# JCC LogMiner Loader Workshop

Jeffrey S. Jalbert
Cheryl P. Jalbert

♦

Tom H. Musson
Keith W. Hare

JCC Consulting, Inc.

# Abstract

Oracle Rdb Engineering and JCC have created – and continue to expand – a fast, powerful, flexible pair of tools to reflect changes made in  an Rdb database to other platforms.

- Oracle Rdb LogMiner
- JCC LogMiner Loader

This no-charge, one-day seminar will provide an introduction to the concepts, architecture, capabilities, and applications of this powerful pair.  The workshop nature will demonstrate installation, configuration, use, and monitoring.  Your own architectural challenges are welcome.

This is a technical seminar intended for Systems Architects and Database Administrators.  A general introduction to the LogMiner Loader is available as the Loader features page on the JCC website.

# Thank you!

- Thanks to Oracle for providing the site.
- Thanks to Kevin Duffy and others on-site for making the arrangements.

# Uses of JCC's LogMiner Loader

- To create and maintain a composite of regional databases
- To maintain multiple copies of the corporate information to scatter to remote locations
- To be able to use tools (such as Oracle's data warehouse tools) that are not available in the source database environment
- To provide web access without performance and security impacts on the production database(s)
- To reorganize a huge, critical, and overworked database without appreciable downtime
- To add coherency to an environment with more than one database platform
- To archive data
- To audit data changes
- To convert to another platform without interruption in support
- To provide realistic data for serious application testing

Who is here and what uses do you envision?

# JCC LogMiner Loader

- This seminar is tuned for LML V3.2.4.
- JCC encourages updates to the latest version of the Loader.
  - New releases of the Loader
    - Provide new features.
    - Address all known bugs.
    - Often, provide protection from bugs in companion products.
  - JCC is careful to protect backward compatibility.
  - We will show you how to run new releases as variants.
- As we get into more detail, we can discuss the versions of other products in your architecture.

# Agenda

- Your questions are important throughout.
  - Agenda is designed to take whatever detours you dictate.
- Discussion and live workshop will illustrate
  - General concepts supporting the LogMiner and the Loader
  - Acquisition and installation of the Loader
  - Configuration of the Loader
  - Running the Loader
  - Using the Loader's monitor to track system performance
  - Tuning system performance
  - Tools to aid the DBA
  - Options and solutions to design challenges

# Table of Contents

Topic 1

# Oracle Rdb LogMiner

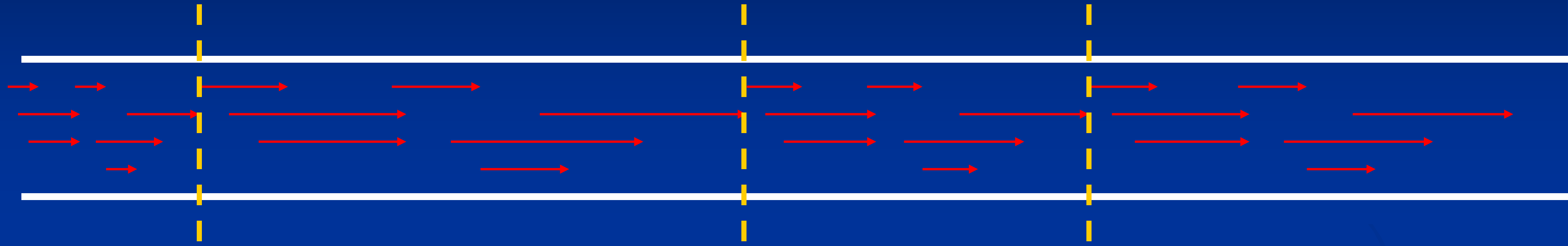
# Rdb's AIJ (After Image Journal)

- With Rdb, an AIJ (After Image Journal) file or set of AIJ files may be associated with a database.
  - The original purpose of the journal is recovery.
- After journaling is turned on, almost all changes to the database are also reflected in the Journal.
  - The number of database properties and events that are *not* reflected in the Journal – for example, having the database enabled for LogMiner – are diminishing over time.
- Conceptually, the Journal is a *single stream record of changes that never terminates.*
  - For management purposes, the active journal is partitioned into multiple fixed-length files.
  - The files can be backed up and/or emptied out, periodically, to provide space for additional changes.
  - The backed-up files are conceptually part of the historical journal.

# Quiet Points

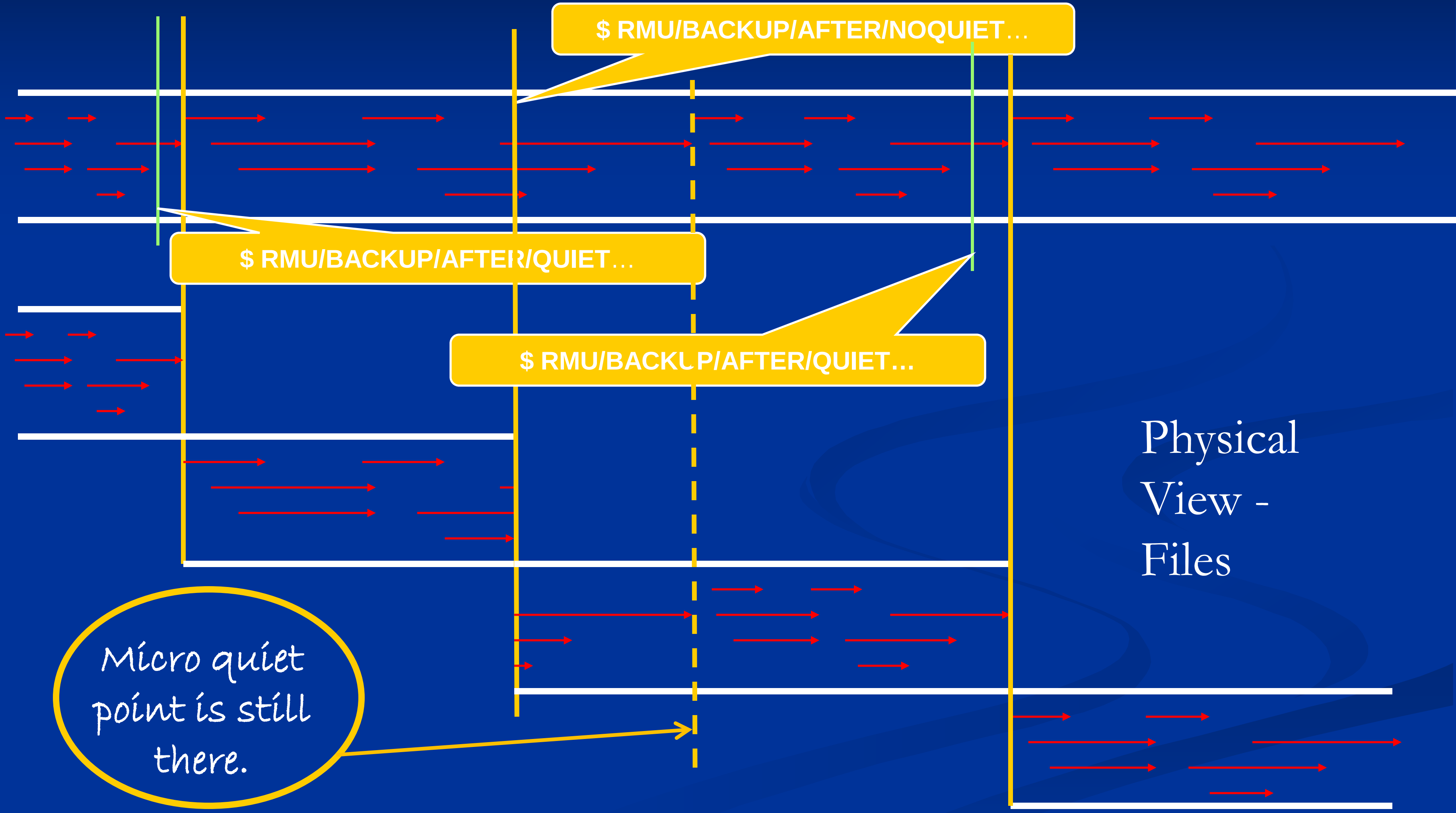
- To be maintain the consistency of the transactions, requires some care.
- A quiet point is a point in the Journal at which all transactions that have written to the Journal have also committed (or rolled back).
  - That is, at a quiet point, the journal has no transactions in progress.
- Each AIJ file contains a header which can record whether a quiet point has occurred when the (previous) AIJ backup occurred.
- Quiet points are
  - Requested
    - By the database administrator
    - By the Rdb engine
  - Spontaneous, a natural result of the work flow
    - Called "micro" quiet points
    - Are surprisingly frequent

# After Image Journal



Conceptual View – continuous record of database changes

# After Image Journal Backups



$ RMU/BACKUP/AFTER/NOQUIET…

$ RMU/BACKUP/AFTER/QUIET…

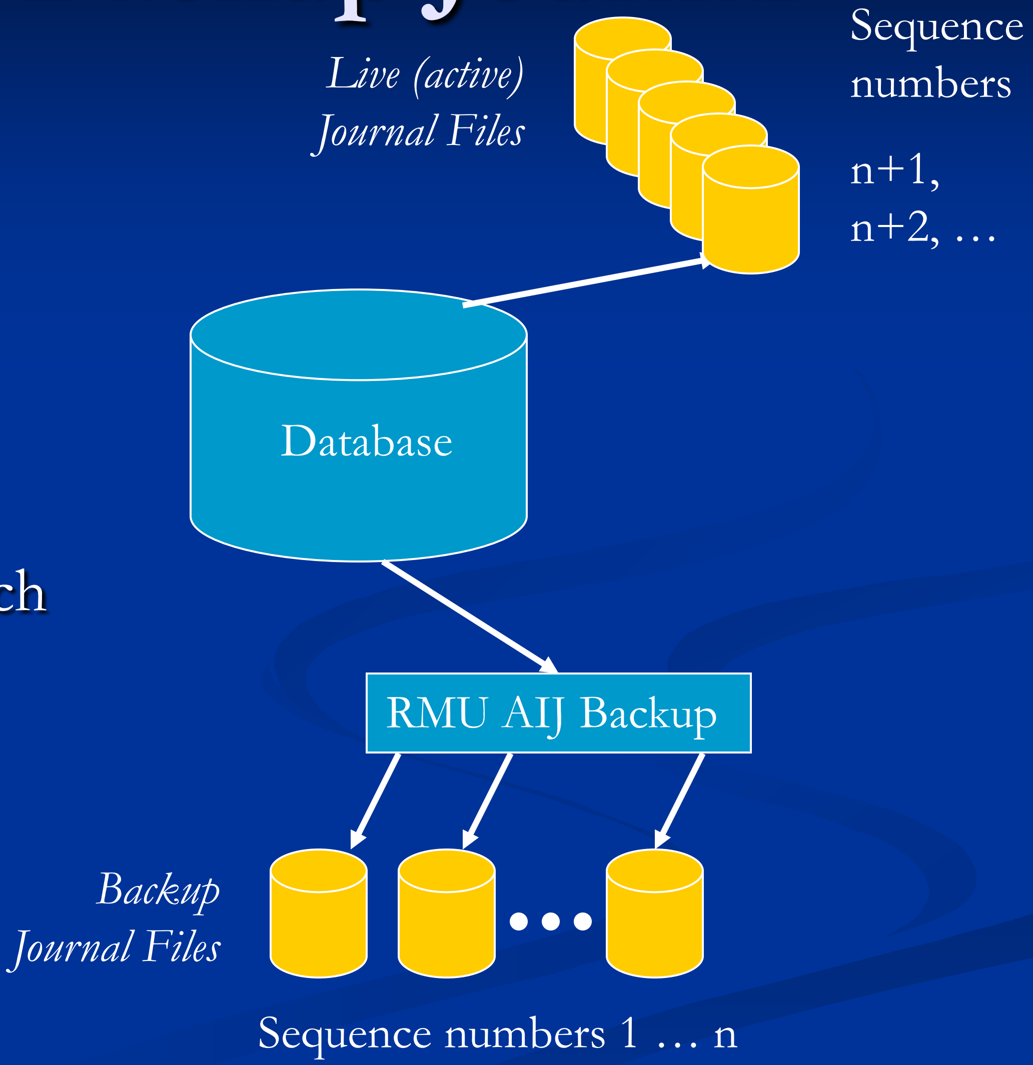
$ RMU/BACKUP/AFTER/QUIET…

Micro quiet point is still there.

Physical View - Files

# Active and Backup Journal

- Conceptual journal is represented in files
  - Active journal
  - Backup journal
- Data in currently active journals is backed up to a single target file during each backup operation.
- Journal sequence numbers are incremented when switching journals.



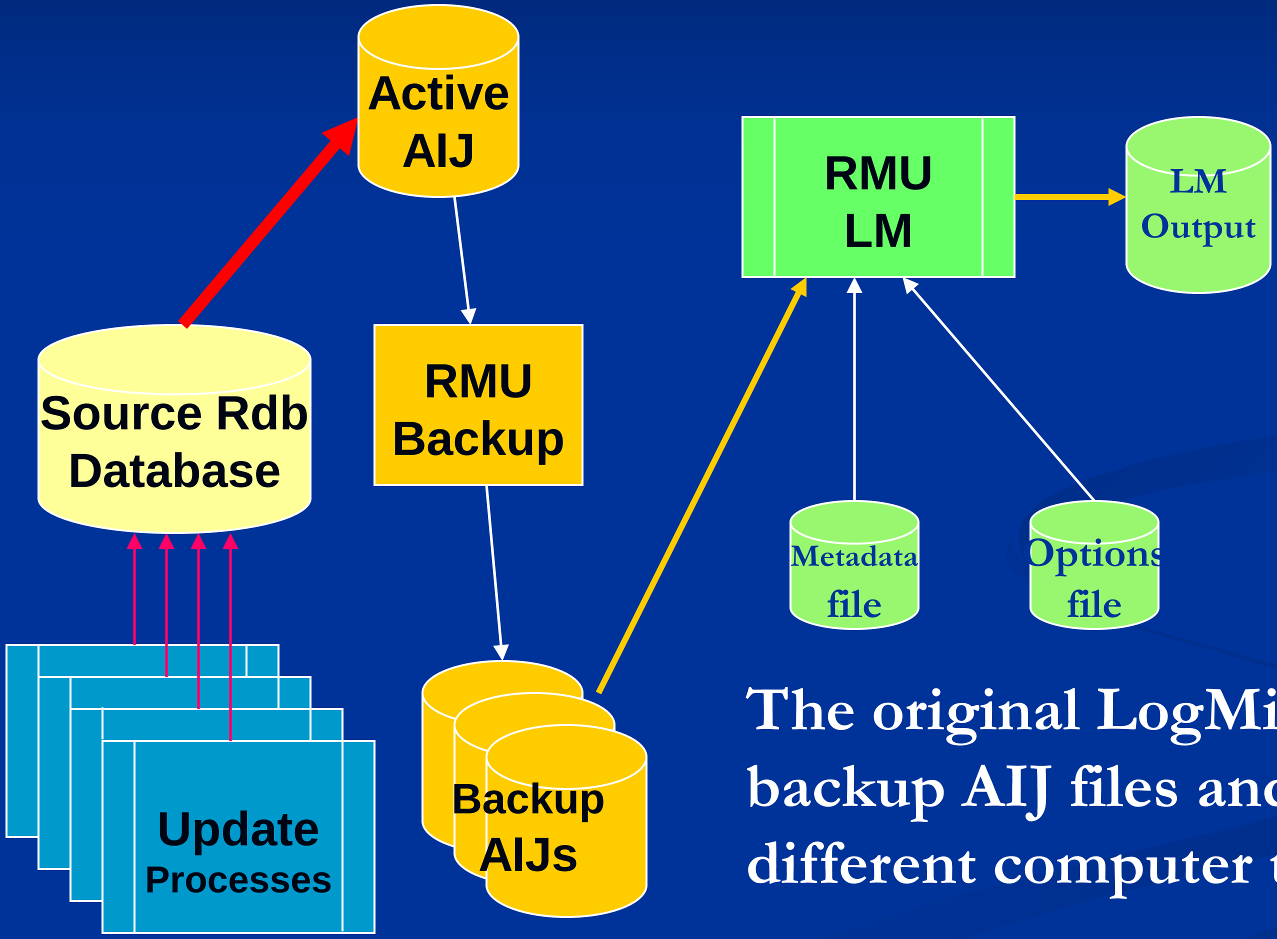
# Rdb LogMiner

- The LogMiner reads the AIJ to produce a stream of changes.
- LogMiner runs in either of two modes
  - Static
    - Runs on backup copies of AIJ files
    - Runs on the same machine as the source db or a different machine
    - Is batch-like
  - Continuous
    - Runs continuously in "Near Real Time" – must wait for the commit on the source
    - Runs on a machine where the source database is open
- The Continuous LogMiner view of the AIJ is comparable to the conceptual view.

# What Rdb LogMiner Does

- Starts at a quiet point
  - Guarantees that work is complete…all rows are included
- Extracts rows from the AIJ.
  - Only rows with changes are reflected in the AIJ.
  - Only rows for tables specified in the options file are processed by the LogMiner.
  - The LogMiner does not interpret the data within the row.
- Writes requested rows to specified output files.
  - Disk files
  - **Mailbox (pipe) – Loader**

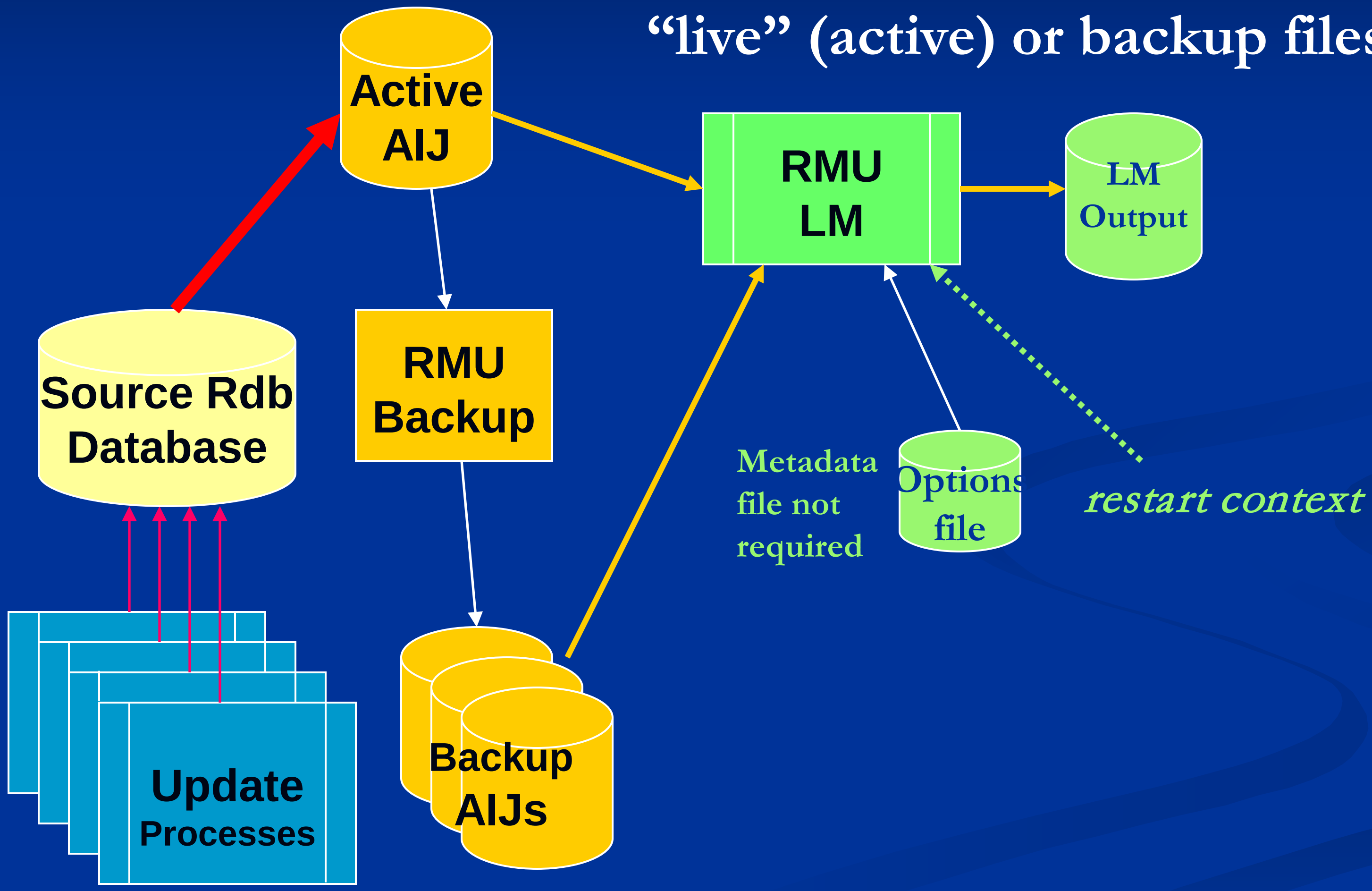
# Static LogMiner



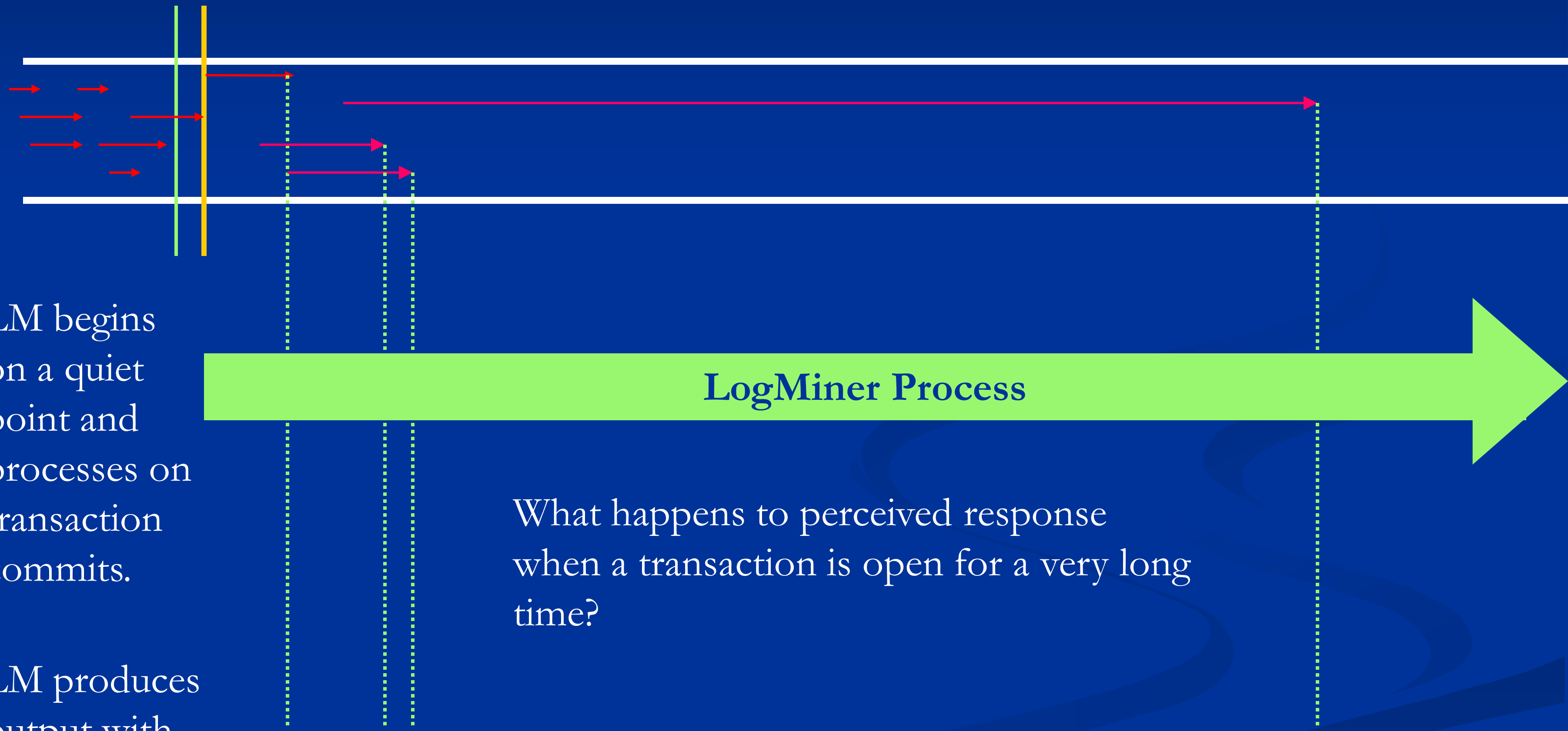
The original LogMiner ran on backup AIJ files and could run on a different computer than the source.

# Continuous LogMiner

CLM uses the AIJs in order whether "live" (active) or backup files.



# Commit on Source DB



LM begins on a quiet point and processes on transaction commits.

LM produces output with little delay.

What happens to perceived response when a transaction is open for a very long time?

# Processing

- The LogMiner may have many transactions "in flight" simultaneously.
- If the volume of data is large for one transaction, data spills onto work files.
  - May imply an I/O issue for work files.
  - May imply a space issue for work files.
  - Managed with assistance from the Loader.
- When the commit record is received, rows are sorted and *only* the last version – within the transaction – of any row is sent to the output file.

# Commits and CLM

- Optionally, CLM writes commit records to the output file.
    - Must be explicitly requested with `/include=action=commit`
    - Required for JCC LogMiner Loader.
- Unlike static LogMiner, CLM can restart at a micro quiet point in the journals.
    - Backing up the AIJ files does not interfere.
- The context is called the AERCP.
    - Journal Sequence number
    - Offset from start of journal
    - Other info
- If requested, the AERCP, for the most recent quiet point is provided with each commit record.

# LogMiner Options File

- The tables to be considered by LogMiner must be specified.  If the table list is extensive, this must be specified in an options file as in the following example:

```
table=account                   ,output=rdb_logminer_output_file
table=account_audit             ,output=rdb_logminer_output_file
table=account_batch_processing  ,output=rdb_logminer_output_file
table=account_contract          ,output=rdb_logminer_output_file
```

- For the example, the file name is a logical name.
- Output to one of these
  - File for each table
  - **Same file for all (Loader requirement)**
  - Application callback in shareable image

Loader kit provides help.

More later.

# Metadata File for Static LogMiner

- LogMiner reads database to get accurate information about record structures.
- For static LM *to be run without access to the source database*, the metadata file provides the alternative to opening the database.

```
$ rmu    /unload                               -
         /after_journal                        -
         /save_metadata=my_db.metadata         -
         /log                                  -
         my_db
```

- For continuous LogMiner, the database must be opened on the node where the LogMiner is running.
  - Lock marks the logical EOF points in the AIJ.
  - Metadata file not appropriate and not used.
  - Information on the metadata comes from the db.

More on modes later.

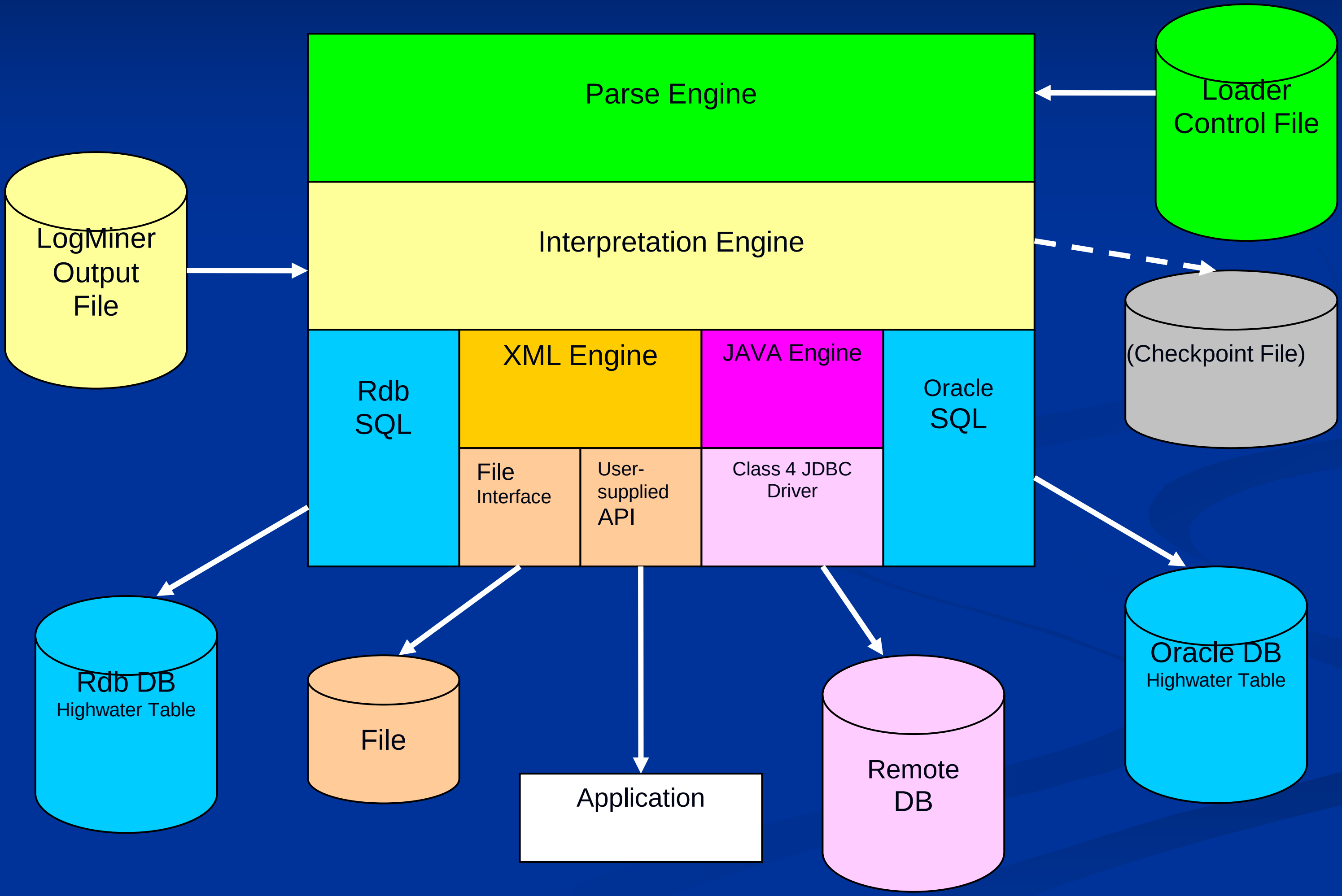
Topic 2

# JCC LogMiner Loader

# JCC LogMiner Loader

- The JCC LogMiner Loader (the Loader) reads the binary files produced by the Rdb LogMiner and emits them to a target.
- The target may be one of the following:
  - Rdb database
  - Oracle database
  - XML document stream (one document per source transaction, to a customer-supplied API)
  - Disk file
  - Tuxedo
  - JDBC Class 4 driver (There are at least 141 different targets.)
    - Oracle (although using the Oracle target directly is better)
    - SQL Server
    - Tandem
    - Your favorite

# JCC LogMiner Loader Architecture



# Partnership – LM and LML

- The Loader manages the restart context for both the Oracle Rdb LogMiner and the JCC LogMiner Loader.
- The Loader kit includes procedures to create the options file for LogMiner, as well as metadata definitions needed for the Loader.
- There are other architectural links (heartbeat) that belong  later in the discussion.
- JCC and Oracle have explored expanding the source to include an Oracle 10 or 11 source.
  - The architecture will need to be altered.
  - The only source supported, at this time, is Rdb.
  - JCC has created application solutions.

We can update this.

# Partnership

- Concepts that we will see in establishing our Loader definition …
  - LogMiner requires
    - AIJs
    - Options File
  - Loader is configured with
    - Control File
    - Some logical names
  - LogMiner and Loader fault tolerance relies on
    - “checkpoint” or “highwater” data

# Workshop

- Pardon the interrupt …
- Let’s see some results.

# Pause for Workshop

- Let’s look at
  - How to acquire the Loader
  - Loader keys
  - Installation
  - Directory structure
  - Product versions supported
  - Post installation steps
  - The example for the Workshop
- We can’t actually run the Loader until we discuss a few choices that tune the results to your needs.

# Download

- Kit is available at FTP.JCC.COM.
  - Documentation
  - Kit, including examples and useful procedures
- Download as a self extracting OpenVMS executable.
  - The documentation provides an example FTP session.
  - Use VMS FTP or make certain that your browser is properly set.  (Do not use passive FTP.)
- Read the documentation online with Adobe Acrobat or print it.
- Get a license by sending mail to info@JCC.com

# License

- Permanent licenses are available for purchase.
  - Permanent licenses include the first year of Basic Support.
  - Basic Support includes
    - JCC LML Help Desk (during JCC's normal hours)
    - Additional releases during the support period
  - "Gold" support for 24 X 7 coverage is also available.
  - Support is renewable.
- Temporary licenses are available for testing the Loader in your shop.
- We will use a temporary license for this workshop.

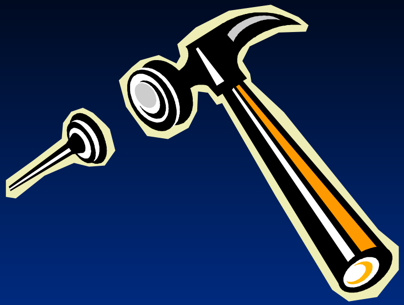
# License

- The license key we will use in this workshop is:

  6C072B24-TEMP:01:JUL:2010-373903836119040-474726881638809061-656A2A42

  - As you can see this is a long value.
  - It is sent by email so you can cut and paste.
- This is a *temporary* license for testing use.
  - For support send mail to JCC_LMLoader@JCC.com
  - Please ask for an extension, if this license runs out before your testing is done.
  - Ask for a permanent license when testing is done or before going to production.

# Install →



- Restore the saveset.
- Modify system startup and manually execute the startup for the first time.
- Run the installation verification procedure.
- Run the IVP cleanup.
- Run any special procedures required for the target you have chosen. (We'll talk about targets in the next section.)

# Restore the Saveset

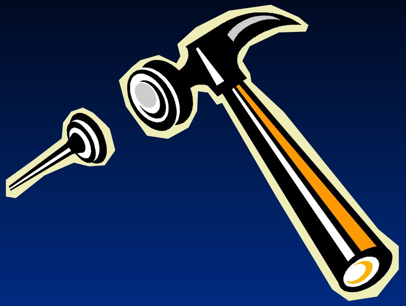
- Download the zip archives and unzip them or get the self-extracting .EXEs.
- Choose the disk and directory where the Loader directory tree will reside. (The root directory on the volume is recommended. )
- $ backup jcclml_MM_mm_pp.sav/save_set <disk>:[<dir>…]/new_version
  - "MM_mm_pp" specifies LML Major, minor, and patch version.
  - <disk> and <dir> should be specified
  - The ellipses (…) and "/new_version" are required.

# Account for Installation

- JCC recommends using the system account when doing the installation.
- Privileges needed for the installation are:
  - SYSNAM to create system logical names and name tables
  - (Additional privileges are needed to *run* the Loader.)
- Various images are installed at system boot time

**TIP** Startup procedures are self referencing, do NOT move before execution.

# Installation Parameters



- @<directory path>:jcc_dba_startup [p1[ p2]]
- Installation has two parameters
  - Version
  - Variant
- Both parameters are optional and defaults are defined.

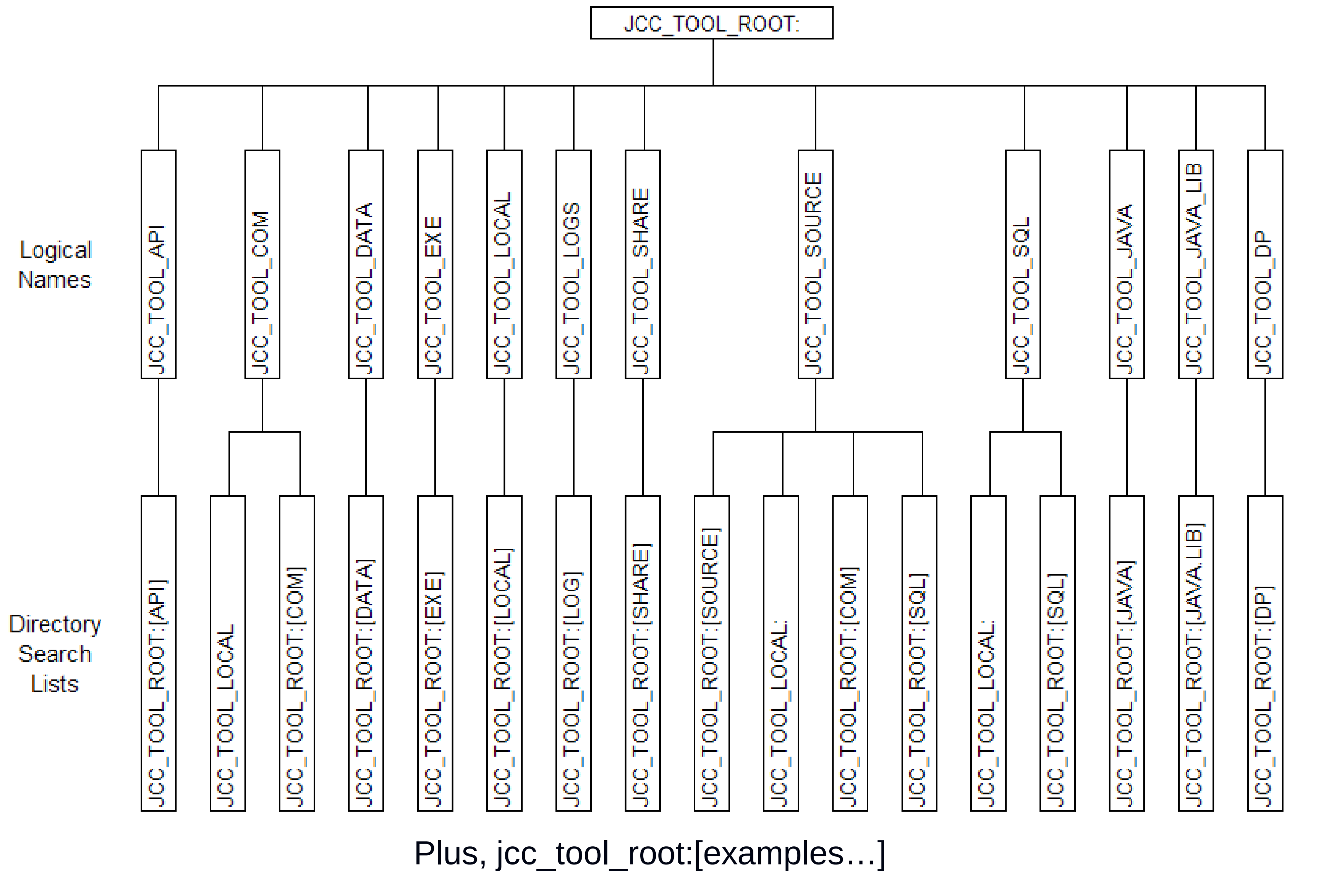
# Versions

- You may want to run more than one version of the Loader at a time.
  - Test new releases
  - Protect the version running in production
  - Use any number of variants
- P1 (version) default is ‘S’
  - S means standard
  - This is the system-wide installation
  - Logical names go into system table
- Alternate P1 value is MV
  - MV means variant
  - Logical names go into a special table

# Variants

- Multiple build types allow you to choose for best performance for your site.
- P2 (variant) default is 'Standard'
  - Standard is linked with the OpenVMS POSIX thread library
    - Required for Oracle and JDBC targets.
- P2 alternatives are
  - EV6 compiled specifically for EV6 optimization.
  - ST (sans threads) supplied to improve performance
    - Use if p-threads are not required.

# Directory Tree



JCC_TOOL_ROOT:

Logical Names

JCC_TOOL_API
JCC_TOOL_COM
JCC_TOOL_DATA
JCC_TOOL_EXE
JCC_TOOL_LOCAL
JCC_TOOL_LOGS
JCC_TOOL_SHARE
JCC_TOOL_SOURCE
JCC_TOOL_SQL
JCC_TOOL_JAVA
JCC_TOOL_JAVA_LIB
JCC_TOOL_DP

Directory Search Lists

JCC_TOOL_ROOT:[API]
JCC_TOOL_LOCAL
JCC_TOOL_ROOT:[COM]
JCC_TOOL_ROOT:[DATA]
JCC_TOOL_ROOT:[EXE]
JCC_TOOL_ROOT:[LOCAL]
JCC_TOOL_ROOT:[LOG]
JCC_TOOL_ROOT:[SHARE]
JCC_TOOL_ROOT:[SOURCE]
JCC_TOOL_LOCAL:
JCC_TOOL_ROOT:[COM]
JCC_TOOL_ROOT:[SQL]
JCC_TOOL_LOCAL:
JCC_TOOL_ROOT:[SQL]
JCC_TOOL_ROOT:[JAVA]
JCC_TOOL_ROOT:[JAVA.LIB]
JCC_TOOL_ROOT:[DP]

Plus, jcc_tool_root:[examples…]

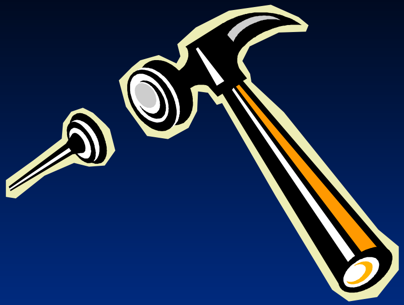
# Modify System Startup

- Edit system startup to include starting the Loader by adding a reference to the DBA startup procedures, e.g.
  - $ @DPA100:[jcc_clml.com]jcc_dba_startup
  - Repeat the line with the MV parameter, if you are starting more than one version of the Loader.
    - Note that running more than one Loader version (3.2.1, 3.1.1, 3.0, etc.) is different than running more than one Loader family.
- Manually execute the startup for the first time

# Additional Preparation

- Ensure your account has the required privileges:
  - SYSNAM to create system logical names
  - PRMMBX to create permanent mailbox
  - SYSGBL to create system global section
  - PRMGBL to create permanent global section
  - SYSLCK to create system wide locks
- Run jcc_tool_com:jcc_lml_user (supplied with the kit)
  - Afterwards, the commands do not require
    - The DCL symbol "@" for JCC procedures
    - The RUN command for JCC executables
  - All examples in the documentation and this presentation assume that this procedure has been run.

# Logical Names



"JCC_TOOL_API" = "JCC_TOOL_ROOT:[API]"
"JCC_TOOL_BATCH" = "SYS$BATCH"
"JCC_TOOL_COM" = "JCC_TOOL_LOCAL:" = "JCC_TOOL_ROOT:[COM]"
"JCC_TOOL_DATA" = "JCC_TOOL_ROOT:[DATA]"
"JCC_TOOL_DP" = "JCC_TOOL_ROOT:[DP]"
"JCC_TOOL_EXE" = "JCC_TOOL_ROOT:[EXE]"
"JCC_TOOL_JAVA" = "JCC_TOOL_ROOT:[JAVA]"
"JCC_TOOL_JAVA_LIB" = "JCC_TOOL_ROOT:[JAVA.LIB]“
"JCC_TOOL_LOCAL" = "JCC_TOOL_ROOT:[LOCAL]“
"JCC_TOOL_LOGS" = "JCC_TOOL_ROOT:[LOG]"
"JCC_TOOL_ROOT" = "DKA300:[JCCLML_03_02_04.]"
"JCC_TOOL_SHARE" = "JCC_TOOL_ROOT:[SHARE]"
"JCC_TOOL_SOURCE" = "JCC_TOOL_ROOT:[SOURCE]”

= "JCC_TOOL_LOCAL:”
= "JCC_TOOL_ROOT:[COM]“
= "JCC_TOOL_ROOT:[SQL]"

"JCC_TOOL_SQL" = "JCC_TOOL_LOCAL:”

= "JCC_TOOL_ROOT:[SQL]"

These are the logical names defined for our workshop.

# Post Installation

- Make certain that your license key is in the correct place. jcc_tool_root:[local]jcc_lml_license.com
- After installation and startup verify your installation:
  - $ jcc_logminer_loader_ivp
  - $ clean_up_ivp

Topic 3

# Loader Options

# Loader Options

- Source
- Target
- Mapping from Source to Target
- Actions to include
- Primary keys
- Filters
- Excludes
- Mode

# Source

- Source is presumed to be a production database that cannot be disturbed.
- Source is presumed to have some data that it is important to have in the target.
    - A subset of the source may be used.
    - Some materialized fields may be added.
- The data loaded represents the *changes* to the source.
    - The Data Pump can assist with initial population of the target.
    - We will talk about initial population of the data later.

# Targets

- You will choose your target depending on what you are trying to accomplish.
- Target platforms currently supported are
  - Rdb
  - Oracle
  - Tuxedo (VMS version)
  - XML to your own API
  - JDBC Class 4 Driver
    (to a wide range of tools and databases)
- Your target choice may require additional configuration.
- Special Loader provisions are added to enhance performance and robustness for each available target.

# Source to Target Mapping

- A single source database may map to multiple targets, using multiple Loader families.
- Targets for a single source may be of multiple types.
  - Through the use of multiple Loader families.
- Schemas for the targets may be different than the source.
  - A source table may map to multiple target tables, even within the same Loader session.
  - With some restrictions, multiple source tables may map to the same target table.
- With some restrictions, multiple source databases may map to a single target database.
- Tuning may be different for source and target.

# Mapping

Granville, Ohio, USA

Somewhere “down the tracks”

- As an analogy, map someone from JCC to some rail destination
  - With a suitable time change to the daily schedule
  - With a suitable clothing change from developer to consultant ☺

# Mapping

- Think of mapping as telling the Loader what to do with a change in the source.
    - Unlike our imaginative analogy of mapping from JCC to <pick your favorite spot> …
    - Loader mapping is only done when the source AIJ records a change.
- More on mapping when we talk data and database transforms.

# Actions

- The action on a row is one of these
  - Insert
  - Update
  - Delete
- LogMiner identifies M or D
- Loader does an "upsert" for LogMiner's M
  - If the row is found, it is modified.
  - If not, it is inserted.

# Selecting Actions

- The Loader can include, or not, each of the actions.
  - Insert / NoInsert
  - Update / NoUpdate
  - Delete / NoDelete
- The choice depends on what results are needed.
- The combinations that have meaning can be included by name.
  - Replication: Insert, Update, Delete
  - Rollup: Insert, Update, NoDelete
  - Audit: Insert, NoUpdate, NoDelete (all actions are interpreted as inserts)
- Other combinations of insert or noinsert, update or no update, and delete or no delete are possible, but should be carefully considered.

# Identifying the Row

- To do anything but inserts, each table in the target must have a *primary key that does not change*.
- For applications that do not have a natural key, there are two options.
  - Loader can materialize a column called originating_dbkey.
    - Identifies the physical location of the row in the source database.
    - Guaranteed to be unique.
    - But DB-keys can be reused so table cannot be reorganized.
  - DBA can add an IDENTITY attribute to the table in the source.
    - Requires changing the source DB.
    - Provides a column to replicate which can be the primary key in the target.
- What limits are implied for a table for which all columns are required to define a primary key?

# Filtering

- It is possible, but not required, to “filter” what is written to the target.
- Examples

```
!
!       Just apply a touch of filtering
!
filtermap~people_filter~                                \
     PEOPLE_ID > 10000                                  \
     and (upper(LAST_NAME) containing 'AS'              \
     or upper(FIRST_NAME) like '%ST%')
!
filtermap~details_filter~                               \
     details_id > 10000 and details_id < 30000
```

# Filtering

- A filter may be written that includes rows that meet any criteria that can be stated in SQL for a *single* row.
- The restriction to a single row is due to the way the Loader receives data from the LogMiner.
- More on filtering if we talk about Data and Data Model transforms.

# Excluding Columns

- If you will not (or, at the moment, do not) need a table in the target or for a filter,
  - Exclude the table *before* it requires any processing
  - Comment it out of the LogMiner options file that is created by the Loader kit
- If just a column should be excluded,
  - Don’t define the column with the MapColumn statements (comment it out of what is generated by procedures in the kit) *OR*
  - Use MapExclude to exclude the column
- Note that table and column definitions and excludes place some ordering requirements on the Control File.

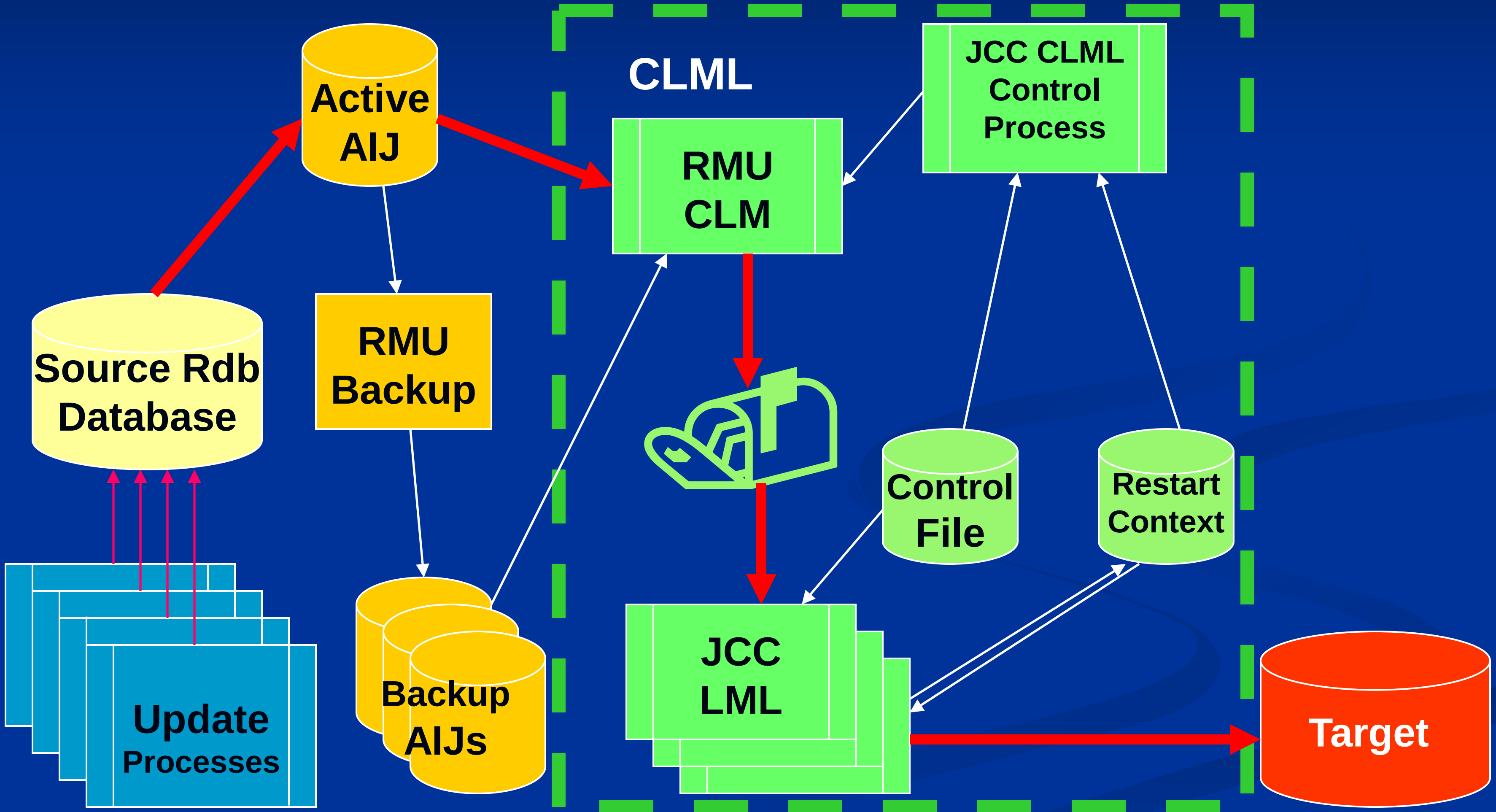
# Mapping from Source to Target

- The Control File for the Loader defines the mapping from source to target with options for
  - Action choices
  - Materialized columns
  - Filters
  - Excludes
  - Some transforms
- Successful mapping requires identification of the primary key in the target.
- We will cover more complex mappings in the Data and Database Transforms section.

# Mode

- The LogMiner can work in either of two modes
  - Static – batch-like, uses backed up AIJs
  - Continuous – runs with the Loader in near realtime, uses live or backed up AIJs
- The Loader can work in
  - Static
  - Continuous
  - Copy
- Continuous mode is the default.

# Continuous LogMiner & the Loader



# Notes on the Target – Physical Database Design

- Indexes
  - Can be different from indexes on the source.
  - Should include an index on the primary key.
  - Should not include more than one unique index per table.
- Do NOT add constraints.
  - Order of rows within a transaction is not guaranteed.
  - If a child row is written before a parent row, some constraints might fail.
- Do NOT repeat triggers that will have fired in the source.
  - If – for example – the source has a trigger that increments a value every time some other column is updated, having the same trigger in the target will cause the count to be incremented twice for each update.
  - Other triggers – that have not already fired in the source – may be justified.

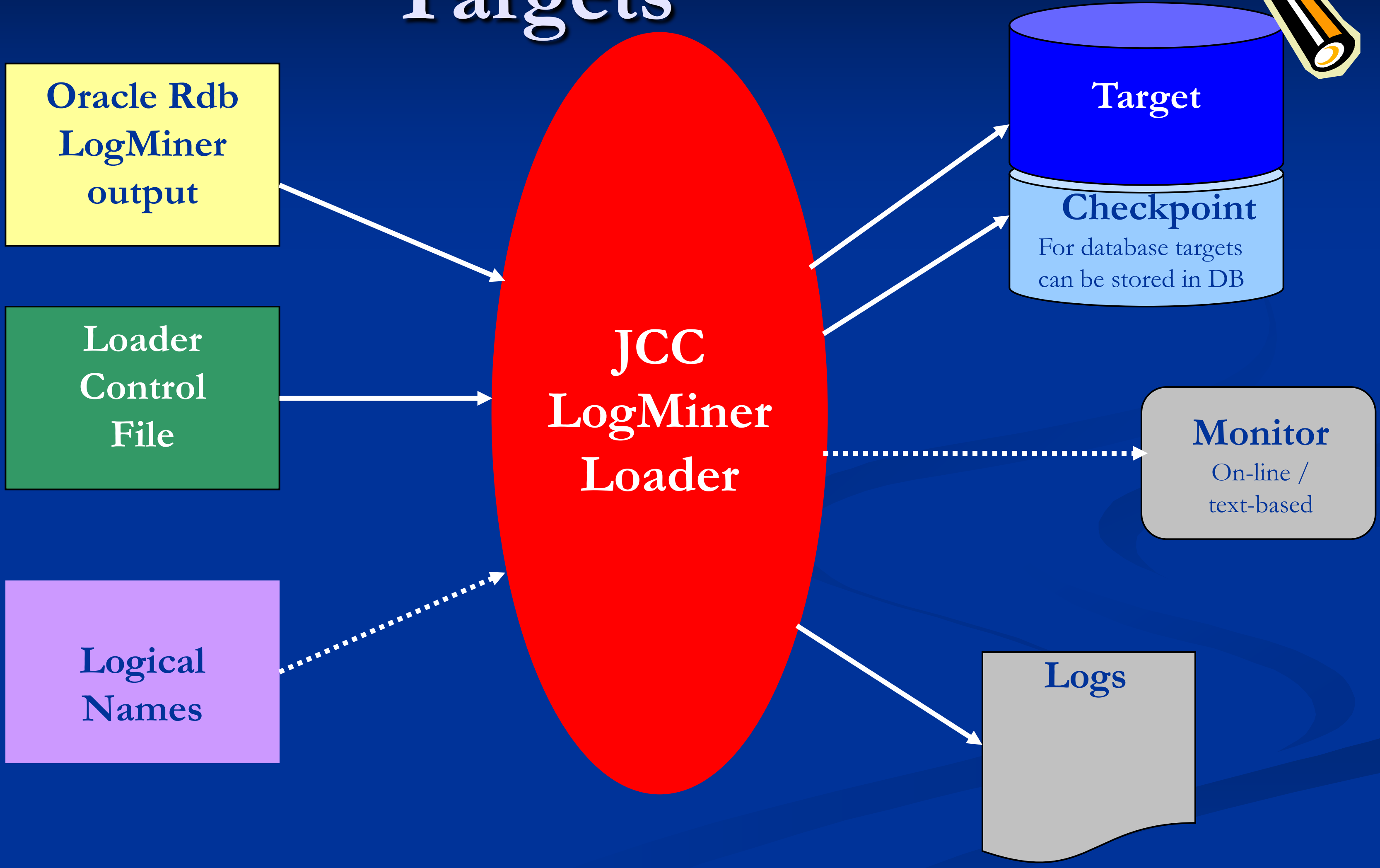
# Workshop

- Pardon the interrupt …
- We have much more to discuss on application options, but …
- Let’s get the source and target ready.

# Workshop

- We have
  - Downloaded the kit, including documentation
  - Gotten a license
  - Installed the Loader
- To use the Loader, you will also …
  - Prepare your source
  - Configure the Loader
  - Prepare your target(s)
  - Run the Loader
- In this section, we will
  - Prepare the source
  - Follow the special procedures for the target types that interest you

# Loader Input & Output: Targets



Oracle Rdb LogMiner output

Loader Control File

Logical Names

JCC LogMiner Loader

Target

Checkpoint

For database targets can be stored in DB

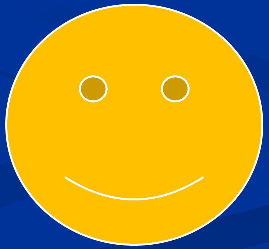
Monitor

On-line / text-based

Logs

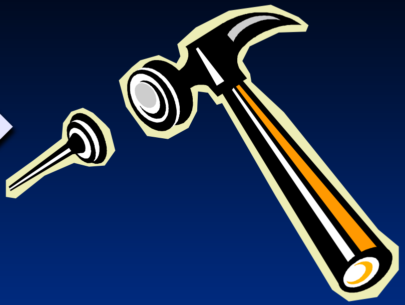
# Prepare Your Source

- For your Rdb source
  - Turn on journaling … *if it isn't already on*
  - Turn on LogMiner
  - Backup



We can update this.

# Establish Your Target →



- Choose the target.
  - Rdb
  - Oracle
  - XML and your API
  - Tuxedo
  - JDBC Class 4 driver and its target
- Complete the steps to ready the target.
- Load the initial data.
  - Data Pump
  - Other
- Provide additional target specific information with the Loader Control File.

# Rdb Targets →

- Create the metadata for the target db.
- Add Loader specific metadata.
    - Create the highwater table.
    - Add dbkey columns and views as necessary
    - Add materialized fields, if desired
- Do physical db work, as required
- Load the initial data (discussed later)
- Link to the Rdb source

# Rdb Target and Physical DB Design

- Add indexes.
  - For the primary keys.
  - Add other indexes as desired.
    - Can be different from the source.
    - Only one unique index per table.
- Do NOT add constraints.
  - Order of rows within a transaction is not guaranteed.
  - If a child row is written before a parent row, some constraints might fail.
- Do NOT repeat triggers that will have fired in the source.
  - If – for example – the source has a trigger that increments a value every time some other column is updated, having the same trigger in the target will cause the count to be incremented twice for each update.
  - Other triggers – that have not already fired in the source – may be justified.

# Add the High Water Table

- When using Rdb for the target, the Loader can store the highwater table in the database.
  - Using the database supports transaction control.
  - Using a local file is possible, but may permit some repetitive sending on restart.
- Script is: `jcc_tool_sql:create_logminer_highwater.sql`
- You should edit the highwater table script to match your target database standards and needs.
  - The highwater table is modified at least once per Loader checkpoint. (How often that is varies with your setting.)

# Oracle →

- Get the necessary Oracle software.
- Define logical names to point to additional images.
- Define JCC functions and names table in the source database.
- Create the metadata for the target Oracle db.
- Add Loader specific metadata.
  - Create the highwater table.
  - Add dbkey columns and views as necessary
  - Add materialized fields, if desired
- Do Physical db work, as required
  - Do not mimic the triggers and constraints of the source
  - Add indexes
- Load the initial data (discussed later)

# Oracle Targets Require Oracle

- The Loader uses the native Oracle call interface [OCI] to communicate with Oracle targets
  - Requires a traditional Oracle installation on OpenVMS
  - Tested with many Oracle versions.  You pick which you wish to install.
    - Version 9.2 requires kits later than 9.2.4
    - Version 3.0 and later of the Loader tested with 10gR1, 10gR2, 9.2 (11 is not available on OpenVMS, yet.)
  - Obtain from Oracle
    - OTN or MetaLink

# Special Startup for Oracle Target

- For Oracle, additional logical names are required to find additional images.
  - $jcc_lml_oracle_user <version> - <full file specification for orauser.com>
  - $jcc_lml_oracle_user 10.1 - disk$software:[oracle10g.db]orauser.com
- For Oracle targets, additional assistance is available. The next several slides show how to create the Oracle targets with procedures in the kit.

# Prepare Source for Oracle Target

- Add JCC functions to the source database

```
SQL> attach 'filename source_db';
SQL> @jcc_tool_sql:vms_functions
%SQL-F-FCNNOTDEF, function SET_SYMBOL is not defined
%SQL-F-MODNOTDEF, module GET_SYMBOL_DATE is not defined
%SQL-F-PROCNOTDEF, procedure GET_SYMBOL is not defined
%SQL-F-MODNOTDEF, module CVT_DBKEY is not defined
%SQL-F-MODNOTDEF, module CVT_DBKEY is not defined
%SQL-F-PROCNOTDEF, procedure EXT_CVT_DBKEY_TO_QUAD is not defined
%SQL-F-PROCNOTDEF, procedure EXT_CVT_QUAD_TO_DBKEY is not defined
SQL>commit;
```

# Prepare Source for Oracle Target

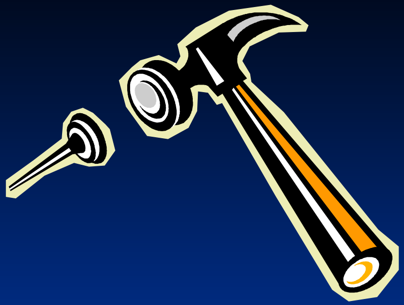
- Create the JCC Names table in the source database
  - Used to resolve Rdb names that are too long for Oracle.
  - Requires a storage area called DEFAULT_AREA
  - You may edit this procedure to substitute your own area name

```
SQL> attach 'filename source_db';
 SQL> @jcc_tool_sql:jcc_generate_oracle_names_tbl;
 SQL> commit;
```

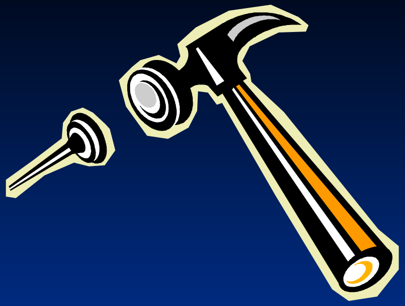
- Generate the default names to be used in the Oracle database:

```
SQL> @jcc_tool_sql:jcc_generate_oracle_names;
```

```
$ table_name :== "EMPLOYEES"
$ SQL
SQL> attach 'filename source_db';
SQL>
@jcc_tool_sql:jcc_generate_oracle_table_sql
set echo on;
-- Generating for table EMPLOYEES
drop table   EMPLOYEES  ;
create table EMPLOYEES
  (EMPLOYEE_ID      CHAR(5 )
  ,LAST_NAME        CHAR(14)
  ,FIRST_NAME       CHAR(10)
  ,MIDDLE_INITIAL   CHAR(1 )
  ,ADDRESS_DATA_1   CHAR(25)
  ,ADDRESS_DATA_2   CHAR(20)
  ,CITY             CHAR(20)
  ,STATE            CHAR(2 )
  ,POSTAL_CODE      CHAR(5 )
  ,SEX              CHAR(1 )
  ,BIRTHDAY         DATE
  ,STATUS_CODE      CHAR(1 )
  );
```



```
pctfree 10
tablespace users
-- There are 1000 rows in the tab
--each row requires an estimated 126 bytes
--allowing 58 rows per block with
-- pctfree of 0.10
-- Requiring 18 blocks of 8192 bytes each
storage (initial     144 K
         next      36 K
      maxextents  unlimited
      pctincrease 25)
   ;
commit work;
```



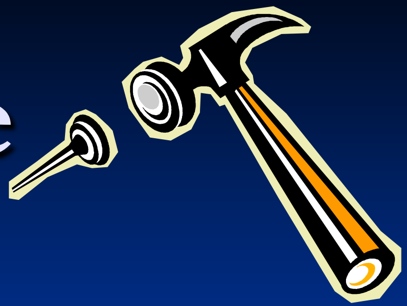
# Generate Oracle Indexes

- At least the primary key index is required in the Oracle target.
- There should be no more than one unique index per table.
- A Loader script can be used to generate target index definitions for all indexes in the source.

```
SQL> @jcc_tool_sql:jcc_generate_oracle_indexes
-- Indexes for table EMPLOYEES
set echo on;
drop index EMPLOYEES_HASH;
create index  EMPLOYEES_HASH
  on EMPLOYEES
    (EMPLOYEE_ID)
  pctfree 5
  tablespace indx
-- There are 1000 rows in the table
-- and each row requires an estimated 13 bytes in the index
-- allowing 593 entries per block with pctfree of 0.05
-- Requiring 3   blocks of 8192 bytes each
  storage (initial       24                  K
           next          6                   K
           maxextents     unlimited
           pctincrease    25)
  compute statistics
  ;
commit work;
```

```
drop index EMP_LAST_NAME;
 create index  EMP_LAST_NAME
  on EMPLOYEES
    (LAST_NAME  )
  pctfree 5
  tablespace indx
-- There are 1000 rows in the table
-- and each row requires an estimated 22 bytes in the
   index
-- allowing 350 entries per block with pctfree of 0.05
-- Requiring 4 blocks of 8192 bytes each
  storage (initial       32    K
           next          8     K
           maxextents     unlimited
           pctincrease    25)
  compute statistics;
commit work;
exit;
```

# Add the High Water Table



- When using Oracle for the target, the Loader can store the highwater table in the database.
  - Using the database supports transaction control.
  - Using a local file is possible, but may permit some repetitive sending on restart.
- Script is:
  `jcc_tool_sql:create_logminer_highwater_oracle.sql`
- You should edit the highwater table script to match your target database standards and needs.
  - The highwater table is modified at least once per Loader checkpoint. (How often that is varies with your setting.)

```
create table logminer_highwater
        (trans_name             char(32)
        ,start_trans_time       date
        ,commit_time            date
        ,update_count           number
        ,commit_interval        number
        ,successful_completion  char(1)
        ,input_name             varchar(255)
        ,thread_number          number
        ,data_version           number
        ,highwater_data         long raw
        ,primary key (trans_name,thread_number) not
  deferrable)
        pctfree 10
        tablespace jcclml
        storage (initial        8K
                 next           8K
                 maxextents     unlimited
                 pctincrease    25) ;
```

# JDBC →

- Get the necessary software.
- Define logical names to point to additional images.
- Create the metadata for the target db.
- Add Loader specific metadata.
  - (A highwater table is not supported with JDBC targets.  A file is used.)
  - Add dbkey columns and views as necessary
  - Add materialized fields, if desired
- Do Physical db work, as required
  - Do not mimic the triggers and constraints of the source
  - Add indexes (one for the primary key and no other unique indexes)
- Load the initial data (discussed later)

# JDBC Targets Require JAVA

- Download from the HP Web site.
- Loader is tested with multiple versions, including:
  - Alpha V1.4.2-2, V5.0-4
  - Itanium V1.4.2-2, V5.0-3, V6.0
- Use latest version
  - May require a number of patches for OpenVMS and TCPIP

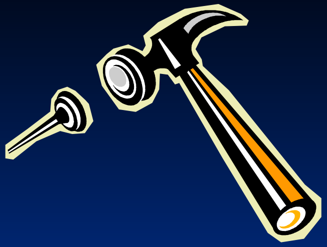
# Special Startup for JDBC Target

- For JDBC targets, additional logical names are required to find additional images.
- For JDBC targets, execute
  - $ jcc_lml_jdbc_user <Java version> - <Java setup procedure> < Java setting>
  - $ jcc_lml_jdbc_user 1.5.0 - sys$common:[java$150.com]java$150_setup.com

# Generate Target Metadata

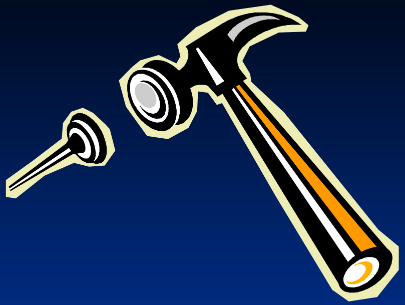
- With the JDBC target for the Loader, a large number of database products are the possible target of JDBC.
  - We will not be precise in referring to the JDBC *target* versus the ultimate data store.
  - For the Workshop, we will use SQL Server for the ultimate data store.
- To generate ANSI SQL table definitions, you can use
  $ rmu/extract/lang=ansi_sql/item=(table,noconstraint)/output=<target>.sql <source db>
- SQL Server Metadata syntax for tables is essentially the same as for Oracle.
  - Except that there is no use of storage clauses.

# SQL Server Metadata



```
drop table   EMPLOYEES  ;
create table EMPLOYEES
  (EMPLOYEE_ID      CHAR(5 )
  ,LAST_NAME        CHAR(14)
  ,FIRST_NAME       CHAR(10)
  ,MIDDLE_INITIAL   CHAR(1 )
  ,ADDRESS_DATA_1   CHAR(25)
  ,ADDRESS_DATA_2   CHAR(20)
  ,CITY             CHAR(20)
  ,STATE            CHAR(2 )
  ,POSTAL_CODE      CHAR(5 )
  ,SEX              CHAR(1 )
  ,BIRTHDAY         DATETIME
  ,STATUS_CODE      CHAR(1 )
  )
commit
```

# Add to the Metadata



- Add Loader specific columns
  - Dbkeys, if necessary
  - Materialized columns, if desired
- Primary key indexes should be added in the target:

```
create unique index EMPLOYEES_HASH
on EMPLOYEES
  (EMPLOYEE_ID)
```

- Other indexes may be needed, as well, but only one unique index per table, please.
- Do not mimic the triggers and constraints of the source.

# High Water File with JDBC

- With a JDBC target,
  - Database tables cannot be used for checkpointing.
  - A local file for checkpointing the highwater context is required.
  - Checkpoint keyword is defined with LML_internal, as `checkpoint~1~LML_internal`
- File name is specified in the Loader Control File
  - May specify a logical name to be translated at run time
  - Default file type is ".JCCLML-CHECKPOINT"
  - Example

```
jdbc_checkpoint_dir:example-jdbc.jcclml-checkpoint
```

# Load the Initial Data

- You will need to initialize the data in your target.
  - LogMiner and Loader work with data *changes* made to the source.
  - Rows are not processed, if not written to the AIJ.
  - You must start at some known point.
- There are many ways to load the initial data.
  - JCC LogMiner Loader customers often prefer to use a companion tool that is provided with the Loader.
  - That tool is the Data Pump.

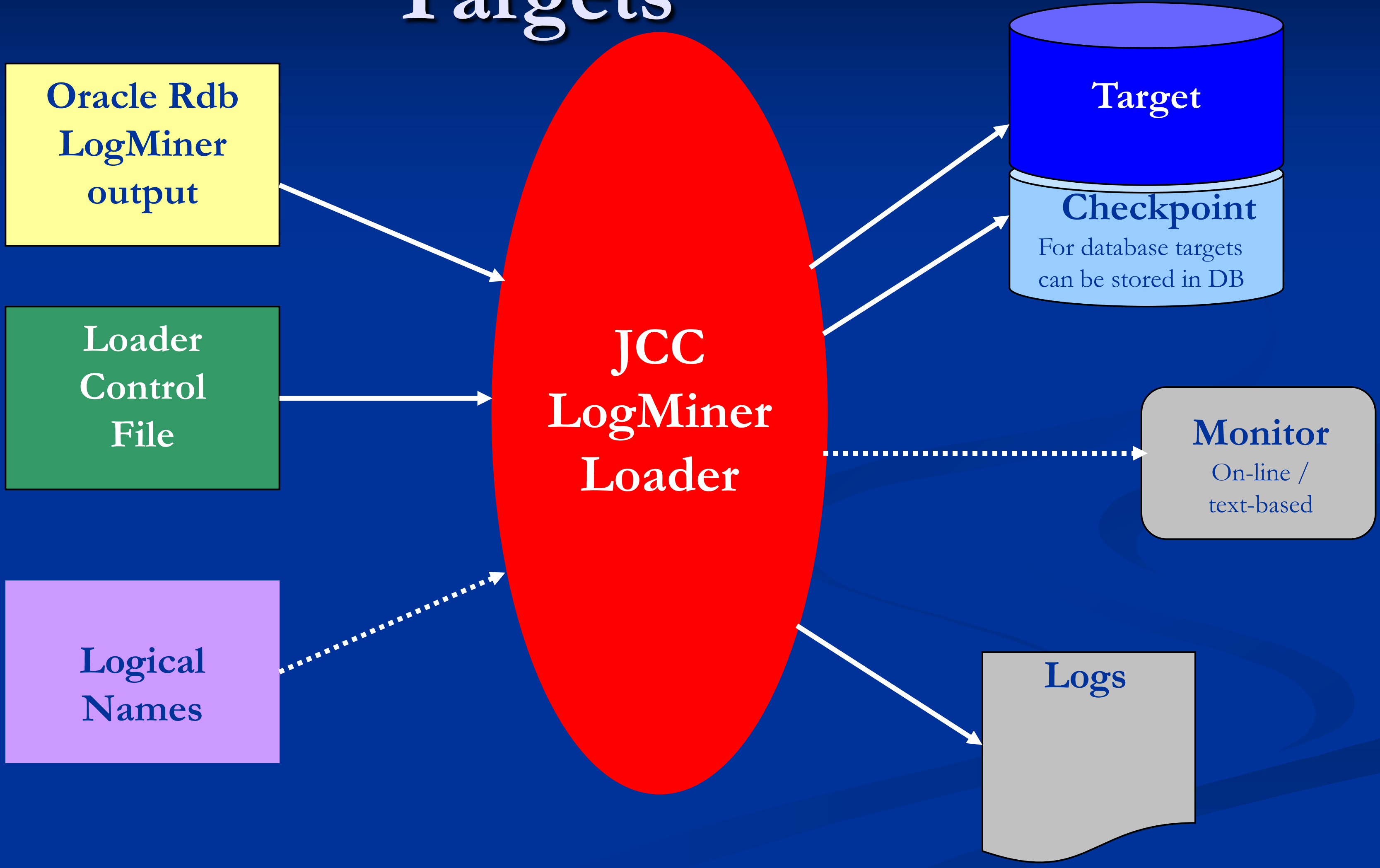
Topic 4

# Configuration

# Status Check

- We have
  - Installed the Loader
  - Applied the license key
  - Modified system startup
  - Run jcc_lml_user
  - Prepared our target(s)
  - Added the highwater table (for Oracle or Rdb targets)
- We still need to
  - Build the LogMiner Options File
  - Build the Loader Control File
  - Establish some logical names
  - Enable LogMiner on the source
  - Run the Loader

# Loader Input & Output: Targets



# Options & Control →

- The LogMiner requires an "Options File."
- The Loader requires a "Control File."
- Each is detailed.
- The Loader kit includes procedures to aid their creation.

# Editing the Options File

- You may edit the options file to remove tables in which you have no interest.
- Aside from *commenting out* lines, do not change any other text in this file.
  - Target filenames are specified at run time as logical names.
  - Changing them can cause tables to be missed.

# Build the Control File →

- For ease of maintenance, segment the Control File into parts. One suitable division is
  - Master
  - Target definition
  - Metadata of the source
  - Virtual (materialized) columns and tables
  - Excludes
  - Filters
  - Target metadata and mapping
  - Performance tuning and statistics related keywords
- Use the include_file keyword for each segmented component of the Control File.
  - Include components in the order listed above or consult the documentation for order requirements.

# Requirements, Recommendations, and Tailoring

- Most keywords have defaults that work for most circumstances.
- Some few are required.
- Others are recommended.
- The remainder are available to tailor things to your environment, change the statistics and logging, or tune performance.

TIP

Don't get lost in the possibilities. Use defaults to get started.

# Master Control File

- Include these keywords
  - Loadername~<the name that you want to use>
    - Must be first, unless
      - Loadername is defined with the logical name jcc_logminer_loader_name
      - Or the default of "JCCLML" is to be used
    - Make it unique in the first eleven characters
  - Logging~initialization
    - Echoes the Control File in the log.
    - Recommended, not required.
  - Keywords that provide Loader session context
  - Include_file~<file name>
  - Include_file~<file name>

# Target Definition – Output Keyword

- The *optional* Output keyword specifies the target
  - The target (API|FILE|JDBC|OCI|Rdb|TUXEDO)
  - SYNCH|ASYNCH (*optional*) Defaults to SYNCH.
  - The name of the target (*optional*)
    - For Rdb, the name of the database
    - For OCI (Oracle), the TNSnames entry
    - For API, shareable image name
    - For FILE, the file or mailbox
    - For JDBC, the driver specific reference to the database
    - For Tuxedo, the qspace used for tpenqueue calls
  - Message contents (*optional*)
    - RECORD is required for Rdb and OCI targets (default)
    - TRANSACTION is useful for XML
  - Output conversion *optional* comma separated list …

# Output Keyword (con't.)

- Output keyword parameters continued …
  - Output conversion
    - TEXT (FILE, API)
    - XML (FILE, API)
    - TRIM (Oracle, FILE, API, and Tuxedo)
    - ESCAPE (FILE, API)
    - HEADER (FILE, API, Tuxedo)
    - Perhaps, others in the future …
  - Obviously, in most environments, TRIM is the only conversion option that is interesting.

# Additional Target Definition

- Output keyword, if defined, must be defined before any metadata.
- If the Output keyword is not defined, by default, the target is defined to be Rdb.
- For JDBC, add JDBC keyword to provide
  - Vendor specific driver name
  - Connect string as specified by the vendor
  - Classpaths are specified in the UNIX style
- Add Validation~<username>~<password>
  - For OCI (Oracle) or JDBC
  - For Rdb to supply credentials over TCP/IP.
- For API, there are other keywords.

# Source Metadata Control File

- The kit includes a procedure to define the metadata of the source
  - jcc_lml_create_control_file <db name> [option 1 [option 2]option3]]]
- The options provide support for more complex target schemas.
  - TABLE is the default (for upward compatibility of older installations).
  - MAPTABLE is a basis for more complex schemas.
  - ALL (if listed as option 1) provides the Table definitions and ONE set of Maptable definitions.
  - DATEFILTER contains FilterMap statements to exclude invalid dates.
- Examples
  - JCC_LML_CREATE_CONTROL_FILE <db> Table
  - JCC_LML_CREATE_CONTROL_FILE <db> MapTable DateFilter
  - JCC_LML_CREATE_CONTROL_FILE <db> ALL

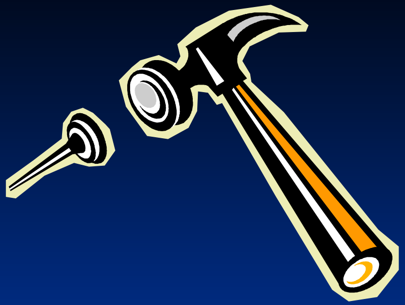
# Tailoring the Metadata Control File

- You may need to edit the default definitions of
  - Primary Keys
    - A primary key is required to identify row to be updated or deleted.
      - One or more columns which do not change.
      - OR the originating DBKeys (with certain restrictions)
      - OR adding the IDENTITY attribute to the source table
    - The procedure “guesses” and uses
      - Primary key constraints
      - Unique indexes, if no primary key constraint
      - Assumption of DBKey mechanism, if nothing else has worked
  - Action to take for each table
    - Possible actions are combinations of
      - [no]insert
      - [no]update
      - [no]delete
      - Rollup (equivalent to INSERT, UPDATE, NODELETE)
      - Replicate (equivalent to INSERT, UPDATE, DELETE)
      - Audit (equivalent to INSERT,NOUPDATE,NODELETE)
    - The default is Replicate (insert,update,delete).

# Workshop

- Let's build a few things …
  - LogMiner Options File
  - Loader Control File
  - Metadata definition

# Build the LogMiner Options File →



```
$ JCC_CREATE_LOG_MINER_OPT_FILE source_db

Directory JCC_ROOT:[JEFF.LOADER]

MF_PERSONNEL_LM_UNL.OPT;1        2/513     20-APR-2006 15:09:42.31

Total of 1 file, 2/513 blocks.
Change mode can be entered only from a terminal
227 substitutions
JCC_ROOT:[JEFF.LOADER]MF_PERSONNEL_LM_UNL.OPT;2 15 lines

Directory JCC_ROOT:[JEFF.LOADER]

MF_PERSONNEL_LM_UNL.OPT;2        2/3       20-APR-2006 15:09:42.41
MF_PERSONNEL_LM_UNL.OPT;1        2/513     20-APR-2006 15:09:42.31
```

# LogMiner Options File

```
$ type MF_PERSONNEL_LM_UNL.OPT;2
!
!  Continuous Logminer Options file
!  Generated at 2006-04-20 15:09:42
!
table=CANDIDATES,output=rdb_logminer_output_file
table=COLLEGES,output=rdb_logminer_output_file
table=DEGREES,output=rdb_logminer_output_file
table=DEPARTMENTS,output=rdb_logminer_output_file
table=EMPLOYEES,output=rdb_logminer_output_file
table=JCC_RDB_TO_ORACLE_NAME,output=rdb_logminer_output_file
table=JOBS,output=rdb_logminer_output_file
table=JOB_HISTORY,output=rdb_logminer_output_file
table=RESUMES,output=rdb_logminer_output_file
table=SALARY_HISTORY,output=rdb_logminer_output_file
table=WORK_STATUS,output=rdb_logminer_output_file
```

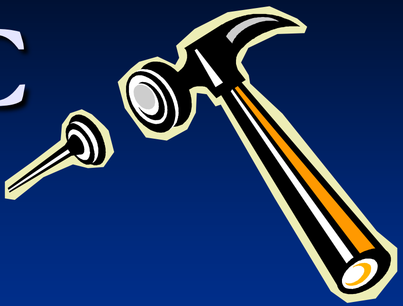
# Master Control File - Rdb

```
! Loader name is defined via the logical name
!
logging~initialization
logging~statistics~runtime,timer
output~rdb~synch~target_db~record
output_failure~1~1000
parallel~1~4~constrained
! Include metadata definitions
include_file~demo_metadata
```

# Master Control File - Oracle

```
! Loader name is defined via the logical name
logging~initialization
logging~statistics~runtime,timer,commit
sort~by_record
!This is an Oracle target which checkpoints in the target
checkpoint~1
input~ipc
input_failure~5
output_failure~1~1000
parallel~4~32~constrained
thread~startup~0.25
thread~shutdown~0.5
include_file~loader_regression_test_control_ora_interface
include_file~loader_regression_test_metadata_ora
include_file~loader_regression_test_excludes_ora
include_file~loader_regression_test_includes_ora
```

# Master Control File - JDBC



```
! Loader name is defined via the logical name
logging~initialization
logging~statistics~runtime,timer,commit
logging~input~ignore_unknown_tables,log_restart
sort~by_record
input~ipc
checkpoint~1~lml_internal~asynch~loader_regress_test_sqlsrv_jdbc_chkpt
input_failure~10
output_failure~1~10
parallel~1~5~constrained
thread~startup~0.1
thread~shutdown~1
include_file~loader_regression_test_jdbc_interface
include_file~loader_regression_test_metadata
include_file~loader_regression_test_excludes
include_file~loader_regression_test_includes
include_file~loader_regression_test_virtual
```

# Target Definition - Rdb

- Nothing is required, as this is the default.
- Equivalent to any of these:

```
Output~Rdb~synch~<target db>~record
Output~Rdb
Output~Rdb~synch
```

# Target Definition - Oracle

```
!
!      Define the Oracle interface.
!
output~oci~synch~jccdb~record~trim
validation~atlas~<password>
```

# Target Definition - JDBC

```
output~jdbc~synch~                                    \
jdbc:Microsoft:sqlserver://kong:1433;DatabaseName=RegressionTest
validation~RegTest~<password>
jdbc~driver~com.microsoft.jdbc.sqlserver.SQLServerDriver
jdbc~connect~                                         \
jdbc:Microsoft:sqlserver://kong:1433;DatabaseName=RegressionTest
jdbc~classpath~/jcc_tool_java_lib/mssqlserver.jar
jdbc~classpath~/jcc_tool_java_lib/msbase.jar
jdbc~classpath~/jcc_tool_java_lib/msutil.jar
```

# Metadata Generation

```
colchi > jcc_lml_create_control_file source_db all
    1       !
319 substitutions
1 substitution
JCC_ROOT:[DEMO]MF_PERSONNEL_TABLE.INI;2 74 lines
%DELETE-I-FILDEL, JCC_ROOT:[DEMO]MF_PERSONNEL_TABLE.INI;1
deleted (513 blocks)
    1       !
111 substitutions
JCC_ROOT:[DEMO]MF_PERSONNEL_MAPTABLE.INI;2 103 lines
%DELETE-I-FILDEL, JCC_ROOT:[DEMO]MF_PERSONNEL_MAPTABLE.INI;1
deleted (513 blocks)
```

# Metadata File Generation

```
$ type MF_PERSONNEL.INI
!
!GeneratedMetadataControlFile
!
TABLE~CANDIDATES~1~Replicate
!
!Table_"CANDIDATES"_has_neither_a_primary_key_nor_a_unique_index.
!
                                                                  o
                                                                  o
                                                                  o
TABLE~EMPLOYEES~1~Replicate
PRIMARY KEY~EMPLOYEES~EMPLOYEE_ID~1~5~0~14~0
COLUMN~EMPLOYEES~LAST_NAME~2~14~0~14~0
COLUMN~EMPLOYEES~FIRST_NAME~3~10~0~14~0
COLUMN~EMPLOYEES~MIDDLE_INITIAL~4~1~0~14~0
COLUMN~EMPLOYEES~ADDRESS_DATA_1~5~25~0~14~0
COLUMN~EMPLOYEES~ADDRESS_DATA_2~6~20~0~14~0
COLUMN~EMPLOYEES~CITY~7~20~0~14~0
COLUMN~EMPLOYEES~STATE~8~2~0~14~0
COLUMN~EMPLOYEES~POSTAL_CODE~9~5~0~14~0
COLUMN~EMPLOYEES~SEX~10~1~0~14~0
COLUMN~EMPLOYEES~BIRTHDAY~11~8~0~35~0
COLUMN~EMPLOYEES~STATUS_CODE~12~1~0~14~0
```

Topic 5

# More Configuration

# Metadata File and Primary Keys

- Procedure looks for
  - Primary key constraint
  - Unique index with the fewest number of columns
- If neither is found, assumes DB-key
- Note:  Some editing of the output may be required.

# Source to Target Mapping

- *Optional* keywords for modifying the mapping of source columns/tables to target columns/tables
  - MapColumn
    - Rename
    - Value if NULL
  - MapKey
  - MapTable
  - VirtualColumn

# Syntax of Map … Keywords

- Syntax in the documentation is shown as
  - MapTable~<source table name>~<map table name> \
    [,<target table rename>][~actions>[~options>]]
  - MapColumn~<map table name>~<source column name> \
    [target column rename][~<value if null>]
  - MapKey~<map table name>~<target column name> \
    [,<target column name>[,<target column name>[…]]]
  - For virtual columns (materialized data)
    - MapColumn~<map table name>~ \
      <source (virtual) column name|output column name> \
      [,<target column rename>][~<value if null>]
    - VirtualColumn~<table name>~<virtual column name> \
      [,<output column name>]

# Sample Mapping

```
!
!       Map the people table for auditing
!
MapTable~people~people_filter~insert,noupdate,nodelete
MapColumn~people_filter~people_id
MapColumn~people_filter~progname
MapColumn~people_filter~progindex
MapColumn~people_filter~seqnum
MapColumn~people_filter~last_name
MapColumn~people_filter~first_name
MapColumn~people_filter~age
MapColumn~people_filter~since_year
MapColumn~people_filter~city
MapColumn~people_filter~state
MapColumn~people_filter~huge_text
! Add materialized data
MapColumn~people_filter~loader_sequence_number
MapColumn~people_filter~originating_dbkey
MapColumn~people_filter~loadername
MapColumn~people_filter~loader_version
MapColumn~people_filter~loader_link_date_time
```

You may notice that this includes no MapKey statement. It is an audit table. Does that explain it?

# Exclude

- It is possible, but not required, to exclude columns or tables from what is written to the target.
- The keyword Exclude
  - Is available for backward compatibility.
  - Excludes the table or column from all targets.
- Use MapExclude to exclude column from specific targets.
- If you are not going to use a table at all, it is more efficient to exclude it in the LogMiner options file.

# Syntax for Exclude and MapExclude

- Exclude~<table>[~<column>]
- MapExclude~<map table name>~ \
<target column name>

# Exclude Tables and Columns

```
!
!       Tables excluded for all targets
!
exclude~degrees
exclude~resumes
exclude~salary_history
!
!       Omit sending some columns to any target
!
exclude~employees~sex
exclude~employees~status_code
!
! Exclude tables and columns from a single target
!
mapexclude~commercial~customer_type
!
! Customer_type may have been included, initially,
! to provide an option for filtering.
```

# Filters

- It is possible, but not required, to “filter” what is written to the target.
- The keyword Filter is available for backward compatibility.
- Use FilterMap.
  - FilterMap is built on the MapTable syntax.
  - FilterMap supports SQL predicates for a specific row.
  - FilterMap does not support SQL on multiple tables.

# Syntax and Example for FilterMap

- FilterMap~<map table name>~[where]      \
  <sql restriction>

```
!
!       Just apply a touch of filtering
!
filtermap~people_filter~                              \
      PEOPLE_ID > 10000                               \
      and (upper(LAST_NAME) containing 'AS'           \
      or upper(FIRST_NAME) like '%ST%')
!
filtermap~details_filter~                             \
      details_id > 10000 and details_id < 30000
```

# FilterMap Architectural Limits

- Because some ask, let us pause for a digression.
- FilterMap does not support SQL on multiple tables because
  - There is no guarantee that all of the tables that you might reference are included in a specific transaction (or checkpoint).
  - The Loader does not have access to data that is not in the transaction (or checkpoint).
  - There is a temporal nature of data that can cause unexpected results if we add to the Loader architecture an option of reading data from the source or filter database.

# Establish Some Logical Names

- The kit includes a facility for maintaining logical names
- Logical names control many aspects of the Loader
  - JCC_logminer_loader_hw_response
  - JCC_aij_backup_spec (use wildcards to specify where backup journals are)
  - JCC_add_clm_shared_read = true, if running multiple CLM processes to read from the same database
  - Logging and statistics logical names
  - Performance control
    - JCC_add_clm_sortwork_files
    - JCC_clml_heartbeat_enable
    - JCC_clml_heartbeat_interval
    - JCC_logminer_loader_lock_threshold
  - Details
  - Target specific logical names
  - Logical names to change default names used

# Get Ready

- The workshop will
  - Prepare for Continuous
  - Assume that the source db uses a recent version of Rdb
- Define a logical name to avoid OPCOM stalls on the first run.
  - jcc_logminer_loader_hw_response = create | quit

# Get Set

- Journaling must be turned on.
- AIJ files must remain available until mined.
- JCC recommends
  - Establish a Quiet Point Backup
  - Backup AIJ and remove this and previous AIJs from the directories that will be used for AIJ files to be mined.
  - Enable LogMiner

# Go

- Initiate the continuous session with
  - jcc_tool_com:JCC_run_clm_lml.com
    <source db name>
    <LogMiner options file>
    <Loader Control File>
  - Additional parameters are *optional* and not desirable under *most* circumstances.

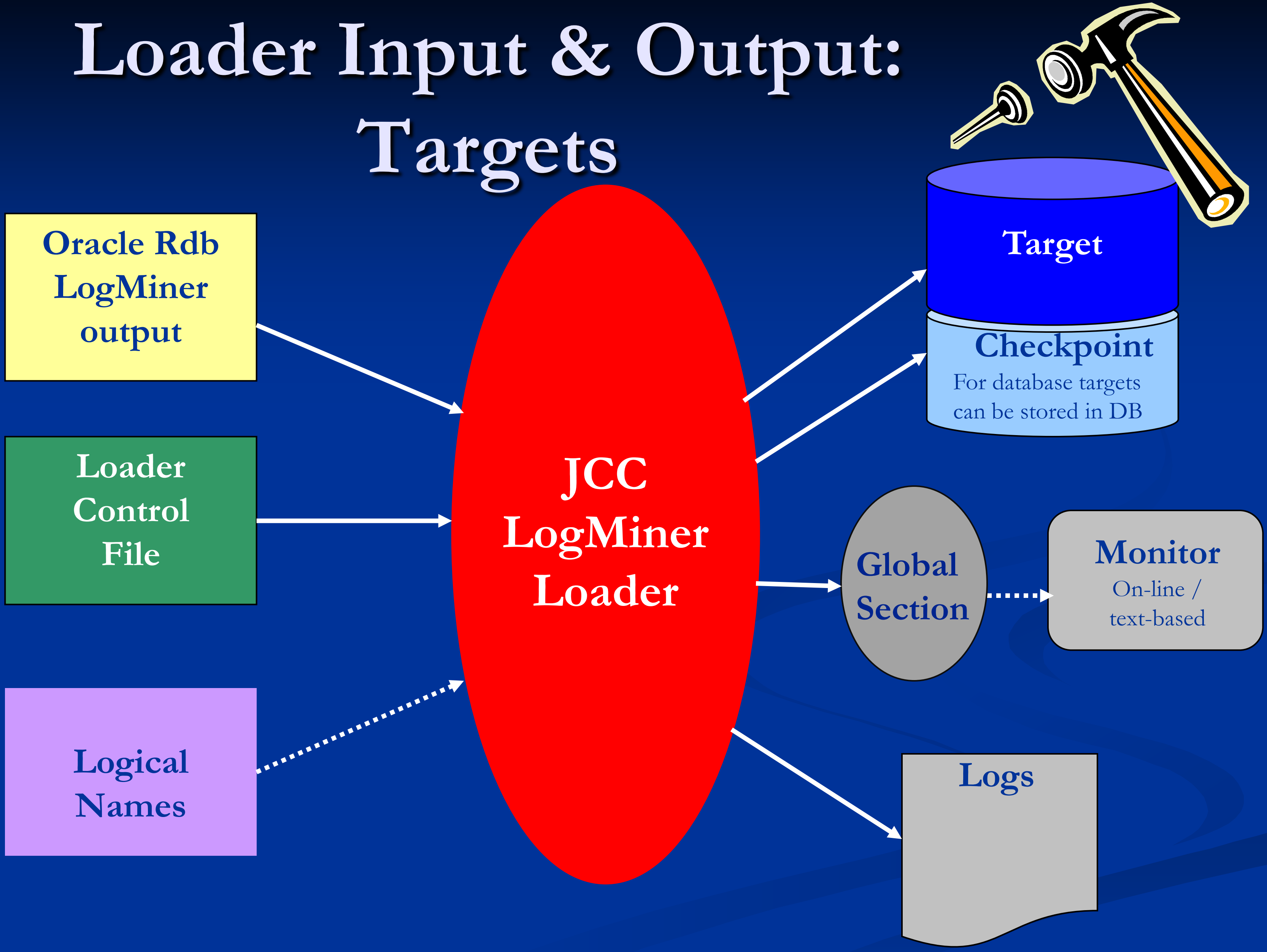
# Wait!

- Before we Run the Loader, let’s jump ahead a bit.
- How do we see what the Loader is doing?

Topic 6

# Monitoring – Sneak Preview

# Loader Input & Output: Targets



# Monitoring Your System →

- Latency alarms
- Statistics Monitor
- Logging
- Additional information from materialized data
- Locking diagnostics

# Statistics Monitor

-
  ```
  JCC_LML_statistics <Loadername> -
  [refresh seconds][report type] -
  [tardy threshold[operator class]]
  ```
  - Requires Loader name
  - Refresh rate of 3 seconds is the default
  - Report types are
    - F full
    - D detailed
    - B brief
    - C CSV (comma separated values)
    - T T4
  - Uses vary
    - Some report types create data for OpenVMS tools
    - Some report types are suitable for online display
    - Some report types create data that can be loaded into a db

# Example of a Detail Report

```
Rate:   6.00                       REGTESTOPO             2-OCT-2007 16:32:30.00
================================================================================
   Input:  2-OCT-2007 16:20:23.53                Output:  2-OCT-2007 16:20:22.69
--[Trail:   12.1m]---------------             ---[Trail:   12.1m]---------------
Transactions                22654             Checkpoints                   4888
Records                    276850                Timeout                       7
   Modify                  253074                BufferLimit(   170)         100
   Delete                    1123                NoWork                        2
   Commit                   22653             Records(   2)               253971
 Discarded                                       Messages(  N/A )            N/A
   Filtered                     0                Filtered                      0
   Excluded                     0             Failure                          0
   Unknown                      0             Timeout                          0
   Restart                      0        - Current ---------------- Ave/Second -
   NoWork                    1657        Checkpoints            43          7.17
   Heartbeat                    5        Records              1765        294.17
Timeout                         0        Rate                67.87%
--- Restart Context ------               - Latency(sec) ------ LML detail ------
M|AIJ#                  4 |              CLM   12.1m |  Inpt  26.0%  Cnvt   6.4%
Q|VBN              447834 |              ------------   Sort   0.1%  Trgt  55.5%
P|TSN               44289 |              LML    0.64    Sync  12.1%  Ckpt   0.0%
 CTSN               44303 |        - Loaders - 01234567 ------------------------
  LSN               20952 |        - States  - >>>R>R<R
```

Oracle target in regression test. Target machine is pegged.

# Thread States

- "Threads" and parallel processing are a performance tuning option to be discussed later.
- The example shows 8 threads (0-7).
- The symbols below each shows its state.
  - R means that the thread is waiting for the read lock.
  - < means that the thread is currently reading from the input mailbox.
  - > means that the thread is writing to the target.
  - Documentation shows the entire table of symbols.

# Workshop

- Now, we can run the Loader!

# Sample Run Procedure

```
$ set verify
$ @jcc_tool_com:jcc_lml_user 3.0
$ jcc_lml_oracle_user 10.1 disk$software:[oracle10g.db]orauser.com
!
$ define JCC_CLML_LOGGING_STYLE "Reuse“
$ define jcc_aij_backup_spec                      dpa200:[AIJ_BACKUP]*.aij;*
$ define source_db                                loader_regression_test_db
$ define loader_regression_test_control
loader_regression_test_control_ora.ini
$ define loader_regression_test_control_ora_interface                      -
                          loader_regression_test_control_ora_interface.ini
$ define loader_regression_test_metadata_ora                               -
                          loader_regression_test_metadata.ini
$ define loader_regression_test_excludes_ora                               -
                          loader_regression_test_excludes_ora.ini
$ define loader_regression_test_includes_ora                               -
                          loader_regression_test_includes_ora.ini
$ define loader_regression_test_opt                                        -
                          jcc_tool_examples:loader_regression_test_lm_unl.opt
$!
$ define JCC_LogMiner_Loader_Name                 REGTESTORA
$ define JCC_LogMiner_Loader_HW_Response          CREATE
$ jcc_run_clm_lml source_db loader_regression_test_opt                     -
                          loader_regression_test_control
```

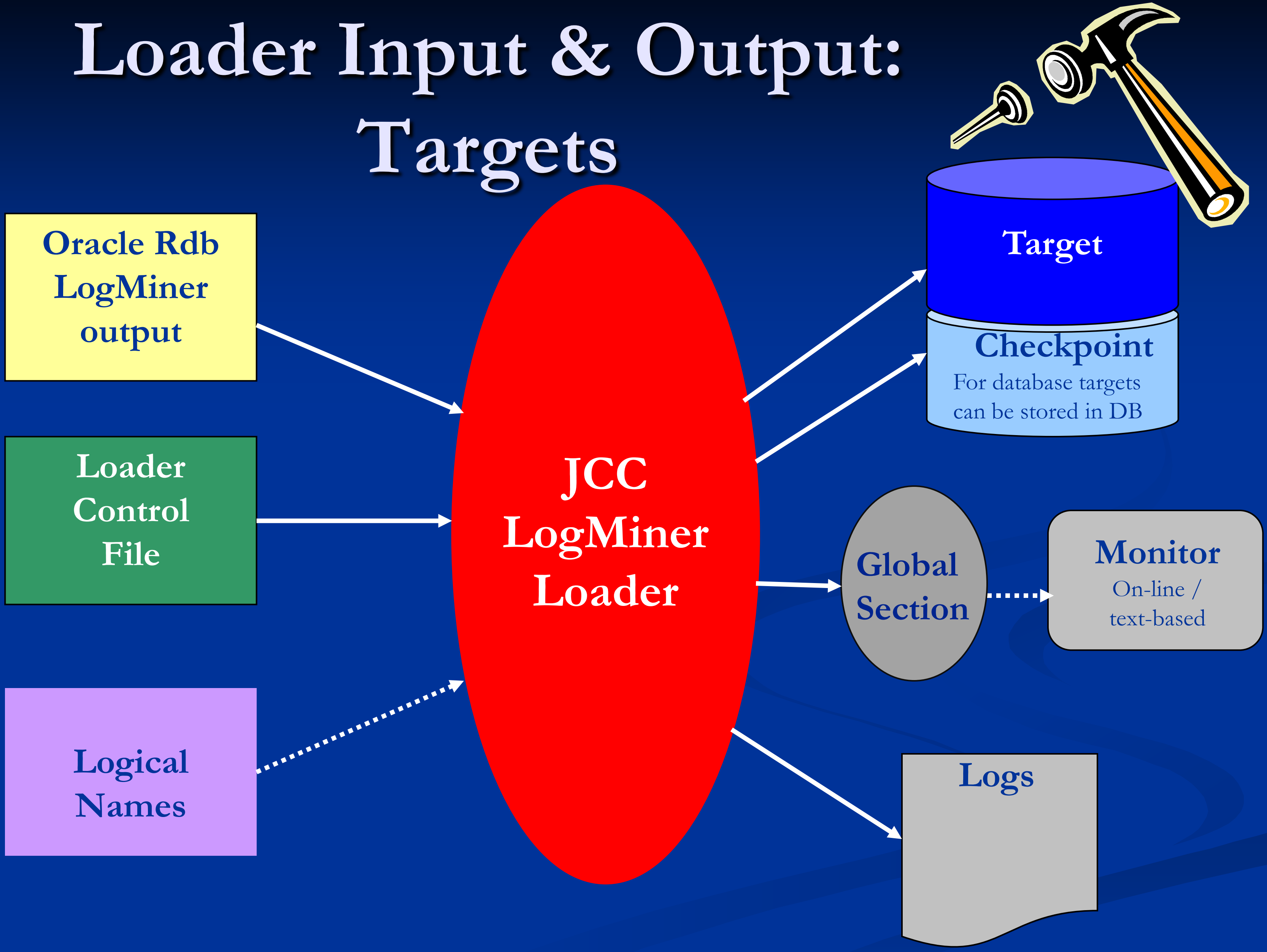
# Ready? What else? →

- Watch your operation
  - Statistics monitoring
  - Logging
  - Materialized data
  - Latency alarms
  - Locking diagnostics
- Test your overall architecture
  - Copy mode
  - Throttles
  - Materialized data
  - Monitoring
- Tune your performance
  - Checkpoint Intervals
  - Parallel Threads
  - Additional Loaders
  - Systems Tuning
  - Exclude what's not needed
  - Boosts for the Target
  - DBA tools
- Other topics …

# Topic 6

# Monitoring

# Loader Input & Output: Targets



# Monitoring Your System →

- Latency alarms
- Statistics Monitor
- Logging
- Additional information from materialized data
- Locking diagnostics

# Latency Alarms

- Loader can report problems to the operator.
  - Operator class is definable
  - Latency trigger is definable
- `JCC_LogMiner_Loader_stat_tardy_field – trailoutput|trailinput|latency`
- `Operator~ALL|<operator class> - [,<op class>[,…]]`
- Note that the problem may be in downstream processing.  The alarm just says that the Loader is behind.

# Statistics Monitor

-
  ```
  JCC_LML_statistics <Loadername>        -
  [refresh seconds][report type]         -
  [tardy threshold[operator class]]
  ```
  - Requires Loader name
  - Refresh rate of 3 seconds is the default
  - Report types are
    - F full
    - D detailed
    - B brief
    - C CSV (comma separated values)
    - T T4
  - Uses vary
    - Some report types create data for OpenVMS tools
    - Some report types are suitable for online display
    - Some report types create data that can be loaded into a db

# Example of a Detail Report

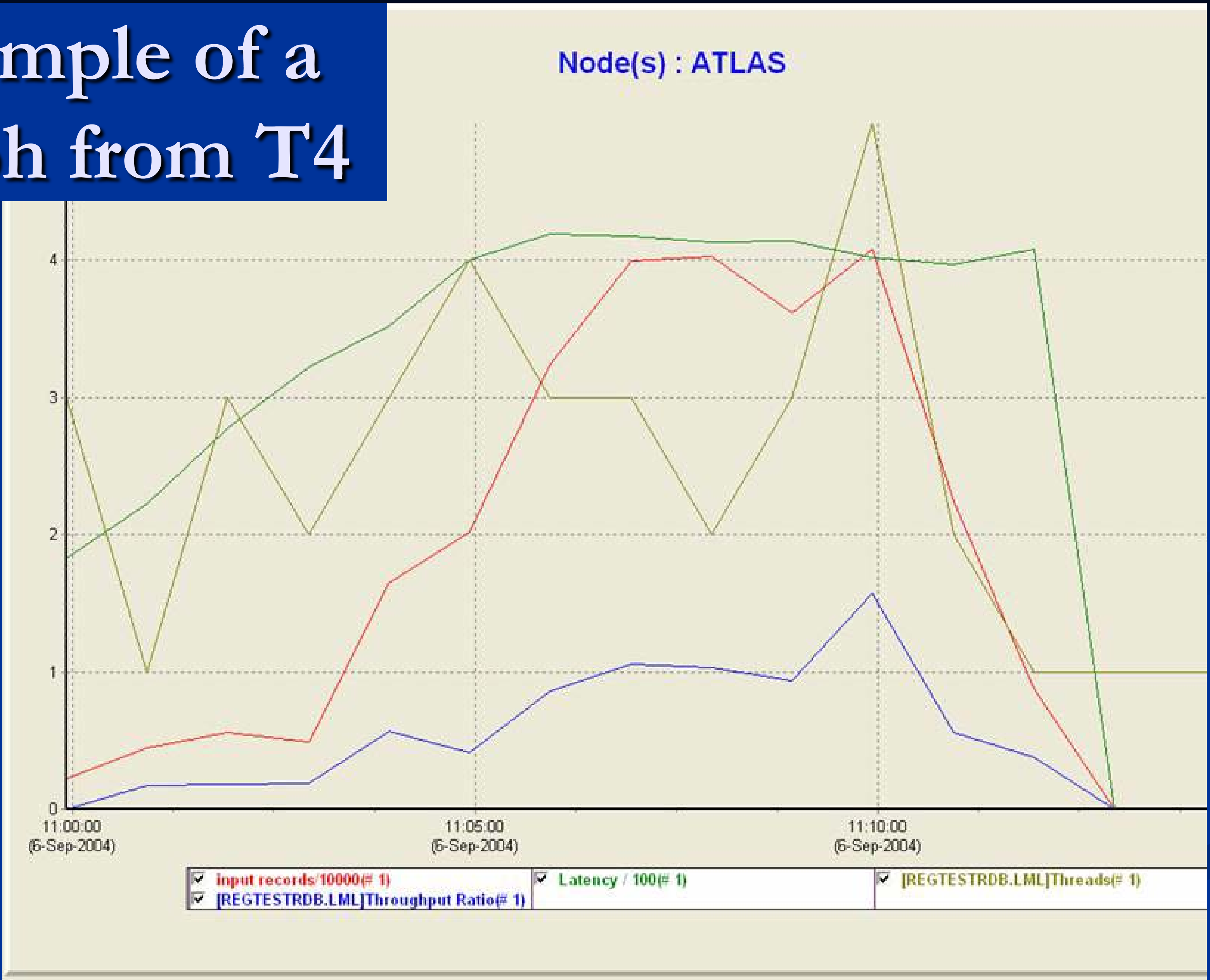
```
Rate:   6.00                       REGTESTOPO             2-OCT-2007 16:32:30.00
================================================================================
   Input:  2-OCT-2007 16:20:23.53                Output:  2-OCT-2007 16:20:22.69
--[Trail:   12.1m]---------------             ---[Trail:   12.1m]---------------
Transactions                22654             Checkpoints                   4888
Records                    276850                Timeout                       7
   Modify                  253074                BufferLimit(   170)         100
   Delete                    1123                NoWork                        2
   Commit                   22653             Records(   2)               253971
 Discarded                                       Messages(  N/A )            N/A
   Filtered                     0                Filtered                      0
   Excluded                     0             Failure                          0
   Unknown                      0             Timeout                          0
   Restart                      0        - Current ---------------- Ave/Second -
   NoWork                    1657        Checkpoints            43          7.17
   Heartbeat                    5        Records              1765        294.17
Timeout                         0        Rate                67.87%
--- Restart Context ------               - Latency(sec) ------ LML detail ------
M|AIJ#                  4 |              CLM    12.1m |  Inpt  26.0%  Cnvt   6.4%
Q|VBN              447834 |              ------------   Sort   0.1%  Trgt  55.5%
P|TSN               44289 |              LML    0.64    Sync  12.1%  Ckpt   0.0%
 CTSN               44303 |        - Loaders - 01234567 ------------------------
  LSN               20952 |        - States  - >>>R>R<R
```

Oracle target in
regression test.
Target machine is
pegged.

# Thread States

- "Threads" and parallel processing are a performance tuning option to be discussed later.
- The example shows 8 threads (0-7).
- The symbols below each shows its state.
  - R means that the thread is waiting for the read lock.
  - < means that the thread is currently reading from the input mailbox.
  - > means that the thread is writing to the target.
  - Documentation shows the entire table of symbols.

# Example of a Graph from T4



# CSV Output in a Database

- CSV – Comma Separated Values – is one of the monitor report types.
- Loader kit includes a procedure for loading CSV output into an Rdb database.
- Some Loader users have found this valuable for analysis.

# Example of Running the Procedure to Load the CSV Output into a DB

```
    jcc_tool_source:jcclml$statistics.sql

    create table jcclml$statistics
           (Report_Datetime    timestamp(2)
           ,Loader_name        char(31)
           ,Commit_Datetime    timestamp(2)
           ,Row_Rate           integer
           ,Transaction_Rate   integer
           ,Source_Txn_Seconds integer(2)
           ,Statistics_Interval      integer(2)
           ,Thruput_Ratio            integer(2)
           ,Trailing_Seconds   integer(2)
           ,Input_Timeouts           integer
           ,Output_Failures    integer
           ,Loader_Threads           integer
           ,Total_Latency            integer(2)
           ,Input_Latency            integer(2)
           ,Output_latency           integer(2));
```

```
jcc_tool_source:JCCLML$STATISTICS.RRD

DEFINE FIELD REPORT_DATETIME DATATYPE IS TEXT SIZE IS 23.
DEFINE FIELD LOADER_NAME DATATYPE IS TEXT SIZE IS 31.
DEFINE FIELD COMMIT_DATETIME DATATYPE IS TEXT SIZE IS 23.
DEFINE FIELD ROW_RATE DATATYPE IS TEXT SIZE IS 12.
DEFINE FIELD TRANSACTION_RATE DATATYPE IS TEXT SIZE IS 12.
DEFINE FIELD SOURCE_TXN_SECONDS DATATYPE IS TEXT SIZE IS 13.
DEFINE FIELD STATISTICS_INTERVAL DATATYPE IS TEXT SIZE IS 13.
DEFINE FIELD THRUPUT_RATIO DATATYPE IS TEXT SIZE IS 13.
DEFINE FIELD TRAILING_SECONDS DATATYPE IS TEXT SIZE IS 13.
DEFINE FIELD INPUT_TIMEOUTS DATATYPE IS TEXT SIZE IS 12.
DEFINE FIELD OUTPUT_FAILURES DATATYPE IS TEXT SIZE IS 12.
DEFINE FIELD LOADER_THREADS DATATYPE IS TEXT SIZE IS 12.
DEFINE FIELD TOTAL_LATENCY DATATYPE IS TEXT SIZE IS 13.
DEFINE FIELD INPUT_LATENCY DATATYPE IS TEXT SIZE IS 13.
DEFINE FIELD OUTPUT_LATENCY DATATYPE IS TEXT SIZE IS 13.
DEFINE RECORD JCCLML$STATISTICS.
   REPORT_DATETIME .
   LOADER_NAME .
   COMMIT_DATETIME .
   ROW_RATE .
   TRANSACTION_RATE .
   SOURCE_TXN_SECONDS .
   STATISTICS_INTERVAL .
   THRUPUT_RATIO .
   TRAILING_SECONDS .
   INPUT_TIMEOUTS .
   OUTPUT_FAILURES .
   LOADER_THREADS .
   TOTAL_LATENCY .
   INPUT_LATENCY .
   OUTPUT_LATENCY .
END JCCLML$STATISTICS RECORD.
```

Creating the table example:

```
$ sql
SQL> create database filename jcclml$statistics_db
cont> reserve 10 storage areas
cont> number of users is 8
cont> number of cluster nodes is 1
cont> default storage area is statistics_default
cont> page size is 12 blocks
cont> create storage area rdb$system filename statistics_rdb$system
cont> page size is 12 blocks
cont> create storage area statistics_default filename statistics_default
cont> page size is 12 blocks;
SQL> @jcc_tool_source:jcclml$statistics.sql
SQL> commit;
SQL> exit
```

Loading V2.1 statistics example:

```
$ define JCC_LOGMINER_LOADER_STAT_CSV_DATE "|!Y4-!MN0-!D0:!H04:!M0:!S0.!C2|"
$ define JCC_LOGMINER_LOADER_STAT_OPTIONS "noheader,nointeractive"
$ define stats_rrd jcc_tool_source:JCCLML$STATISTICS.RRD
$ define error_file jccstat$errors.txt
$ pipe jcc_lml_statistics regtestrdb 60 csv | -
_$       rmu/load/commit=1/row=1/rec=(file=stats_rrd,form=delim,pre="",suff="",
null="(none)",except=error_file) -
_$       /log jcclml$statistics_db JCCLML$STATISTICS sys$pipe:
%RMU-I-DATRECSTO,   1 data records stored.
%RMU-I-DATRECSTO,   2 data records stored.
%RMU-I-DATRECSTO,   3 data records stored.
%RMU-I-DATRECSTO,   4 data records stored.
%RMU-I-DATRECSTO,   5 data records stored.
%RMU-I-DATRECSTO,   6 data records stored.
%RMU-I-DATRECSTO,   7 data records stored.
%RMU-I-DATRECSTO,   8 data records stored.
```

# Logging →

- The CLML is a logging sink for LogMiner and for Loader threads.
- JCC requests that you add to your Control File `logging~initialize`
  - It echoes the Control File in the log.
  - It makes it much easier to analyze any issues that arise.
- Additional logging is controlled with the logging keyword and various logical names.

# Logging

- Logs with runtime information will include
  - LM
  - LML thread
  - Control process
- Different logging is recommended for problem resolution than for general production use.  (See the documentation.)
- If extensive logging is requested, care is required to provide sufficient disk space and management of the logs.

# Logging

| | |
|---|---|
| 🟥 Logging~output~trace | 🟨 Logging~input~nowork |
| 🟥 Logging~output~dynamic_data | ▨ Logging~input~synchroniztion |
| ▨ Logging~output~record | ▨ Logging~statistics~runtime |
| ▨ Logging~output~synchronization | ▨ Logging~statistics~commit |
| 🟥 Logging~input~trace | ▨ Logging~statistics~timer |
| 🟥 Logging~input~record | 🟥 Logging~trace |
| 🟥 Logging~input~brief | 🟥 Logging~locking |
| ▨ Logging~input~[NO]log_restart | 🟥 Logging~NoSort |
| ▨ Logging~input~[NO]ignore_unknown_tables | 🟥 Logging~heartbeart |
| 🟨 Logging~input~filter | 🟩 Logging~initialization |

| | | | |
|---|---|---|---|
| 🟩 Green = Include | ▨ Gray = Use when needed | 🟨 Yellow = Caution | 🟥 Red = Extremely verbose. Use briefly only |

# Materialized Data as Clues

- Some of the possible VirtualColumns are used to provide information on where time is spent.
  - Transaction_commit_time
  - Transaction_start_time
  - JCC_LML_read_time
  - Transmission_date_time

# Locking Diagnostics

- JCC_LML_show_locks - [LoaderName] - [-b[locking]]
  - Specifying LoaderName limits the output to one Loader family.
  - Specifying blocking limits the output to locks that are blocked by another process.
- Enables discovery of “friction” between Loader threads.
  - In one example, a row was updated in *every* transaction. Effectively, locking for that row made the Loader family single-threaded. (We will cover how to make a Loader family multi-threaded in the Tuning section.)

## Topic 7

# Tuning

# Tuning Your System

- For the purposes of tuning a Loader instance, the word "target" means anything downstream from the Loader.
  - If the target is across a network, the network cost is included in the cost of the target.
  - The Loader cannot write faster than the target and the network can absorb the data.
- The Loader monitoring helps you identify the bottlenecks.
- The Loader provides tools to help you tune your system.
  - Additional Loader families can, sometimes, be used to improve performance, but there may be tradeoffs.
  - Specific keywords may help Loader performance *or target performance.*

# Tuning

- *Optional* keywords of interest for tuning
  - Checkpoint
  - Input_failure
  - Output_failure
  - Parallel & Thread
  - Sort
  - VirtualColumn
  - Target specific keywords
- Logical names of interest in tuning
- Open VMS and tuning

# A Brief Aside - Restartability

- The first guarantee by the Loader is that it will deliver each row at least one time to the target.
  - The guarantee is achieved by recording the high water context as part of a checkpoint.
    - Information from the LogMiner about the AERCP
    - Loader sequence number
    - TSN of the committed transaction
  - The information is recorded
    - Within the transaction in a database table, whenever possible.
    - In a file, otherwise after the transmission is acknowledged.
  - During restarts from system crashes and after errors in transmission, data may be resent.
    - This is not a problem for replication.
    - This is not a problem for most Rdb and Oracle targets.
    - Relative age of the change may be determined by examining the materialized column Loader Sequence Number.

# What Interrupts the Loader and Requires Fault Tolerance?

- Database Administration and maintenance
- Downstream crises
- Network challenges
- Accidents
- Power failures
- What interrupts have you had?

# Fault Tolerance Requirements

- LogMiner and Loader require the AIJs.
  - If an AIJ is not processed by LM, it must be saved until it is processed. (LM and LML cannot invent the changes that have occurred.)
  - CLML can be restarted in the backup AIJs and process them in order until caught up.
- LogMiner and Loader require a consistent database definition.
  - The LM options file and the LML control file include a database definition of tables and columns.
  - Changes can be made, but the AIJs with one set of definitions must be processed with the correct definitions for LM and LML

# Commit Interval

- Each Loader thread "swallows" some consecutive number of transactions from the LogMiner.
- How many transactions in this "gulp" is called the "Commit Interval" and may be specified in the Control File using the Checkpoint keyword
  - For API (File or XML) targets, checkpoint is limited to one.
  - For other targets, any number of source database transactions may be bundled together.
- The source transactions in a commit interval are transmitted as a group to the target.
  - Commits are done to the target as a group.
  - If any failure occurs [e.g. deadlock] the transaction is rolled back and the entire unit is retried.

# Checkpointing Efficiencies

- Checkpoints with more than one source database transaction involved can lead to greater efficiencies.
  - For database targets, transaction commit overhead is reduced.
  - For Oracle targets, dynamic SQL statements are prepared less frequently leading to reduced network traffic (provided the data is sorted by table name).

# Checkpointing Issues

- The latency for delivery of rows to the target will be increased for checkpoint intervals > 1.
  - First transaction is not delivered until remainder of transactions are received.
  - If source database becomes quiescent, a Loader thread may stall waiting to complete the commit interval.
    - See description of timeouts later in this presentation.
- The Loader stores data in virtual memory.  Larger commit intervals imply larger memory demands.
- If the Loader is running in "constrained" mode, larger commit intervals imply the OpenVMS system should be tuned for a larger number of resources and locks.

# Loader Checkpoint Files

- Special file format reserved for the Loader.
- Two records in this file for each thread.
  - Loader writes to these records alternately.
  - One is always good.
- Each record stores:
  - Rdb high water context (AERCP) of last transaction sent to the target.
  - Loader sequence number of that transaction.
    - Loader sequence number is assigned by the Loader.
      - Incremented by 1 as commit records are read.
    - Represents the relative order in which transactions were committed to the source database.
    - May be materialized in some output messages.
  - TSN of the last transaction committed to the target

# Checkpoint Keyword Examples

- Syntax for checkpoint keyword

```
checkpoint~<commit interval>                              \
[~<checkpoint stream type>[~<synchronous>     \
[~<checkpoint target>]]]
```

- Checkpointing to a file with a commit interval of 1.

```
checkpoint~1~lml_internal~async~    \
  loader_regression_test_chkpt
```

- Checkpointing to a database with a commit interval of 10.

```
checkpoint~10
```

# Dynamic Checkpoint Adjustment

- If the workload suddenly changes, e.g. the number of rows in a transaction becomes large, the Loader will dynamically adjust the commit interval down.
  - The goal is to reduce the number of locks being held in the target.
  - There is no tuning knob for this feature.
- When transaction size decreases, the checkpoint interval will gradually return to the original size.

# Backing Up the Local Checkpoint File

- If the local checkpoint file is used, it will contain the only context for restart in case of failure.
  - Where to start in the AIJ.
  - Loader sequence number.
- The checkpoint file may be backed up while it is open.
  - $ backup/ignore=interlock.
  - Requires some OpenVMS privileges to do this.

TIP

The checkpoint file is critical to the recovery strategy!

# Content of a Checkpoint File

- Current checkpoint information may be displayed by using the `jcc_lml_dump_checkpoint` command.

```
Pollux> jcc_lml_dump_checkpoint REGTESTPRDB loader_regression_load_db rdb

JCC LML Dump Checkpoint V02.01.00ev6 (built  4-MAY-2004 10:09:01.43)

 -- Checkpoint restart information --

--- Parallel mode ---
Write Timestamp:      5-MAY-2004 10:01:02.57
LoaderName:          REGTESTPRDB
Completion Flag:     N
Checkpoint Interval: 7
Input Data Source:   LML_CONT_REGTESTPRDB
Last Transaction:
        Start Time:   5-MAY-2004 09:15:33.59
        Commit Time:  5-MAY-2004 09:15:33.62
        TSN:         117784
        LSN:         72027
        AERCP:       1-28-5-1299557-117730-117784
        RM TID:
```

The real-time monitor also displays the restart context at each refresh.

# Checkpoint & DCL Procedures

- One of the purposes of the dump checkpoint utility is to create and populate DCL symbols for use in DBA procedures:

```
Pollux> show sym jcclml$*
  JCCLML$AERCP == "1-28-5-1299557-117730-117784"
  JCCLML$CHECKPOINT_INTERVAL == "7"
  JCCLML$COMMIT_TAD == " 5-MAY-2004 09:15:33.62"
  JCCLML$COMPLETION_FLAG == "N"
  JCCLML$INPUT_SOURCE == "LML_CONT_REGTESTPRDB"
  JCCLML$LOADERNAME == "REGTESTPRDB"
  JCCLML$LSN == "72027"
  JCCLML$RM_TID == ""
  JCCLML$START_TAD == " 5-MAY-2004 09:15:33.59"
  JCCLML$TSN == "117784"
  JCCLML$WRITE_TAD == " 5-MAY-2004 10:01:02.57"
```

# Input_failure

- Checkpoint allows the Loader to batch transaction commits.
    - Generally, this enhances performance.
    - Some scenarios make a commit interval of more than one a problem.
        - Filling part of the commit interval and, then, having a lull with no new source database commits
        - Filling part of the commit interval and, then, having source rows committed that are not destined for the target ("no work" transactions)
- Input_failure provides a timeout to address this situation.

# Input_failure

- Syntax

  ```
  input_failure~<timeout seconds>
  ```
- Sets the number of seconds to wait on input before committing the buffered rows.
- Default is 0.00, which disables input_failure.
- Requires
  - Input_type of IPC (mailbox)
  - Checkpointing enabled
- Note need for order in the Control File.

# Output_failure

- Loader operation can be impeded by failure of the target.
- Syntax

```
output_failure~<timeout seconds>      \
            ~<message retry attempts>
```

- <timeout seconds>
  - Number of seconds to wait before requesting a re-send.
  - Default of one second.
  - Used only with API targets.
- <Message retry attempts>
  - Number of re-sends to attempt before failing
  - Default is ten.
- Addresses deadlock, dropped networks, target overload

# Parallel Processing

- You may run more than one concurrent Loader family against the same source database
- Each Loader session can run with many concurrent threads.
  - The word "thread", here, is used to mean an independent OpenVMS process.
  - Threads can be set to synchronize with one another to avoid buried updates.
    - Uses the OpenVMS Lock Manager
  - The number of threads running can be dynamic or they can be manually controlled.
  - Each thread will maintain its own restart context in the highwater table or checkpoint file.
    - Restart is done from the earliest restart point recorded in the checkpoint context.

# Requirements for Multi-threading

- Input type must be IPC (mailbox).
  - Running CLM.
- Parallel keyword set in the Control File.
  - Defines the min and max number of threads and the mode.
- Thread keyword set in the Control File.
  - To define dynamic characteristics of threads.
- Input keyword must not be set to ASYNC.
  - With continuous LogMiner each thread's input activities are synchronous.
- Input_failure must be set in the Control File.
  - To permit timeouts.

# Parallel Keyword

- Syntax:

```
parallel~<minimum threads>~<maximum threads>  \
    [~CONSTRAINED|AUTOMATIC|UNCONSTRAINED]
```

- <Minimum threads>
  - Must be set.
  - Must be between one and maximum threads, inclusive.
  - May be re-set while the Loader is running with the procedure.

    `JCC_CLML_minimum_threads <LoaderName> <new minimum>`
- <Maximum threads>
  - Must be set.
  - Must be between the minimum and 32, inclusive.
  - May be re-set while the Loader is running with the procedure.

    `JCC_CLML_maximum_threads <LoaderName> <new maximum>`

# Parallel Keyword

```
parallel~<minimum threads>~<maximum threads> \
       [~CONSTRAINED|AUTOMATIC|UNCONSTRAINED]
```

- Consistency mode (*optional*)
  - Constrained
    - Constrained uses VMS locking to synchronize between threads
    - Updates of a specific row will occur in the same order as for the source commits, giving the same results as in the source.
    - Constrained is the default.
  - Automatic
    - Uses unconstrained for update only tables that are audit trails (insert, noupdate, nodelete)
    - Uses constrained in all other cases.
  - Unconstrained
    - Provides ***no*** synchronization between threads.
      - Updates can occur out of order and lead to buried updates.
    - Is safe in insert only situations, like audits, particularly if virtualcolumns for the commit timestamp and LSN are used.
    - Updates and deletes can provide surprises.

# Parallel Keyword

- If a Loader family is started *without the parallel keyword*, it will intrinsically run with one thread.
- This cannot be changed while the session is instantiated.
  - Thread startup and shutdown commands have no effect.
- To convert to threaded, edit the Control File, shutdown and restart.

# Thread Keyword

- The thread keyword controls the rapidity with which the number of threads is adjusted.
- Syntax

```
thread~startup~<delay seconds>
thread~shutdown~<delay seconds>
```

- Delay seconds is the number of seconds to wait before performing the action.
  - Default is 30.00 in both cases.
  - The parameters may include fractional seconds.
  - Recall that time on OpenVMS is intrinsically synchronized to hundredths of a second.

# Issues With Thread Dynamics

- A single Loader thread can do a great deal of work, but throughput intrinsically depends on the performance of the target.
  - If the target is extremely parallelized, then large numbers of threads can increase throughput.
  - If the target is not parallelized, more threads can interact with each other and reduce net throughput.
  - Some of the materialized columns can assist you with partitioning in some targets.
- Loader threads require OpenVMS resources.
  - Creating and deleting threads consumes resources in the starting and stopping of threads. Short fused threads may slow results.
  - Constrained threads use the OpenVMS lock manager. Large numbers of constrained threads can have a performance impact.
  - Threads share Open VMS "pooled quotas" such as PGFLQUO (page file quota). Ensure that pooled quotas are sufficient on an account using threads.

# Materialized Data for Tuning

- The VirtualColumn keyword can be used to add materialized data to a row.
- The VirtualColumn keyword offers options for partitioning the target to enhance performance.
  - VirtualColumn~RandomValue[,<output column name>]~\
    <high value>[<low value>]
  - VirtualColumn~ModValue[,<output column name>]     \
    ~<input column>~<mod value>
- The VirtualColumn keyword also offers many options for adding information which can aid in tracking issues.
  - VirtualColumn~transaction_commit_time
  - VirtualColumn~transmission_date_time
- For additional information on the numerous VirtualColumn options, see the documentation.

# Automatic, Dynamic Tuning

- For checkpointing (already mentioned)
- For extremely large transactions
  - Locking architecture adjusted at threshold
    - From row locks
    - To request WriteLock in protected write mode
  - Threshold default 100,000 rows in a transaction
  - Default configurable with logical name JCC_LOGMINER_LOADER_LOCK_THRESHOLD
  - Gets the large transaction out of the way and returns to normal processing

# Additional Performance Options

- Locking control with JCC_LML_LOCKING_LEVEL
- 64 bit processing with JCC_COMC_VA_MEMORY_MODEL
- Ensure sufficient OpenVMS resources
- others

# Locking Level

- JCC LogMiner Loader uses the Open VMS lock manager.
- Logical name JCC_LML_LOCKING_LEVEL enables user control of locking levels.
  - Page, Page(0) – Source page
  - Page(1) – Block of 16 source pages
  - Page(2) – Block of 256 source pages
  - LAREA – logical area
- As locking level increases, synchronization between Loader threads must increase.
- However, increasing locking level can reduce lock conflicts in the target.
- As with many tuning options, there is no "right" answer for all.

# OpenVMS

- Memory and Pagefile
  - Amount needed is proportional to the maximum number of rows/transaction
  - About 70 bytes of overhead per row
  - Using 64-bit addressing
    - $ define JCC_COMC_VA_MEMORY_MODEL 1
    - Makes it possible to handle much larger transactions
    - Requires care with quotas, especially page file quota
- CPU
  - Demand is directly proportional to source database rows updated/second
  - Impact of Loader on the CPU is very light

# Quotas to Run the Loader

- Ensure that your account has sufficient resources.
  - The Loader is replicating all of the data that is written by *all* of the processes that update the source database!
  - Quotas will need to be higher than you are accustomed to.
  - If using the 64-bit model, you will need even higher quotas.
  - If you have used threads, you may need more resources.
- You are likely to tune quotas as you work with the Loader.

Topic 8

# Tools and Topics for the Database Administrator

# DBA Tools

- Documentation
- Support "Desk"
- Kit procedures
  - Logical Name Maintenance facility
  - AIJ Backup support
  - Additional tools
  - Examples

# Additional Topics

- In this chapter
  - Heartbeat
  - Data Pump
- In later chapters
  - Copy Mode
  - Testing your architecture
  - Advanced schema mapping
  - Problem resolution & support

# Heartbeat

- The LogMiner maintains an Rdb checkpoint in the AIJ file it is working on.
  - Prevents RMU backup from removing the data as it is actively being processed.
- The LogMiner does not respond to journal switch signals until a transaction occurs.
- The result is that AIJ backups can stall if the database becomes quiescent.
- The solution to this dilemma is the JCC CLML heartbeat mechanism.

# Heartbeat Mechanism

- For Heartbeat the CTL process writes a transaction to a special JCC heartbeat table.
  - If the table does not exist it will be created.
  - One row in the table for the CTL process to increment.
  - Updates occur at specified intervals.
- This table will be placed in the default storage area, or the RDB$system area, if no default is specified.
- DBA may move the table after it is created.
- The CTL process adds the heartbeat table to the agenda for the LogMiner.
  - Heartbeat rows are noticed by the Loaders and processed appropriately.

# Enabling Heartbeat

- Heartbeat is enabled by use of a VMS Logical name:

  ```
  jcc_clml_heartbeat_enable
  ```

  - Set the translation value to 1
- Define the heartbeat frequency

  ```
  jcc_clml_heartbeat_interval
  ```

  - Set the translation value to the number of seconds between heartbeats.
  - A value of zero stops writing of heartbeat transactions.
- Heartbeat *should* be enabled on all Loader families acting on a source database.
- *Heartbeat writing should be done by only one Loader family.*
  - Since the table is not indexed, concurrency stalls can occur if multiple CTL processes are attempting to update.

# Data Pump

- Initially added because a major Loader user spent weeks loading data and, then, in testing the downstream processing messed it up.
  - Data Pump permitted "pumping" the source data known to be modified by the incorrect downstream processing.
- Various Loader users have found it the fastest possible way to load the target, initially.
  - The ability to "pump" only the data required is also useful for targets that are not complete replications.

# Data Pump Flexibility

- Data Pump makes ***NO CHANGE*** updates to cause the LogMiner and Loader to pick up the row and push it to the target.
- Data Pump "understands" parent-child structures.
- Data Pump is parameter driven in what it "pumps."
- Data Pump can be tuned with commit.
- Data Pump can be slowed to reduce impact on source.

# Data Pump Components

- Structure File
  - Contains one or more Table Hierarchies
    - Table hierarchy can be restricted with '?' as a parameter marker.
    - Example: Account might have child tables for bill history and for products or services used.  Bill history might have a child table for line items on the bill.
  - Modified with optional syntax
- Data Driver File
- Data Pump Log

**TIP**

Data Pump is included with the Loader.  Data Pump is NOT *required* for running the Loader.

# Structure File and Table Hierarchies

- Structure File contains one or more Table Hierarchies.

  -
    ```
    <table name=“your parent table name”>
           <restriction>where your restriction
           </restriction>
           <update name=“column to no-change update”/>
    </table>
    ```
    - Words in gold and underlined must be replaced with your values.
    - Your restriction is the where clause.
    - For the top-most parent table, the where clause may contain a ? as a parameter marker to be filled in from the Data Driver File.
  - Child tables
    - May not have the question mark, parameter marker
    - Are placed before the </table> of the parent
  - Hierarchies may be up to 12 generations deep
  - Each <table> definition can have up to 36 children.
  - Table hierarchies are separated by a line with only a period.

# Optional Syntax for the Structure File

- Hierarchy
  - By default, the hierarchy name is the table name.
  - An alternate hierarchy name can be specified for the top-level parent.
  - Hierarchy name is used in the Data Driver File.
- Commit
  - Commits are done when
    - The maximum number of rows specified by the commit interval is reached.
    - Processing of a table is complete.
  - Commit interval default is 10.
  - Commit intervals are inherited, if not specifically defined.
- Delay and Sleep
  - Provide a mechanism for slowing the Data Pump.
    - Delay is the number of commits before a delay is imposed.
    - Sleep is the number of seconds to delay.
  - Default for each is zero.
  - Value of zero for either disables.

# Data Driver File

- Provides the data to satisfy the parameter markers in the Structure File.
- Provides a set of column-value pairs.

  -
    ```
    <hierarchy name>                          \
    [<column name>=<data value>               \
    [<column name>=<data value>[ … ]]]
    ```
- Has some requirements
  - Column-value pairs must be in the same order as the parameter markers in the Structure File.
  - Text and Date data values must be in quotes.
  - Backslash (“\”) can be used as a line continuation.
  - Hierarchy name and first column-value pair, as well as subsequent column-value pairs must be separated by (one or more) spaces or backslash and line feed.
  - Comment lines begin with an exclamation mark and are ignored.

# Data Pump Examples

- Example shows
  - Structure File
  - Data Driver File
  - Log (showing the results)
- Additional examples are available in the documentation.
  - Example in the documentation is designed to show all the features.
  - That is, realism is not the prime goal.

**TIP**

Data Pump is included with the Loader. Data Pump is NOT *required* for running the Loader.

# Data Pump Performance

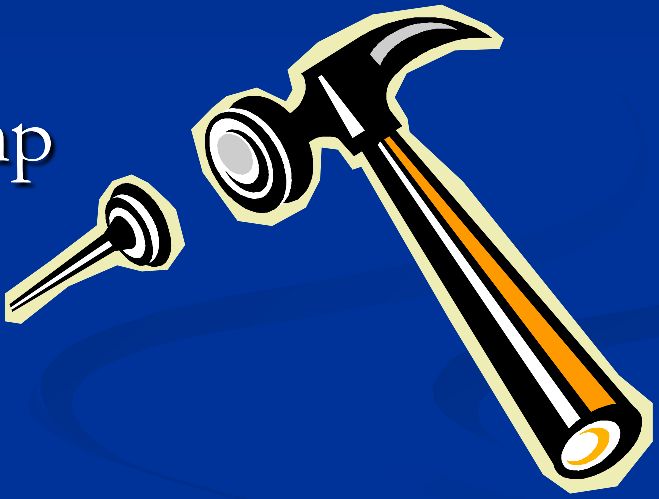
- Biggest problem with the Data Pump is …
  - Designed for performance.
    - Uses temporary tables and an SQL compound statement.
    - Runs in exec mode in Rdb.
      - Includes few boundaries where ^C will be recognized.
  - **Watch what you ask for!**
    - Practice with small requests until you understand the tool.
    - Do not overwhelm your production database while populating the target.

# Data Pump Tuning

- For the most rapid *initial* loading – after you understand all the concepts – consider
  - Small transactions (few hundred or a thousand rows)
  - Unconstrained families.
  - Table actions set to insert, noupdate, nodelete.
  - One Loader and Data Pump can be used for each table.
  - Examine the Rdb documentation for hints on tuning temporary tables.
- Consider using delay and sleep to slow the Data Pump and minimize the impact on Production.

# Workshop

- Pardon the interrupt …
- Let's look at the data pump in action.



# Example Data Pump Structure File

```
<table name='departments'
        hierarchy='DeptHistory'
        commit='3'
        delay='3'
        seconds='2.0'>
        <restriction>
          where department_code = ?
        </restriction>
        <update name='department_name'/>
        <table name='job_history' commit='5'>
           <restriction>
                 where department_code = departments.department_code
           </restriction>
```

Beginning of a hierarchy definition. Two tables and 2 restrictions shown. Commit and delay shown. Notice specification of column to update.

Continued on following pages.

```
 <update name='supervisor_id'/>
      <table name='jobs' commit='7' seconds='0'>
          <restriction>
             where job_code = job_history.job_code
          </restriction>
          <update name='job_title'/>
      </table>
      <table name='employees'>
          <restriction>
               where employee_id = job_history.employee_id
          </restriction>
          <update name='last_name'/>
      </table>
    </table>
</table>
.
```

Note the period (on its own line) that separates the two hierarchy definitions.

Continued on following page.

```
<table name='employees'>
    <restriction>
        where employee_id=?
    </restriction>
    <update name='last_name'/>
    <table name='job_history'>
      <restriction>
        where employee_id = employees.employee_id
      </restriction>
      <update name='supervisor_id'/>
         <table name='jobs'>
          <restriction>
            where job_code = job_history.job_code
          </restriction>
          <update name='job_title'/>
         </table>
      <table name='departments'>
        <restriction>
          where department_code = job_history.department_code
       </restriction>
       <update name='department_name'/>
      </table>
    </table>
</table>
```

.

The second hierarchy definition. Hierarchy shown is three levels deep. Restriction shown on the top level and on each child table.

# Example Data Pump Data Driver File

```
DeptHistory                    \
  DEPARTMENT_CODE='ENGR'
employees                      \
  employee_id='00471'
employees                      \
  employee_id='02000'
DeptHistory                    \
  DEPARTMENT_CODE='ADMN'
```

For the two parameters in the Structure File.

# Log File

```
$ jcc_lml_data_pump JCC_ROOT:[DEMO.SOURCE]MF_PERSONNEL
    PUMP_01.structure PUMP_01.DRIVER

JCC LML Data Pump V03.01.00st (built 17-MAR-2007 13:35:36.31)

This application is licensed to DEMO_NOV_15_2007.
Start time: Mon Sep  3 14:15:42 2007
```

Some header information.

Continued on following pages.

```
Structure> <table name='departments'
Structure>  hierarchy='DeptHistory'
Structure>  commit='3'
Structure>  delay='3'
Structure>  seconds='2.0'>
Structure>  <restriction>
Structure>         where department_code = ?
Structure>  </restriction>
Structure>  <update name='department_name'/>
Structure>  <table name='job_history' commit='5'>
Structure>    <restriction>
Structure>              where department_code = departments.department_code
Structure>    </restriction>
Structure>    <update name='supervisor_id'/>
Structure>    <table name='jobs' commit='7' seconds='0'>
Structure>    <restriction>
Structure>                 where job_code = job_history.job_code
Structure>    </restriction>
Structure>    <update name='job_title'/>
Structure>    </table>
Structure>    <table name='employees'>
Structure>         <restriction>
Structure>                    where employee_id = job_history.employee_id
Structure>         </restriction>
Structure>         <update name='last_name'/>
Structure>    </table>
Structure>       </table>
Structure> </table>
Structure> .
Hierarchy 'DeptHistory' successfully declared
```

Begins echoing the Structure File.

First hierarchy.

Continued on following pages.

```
Structure> <table name='employees'>
Structure>   <restriction>
Structure>                 where employee_id=?
Structure>   </restriction>
Structure>   <update name='last_name'/>
Structure>   <table name='job_history'>
Structure>      <restriction>
Structure>                 where employee_id = employees.employee_id
Structure>      </restriction>
Structure>      <update name='supervisor_id'/>
Structure>     <table name='jobs'>
Structure>        <restriction>
Structure>                         where job_code = job_history.job_code
Structure>        </restriction>
Structure>        <update name='job_title'/>
Structure>     </table>
Structure>      <table name='departments'>
Structure>          <restriction>
Structure>                      where department_code = job_history.department_code
Structure>         </restriction>
Structure>         <update name='department_name'/>
Structure>      </table>
Structure>   </table>
Structure> </table>
Structure> .
Hierarchy 'employees' successfully declared
```

Completes echoing the Structure File.

Second hierarchy.

Continued on following pages.

```
Driver> DeptHistory                             \
Driver>    DEPARTMENT_CODE='ENGR'
departments: 0 selected row(s)
DeptHistory                              DEPARTMENT_CODE='ENGR'
^
No rows found matching the selection criteria.

Driver> employees                \
Driver>    employee_id='00471'
employees: 1 selected row(s)
job_history: 3 selected row(s)
jobs: 1 selected row(s)
departments: 3 selected row(s)
2007-09-03 14:15:48.75: employees          : updated: 1     missing: 0     remaining: 0
2007-09-03 14:15:48.92: job_history        : updated: 3     missing: 0     remaining: 0
2007-09-03 14:15:49.08: jobs               : updated: 1     missing: 0     remaining: 0
2007-09-03 14:15:49.24: departments        : updated: 3     missing: 0     remaining: 0
Updates to the 'employees' hierarchy completed successfully. Updated: 8 Missing: 0
```

Begins processing the Driver File.

Spacing adjusted for readability.

Continued on following pages.

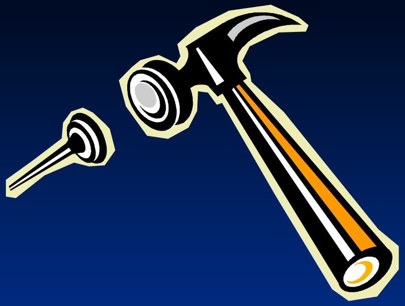
```
Driver> employees           \
Driver>    employee_id='02000'
employees: 0 selected row(s)
employees            employee_id='02000'
^
No rows found matching the selection criteria.

Driver> DeptHistory                        \
Driver>    DEPARTMENT_CODE='ADMN'
departments: 1 selected row(s)
job_history: 15 selected row(s)
jobs: 5 selected row(s)
employees: 10 selected row(s)
2007-09-03 14:15:49.93: departments      : updated: 1      missing: 0     remaining: 0
2007-09-03 14:15:50.14: job_history      : updated: 5      missing: 0     remaining: 10
2007-09-03 14:15:50.19: job_history      : updated: 10     missing: 0     remaining: 5
2007-09-03 14:15:50.25: job_history      : updated: 15     missing: 0     remaining: 0
2007-09-03 14:15:50.40: jobs             : updated: 5      missing: 0     remaining: 0
2007-09-03 14:15:50.68: employees        : updated: 5      missing: 0     remaining: 5
2007-09-03 14:15:50.73: employees        : updated: 10     missing: 0     remaining: 0
Updates to the 'DeptHistory' hierarchy completed successfully. Updated: 31 Missing: 0
```

Continues processing the Driver File.

Spacing adjusted for readability.

Continued on following page.



```
Exception records have been written to
   JCC_TOOL_DP:JCCDP_20070903141542.JCCDP_EXCEPTIONS

%dba_process_dp_input_file: %DBA-W-DRIVER_EXCEPTS, Completed with
   exceptions in the driver file.  See the exceptions file.

%DBA-W-DRIVER_EXCEPTS, Completed with exceptions in the driver
   file.  See the exceptions file.
colchi >
```

Completes the run with exception messages.

In this case, the exception messages are due to the cases for which no rows were found. Generally, when setting criteria for the Data Pump, you expect to find rows that meet the criteria.

Topic 9

# Additional Topics in Application Architectures

# Topics for Application Architectures

- Requirements Review
  - Beginning on slide 207
- Modes (specifically, copy mode)
  - Beginning on slide 215
  - Including throttles and testing, slide 218
- Data and Database Schema Transitions
  - Beginning on slide 220
  - Including materialized data, slide 223
  - Summary, slide 244

# Basic Design Restrictions

- Table Order
- Triggers
- Constraints
- Primary Keys
- AIJs and management concerns

# Table Order

- Table Order
  - Table update order (order of rows within a transaction) is not guaranteed.
  - Table update order can be artificially enforced, but doing so is not generally recommended.
- By default tables are sorted
  - Alphabetically by table name
    - With JCC_LML_Commit, the virtualtable, last, if it exists
  - With deletes before updates

# Triggers

- Triggers
  - Triggers in the target should be triggers that have not already fired in the source.
    - If – for example – the source has a trigger that increments a value every time some other column is updated, having the same trigger in the target will cause the count to be incremented twice for each update.
    - Other triggers – that have not already fired in the source – may be justified.

# Constraints

- Constraints
  - Constraints – and unique indices – in the target can interfere with Loader operation.
    - Order of rows within a transaction is not guaranteed.
    - If a child row is written before a parent row, some constraints might fail.
    - If a constraint is enforced in the source, it is not needed in the target.
- Indexes
  - The primary key needs an index.
  - The one and *only* unique index in a table should be on the primary key.

# Primary Keys

- Primary Keys
  - The Loader cannot update rows without a defined primary key.
  - Primary keys uniquely define a row and are not updated.
  - Primary keys in the target should be indexed.
  - The one and *only* unique index in the target should be on the primary key.

# AIJ

- AIJ files must be retained until the LogMiner has used them.
- May imply a change in operations.
- Loader kit includes tools that you can use to monitor where you are.
- The Data Pump may be used to recover if you have a serious failure in maintaining the AIJs.

# Basic Goals

- Basic goals for the Loader include
  - Reflect data that has changed
    - From an Rdb source
    - To a target
  - Control what happens to the data
  - Restart after shutdown with zero data loss
  - Monitor the performance of the transmission
  - Have minimal impact on the source
  - Work in "near real-time"
- Basic goals for the Loader are abridged by
  - Configuration
  - Tuning choices for the source
  - Architecture of the target
  - Network capabilities

# Modes

- Continuous
  - The default
  - The "near realtime" model
  - Uses backup and live journals
- Static
  - Used for database reorganization with minimal downtime
  - Uses backup journals
- Copy
  - Useful for *some* application requirements
  - Uses backup journals

# Copy Mode

- What is it?
  - Copy mode disconnects the source and target.
  - Copy mode does not require simultaneous operation.
  - Copy mode does not require that the Loader be run on the same machine that accesses the source database.
  - Copy mode *appears* to run like continuous mode, but not in realtime.
- Steps
  - Run the LogMiner in Static mode.
  - Provide the LogMiner output to the Loader.
  - Run the Loader in Copy mode.

# Copy Mode Uses

- Testing and Tuning
  - Repeatable
  - Real workload
- Production
  - No reliable connection between remote sites
  - Security concerns
  - Political or technical reasons not to coordinate
  - Preference for controlled times for updating the target

# Testing with Copy Mode

- Copy mode combines with other Loader tools for excellent testing.
- Throttles
  - Real time – approximate (within 1/100 sec) real time
  - Fixed – fixed (secs) delay between commits
  - Real time percentage – speeds commits up
    - 50% is twice real time; 25% is four times the original work load.
    - You can see how well your downstream application will scale!

# Advantages and Disadvantages of Copy Mode

- Advantages
  - Repeatable
  - Tunable
  - Does not impact the source in repeated trials
  - Can provide efficiencies for network traffic
- Disadvantages
  - Does not use the live journals
  - Is not ongoing without special management
  - Fault tolerance requires additional management

# Transforms

- "ETL", ("Extract, Transform, and Load") has sometimes been hot topic.
- What people mean by "transform" is defined by what they are trying to accomplish.
- This section discusses what the Loader can do relative to changing data or data models.

# Simple Data Transforms

- The output conversion parameter of the output keyword uses
  - Text – convert input numerics to strings
  - Trim – remove trailing spaces from string data
  - Additional options that are target specific.
- Values to use if a column is NULL may be defined with the MapColumn keyword
  - Care is required with the Oracle target in combining trim and valueifnull.
- Date format may be defined to any OpenVMS date format with the date_format keyword.
  - Applies to XML and Tuxedo targets
  - Loader sends dates to Oracle targets appropriately for the version of Oracle used.
- The DBkey – generally binary character string – can be defined as BigInt
  - BigInt may be more readable for support purposes with Rdb and Oracle targets
  - BigInt is required for JDBC

# Adding Data

- The Loader includes the keyword virtualcolumn for adding materialized data to existing tables.
- The Loader includes the keyword virtualtable for adding an entire table of materialized data.
- There are over 20 virtual columns that can be added.

# Using Materialized Data

- There can be many different reasons for using these keywords.
  - For merging databases
    - JCCLML_constant
  - For adding timing information
    - Transaction_start_time
    - Transaction_commit_time
    - Transmission_date_time
  - For target performance tuning
    - Randomvalue
    - Modvalue
  - For auditing
    - JCCLML_username
    - Action
    - Table_name

# Easy Data Model Changes

- Tables can be excluded from the target.
- Columns can be excluded from the target.
- Changes can be made based upon the "action."
  - Replication – Replicate all tables or a subset of tables, all columns or a subset of columns. (insert, update, delete actions)
  - Rollup – Write all tables or a subset of tables, all columns or a subset of columns *without deleting the last version of a row to appear in the source.* (insert, update, nodelete actions)
  - Audit – Write all tables or a subset of tables, all columns or a subset of columns *without overwriting any row.* (insert, noupdate, nodelete actions)
  - Other
- Combinations of options can be used.
- Multiple targets can be used.

# Why Data Models Change

- Data models designed ("normalized") for the best support of information storage and retrieval may not be ideal for
  - Reports
  - Decision Support
  - ODS (Operational Data Store)
  - Data Warehousing
  - Specific tools
- Corporate and departmental mergers can bring different focuses together.
- Data models may have been built on an incomplete understanding or the business may have changed.
- Technology may support approaches not previously available.

# Data Model Changes and the Loader

- We will examine several example scenarios.
  - One source to many targets
  - Multiple sources to one target
  - Processing queue challenge
  - Warehousing before deleting
- Please interrupt with your own questions and challenges
- We can also discuss other architectural challenges that don't necessarily involve data model changes
  - Rdb – Oracle coordination
    - Example presented at the Rdb Forums
    - New options with recent JCC development work
  - Occasionally connected databases
  - Additional processing in the target

# One Source to Multiple Targets

- Example:  CSP (customer service provider) data in the source is to be divided by customer type for the target tables.
  - Some, but not all, of the other attributes will be appropriate for each customer type.
  - Account number remains the key in all source and target tables.
- Solution:  Use MapTable and MapFilter.
  - MapTable establishes each of the source to target mappings.
  - MapFilter uses the customer type to filter out all rows that do not have the customer type that is wanted in a particular target table.

# One Source – Many Targets

| Source Table with Combined Data | | | | | | | |
|---|---|---|---|---|---|---|---|
| Acct # | Acct Type | A | B | C | m | n | Balance |
| 1 | REG | xxxxxxx | xxxx | xxxxx | x | xxxxxxx | xxxxxxx |
| 2 | COM | xxxxxxx | xxxx | xxxxx | x | xxxxxxx | xxxxxxx |
| 3 | COM | xxxxxxx | xxxx | xxxxx | x | xxxxxxx | xxxxxxx |
| 4 | REG | xxxxxxx | xxxx | xxxxx | x | xxxxxxx | xxxxxxx |
| 5 | REG | xxxxxxx | xxxx | xxxxx | x | xxxxxxx | xxxxxxx |

| Target Table – REG Accounts | | | | |
|---|---|---|---|---|
| Acct # | A | B | C | Balance |
| 1 | xxxxxxx | xxxx | xxxxx | xxxxxxx |
| 4 | xxxxxxx | xxxx | xxxxx | xxxxxxx |
| 5 | xxxxxxx | xxxx | xxxxx | xxxxxxx |

| Target Table – COM Accounts | | | |
|---|---|---|---|
| Acct # | m | n | Balance |
| 2 | x | xxxxxxx | xxxxxxx |
| 3 | x | xxxxxxx | xxxxxxx |

# One Source – Many Targets

- Note that the one source could also be divided into many target *databases*.
- For example, a field that identifies a region can be used to divide a source database into multiple regional databases.

# Multiple Sources to One Target

- Combining multiple sources presents additional issues.
- Consider the following
  - Unique keys
  - Possibly duplicate keys
  - Table differences
  - Key column differences

# Primary Key Access in the Target

- If you have source tables with the same metadata definition, they may be combined in a target table
    - if the row keys are unique
- Consider the CSP example.
    - If you had the target tables, you could create the source tables because the Acct #'s are unique.
    - You would lose information, if you did not re-create the account type.

# Many Sources – One Target

| Source Table – REG Accounts | | | | |
|---|---|---|---|---|
| Acct # | A | B | C | Balance |
| 1 | xxxxxxx | xxxx | xxxxx | xxxxxxx |
| 4 | xxxxxxx | xxxx | xxxxx | xxxxxxx |
| 5 | xxxxxxx | xxxx | xxxxx | xxxxxxx |

| Source Table – COM Accounts | | | |
|---|---|---|---|
| Acct # | m | n | Balance |
| 2 | x | xxxxxxx | xxxxxxx |
| 3 | x | xxxxxxx | xxxxxxx |

| Target Table with Combined Data | | | | | | | |
|---|---|---|---|---|---|---|---|
| Acct # | | A | B | C | m | n | Balance |
| 1 | | xxxxxxx | xxxx | xxxxx | | | xxxxxxx |
| 2 | | | | | x | xxxxxxx | xxxxxxx |
| 3 | | | | | x | xxxxxxx | xxxxxxx |
| 4 | | xxxxxxx | xxxx | xxxxx | | | xxxxxxx |
| 5 | | xxxxxxx | xxxx | xxxxx | | | xxxxxxx |

# Duplicate Keys

- Duplicate keys in the target provide no way to update or delete rows.
- If source tables to be combined *may* have duplicate keys → it is necessary to create a unique key.
- The most likely way to create a unique key is to add a column to the key that identifies which database is the source of the row.
  - This is done with the materialized column JCCLML_CONSTANT.
- Examples:
  - In the CSP example, if the acct #'s were not unique, adding a column to the table and the key called acct type and giving it a value based on the database of origin would be sufficient.
  - When regional databases are to be combined, adding a column to the table and the key called Region provides the basis of a unique key.

# Table Differences

- The target table can have columns for which one of the source tables does not have data
  - Provided the columns that differ are *NOT* key columns.
- Examples
  - The CSP mapping from REG and COM sources will have some values that are missing in each row.
    - Note that you *can* provide the acct type column with a materialized value.
  - If you are mapping truck data and car data into a vehicle table, there will be some vehicle columns that are not needed for each type of vehicle.

# Key Differences

- There may be a temptation to combine data in one target table for which you cannot name a consistent primary key.
  - In our CSP example, you could define
    - Acct # as the key for the mapping from the table for REG data and
    - n as the key for the mappings from the table for COM data
- This mapping is not supported.
- The resulting target table is unlikely to be useful for other tools.

# Processing Queue Challenge

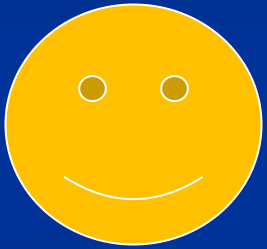
- Goal:
  - Create queues for further processing.
  - Inserts, updates, and deletes should be batched by the hour in which they occurred.
- Solution:
  - Create 24 target tables
  - Materialize a timestamp
  - Filter on the timestamp to place the data in the correct table

# Warehousing Before Deleting

- Goal:
  - Keep the production database small.
  - Do not throw away data.
- Solution:
  - Capture all the data in a target database.
  - Use NOdelete.
  - Add a timestamp, action, username, or other materialized values.

# Changes on Multiple Databases

- Goal:
  - Provide a tool that manages data on another database.
  - Maintain most of the operations on an Rdb database.
  - Prevent data anomalies from updating the same data with both.



We can update this.

- Solution 1 for an Oracle target:
  - Define an Rdb database as the authority.
  - Use database links to write the tool's input to a linked Oracle table in the Rdb database.
  - Use the LogMiner and Loader to keep the Oracle database updated.
- Solution 2 for an SQL Server target:
  - Create a custom procedure to replicate changes from the Loader target to the Rdb source

# Occasionally Connected Databases

- Goals:
  - Reduce network traffic on WAN link
  - Provide local database for reporting
  - Freeze data at a specific point-in-time (yesterday's close, for example)
- Solution:
  - Nightly quiet point AIJ backup
  - Run Oracle Rdb LogMiner on source computer
  - Copy LogMiner output file to target computer
  - Run Loader in Copy mode on copied LogMiner output

# Additional Processing on the Target

- Issue: There are cases for which the Loader does not have the information necessary to provide the transforms desired.
  - Need to make decisions based on a join
  - Need to maintain summary information
  - Need data type conversion not available through Loader
  - Others
- Solutions:
  - Triggers on the target
  - Processing on the target as outlined by the changes on multiple databases example
  - Data conversion with triggers or additional processing
  - Processing as in the queue example

# Column Dependence

- Goal
  - Only update the target table when certain columns change values.
- Fact
  - The Loader gets a row from the LogMiner whenever any portion of the row is updated.
- Implementation without change to source application
  - On the source, add after update trigger to record before and after values for necessary columns to audit table
  - Configure Loader to write the target table from the audit table
  - Create filter on table so that table is only written on desired changes
  - Source audit table can be deleted or truncated, as necessary, depending on needs

# Primary Key

- Issue
  - I have tables that have no primary key or a primary key that changes value.
  - I do not want to use originating_dbkey.
- Implementation without change to source application
  - Add identity column to source table
  - Configure Loader to use identity column as MapKey
  - In case of a database reorganization, you will need to do some maintenance for the identity column.
    - $ define rdms$set_flags auto_override can be used to re-load data preserving the original values

# Comment

- Note that, for the last two examples, the emphasis has been on no change in the existing source applications.
- Sometimes, the focus must be no change to the metadata of the source database.
- Sometimes, there must be no metadata change to the target or the applications that use the target.
- Sometimes, the focus is on solving the issue with the least effort.
- Each issue needs to be solved in context.

# Advanced Schema Mapping Summary

- The Loader supports mapping a source to a target with a different schema and/or different tuning.
- Rule of thumb: The Loader can do the mapping if for each source column, you provide
  - Target table and column
  - Primary key to identify the row
- Examples
  - Rollup of regional databases
  - Normalization correction by subsetting the data with filtering
  - Audits
  - Your issue?

Topic 10

# Problem Resolution

# Complexity Meter

- **Keeping it simple**
  - The JCC LogMiner Loader may be fairly easily established for simple replication.
  - The defaults for many of the keywords and other choices will be useful for many applications.
  - The most important things to know are that the Loader cannot process AIJ files that have been removed from the system and cannot find rows in the target (to update) without a key.
- **Making it complex**
  - The Loader is often used in very complex architectures demanding a great many "knobs" for setting its behavior.
  - The Loader is often used in demanding applications in which the performance tuning options are critical.
  - The Loader is, increasingly, asked to translate between different data models.
- **Adding Sanity**
  - The Loader has extensive aids to the DBA and extensive monitoring.
  - New features support testing your application in ways that have seldom been possible before.

# How to Solve Issues

- Learn
  - Training
  - Documentation
- Examine the logs
- Experience
  - Because we see some of the same issues over and over …
    - We invent tools or change the documentation.
    - We present "lab exercises" for your analysis.
- Ask

# Memory – Issue

I started a Continuous LogMiner Loader session on 11-DEC-2008 that was moving data from one Rdb database to another Rdb database on the same OpenVMS node.  It failed last Sunday morning (4-JAN-2009 05:42:15) because of quota problems:

```
4-JAN-2009 05:41:18.22 20605904 LML PROD_COPY_D  %comc_va_write: Unable to allocate memory for buffer
4-JAN-2009 05:41:18.24 20605904 LML PROD_COPY_D
4-JAN-2009 05:41:18.24 20605904 LML PROD_COPY_D  %dba_buffer_input: unable to write VA for input buffer.
4-JAN-2009 05:41:18.24 20605904 LML PROD_COPY_D
4-JAN-2009 05:41:18.24 20605904 LML PROD_COPY_D   %COMC-E-NO_MEMORY,  Unable to allocate memory for the virtual array.
```

My first question is, what quotas should I be concerned with and what's considered a reasonable quota for the process running the LML?  Current settings:

| | |
|---|---|
| WSdef: | 4096 |
| WSquo: | 8192 |
| WSextent: | 204800 |

# Memory – Response

In the full documentation, we document the possibility of using a 64-bit memory model for the Loader.

The solution involves two steps

- ❑ Ensure that you have sufficient page file quota for the account running the Loader. When I say sufficient, I mean BIG!!!
- ❑ Turn on 64-bit memory addressing in the loader ($ define JCC_COMC_VA_MEMORY_MODEL 1)

So how big is big? 2 million pages = 1 gig of memory, enough to completely map P0 address space. However, with 64-bit addressing we can use 4 thousand million times more. So you will want page file quotas in the multiple millions of pages. You may have to experiment. If you do not yet use 2 million you might try raising to that value (and you won't have to use the 64-bit model) if you already are above 2 million then try adding a zero.

Remember, we may require real space in the page file to get this to go through.

# OLD! AIJ – Issue

I restored the AIJ’s from a backup created on the evening of 5-JAN-2009. This AIJ backup should include all transactions from the evening of 2-JAN-2009 through the evening of 5-JAN-2009. The backup sequence number for the first AIJ backed up is 9801. When I attempt to restart, the CLM fails stating:

```
6-JAN-2009 13:49:01.00 20528CEE CLM PROD_COPY_D  %RMU-I-LOGOPNAIJ, opened journal file V403:[LOGMINER.AIJ]MAIN$54836-2108-9801.AIJ;1 at  6-JAN-2009 13:49:01.00
6-JAN-2009 13:49:01.03 20528CEE CLM PROD_COPY_D  %RMU-F-AIJSEQAFT, incorrect AIJ file sequence 9801 when 9759 was expected
6-JAN-2009 13:49:01.03 20528CEE CLM PROD_COPY_D  %RMU-F-FTL_RMU, Fatal error for RMU operation at  6-JAN-2009 13:49:01.03
```

The sequence it’s looking for is about 2 weeks old, dating back to something like 23-DEC-2008. My first thought was that the whole process was really backed up but when I query my target database I find transactions as late as 4-JAN-2009 4:18.

That leads to my second question, why would the CLM be looking for such an old journal sequence number when it was nearly caught up?

# OLD! AIJ – Response

- At JCC and our client sites, we think we found a bug that involves an unpleasant interaction between AIJ backups and the LogMiner execution. If the LogMiner is started while AIJ backup is running then AIJ backup has the ability to “pull the rug” out from the LogMiner. That is, it will copy the journal and zero the journal while the LogMiner is running.
- The LogMiner encounters the zeroed blocks and skips to the next journal just fine, but it is possible that some data is not processed by LogMiner. This means that any transactions that were committed in the section of journal not processed will be considered by the LogMiner to be “open” and therefore the AERCP cannot be updated. This can be the source of old AERCPs.
- If you have this issue, you may be forced to start either in backup journals or the live journals if you really are caught up.
- Do you remember if you started a Loader family session just after an AIJ backup kicked off?
- We are waiting for Oracle to resolv s.

Fixed!

# Slow Target – Issue

- I am replicating an Rdb database to a SQL Server.
- I have a problem that one table is very slow. According to the JCC statistics screen, 98% of the time is in the target.
- Is it possible to see the Target SQL statement?

# Slow Target – Response

- Generally, a slow update is a sign of a missing index in the target.
  - To help identify the table with LML 3.2.0 or later, add "logging~output~trace" to your control file.
  - This will write a significant amount of information to the log file including the SQL statements.
  - JCC does not recommend using this in production, unless diagnosing a problem.

# Requirement for Consistent Source Database Definition

- Consistent database definition is cited as a requirement for LML operation.
- Changing the source metadata for columns and tables that are not touched – directly or indirectly – in the LogMiner operations does not impact LogMiner or Loader definitions.
- Changing the source metadata for columns and tables used by the Loader
  - Requires care to ensure that the Loader runs against a defined set of metadata.
  - Can be accomplished with the steps on the next slide.

# Changing Metadata Used by the Loader

- Shutdown the application.
- Process all existing AIJs.
- Backup AIJs.
- Shutdown the Loader.
- Make the metadata changes.
- Update the Loader Control File and/or LogMiner options file.
- If necessary for replication, update the target metadata.
- Move or rename the backup AIJs so they will not confuse the issue with old definitions.
- Restart CLML in the live journal.
- Restart the application.

Topic 11

# Support

# Support Options

- Basic support
  - One year of Basic support is bundled with the license fee.
  - Basic support includes
    - New releases during the support period.
    - Access to JCC Consultants, Developers, and Architects during JCC business hours.
- GOLD support
  - Includes Basic support
  - Extends the availability to 24 X 7
- More in-depth architectural questions or direct involvement may be addressed with consulting support.
  - On-site
  - Remote

# Documentation

- Is in English
- Includes details of keywords for Control File
- Is indexed
- Has detailed Table of Contents
- Has FAQ (Frequently Asked Questions)
- Is available from our FTP site

# Presentations

- Multiple presentations on the Loader have been created.
- Some of these focus on application architectures.
- These are available on the JCC FTP site.

# Ready?

- You have a copy of the Loader kit on your CD.
- You have a temporary license key on your CD and in this handout.
- You have the handout in hardcopy and the complete (3.1) documentation on the CD.
- Have fun!
- Contact JCC_LMLoader@JCC.com with questions.

# In the Forum

- JCC LML for Oracle Sources

This is what we have been smiling about.