(ses_id)
6. Disable full row logging:
   EXECUTE FUNCTION cdc_set_fullrowlogging()
7. End CDC session:
   EXECUTE FUNCTION cdc_closesess(ses_id)

Configuring
Push Data Triggers
aka
Smart Triggers

Configuring the server for Push Data Notification

Users who will execute sessions that will register for Push Data Notification must have privileges in the sysadmin database:

```
$execute function task('grant admin', ‘user1’, 'replication');
```

Push Data Notification depends on Enterprise Replication which needs a storage pool defined to hold spooled notices. If ER is not currently running and you do not have a storage pool defined, you will need to define at least one storage pool with:

```
$execute function task( 'storagepool add', '/informix/storage', '0', '0', '20000', '1' );
```

Opening a Push Data Session

$execute function task('pushdata open') into :retval;

Returns an integer which is the session id.

The session id is needed when you call the SBLOB API function to retrieve data events!

Registering for Push Data Events

This call registers to receive three columns from the creditcardtxns table in the creditdb database when the amount is greater than 100. Event records will be labeled “card txn alert”.

```
$execute function task('pushdata register',
            {table:"creditcardtxns",
             owner:"informix",
             database:"creditdb",
             query:"select uid, cardid, carddata from creditcardtxns where carddata.Amount::int
>= 100",
             label:"card txn alert"})
```

Setting Session Attributes

You can also set session level attributes in the same way.

```
execute function task( 'pushdata register', { timeout:"60", max_pending_ops:"0", maxrecs:"1" } );
```

The Smart BLOB Read

You use the SBLOB API read function from an ESQL/C client application to retrieve pending data from registered events:

```
bytesread = ifx_lo_read( sessionid, databuf, nbytes, &errcd );
```

sessionid = the sessionid returned from the pushdata open call
databuf = a pointer/address of a data buffer area to hold returned data
nbytes = number of bytes that should be read into databuf
errcd = an integer variable to hold an error code if returned by the call

The buffer must be able to hold maxrecs return records. Each record will be 1024 bytes plus the size of a pre-image plus the size of a post-image of the requested columns.

The Smart BLOB Read

Sample INSERT record read from the ifx_lo_read() call

{“operation”:"insert",”table”:"creditcardtxns",”owner”:"informix",”database
”:"creditdb",”label”:"card txn alert",”txnid”:2250573177224,”commit_time
”:1488243530,”op_num”:1,”restart_logid”:31,”restart_logpos”:24,”rowdata”:
{“uid”:22,”cardid
”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams
Club","Amount":200,"Date":2017-05-01T10:35:10.000Z } }}

Opening a Detachable Push Data Session

By default, if a registered client disconnects from the server its Push Data session is destroyed. It is now possible to set up a session as detachable so that a client can reconnect to a running Push Data session and retrieve data with no data loss.

To open a detachable session:

execute function informix.task('pushdata setdetach') into :retvalstr;

Then if a client losses its connection to the server or crashes, another client can connect to the server and rejoin the orphaned session with:

$execute function informix.task('pushdata join', "{session_id:"245"}) into :retvalstr;

* Note that if the server goes offline all Push Data sessions are destroyed and cannot be rejoined.

You can also begin to retrieve data from an earlier log position with:

execute function admin("pushdata reset_capture", '{"logid”:”%d”, "logpos”:”%d”}’);

Closing Push Data Sessions

You can deregister a Push Data event with:

```
execute function task('pushdata deregister',{table:"usertable",owner:"informix",
database:"ycsb”});
```

You can deregister from all events with the same label with:

```
execute function task('pushdata deregister', { label:"card txns"});
```

You can end a detachable session from within the session itself with:

```
execute function informix.task('pushdata delete') into :retvalstr;
```

To end a detachable session that is not attached to any client use:

```
execute function task('pushdata delete', '{session_id:"12"}') ;
```

To end all detachable sessions that do not have an attached client use:

```
execute function task('pushdata delete', '{delete_all:"1"}');
```

Configuring Asynchronous Post Commit Data Triggers

Configuring Asynchronous Post Commit Triggers

- Uses loopback ER to replicate to a stored procedure which can be written in SPL, Java, or “C”.
- The procedure becomes the target of the replication and all replicated data is handed off to the procedure for further processing.
  - The proc can save the data in a table, manipulate it and perform other operations, or simply hand it off to an external application stream using standard IPC mechanisms like shared memory, Unix message queues, MQTT, IBM MQ, network data transfer protocols such as FTP, SFTP, etc.

Configuring Asynchronous Post Commit Triggers

- Define an ER group for the source server
- Define an ER group for the ER loopback service
- Create ‘replication to SPL’ type replicate with same “database and table” information for both source and target participants. Loopback server group name is specified with target participant definition:
  - 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 loopback server group.
- If the procedure is passed with –jsonsplname=<name> data is sent to the procedure as JSON documents.
- The optional parameter –cascaderepl configures the replicant to also replicate data changes that are applied by ER from another server permitting the trigger to process data from across the ER Cluster

Wrapping it all up:

Wrapping it up

Change Data Capture API

- External program registers to receive notice of data availability
- Pulls the data from the logical logs
- Change data is normally available beginning with the current logical log record when the capture is registered, though it is possible to begin retrieval from a specified earlier logical log position
- Once a capture is defined, if the client application disconnects, it can reposition to pull data continuing from the logical log record following the last record forwarded.
- Data is returned in Informix binary format by default and some data types may have to be converted to standard types. Standard ESQL/C functions are available for the conversions.
- It is possible to request data sent using Java standard data types. (Primarily applicable for clients written in Java.)

Wrapping it up

Change Data Capture API

Resources:

Change Data Capture API Programmer’s Guide:

https://www.ibm.com/support/knowledgecenter/SSGU8G_14.1.0/com.ibm.cdc.doc/cdc.htm

Monitor Change Data Capture session information:

onstat -g cdc

Wrapping it up

Push Data Triggers

- Data changes are pushed to the registered external application as they become available.
- Optionally the trigger an be defined as detachable making it possible to continue receiving data changes without missing anything if the application disconnects and reconnects later.
- It is possible to begin the session beginning at a specified log and log position.
- Data and meta-data are forwarded to the client application in JSON format.
- Each JSON document may contain multiple records.
- Trigger definition can contain filters to limit what changes are pushed to the client.
- While notifications are pushed to the client, the “C” API requires the client application to actively read the actual data using large object read functions.

Wrapping it up

Push Data Triggers

Resources:

Push Data Trigger Documentation:

https://www.ibm.com/support/knowledgecenter/en/SSGU8G_14.1.0/com.ibm.erep.doc/ids_erp_pushdata.htm

Java Smart Trigger API Documentation:

https://www.ibm.com/support/knowledgecenter/SSGU8G_12.1.0/com.ibm.jdbc_pg.doc/ids_jdbc_st_01.htm

Wrapping it up

Asynchronous Post Commit Triggers

- Uses ER either to another server or within the same server using ER Loopback support.
- Replicate is defined with a stored procedure as the target of the replication.
- For an SPL target procedure data is passed to the procedure in the argument list.
- For a JSON target procedure data is passed to the procedure as a JSON document.
- Processing is initially done by the stored procedure inside the Informix instance’s CPU VP and memory architecture, though the procedure can pass the data outside to external applications.

Wrapping it up

Asynchronous Post Commit Triggers

Resources:

Replication to SPL documentation:

https://www.ibm.com/support/knowledgecenter/SSGU8G_14.1.0/com.ibm.erep.doc/ids_erp_spl_replication.htm

Nagaraju Inturi's Presentation, Sample Code, and Video:

https://github.com/nagaraju-inturi/informix-async-postcommit-triggers

https://www.youtube.com/watch?v=STAo7wsEKgE

Nagaraju Inturi's Presentation on ER Loopback Replication:

https://github.com/nagaraju-inturi/loopback-replication

https://www.youtube.com/watch?v=8_B6qDdPZ68&t=9s

This presentation is showing using Loopback Replication to migrate a table to new fragmentation scheme bypassing the requirement to take the table offline during an alter operation. The configuration is basically the same as for Post Commit Triggers.

Wrapping it up

This is just an overview with enough details to help you research further and understand what options are available to you for the kind of data change processing that you may be called upon to implement.

That’s all folks!

Questions?

That’s all folks!

Art S. Kagel, ASK Database Management

art@askdbmgt.com
art.kagel@gmail.com

Thank You

New Informix Tech Talks by the IIUG

We are launching a new channel on YouTube for Informix Users! This will be a place for Informix how-to videos. More information will be coming soon.