

Oracle Streams

Drinking from the Fire Hose

John Garmany

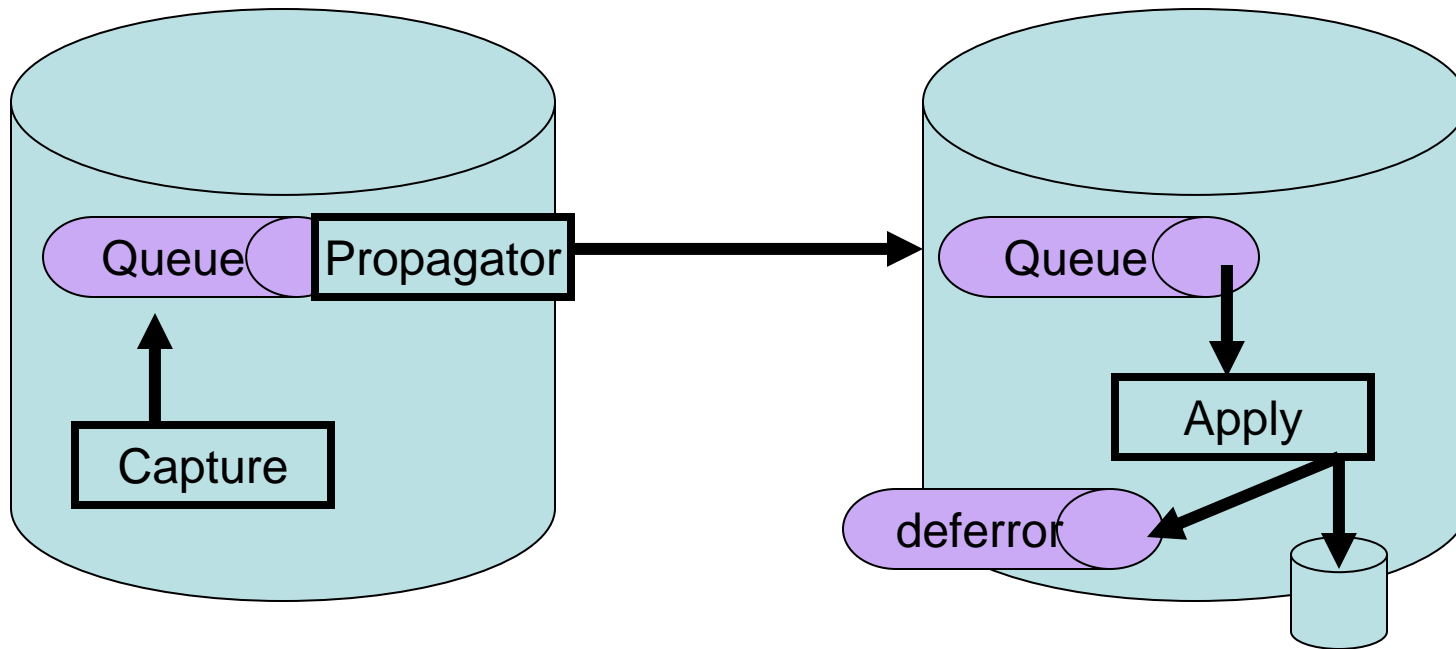
garmanyj@proxitec.com

www.proxitec.com.com

Streams Replication

- Low impact form of Advanced Replication.
- One-way or Bi-directional
- Four Main Components.
 - Capture – Log Miner
 - Propagator – job driven
 - Advance Queues
 - Apply - custom handlers, conflict resolution

Streams Components



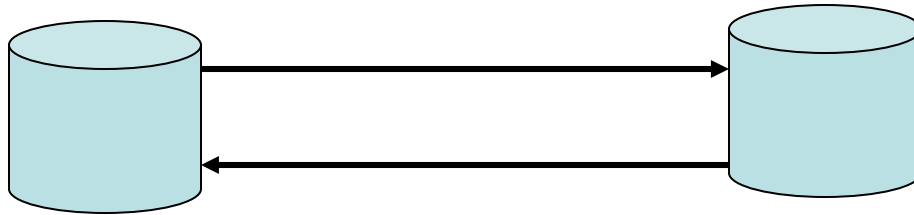
Change Data Capture

- Uses Streams Components
- Replicates change data.
- Table on destination only contains changes.

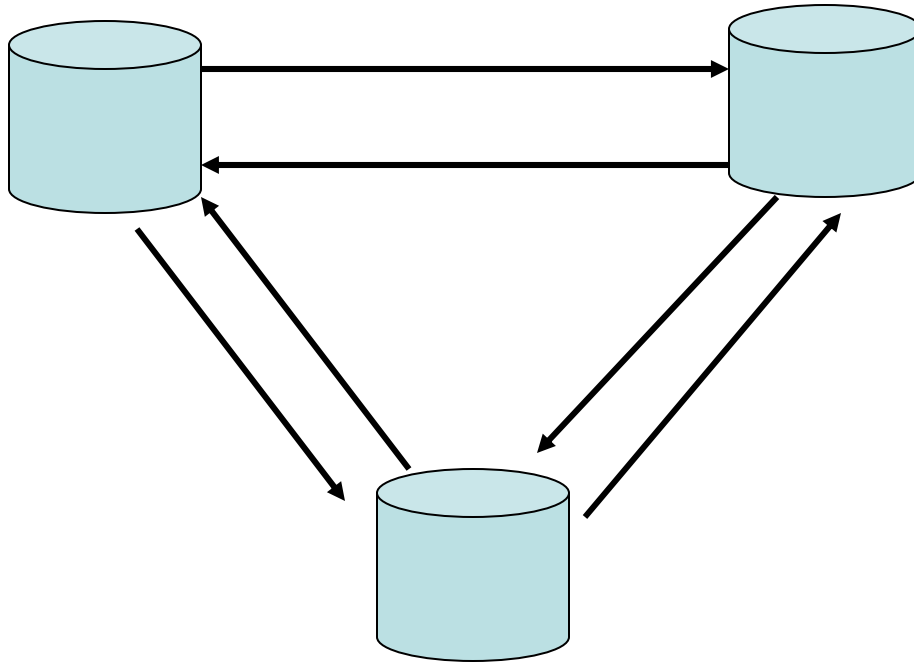
Streams Setup

- Database
 - Everything but SYS.
- Schema
 - Everything in a schema
- Table
 - A table

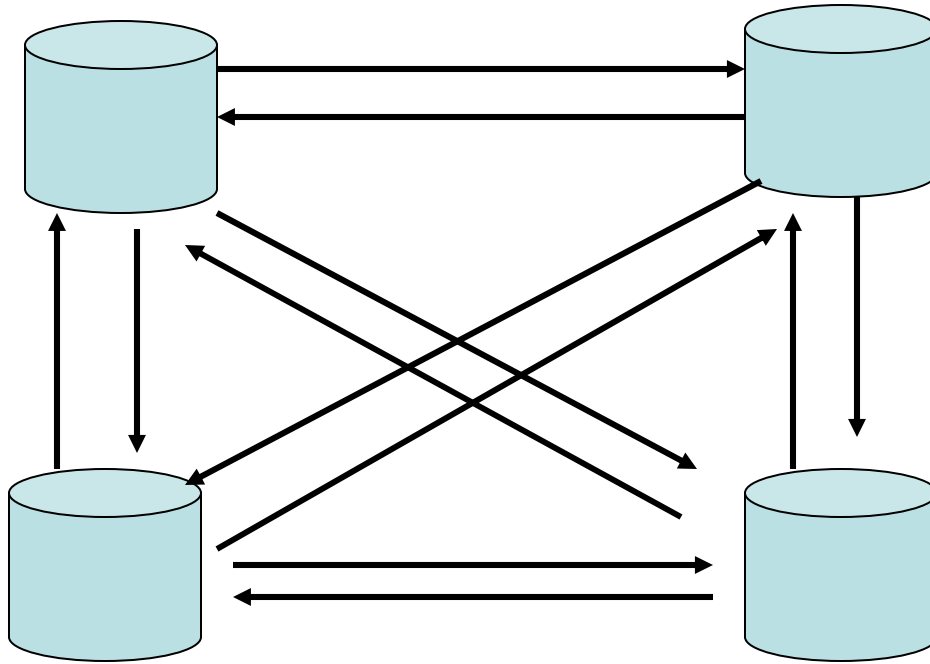
True Multi-Master



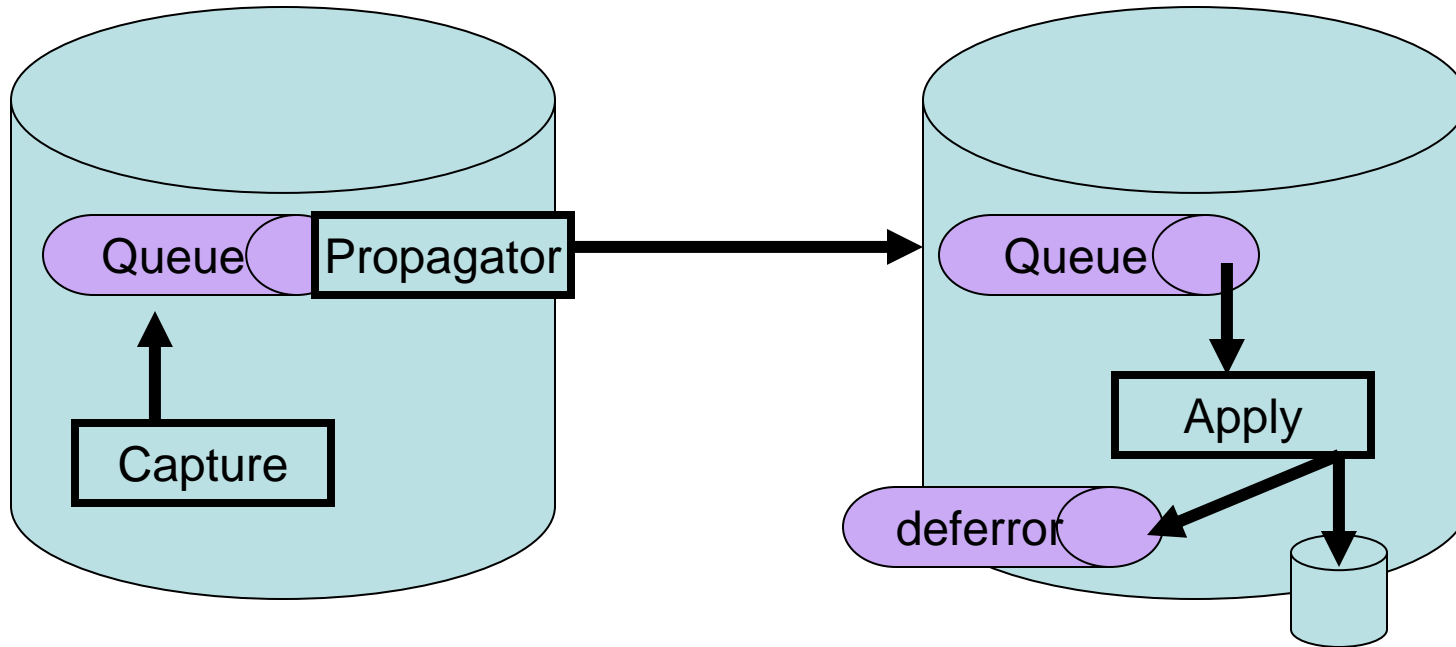
True Multi-Master



True Multi-Master



Streams Components



Checking for Unsupported Types

```
SELECT
  OWNER,
  TABLE_NAME,
  REASON,
  AUTO_FILTERED
FROM DBA_STREAMS_UNSUPPORTED
WHERE OWNER IN ('PROD_USER', 'RPT_USER');
```

- If a row is returned and auto_filtered is 'YES' then the object is valid but not replicated.
- If auto_filtered is no then the streams will error when it attempts to replicate the object changes.
- If auto_filtered is no, then there are probably errors in the error queue or the capture process has aborted.

Adding Rules

If you wish to add new rules after the initial instantiation, you can follow these steps:

- Stop all Streams processes
- Define Apply Rules with `dbms_streams_adm.add_table_rules`
- Define Propagation Rules with `dbms_streams_adm.add_table_propagation_rules`
- Define Capture Rules with `dbms_streams_adm.add_table_rules`
- Start the Streams processes

Streams Tags

- Tags are used to define actions.
 - Null Tags are the default.
 - Changes by the Apply Process have non-null tags (will not replicate further)
 - Certain Tags can be used to identify LRCs that must be forwarded, even if not locally applied.

Streams Tags

- What if you want to make changes that will not replicate?
 - Set the TAG to a non-null value for the current session.

```
begin
  dbms_streams.set_tag(tag=> HEXTORAW('2A'));
End;
/
Select dbms_streams.get_tag from dual;
```

LCR

- Two Types of LCR
 - SYS.LCR\$_ROW_RECORD
 - SYS.LCR\$_ROW_LIST
 - SYS.LCR\$_DDL_RECORD

```
sql> desc sys.lcr$_row_record
```

LCR

- Constructing and Loading an LCR

```
CREATE OR REPLACE PROCEDURE construct_row_lcr(  
    source_dbname VARCHAR2,  
    cmd_type VARCHAR2,  
    obj_owner VARCHAR2,  
    obj_name VARCHAR2,  
    old_vals SYS.LCR$_ROW_LIST,  
    new_vals SYS.LCR$_ROW_LIST) AS  
row_lcr SYS.LCR$_ROW_RECORD;
```

LCR

```
BEGIN -- Construct the LCR based on information
        passed to procedure
row_lcr := SYS.LCR$_ROW_RECORD.CONSTRUCT(
    source_database_name => source_dbname,
    command_type => cmd_type,
    object_owner => obj_owner,
    object_name => obj_name,
    old_values => old_vals,
    new_values => new_vals);
-- Enqueue the created row
LCR DBMS_STREAMS_MESSAGING.ENQUEUE(
    queue_name => 'strm04_queue',
    payload => ANYDATA.ConvertObject(row_lcr));
END construct_row_lcr; /
```

DML Handler

Custom Apply Handler

- A production table that housed important data
- A history table that was inserted into every time a record on the production table changed
- A remote database meant to hold history

The client wished to set up Streams as follows:

- On insert to the history table, attempt to stream the row across to the target DB
- On successful apply to the target DB, remove the row from the source DB history table
- Do not propagate deletes from the source to the target

Custom Apply Handler

- Start with Stopping the DELETES.
- Add a Rule to the Capture Process to not capture DELETES.

Capture Rule

Positive and Negative Rules.

Positive Rule is capture dml.

Negative Rule is to exclude a DELETE command.

```
inclusion_rule          => false,  
and_condition         =>  
    ':lcr.GET_COMMAND_TYPE() = ''DELETE'''
```

Custom Apply Handler Capture Rule

```
begin
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'pubs.t1',
    streams_type    => 'CAPTURE',
    streams_name    => 'capture_pubs',
    queue_name      => 'rep_capt_queue',
    include_dml     => true,
    include_ddl     => true,
    source_database => 'ORCL1',
    inclusion_rule  => false,
    and_condition   =>
      ':lcr.GET_COMMAND_TYPE() = ''DELETE''');
end;
/
```

Custom Apply Handler

Create the Handler

Apply Handlers have the format:

```
PROCEDURE user_procedure (  
    parameter_name IN ANYDATA);
```

The Apply Handler accepts an entry from the ANYDATA queue and does what it wants with it.

Custom Apply Handler

Create the Handler

Use the `Irc.getobject` Function to load the LCR to a variable.

```
create procedure del_orig_lcr (in_any in sys.anydata)
is
  lcr sys.lcr$_row_record;  -- define variable
  rc pls_integer;
...
Begin
  rc := in_any.getobject(lcr);
```

Custom Apply Handler

Create the Handler

Load the new values from the LCR

```
create procedure del_orig_lcr (in_any in sys.anydata)
is
  lcr sys.lcr$_row_record;
  rc pls_integer;
  lcrrow sys.lcr$_row_list;  -- column values
...
Begin
  rc := in_any.getobject(lcr);
  lcrrow := lcr.GET_VALUES('NEW');  -- load the record
```

Custom Apply Handler

Create the Handler

We want to apply this LCR to the database.
Execute the LCR to Apply it to the database.

Begin

```
rc := in_any.getobject(lcr);  
lcrrow := lcr.GET_VALUES('NEW');  
lcr.execute(true); -- apply the LCR
```

Custom Apply Handler

Create the Handler

Now delete this record from the source database.
PK is ID, Use DBLink to execute delete.

```
for i in 1..lcrrow.count
  loop
    if lcrrow(i).column_name = 'ID' THEN
      pkdata := lcrrow(i).data;
      rc := pkdata.getNumber(pkval);
      delete from streams_test.test_hist@primarydb
        where id = pkval;
      commit;
    end if;
  end loop;
```

Custom Apply Handler

The Handler

```
create procedure del_orig_lcr (in_any in sys.anydata) is
  lcr sys.lcr$_row_record;
  lcrrow sys.lcr$_row_list;
  rc pls_integer;
  pkdata sys.anydata;
  pkval number;
begin
  rc := in_any.getobject(lcr);
  lcrrow := lcr.GET_VALUES('NEW');
  lcr.execute(true);
  for i in 1..lcrrow.count
  loop
    if lcrrow(i).column_name = 'ID' THEN
      pkdata := lcrrow(i).data;
      rc := pkdata.getNumber(pkval);
      delete from streams_test.test_hist@primarydb
        where id = pkval;
      commit;
    end if;
  end loop;
26 end;
/
```

Custom Apply Handler

Install the Handler

```
BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name      => 'streams_test.test_hist',
    object_type      => 'TABLE',
    operation_name   => 'INSERT',
    error_handler    => false,
    user_procedure   => 'streams_test.del_orig_lcr',
    apply_database_link => null,
    apply_name       => null);
END;
/
```

Adding Rules

```
DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(  
  schema_name    IN VARCHAR2,  
  streams_type   IN VARCHAR2,  
  streams_name   IN VARCHAR2 DEFAULT NULL,  
  queue_name     IN VARCHAR2 DEFAULT 'streams_queue',  
  include_dml    IN BOOLEAN  DEFAULT TRUE,  
  include_ddl    IN BOOLEAN  DEFAULT FALSE,  
  include_tagged_lcr IN BOOLEAN DEFAULT FALSE,  
  source_database IN VARCHAR2 DEFAULT NULL,  
  dml_rule_name  OUT VARCHAR2,  
  ddl_rule_name  OUT VARCHAR2,  
  inclusion_rule IN BOOLEAN  DEFAULT TRUE,  
  and_condition  IN VARCHAR2 DEFAULT NULL);
```

Adding Rules

The meat and potatoes.

```
inclusion_rule IN BOOLEAN DEFAULT TRUE,  
and_condition IN VARCHAR2 DEFAULT NULL);
```

Inclusion Rule is TRUE or FALSE (Negative Rule or Positive Rule).

Condition: Rule is executed if true.

```
:lcr.get_tag() = HEXTORAW('02')
```

Conflict Resolution

- Happens when the 'old' values do not match the current data.
- Two users on different databases update the same data.
- Data in a replicated table is diverging.
- Only the Apply Process encounters the conflict.

Conflict Avoidance

- Design to avoid conflicts.
 - Separate Primary Keys.
 - Key Logging vs All Columns
 - Use Unique Numbers.

```
select sys_guid() oid from dual;
```

Returns a 16-byte unique number.

- Avoid Delete conflicts by updating instead

Conflict Types

- Update
 - Old Image vs Current Data
- Uniqueness
 - PK or Unique Column Violation
- Deletes
 - Referenced Row already deleted
- FK Violation

Conflict Resolution

- Local Database
 - Database enforces integrity.
 - Transaction Rolled back.
- Remote Database
 - Transaction Goes in the Error Queue if not resolved.
 - Can't Roll Back as transaction is committed on source database

Conflict Detection

- Apply Process Detects Conflicts
 - Uniqueness on INSERT and UPDATE
 - Missing PK on DELETE, UPDATE
 - Data Divergence on UPDATE
 - FK Violation on Apply.
- Apply does not catch: Data matches New Value, not Old Value on Update.

Conflict Resolution

- Prebuilt Conflict Resolution
 - Overwrite
 - Discard
 - Maximum
 - Minimum

Conflict Resolution

- Column Handlers must have a column list.
 - List of Columns that handler resolves.
 - May have more than one resolution handler on a table
 - Only one handler per column list.
 - A column can be in only one list.
- Resolution Column is the column used to resolve the conflict.

Conflict Resolution

- Column List
 - Prebuilt Handlers must have a column list.
 - List of Columns that handler resolves.
 - May have more than one resolution handler on a table
 - Only one handler per column list.
 - A column can be in only one list.
- Resolution Column is the column used to resolve the conflict.

Conflict Resolution Maximum

- Latest update wins!
- Common resolution method.
- Must have or add a timestamp column that is updated on row changes.

Conflict Resolution Maximum

```
ALTER TABLE pubs.t1 ADD (time TIMESTAMP WITH TIME
    ZONE);
CREATE OR REPLACE TRIGGER pubs.insert_time
BEFORE
INSERT OR UPDATE OF c1, c2 ON pubs.t1
FOR EACH ROW
BEGIN
IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP
    THEN
        :NEW.TIME := SYSTIMESTAMP;
    ELSE
        :NEW.TIME := :OLD.TIME + 1 / 86400;
END IF;
END;
/
```

Conflict Resolution Maximum

Set the Conflict Resolution

```
DECLARE
    cols DBMS_UTILITY.NAME_ARRAY;
BEGIN
cols(1) := 'C1';
Cols(2) := 'C2';
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
object_name => 'pubs.t1',
method_name => 'MAXIMUM',
resolution_column => 'time',
column_list => cols);
END;
/
```

Conflict Resolution Maximum

Removing Conflict Resolution

```
DECLARE
    cols DBMS_UTILITY.NAME_ARRAY;
BEGIN
cols(1) := 'C1';
Cols(2) := 'C2';
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
object_name => 'pubs.t1',
method_name => NULL,
resolution_column => 'time',
column_list => cols);
END;
/
```

Conflict Resolution Maximum

Excluding Non-key Columns

```
DECLARE
    cols DBMS_UTILITY.NAME_ARRAY;
BEGIN
    Cols(2) := 'C2';
    DBMS_APPLY_ADM.COMPARE_OLD_VALUES(
        object_name => 'pubs.t1',
        operation => '*',
        column_table => cols,
        Compare => false);
END;
/
```

Conclusion

- Streams Replication add huge capability to the replication suite.
- Rules and Custom Handlers provide unlimited customization.
- Conflicts are going to happen in multi-master replication.
 - Plan for avoidance
 - Implement automatic resolution

Contact Information

garmanyj@proxitec.com

www.proxitec.com

