Action Diagram Performance

Session 700

Tracy Wood
Texas Instruments

Introduction

- This session will discuss PrAD/PAD design techniques that may be used to increase performance of a Composer application
- Objective: To empower attendees with the information needed to make critical design decisions geared toward maximizing PrAD/PAD performance

Assumptions

This session assumes attendees have:

- Basic understanding of Composer analysis and design concepts
- Familiarity with SQL and relational database concepts
- Understanding of Procedure Action Diagramming (PrAD) and Action Blocks (ABs)

Topics

- View control
- Read optimization
- Action Block usage

View Control

- View
  - Matching
  - Usage
  - Optimization

View Matching Tips

- Create identical information view structures
  - Same attributes
  - Same order
- Create view structure for matching and one for other operations, when necessary
- Problems not likely to occur unless view structure is complex (i.e., many attributes) and the view is passed many times (e.g., an action diagram is called thousands of times)

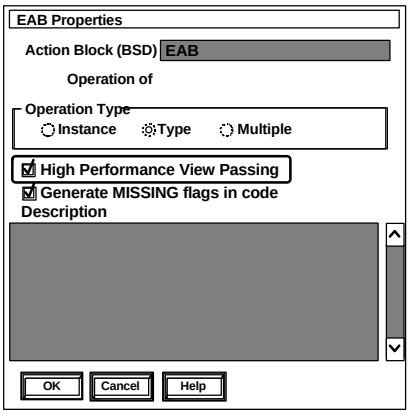
View Usage

- Group views
- Persistent views
- Load validation into group views

Group Views

- Eliminates intermittent move statements
- Eliminates additional structures
- Reduces complexity of view matching
- Composer will pass starting address of group view
- Cardinality of one (non-repeating)

High Performance View Passing

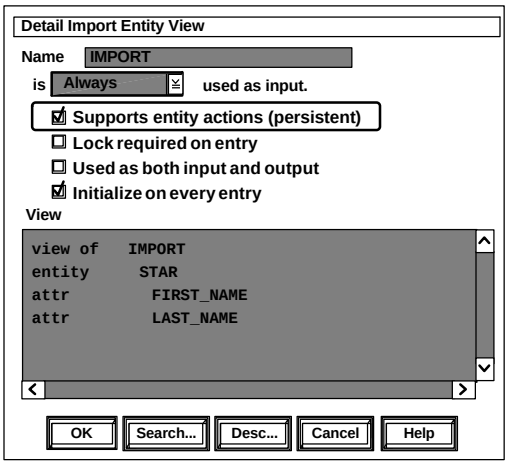


- Efficient view passing
- Allows passing of pointers
- Eliminates intermittent moves
- Eliminates additional structures

Persistent View Usage

- Allows currency of entity action to be passed from parent to child
- Greatest benefits:
  - High-volume batch transactions
  - High-volume complex on-line transactions
- Re-reading stable entity action views within an action diagram adds I/O overhead to a unit of work

Defining Persistent Views



- Entity action views may be defined as persistent
- Local and workset views may not be persistent

Persistent View Actions

```
Entity View import star (mandatory, persistent, exported, locked)
     first name (mandatory)
     last name (mandatory
     agent (optional)
     phone number (optional)
```

- If the view will be used in either an *UPDATE* or *DELETE*, it must be defined as persistent LOCKED (will hold currency for that occurrence until that action committed)

- When locking option used, *SELECT FOR UPDATE* is established in calling procedure (ensures currency and Update Intent Lock on the referenced page/block)

Persistent Import Only View

- If a view is defined as PERSISTENT in a called action diagram, and is defined as IMPORT ONLY
- the SELECT statement generated from a READ action in the calling action diagram will *NOT* result in a *SELECT FOR UPDATE*
- It will *NOT* be able to be used in any *UPDATE/CREATE/ASSOCIATE* constructs

Persistent Import/Export View

- If the view in the called action diagram is defined as PERSISTENT, *IMPORT*, and *EXPORT*
- Then a *SELECT FOR UPDATE* will be generated in the calling action diagram
- The view *CAN* be used within the *UPDATE/ CREATE /ASSOCIATE* constructs

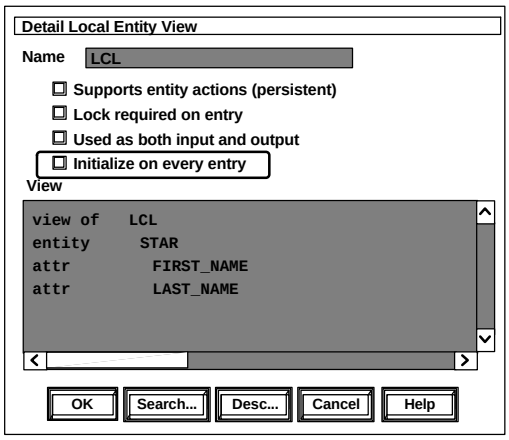
Load Validation into Group Views

- Validation table is loaded into group view once
- Validation logic processes against group view
- Reduces database reads
- Can pass as hidden view

View Optimization

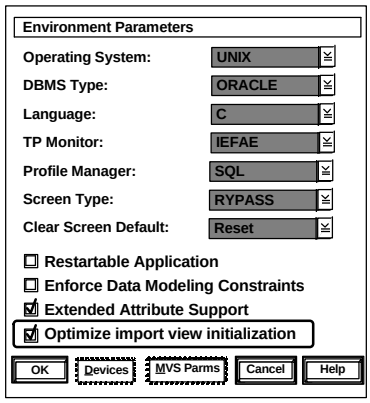
- Turn off local view initialization
- Turn off import view initialization
- Importable Export/Exportable Import usage

Turn Off Local View Initialization



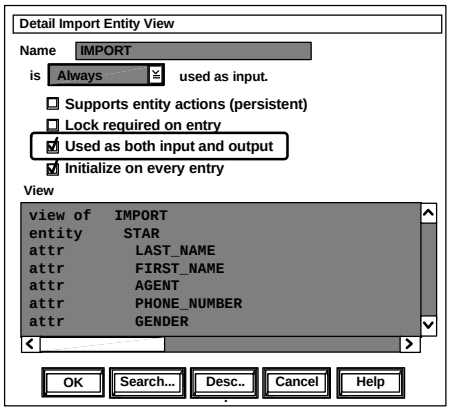
- *Local View Optimization*: Initialization may be turned off for individual local views
- This option will enable the application to set a value and retain it for the duration of the application

Turn Off Import View Initialization



- Import view Optimization
  - Initialization logic may be turned off as a Generation Environment option, reducing initialization of import views (This feature is optional, as it will cause incompatibility with applications generated prior to IEF 5.0)

Importable Export/Exportable Import Usage



- Eliminates initialization section
  - Reduces move statements
- Eliminates moving import to export
- Reduces number of views
- Relies on calling AB for initialization

View Control Summary

- View
  - Matching
    - » Structure
  - Usage
    - » Group Views
    - » Persistent Views
  - Optimization
    - » Turn off initialization
    - » Importable/Exportable

READ Optimization

- ERD access path strategy
- Extended READs
- READ statement guidelines
- READ statement cursor option
- READ EACH options

ERD Access Path Strategy

- *Involve DBA*
- Volumetrics of entity types
- Selection criteria
- Use most efficient data access path
- Reduce the number of reads
- Dependent on processing required
- Use primary identifiers when possible
- *Communicate* primary entity access strategy

Extended READ

- Achieving currency on more than one table with a single read statement

```
READ EACH Customer
                Order
        WHERE DESIRED customer places DESIRED order
AND DESIRED customer_code = import customer_code
```


Extended READs

- Allows DBMS to perform join
- Saves read of associated entity type
  - As foreign key or through denormalization
- Cannot distinguish which entity is not found
- Requires proper view management
- Usage based heavily on denormalization

Extended READ Targeting a Single Table

- Entity action views customer order code
- Number delivery address posted date

```
READ customer
        order
      WHERE DESIRED customer_code = import
      customer_code AND DESIRED customer
      will place DESIRED order AND DESIRED order_number =
      import order_number
```


Resulting SQL

```
SELECT
      ORDER002.FK_CUSTOMERCODE,
      ORDER002.STATUS,
      ORDER002.DATE0,
      ORDER002.NUMBER
   FROM
      ORDER0                ORDER002
   WHERE
      ORDER002.FK_CUSTOMERCODE = :CODE-001TP AND
      ORDER002.NUMBER = :NUMBER-002TP
   END-EXEC
```

Extended READ Resulting in a Join

Entity Action Views

Customer	Order
Code	Status
Name	Date
	Number

READ customer
order

WHERE DESIRED customer_code = import customer_code AND DESIRED customer will place DESIRED order and DESIRED order_number = import_order_number

Resulting SQL

```
SELECT
        CUSTOMER01.NAME,
        CUSTOMER01.CODE,
        ORDER002.FK_CUSTOMERCODE,
        ORDER002.STATUS,
        ORDER002.DATE0,
        ORDER002.NUMBER,
        ORDER002.FK_CUSTOMERCODE
    FROM
        ORDER0          ORDER002,
        CUSTOMER        CUSTOMER01
    WHERE
        CUSTOMER01.CODE = CODE_001TP AND
        ORDER002.FK_CUSTOMERCODE
        CUSTOMER01.CODE AND
                        ORDER002.NUMBER =
    :NUMBER-002TP
    END-EXEC
```

READ Statements

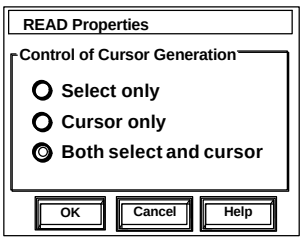
- Minimize READs when possible
  - Retrieve data
  - Create currency
- Attributes used as qualifiers in reads should have identical properties as comparison attributes
- Retrieve only required data (starve entity action views)
- Search minimum number of rows

READ Statement Guidelines

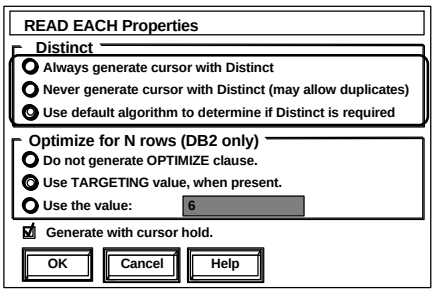
- Never use functions or mathematical computations in READ statements
  - Define local view to be set using function/or compute
- Use positive logic, avoid NOT
  - May eliminate index usage
- Avoid comparing attributes of different domain or length

READ Statement Cursors

- SELECT ONLY (if one row guaranteed)
  - Avoids overhead of opening cursor
  - Will fail if more than one row
- CURSOR ONLY
  - Cursor will always be opened, even if only one row retrieved
- BOTH SELECT AND CURSOR
  - Select executed
  - If more than one row retrieved, a cursor is opened



Select Distinct



- Can be turned off, forced on, or left up to Composer to decide (default is Composer decides)
- Can be set for individual READ EACH statement (5.3 +)

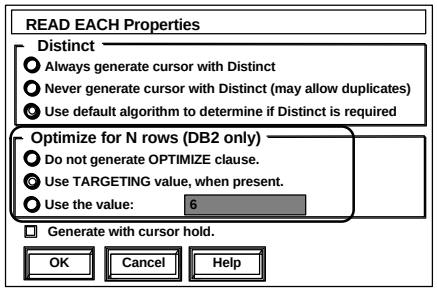
- Used to select unique occurrences
- Avoid using if large number of occurrences in table
- Causes sort of composite table (reference DB2 Version)

Default Algorithm for Distinct

- Generates DISTINCT if the generated SQL accesses a table whose corresponding entity is not specified in the READ list

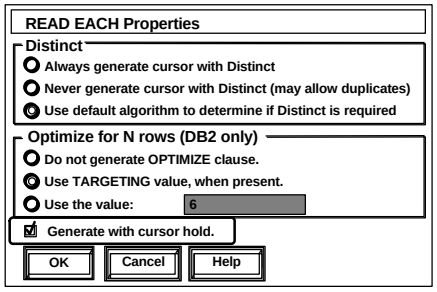
```
READ EACH order_line
   WHERE DESIRED order_line is_history_data_for SOME order AND THAT order
    nett_value = 100
EXEC SQL DECLARE CUR_EXAMPLE CURSOR FOR
    SELECT DISTINCT
          ORDER_LINE_DATE,
           ORDER_LINE_STATUS
      FROM        ORDER    O,       ORDER_LINE L
WHERE        L.FK_ORDER_NUMBER = O.NUMBER AND ONETT_VALUE = 100;
```


Optimize for N Rows



- DB2 2.3 feature, optimizing access path for return of *N* rows
- Normally used for a READ EACH with a TARGETING clause, the TARGETING value (maximum of the group view) is used
  - Does NOT affect the number of rows fetched
  - Used on READ statements as *optimize for 1 row* when cursor is selected
  - If the READ EACH clause does not have a TARGETING clause, then *optimize for* will not be used in the generated SELECT

Cursor with Hold



- Available for READ EACH statements only
- Generated if the target environment is MVS and DB2
- DB2 will ignore it when used in Composer online transactions (either pseudo-conversational CICS or message-driven IMS applications)

Cursor with Hold

Read Each Customer
.....
Update Customer
External Commit

- Commit must be issued within *READ EACH* construct after any entity view logic
- Commits need to be performed in an external action block because Composer does not support explicit commits (to avoid misuse of commits in PADs)
- The external action block used for issuing commits must include SQLCA, and must be precompiled and bound with the Composer-generated batch application's plan

READ Optimization Summary

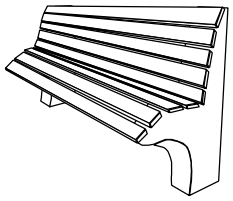
- ERD access path strategy
- Extended READs (powerful with proper views)
- READ statement guidelines
- READ statement cursor option
- READ EACH options (select Distinct, Optimize for N, Cursor with Hold)

Action Block Usage

- Customize function calls
- Minimize number of USE statements
- Server design issues

Customize Function Calls

- *Benchmark* to determine if required
- Allow customization for system
- Speed up performance
- Increase project EAB maintenance
- Reusable



Minimize Number of Use Statements

- Combine process action blocks with high-level action diagrams
- Reduce load module size
- Review action block reusability
- Avoid generic action blocks
  - Should have specific task and be used consistently

Server Design Tips

- A server should perform a specific task
  - minimize input/output data
- Minimize volume of data flowing between server and client
  - Force selection criteria prior to server execution
  - Provide for next and previous database searches
- Turn trace off at client manager

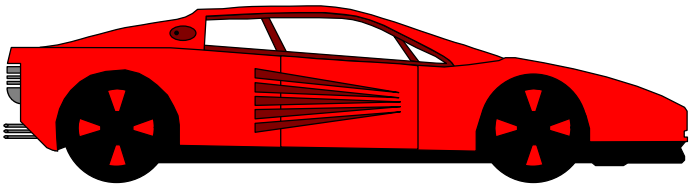
Action Block Usage Summary

- Customize function calls
  - Primarily of benefit for repetitious calls
- Minimize number of Use statements
  - specific AB, more inline code

In Summary

- Make DBA active participant of application design
- Communication among team members is critical
- Proper view management is critical
- Optimize READ statement usage
  - Understand and utilize Composer features
- For extreme performance requirement minimize action block usage

Perform with Composer!



Action Diagram Performance

Session 700

Tracy Wood
Texas Instruments