Smart Triggers/Push Data

NAGARAJU & BRIAN
Value Proposition

- Selectively trigger events based on changes in server data
- Real time ‘push’ notifications help clients avoid polling the server
- Small data flow allows simple small clients to work with many triggered events at once
What are Smart Triggers in JDBC

- Smart Triggers are registered events on the server that you subscribe to from your JDBC client
  - Triggers are based on a SQL statement query that matches changes made to a table
    - SELECT id FROM CUSTOMER WHERE cardBalance > 20000;
- One client can listen to many events from many tables, allowing a wide range of monitoring opportunities
  - Monitor account balances
  - Take action on suspicious behaviors
What does a Smart Trigger Look Like?

- It’s designed to be a simple set of classes/interfaces in Java.
- Designed for both simple standalone monitor applications as well as integration into multi-threaded environments.
- Leverages the Push Notification feature in the server to do the heavy lifting.
- Receives JSON documents when a trigger occurs.
Use case: Banking

- Bank accounts
  - I want to be alerted when an account balance drops below zero dollars
  - I don’t want to write SPL or install stored procedures and triggers
  - I want to be notified in my client application
  - I don’t want to poll the database for this information or re-query each time a balance changes from the client
public class BankMonitor implements IfmxSmartTriggerCallback {
    public static void main(String[] args) throws SQLException {
        IfxSmartTrigger trigger = new IfxSmartTrigger(args[0]);
        trigger.timeout(5).label("bank_alert");
        trigger.addTrigger("account", "informix", "bank",
            "SELECT * FROM account WHERE balance < 0", new BankMonitor());
        trigger.watch(); //blocking call
    }

    @Override
    public void notify(String json) {
        System.out.println("Bank Account Ping!");
        if(json.contains("ifx_isTimeout")) {
            System.out.println("-- No balance issues");
        } else {
            System.out.println("-- Bank Account Alert detected!");
            System.out.println("   " + json);
        }
    }
}
Demo!
Adding Smart Triggers to the JDBC driver allows other languages to have this support.

- Groovy, JavaScript (NodeJS), Python, Scala and more
NodeJS Smart Trigger Example

```javascript
var java = require("java");
java.asyncOptions = {
    syncSuffix: ""
};
java.classpath.push("ifxjdbc.jar");


smartTrigger.timeout(10);

smartTrigger.open();
smartTrigger.addTrigger("pushtest", "informix", "ewdb", "SELECT * FROM pushtest", "smart-trigger");
smartTrigger.tableRegistration();

var foo = smartTrigger.readFromSmartBlobObject();
console.log(foo);
```
Use Case: Blockchain

- With Smart Triggers we can integrate into Blockchain use cases
  - Changes to the data with Smart Triggers can initiate a smart contract!
  - Many blockchain examples/demos/applications use NodeJS
    - Having JDBC work with NodeJS allows us to be part of these examples and leverage our technology alongside blockchain
- Conference registration blockchain demo using Ethereum public blockchain
  - Conference registration smart-contract
    - Callback registered on conference registration blockchain smart-contract inserts data into Informix database.
    - Smart-trigger registered on Informix conference database database executes Hotel Reservation smart-contract in blockchain.
Blockchain Demo!
Push data to Client (Server functionality)

- Push vs Pull architecture
- Event driven programming model
sesid = task("pushdata open")
Task("pushdata register", {json})
Task("pushdata register", {json})
While (1)
{
  bytes=Ifx_lo_read(sesid, buf, size, err)
  Execute action;
}
API Calls

- TASK('pushdata open');
  - Register client session as a push data session
  - Returns session id, need this id to read event data.
- TASK('pushdata register', {event and session attributes});
  - Register event conditions, and session specific attributes
- Smart blob read API (ifx_lo_read() or equivalent call) to read event data
  - Pseudo smart blob interface to read event data.
  - Returns JSON document(s).
  - Can be configured as blocking or non-blocking call
- TASK('pushdata deregister', {event condition details});
  - De-register event conditions.
## JSON attributes for registering new event conditions

<table>
<thead>
<tr>
<th>Input attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>Table name to be registered</td>
</tr>
<tr>
<td>owner</td>
<td>Table owner</td>
</tr>
<tr>
<td>database</td>
<td>Database name</td>
</tr>
<tr>
<td>query</td>
<td>Select statement including projection list and where clause to register for changes in a data set.</td>
</tr>
<tr>
<td>label</td>
<td>User defined string to be returned along with event document – useful to differentiate between events when more than one push-data event registered within the same session</td>
</tr>
<tr>
<td>timeout</td>
<td>How long client gets blocked in smartblob read api for new events to be returned by server before returning timeout document.</td>
</tr>
<tr>
<td>commit_time</td>
<td>Return event data committed after this transaction commit time.</td>
</tr>
<tr>
<td>txnid</td>
<td>8 byte unique id. Higher order 4 bytes: commit work log id, lower order 4 bytes: commit work log position.</td>
</tr>
<tr>
<td>max_pending_ops</td>
<td>Maximum number of event records to be kept in the session pending</td>
</tr>
<tr>
<td>maxrecs</td>
<td>Maximum number of records to be returned by smartblob api read call.</td>
</tr>
</tbody>
</table>

Example Command:
```
execute function informix.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")}
```
# Event Data JSON Attributes:

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation</td>
<td>Operation type: Insert/Delete/Update</td>
</tr>
<tr>
<td>table</td>
<td>Table name</td>
</tr>
<tr>
<td>owner</td>
<td>Table owner</td>
</tr>
<tr>
<td>database</td>
<td>Database name</td>
</tr>
<tr>
<td>label</td>
<td>Optional user specified data for the even condition.</td>
</tr>
<tr>
<td>txnid</td>
<td>8 byte unique id. Higher order 4 bytes: commit work log id, lower order 4 bytes: commit work log position.</td>
</tr>
<tr>
<td>commit_time</td>
<td>Transaction commit time for the event data.</td>
</tr>
<tr>
<td>op_num</td>
<td>Increasing sequence number for the event document within a given transaction. If transaction generate 10 events, then each document returned will have incrementing op_num starting from 1 to 10.</td>
</tr>
<tr>
<td>rowdata</td>
<td>Row data in JSON document format. Data is returned in column name as key and column data as value.</td>
</tr>
<tr>
<td>before_rowdata</td>
<td>Before row data for “update” operation.</td>
</tr>
<tr>
<td>ifx_isTimeout</td>
<td>Document with this attribute is returned with value set to “true” if no events gets triggered within the timeout interval specified by the client.</td>
</tr>
<tr>
<td>ifx_warn_total_skipcount</td>
<td>Warning document with this attribute is returned with cumulative number of discarded events due to max_pending_ops attribute threshold.</td>
</tr>
</tbody>
</table>
Example event data documents

Sample output for **Insert** operation:

```
```

Sample output for **Update** operation:

```
```

Sample output for **Delete** operation:

```
```

Sample output for multi row document when maxrecs input attribute set to greater than 1:

```
```
Command to print all sessions:
Onstat –g pd

push-data session structure at 0x5950e028
push-data session id: 70 0x46
Smartblob file descriptor: 39
Number of event conditions: 1
Number of pending event operations: 51
Number of discarded event operations: 0
Total event operations returned to client: 11361

Frist tx begin work logpos: 304:e9e0f0, commit work pos: 305:b697a8
Last txn begin work logpos: 304:e9e0f0, commit work pos: 305:b697a8

Command to print all event conditions:
Onstat –g pd event

push-data subsystem structure at 0x584cc028
push-data session structure at 0x588f5028
push-data session id: 39 (0x27)
Number of event conditions: 1

Push-data event structure at 0x461ed028
Full Table Name: ycsb:informix.usertable
User data: testing...
Replicate name: pushrepl_250_1487957951_1352060721
Onstat commands

Command to print information about specific session:
Onstat –g pd 70

push-data session structure at 0x5950e028
  push-data session id: 70 0x46
  Smartblob file descriptor: 39
  Number of event conditions: 1
  Number of pending event operations: 51
  Number of discarded event operations: 0
  Total event operations returned to client: 11361
  Frist tx begin work logpos: 304:e9e0f0, commit work pos: 305:b697a8
  Last txn begin work logpos: 304:e9e0f0, commit work pos: 305:b697a8

Command to print event conditions for specific session:
Onstat –g pd 39 event

push-data subsystem structure at 0x584cc028
  push-data session structure at 0x588f5028
    push-data session id: 39 (0x27)
    Number of event conditions: 1
    Push-data event structure at 0x461ed028
      Table Name: ycsb:informix.usertable
      User data: testing...
      Replicate name: pushrepl_250_1487957951_1352060721
### Comparing Smart Trigger and Regular I/U/D Trigger

<table>
<thead>
<tr>
<th>Smart Trigger</th>
<th>Regular Trigger(I/U/D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post Commit</td>
<td>Pre Commit</td>
</tr>
<tr>
<td>Register Trigger on a specific Dataset/Event</td>
<td>Trigger gets fired for all changes</td>
</tr>
<tr>
<td>Asynchronous and Linear Scalability</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Data is in JSON format</td>
<td>SQL format</td>
</tr>
<tr>
<td>Trigger logic gets executed in the client</td>
<td>Trigger logic gets executed in the server</td>
</tr>
<tr>
<td>Natural fit for event driven programming model</td>
<td>-</td>
</tr>
<tr>
<td>No schema changes required to define new smart trigger</td>
<td>Require schema changes and exclusive lock on the table to modify trigger definition</td>
</tr>
</tbody>
</table>
## Comparing Push data and CDC

<table>
<thead>
<tr>
<th>Push data</th>
<th>CDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designed for Smart Triggers</td>
<td>Designed for Data streaming/replication</td>
</tr>
<tr>
<td>Can register where clause</td>
<td>No where clause support</td>
</tr>
<tr>
<td>Data in JSON format</td>
<td>Byte stream</td>
</tr>
<tr>
<td>Push technology</td>
<td>Push technology</td>
</tr>
<tr>
<td>Only committed transactions are sent to</td>
<td>All records returned to the user including</td>
</tr>
<tr>
<td>Smart Trigger analysis</td>
<td>rollbacked operations</td>
</tr>
<tr>
<td>*Once the client disconnect from the server,</td>
<td>CDC can read old log files</td>
</tr>
<tr>
<td>events for the client aren’t captured/staged</td>
<td></td>
</tr>
</tbody>
</table>

*Once the client disconnect from the server, events for the client aren’t captured/staged.*
Questions ?