

Logical Recovery Design under DB2

- **Separate set of Audit Tables to Store Changed Rows**
 - Organized by time
 - Provides for fast logical recovery from application data corruption
 - Provides an online audit trail of changes to the data
- **Purpose of Audit Database**
 - When a change is made to a DB2 table, the image of the row as it looked before the change is replicated to the corresponding audit table
 - These audit tables will be accessed during a logical recovery situation where backout of corrupt/damaged data is needed
 - Also used as a transaction log, recording all changes to the data over time
 - Allows the base database to be kept smaller in size and therefore manageable



© YL&A 1999-2007

Audit Database Design

- **Audit tables are almost identical to their corresponding base tables**
- **Primary key contains a “before” or “starting” timestamp of the image of the data**
 - Equates to the “update” timestamp in the base table *before* the row is actually updated
- **Tables contain an “ending” timestamp**
 - Equates to the “update” timestamp in the base table *after* the row is updated
 - For deletes, this equals the DB2 current timestamp special register
- **These timestamps represent the period of time the row was active**

BASE_TABLE
<u>KEY1</u>
DATA1
UPD_TSP

AUDIT_TABLE
<u>KEY1</u>
<u>STRT_TSP</u>
DATA1
END_TSP



© YL&A 1999-2007

Application Update Procedure

■ “Blind Updates”

- Application does not know which specific tables to change
- Therefore, application attempts to update all data/rows in all DB2 tables for that particular key
- The triggers analyze the update statement coming in and avoid unnecessary updates
 - Saves database storage
 - Reduces the amount of I/O and CPU use during logical recovery or audit queries
- How? The triggers compare all of the data before the update to the data after the update; if exactly the same, then the update is not processed and a -438 SQLCODE and “75001” SQLSTATE exception is returned to the application, along with a SQL message of “NO UPDATED DATA”
- The Actual update never happens
- Application is responsible for checking for this expected exception code and continuing normal processing



© YL&A 1999-2007

Capturing Data Changes via Triggers

- Define DB2 triggers to replicate changed or deleted rows from the base tables to the audit tables

- 3 types defined to each base table:

- **After Delete** of a row on the base table, insert that row to the audit table with appropriate starting and ending timestamps

```
CREATE TRIGGER YLA.UDANTRG3
AFTER DELETE ON YLA.BASE_TABLE
REFERENCING OLD AS OLDROW
FOR EACH ROW MODE DB2SQL BEGIN ATOMIC
INSERT INTO PPOCSSR.AUDIT_TABLE VALUES (OLDROW.KEY1, OLDROW.UPD_TS , OLDROW.DATA1 , CURRENT_TIMESTAMP);END!
```

- **Before Update** of a row on the base table, check all columns to see if data has actually changed; if not, return a -438 SQLCode, and SQLSTATE of “75001” to the application and do not perform the update. An SQL message of “NO UPDATED DATA” is also returned.

```
CREATE TRIGGER YLA.UDANTRG1
NO CASCADE BEFORE UPDATE ON YLA.BASE_TABLE
REFERENCING OLD AS OLDROW
NEW AS NEWROW
FOR EACH ROW MODE DB2SQL
WHEN ( (OLDROW.DATA1 = NEWROW.DATA1) )
BEGIN ATOMIC
SIGNAL SQLSTATE '75001' ('NO UPDATED DATA') ;END!
```

- **After Update** of a row on the base table has processed, insert the before image of row to audit table with appropriate update and ending timestamps

```
CREATE TRIGGER YLA.UDANTRG2
AFTER UPDATE ON PPOCSSR.BASE_TABLE
REFERENCING OLD AS OLDROW
NEW AS NEWROW
FOR EACH ROW MODE DB2SQL BEGIN ATOMIC
INSERT INTO PPOCSSR.AUDITBL VALUES (OLDROW.KEY1, OLDROW.UPD_TS , OLDROW.DATA1 , NEWROW.UPD_TS); END!
```



© YL&A 1999-2007

Logical Recovery

- Automated and instantaneous process
- Logical Recovery Table
 - Used to initiate a logical recovery
 - Contains a timestamp
 - Empty of data most of the time
 - In a logical recovery situation, a row is inserted into this table
 - The logical recovery timestamp indicates the point in time to recover to, and must be carefully set to a precise timestamp

LGCAL_REC
<u>LGCAL_REC_TS</u>

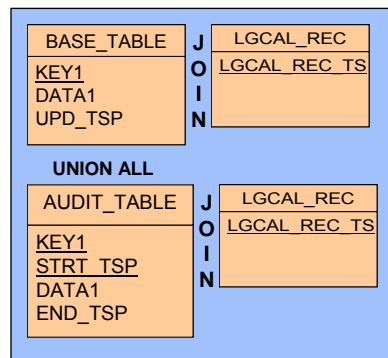


© YL&A 1999-2007

Detection and Data Selection During a Logical Recovery

- Application programs and ad-hoc queries must be coded to detect a recovery situation in order to continue processing and selecting data
- This can be accomplished by accessing a set of DB2 “as of” views encompassing the base and audit tables
- The application will always read the LOGRCVRY table first
- If data is present (indicating a recovery situation), the application should branch to code that accesses the “as of” views
- If data is not present, the application processes normally against the base tables
- Logical Recovery Views

- Each view is actually a query that accesses both the base table and audit table in a “Union” SQL statement.
- The logical recovery table provides the timestamp in the views for the point of recovery
- The views are then accessed by the application when it needs the “as of” image of the data
- Column names in each view are identical to column names in the base table; only the table name is different



© YL&A 1999-2007

SQL Recovery View Example

- Once a logical recovery is detected, all read access is via the views
 - UNION in view utilized
- Accessing the views is accomplished through a separate set of DB2 cursors

```
CREATE VIEW YLA.LREC_VIEW
  ( KEY1 , DATA1 , UPD_TSP )
AS
  SELECT BASETB.KEY1 , BASETB.DATA1 , BASETB.UPD_TSP
  FROM   YLA.BASE_TABLE BASETB
  INNER JOIN
    YLA.LGCAL_REC AS RCVRY1
  ON BASETB.UPD_TS <= RCVRY1.LGCAL_RCVRY_TS

  UNION ALL

  SELECT AUDITB.KEY1 , AUDITB.DATA1 , AUDITB.STRT_TSP
  FROM   YLA.AUDIT_TABLE AUDITB
  INNER JOIN
    YLA.LGCAL_REC AS RCVRY2
  ON AUDITB.STRT_TS <= RCVRY2.LGCAL_RCVRY_TS
  AND AUDITB.END_TS > RCVRY2.LGCAL_RCVRY_TS ;
```



© YL&A 1999-2007