# Verifying the Quality of Data Replicated by Oracle GoldenGate

Nov 09, 2017

Dong Wang

PayPal Holdings, Inc.

## About PayPal Holdings

- A leading technology platform company that enables digital and mobile payments on behalf of consumers and merchants worldwide.
- As of Q3 2017:
  - 218 million active customer accounts
  - 1.9 billion payment transactions
  - $114 billion in total payment volume (TPV)

## About Speaker

- Ph.D. in Biochemistry
- Principal MTS , Database Infrastructure
- Joined PayPal in 2006
- Worked on OGG since 2008

- Data Replication/Verification Approaches
- Data Quality and RTDV v1 Review
- RTDV v2 Design Principles, Architecture, Features

## Physical Data Replication for Oracle

| Mechanism – Commercial Offering | Pros | Cons |
|---|---|---|
| Storage array/appliance remote mirroring<br>- EMC VNX, Hitachi HDID, NetApp SnapMirror | • DB agnostic | • Zero Oracle validation<br>• Cold data<br>• Higher network IO |
| Database block replication<br>– Oracle standby database, ADG | • Oracle block level validation<br>• Warm data for reads<br>• Efficient network IO | • No table level flexibility<br>• Replicated schema structure must be the same as the source<br>• Replicated data not available for writes |

# Logical Data Replication for Oracle

| Mechanism – Commercial Offering | Pros | Cons |
|---|---|---|
| Convert shipped redo to SQL statement, then apply the SQL<br>– Oracle Logical Standby Database | • Target DB open for read/write<br>• Selective data replication<br>• Different schema structure allowed to enhance reads | • Performance limitation<br>• Vendor preference shift to OGG |
| Trigger based extraction of SQL<br>- Quest SharePlex | • Low cost | • Intrusive with triggers<br>• Performance issues |
| Logminer extraction of records from source redo, then convert to target SQL to apply<br>- Oracle GoldenGate | • Major product focus from vendor<br>• Flexible/heterogeneous targets<br>• Security/encryption | • Data quality<br>• Multiple hops before reaching target<br>• Longer latency |

## Data Quality Issues of OGG in PayPal

| Issues | Causes | Fix |
|---|---|---|
| Data being silently dropped when being applied | • Column used in FILTER() for workload assignment not changed at source, and thus not have value in trail | • Enable full supplemental logging at schema level<br>• New upcoming patch to abend replicats |
| Human Errors<br>GG Bugs<br>Application logic | • Lack of automation<br>• Exotic data types (LOBs, ROWIDs)<br>• Software bug<br>• Truncate and immediate reload | • Automation tools development<br>• Schema model design<br>• Bug fix<br>• Better truncate DDL handling |
| Lack of continuous data validation | • No out-of-box solution on market<br>• One time data validation is not enough | • Real Time Data Validation |

Manual data validation does not cut it!

## Data Quality Verification at PayPal

DB file binary verification for ADG

-- no protection from SAN corruption

OGG Manual Data Validation

-- One time deal

-- Data freeze

-- Table level. No visibility to rows

OGG RTDV v1

-- OGG dependency

-- Extra extract

-- XML dependency

OGG RTDV v2

## One Time Data Verification with Ora_Hash()

- Use of ora_hash() at column level

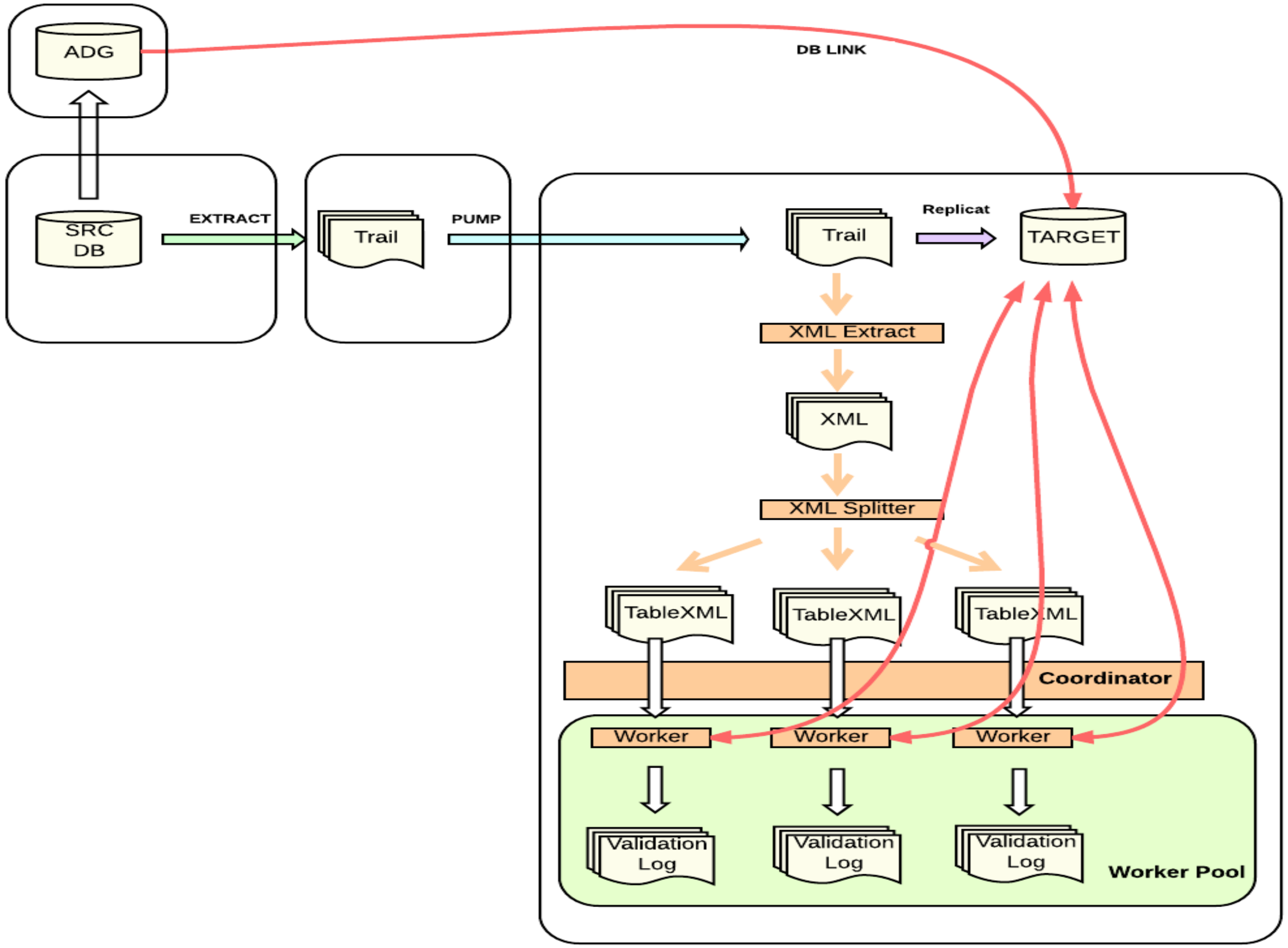
```
select /*+ full(t) parallel(t, 40) */
          count(1),
         sum(ora_hash(col1)),
         sum(ora_hash(col2))
 from table1 t;
```

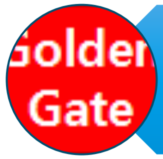
- Validation target:
  - Stop replicats at a consistent time point

```
END <yyyy-mm-dd hh:mi:ss>
```

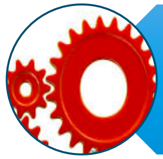
- Validation source:
  - Obtain corresponding source SCN from OGG trail files
  - ADG with recovery stopped at same SCN

## RTDV v1 Review





Dependency to OGG Extraction



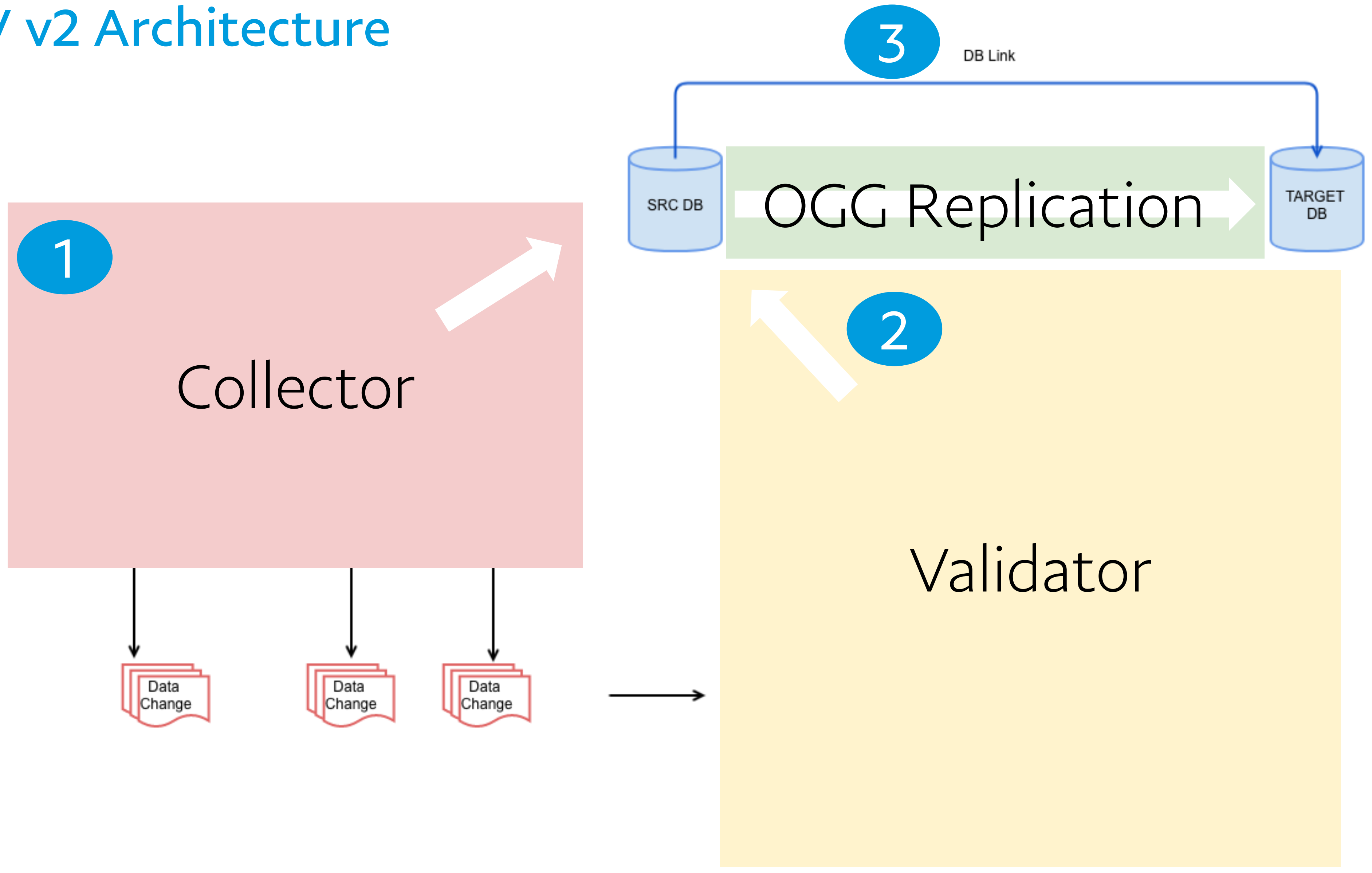
Extra set of XML extracts

Unreliable XML generation

## RTDV v2 Design Principles

| | |
|---|---|
| OGG Independent Change Collection | • Obtain data changes from source DB directly. |
| Data Change Sampling | • Take small sample for validation, not trying to cover all the changes. |
| Self Contained | • No dependency on software components outside of a standard Oracle binary installation |
| Configurable for multiple DB flows | • RAC<br>• One source -> multiple targets<br>• Multiple source DBs on one host |

## RTDV v2 Architecture



## Sampling Changed Data from v$session/v$transaction

Obtain XID:

```
select s.user#, t.xid
  from v$session s,
       v$transaction t
 where s.taddr = t.addr
   and s.user# not in (<exclude_user_ids>);
```

Handling RAC instances by avoiding gv$ views

# Sampling Changed Data from ASH

## Obtain XID:

```
WITH x AS (
    SELECT sample_time,
           user_id,
           xid,
           max(sample_id) over () - sample_id AS diff_sample_id
     FROM  v$active_session_history
    WHERE  sample_time >= trunc(SYSDATE, 'MI') - 60 / 86400
       AND sample_time < trunc(SYSDATE, 'MI')
       AND xid IS NOT NULL
       AND user_id not in (<exclude_user_ids>)
)
SELECT UNIQUE user_id, xid FROM x WHERE diff_sample_id > 0
MINUS
SELECT user_id, xid FROM x WHERE diff_sample_id = 0
   and extract(second from sample_time) >= 59;
```

## Map XID to ROWIDs

```
select logon_user, start_scn, start_timestamp, commit_scn, commit_timestamp,
       table_owner, table_name, row_id, operation
  from (
        select logon_user, start_scn, start_timestamp, commit_scn,
               commit_timestamp, table_owner, table_name, row_id, operation,
               dense_rank() over (partition by table_owner, table_name
                                  order by commit_scn, row_id) r
          from flashback_transaction_query
         where xid = hextoraw(?)
               and commit_timestamp >= to_date(?, 'yyyymmddhh24mi')
               and commit_timestamp < to_date(?, 'yyyymmddhh24mi') + 60/86400
               and row_id is not null
               and <include_exclude_schema>
               and <include_exclude_table>
       )
 where r <= <rows_to_compare>;
```

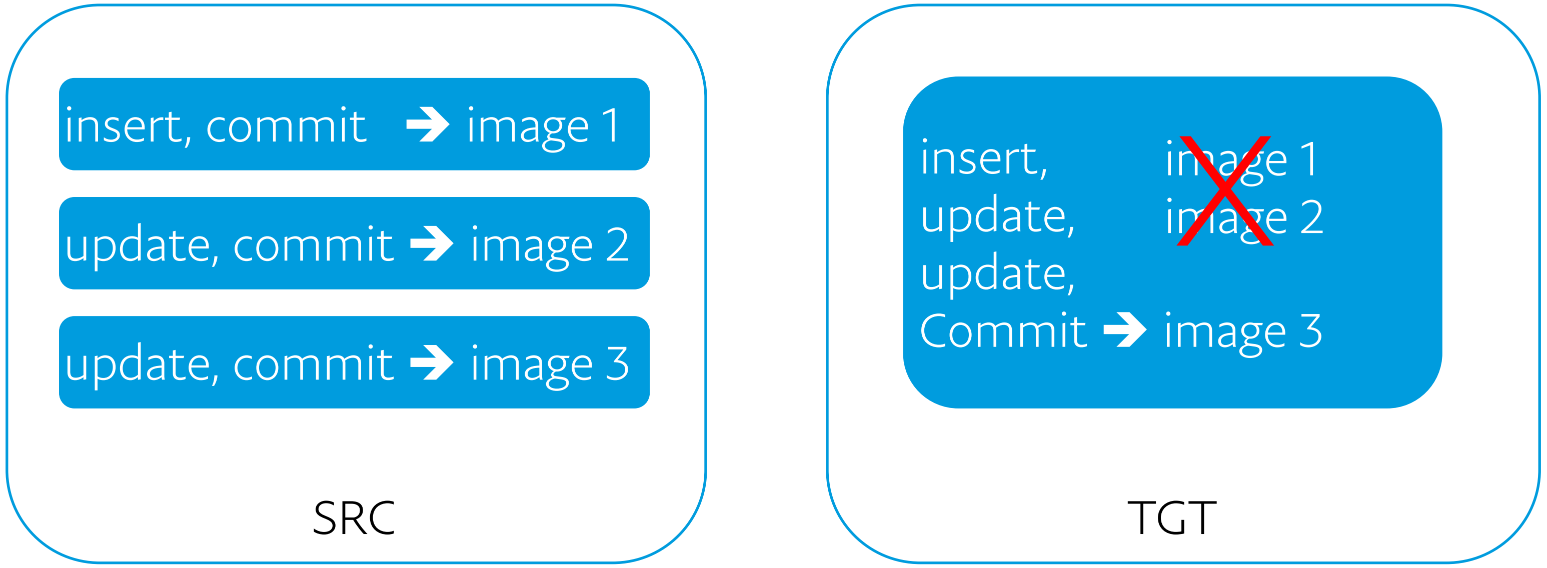
## Sampling Changed Data from Logminer

```
select username, start_scn, start_timestamp, commit_scn, commit_timestamp,
       seg_owner, table_name, rid, operation
  from (
       select username, start_scn, start_timestamp, commit_scn, commit_timestamp,
              seg_owner, table_name, rid, operation,
              row_number() over (partition by seg_owner, table_name order by commit_scn) r2
         from (
              select username, start_scn, start_timestamp, commit_scn, commit_timestamp,
                     seg_owner, table_name, row_id rid, operation,
                     row_number() over (partition by row_id order by commit_scn desc) r1
                from v$logmnr_contents
               where username not in (<exclude_user>)
                 and <include_exclude_schema>
                 and <include_exclude_table>
               )
        where r1 = 1
       )
 where r2 <= <rows_to_compare>;
```

## Running Flashback Queries for Data Comparison

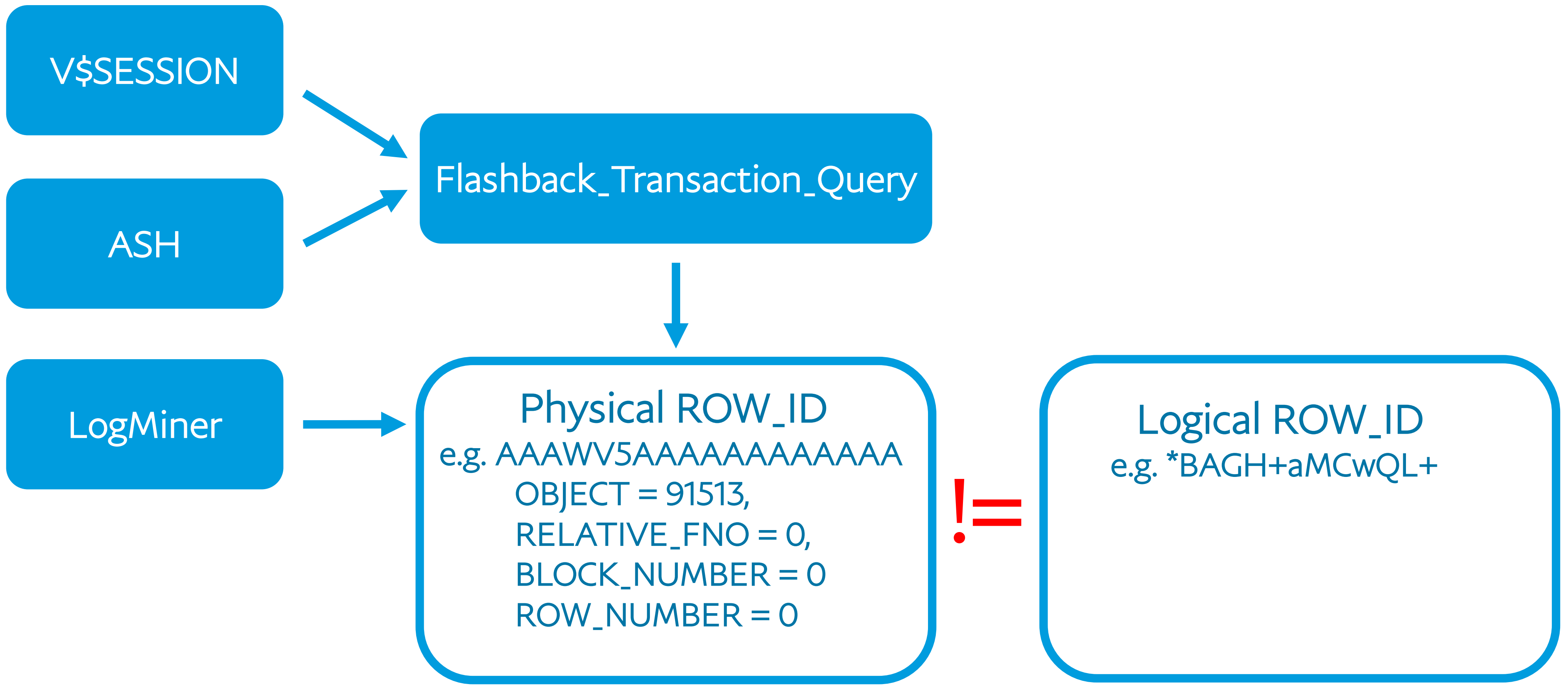
```
select count(1) from
(
  select $column_list
    from $owner.$table as of SCN ?
   where rowid = ?
  intersect
  select $column_list
    from $owner.$table\@$config{target_db_link}
         versions between timestamp
             CAST (to_date(?) AS TIMESTAMP)
             and
             CAST (to_date(?) AS TIMESTAMP)
   where $tgt_where_clause
);
```

## Handling Grouped Transactions



Workaround: expand the flashback query version range to cover all images at source, so SRC Image 3 = TGT Image 3.

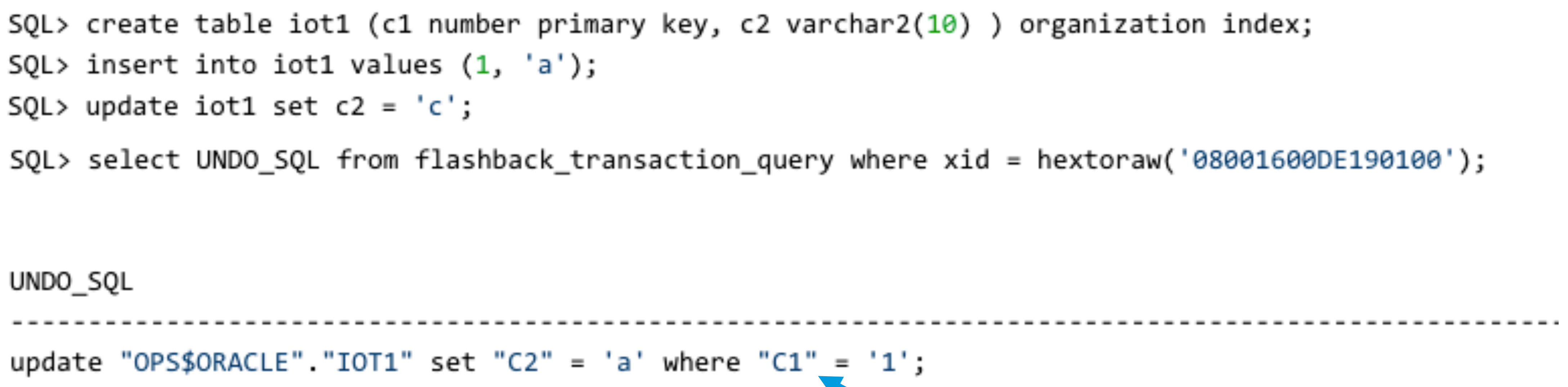
# Handling Index Organized Tables (1)



```
SQL> select * from iot1 where rowid = 'AAAWV5AAAAAAAAAAAA';
no rows selected
```

## Handling Index Organized Tables (2)

- Identify IOT rows with PK values from Flashback_Transaction_Query



```
SQL> create table iot1 (c1 number primary key, c2 varchar2(10) ) organization index;
SQL> insert into iot1 values (1, 'a');
SQL> update iot1 set c2 = 'c';
SQL> select UNDO_SQL from flashback_transaction_query where xid = hextoraw('08001600DE190100');


UNDO_SQL
--------------------------------------------------------------------------------
update "OPS$ORACLE"."IOT1" set "C2" = 'a' where "C1" = '1';
```

PK value(s)

## Handling Functional Unique Indexes

```
SQL> create table modtest (c1 number, c2 number);
SQL> create unique index modtest_ui on modtest (mod(c1, 10));
SQL> select column_name from dba_ind_columns where table_name = 'MODTEST';

COLUMN_NAME
-------------
SYS_NC00003$

Current Workaround (via configuration):
       where_clause.user1.modtest=MOD(c1,10)=MOD(:c1,10)

Enhancement (dynamic handling):
       using DBA_IND_EXPRESSIONS

       COLUMN_EXPRESSION
       -----------------
       MOD("C1",10)
```

## RTDV v2 Features

- Multiple data collection modes to adapt to the source DB activity and data coverage goal
- Optimization for production issues associated with RAC gv$ usage
- Flexible schema/table/column inclusion/exclusion/mapping by regular expression
- Table level customization on validation query where clause; DDL handling
- Integration of third party alert/notification

## RTDV v2 Benefits

- Monitoring of data quality of A/A databases bi-directional replication
- High confidence of data quality on RO DBs
- High confidence of data quality on ETL DBs
- High confidence of data quality on DB cutover/migration

## How to Get Involved?

- Provide feedback on open sourcing this tool.
- Take is for a spin in your env.
- Contribute to the enhancements.

## Q/A