

# Using the Oracle Logminer With the JCC Loader To Reorganize a Database

**Jeffrey S. Jalbert**  
**Thomas H. Musson,**  
**Keith W. Hare**  
**JCC Consulting, Inc.**

# Abstract

The LogMiner capabilities recently added to Rdb provide a platform to build uniquely new architectures for managing and deploying Rdb databases. Applying this capability requires new tools to properly utilize these capabilities.

The JCC LogMiner loader is this tool.

This presentation will focus on the loader and share our experience in using this product to completely reorganize a significant (50 GB) production database (not counting snapshots) while requiring minimal down time.

The presentation will also discuss planned features that will allow the loader to maintain data in a reporting oracle 8i database and other data stores to be synchronized to a transactional Rdb database.

# Introduction – Requirements

We wanted to accomplish three goals:

- Reorganize databases with minimal disruption to production
- Replicate changes from a production database to a target database without impacting the production database. The target database could be:
  - Rdb database
  - Oracle database
- Provide an API into callable routines that can format and transmit formatted Rdb data into user-specific external data stores

# What We Will Discuss

- LogMiner for Rdb
- JCC LogMiner loader
- JCC LogMiner loader methodology
- How to use the tool
- Other applications of the technology
- Case study of Rdb LogMiner and JCC loader performance

# LogMiner For Rdb

- What is LogMiner?
- What isn't LogMiner?
- How to use LogMiner
- LogMiner restrictions

# What Is LogMiner?

- Extract changed *tables* from AIJ backup file
- Access to AIJ file contents:
  - Oracle does not publish internal AIJ format
  - Command line interface
- Use AIJ contents without performing recovery
- High performance *changed data extraction*
- Addresses extraction, not loading
- Journal based replication... which does not require the target database to be precisely the same as the source database

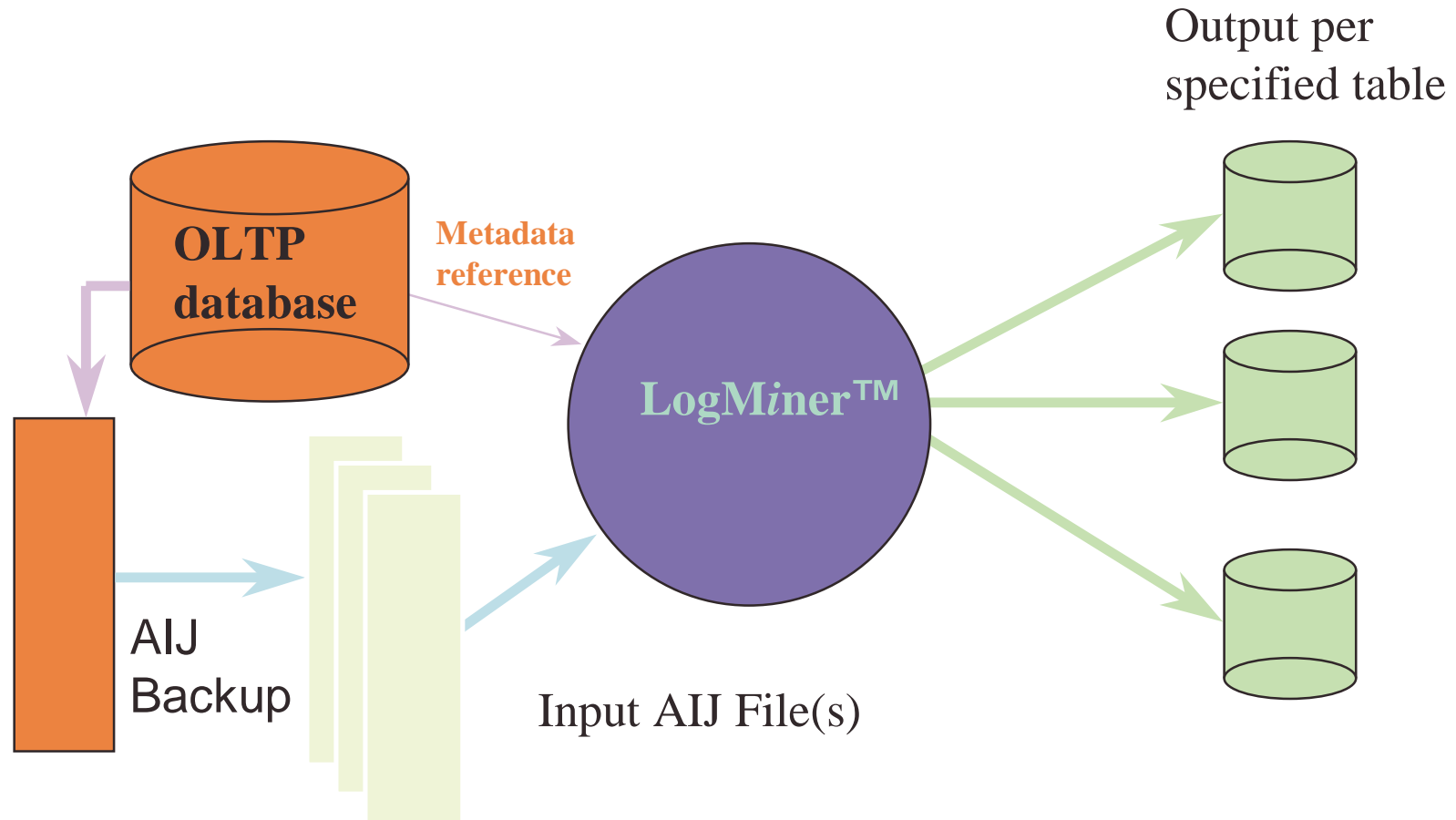
# What Is LogMiner?

- Most information is already in the journal
  - Additional information stored - ignored by recovery
  - System relations used to retrieve metadata
- Minimal impact to production database
  - Single, quick metadata retrieval
    - Tables in mixed areas must be marked in the area inventory
  - Data retrieved from backed up AIJ
  - AIJ size *will* increase
    - Deleted rows are now written to the AIJ (excluding truncate table, etc.)
    - Large delete jobs (monthly purge, etc.) may experience noticeable increases in AIJ I/O
  - No application changes required. Everything done by Rdb

# What Isn't LogMiner ?

- Data loader
  - The data extracted from AIJ backup files by LogMiner can have several targets; None of them are a database
- Data unloaded is not in the format of the table requested
  - Additional information has been added to the record by LogMiner
- Real time
  - Data is extracted from AIJ *backups*

# Extracting Data From AIJ



# Extract Operation

- All specified tables extracted at once (one pass through AIJs)
- Multiple output streams are possible
  - One may cause multiple tables or all tables to be delivered to one output file
  - One may skip some tables
- One or more input AIJ files (processed sequentially)
- Output record order within transaction not predictable
- Only committed transactions are output
  - Output in transaction commit sequence

# What Does LogMiner Output?

- Data from user tables
- Inserts and modifies
  - ‘Final’ record content per transaction
- Deletes
  - Record content just prior to delete
- Committed transactions
  - Rolled-back transactions ignored
- Final form of row returned
  - Transaction that inserts and deletes a row results in delete record being extracted
  - Row modified many times in one transaction results in a single modify output record

# What Does LogMiner Output?

ACTION	“M” or “D”
TABLE_NAME	table name string
RECORD_ID	relation ID
RECORD_LEN	length of data record
NBV_LEN	null bit vector count
LDBK	logical DBKey
START_TAD	date/time of transaction start
COMMIT_TAD	date/time of transaction commit
TSN	transaction sequence number
RECORD_VER	metadata version of record
RECORD	data record contents
RECORD_NBV	null bit indicator array (1 bit per field)

# LogMiner Restrictions

- Blobs (segmented strings) are not supported
- Metadata changes are not allowed within an AIJ
- No quiet point AIJ backups
  - Can process a sequence that start and end with a quiet point
- COMPUTED BY columns are not stored in the AIJ
- DROP or TRUNCATE do not include details in the AIJ
- Optimized AIJs, the logminer works on transaction boundaries
- Vertical record partitioning (VRP)

# How to Use LogMiner

- Easy as 1, 2, 3...
  1. Enable LogMiner
  2. Backup AIJ(s)
  3. Unload AIJ(s)

# Enable / Disable LogMiner

- Logminer and AIJ must be enabled for
  - Logging of delete contents
  - Logging on transaction start time
- Exclusive database access
- Backup AIJ & database after change

```
RMU /SET LogMiner <rootfile> -  
[ /ENABLE ][ /DISABLE ]
```

# Command Line Interface

```
$ RMU /UNLOAD /AFTER_JOURNAL <rootfile>
    @<aij> [, <aij2>...]
/TABLE = (NAME = tnam1, OUTPUT = <ofill>,
          FORMAT={TEXT|BINARY},
          {RECORD_DEFINITION=<rnam1> |
           CONTROL=<cnam1>},
          TABLE_DEFINITION=<snam1>)
/OPTIONS = (FILE = <file>)
/[NO]TRACE
/[NO]LOG
/EXTEND_SIZE=<n>
/SORT_WORKFILES=<n>
/IO_BUFFERS=<n>
/OUTPUT=<file>
/SINCE=<tad>
/BEFORE=<tad>
/SELECT={START_TRANSACTION | COMMIT_TRANSACTION}
/STATISTICS_INTERVAL=<n>
```

# Options File

- Specify tables and outputs
- In addition to command line /TABLE qualifiers

```
TABLE=tblnam1,OUTPUT=ofil1
```

```
TABLE=tblnam2,OUTPUT=ofil2
```

```
TABLE=tblnam3,CALLBACK_MODULE=x,CALLBACK_ROUTINE=y
```

```
TABLE=tblnam4,OUTPUT=ofil4,RECORD_DEFINITION=rfil4
```

```
•  
•  
•
```

# Security / Required Privileges

- RMU /unload /after\_journal
  - RMU\$dump
- RMU /SET logminer
  - RMU\$BACKUP, RMU\$RESTORE or RMU\$ALTER
- All data is in AIJ → all tables visible when extraction allowed

## Example → Extract 4 Tables

```
$ RMU /UNLOAD /AFTER_JOURNAL MYDB.RDB MYAIJ.AIJ -  
      /TABLE = (NAME = SALE, OUTPUT = SALEDATA.DAT) -  
      /TABLE = (NAME = EMPL, OUTPUT = EMPLDATA.DAT) -  
      /TABLE = (NAME = CUST, OUTPUT = CUSTDATA.DAT) -  
      /TABLE = (NAME = SHIP, OUTPUT = SHIPDATA.DAT)
```

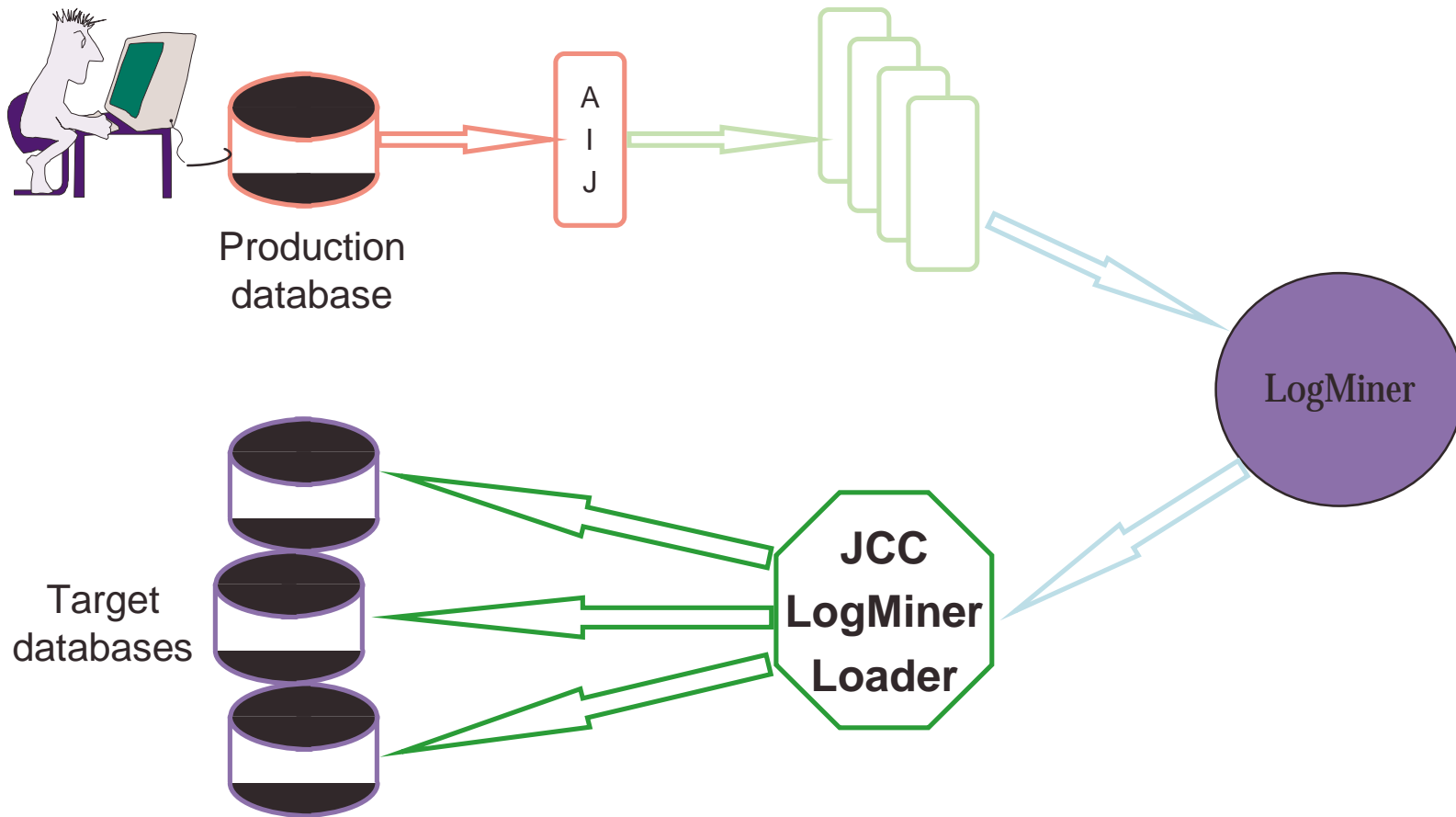
---

```
Table "SALE" : 16334 records written  
Table "EMPL" : 933 records written  
Table "CUST" : 5221 records written  
Table "SHIP" : 57993 records written
```

# JCC LogMiner Loader

- Reads output from LogMiner for Rdb
  - All requested tables to the same output stream
- Applies to target database in transactionally consistent manner
  - Transaction based commit interval specified in control file
- Uses SQL interface to target database
  - Let Rdb maintain the necessary structures (indices, SPAM, ABM)
- Maintains high-water mark information to enable restart from failure
- Deadlock retry
  - Buffers input records since last checkpoint

# Where Does It Fit?



# JCC LogMiner Loader Methodology

- Identifying the record structure
- Uniquely identifying records
  - Primary key
  - Unique index
  - Original DB key
- Dynamic SQL
- Target database structure
  - Constraints
  - Triggers
  - Indices

# Identifying The Record Structure

- Record output from LogMiner for Rdb is variable length
  - 74 bytes fixed LM information
  - Remaining is variable length depending on the table
  - Very wide tables may require multiple records, RMS record size limitation
- Control file to load the column data type and record layout for each table
  - Allows wide control of the actions that the Loader performs
  - DCL/SQL script generates initialization file automatically
    - Makes guesses at unique columns

# Unique or Not?

- How does one update or delete a row in a database if there is no way to uniquely select that record?
- All tables loaded by the JCC LogMiner Loader must either:
  - Have a unique column or set of columns
  - Be insert only

# Uniquely Identifying Records

- Primary key
  - Only if application does not modify primary key columns
- Unique index
  - Only if indexed columns can not change
- Original DBKey
  - This is the DBKey of the row in the source database
  - Utilized and maintained by the JCC LogMiner Loader
  - Catch-all for those tables that do not have primary keys, unique indices, or that have columns within those are updated
  - Assumes that Original DB Keys are not reused
    - Valid only for limited periods of time, usually

# Original DBKey

- In cases where the table has
  - No primary key
  - No unique indices
  - *Not* insert-only
- We create a column in the target database table and populate it
- Could also be done with RMU unload/RMU load

```
SQL> alter table add originating_dbkey char(8);
```

```
SQL> update table set originating_dbkey = dbkey;
```

```
SQL> create unique index table_t_01 on table  
      (originating_dbkey) ...
```

# SQL Interface

- Question: what is the most efficient way to write an SQL statement for Rdb such that an insert, update, or delete is completed?
- Possible answers:
  - Stored procedure, one per table
    - Static; Maintenance headache if tables change; Inflexible
  - Individual insert, update, deletes
    - Static; Maintenance headache if tables change; Inflexible
    - Dynamic; Three compiles; No maintenance
    - Multiple messages between Rdb and the Loader
  - Multi-statement procedure
    - Dynamic; Compiled only once; No maintenance

# Dynamic Multi-statement SQL

- A single multi-statement procedure to insert, update, or delete a row in a table based on the LM Action field
- Handles 'M' being insert or modify by testing for existence
- Add original dbkey column reference if necessary

# A “Simple” Example

```
begin
declare :act char(1);
declare :ind1 integer;
declare :ACCOUNT_ID INTEGER;
declare :ind2 integer;
declare :SEQ_NUM INTEGER;
declare :ind3 integer;
declare :CODE_1 CHAR(4);
declare :ind4 integer;
declare :CODE_2 CHAR(4);
declare :ind5 integer;
declare :AMT INTEGER(2);
```

# A “Simple” Example

```
set :act = ?;
set :ind1 = ?; if (:ind1 = 1) then set :ACCOUNT_ID = NULL; else set :ACCOUNT_ID = ?; end if;
set :ind2 = ?; if (:ind2 = 1) then set :SEQ_NUM = NULL; else set :SEQ_NUM = ?; end if;
set :ind3 = ?; if (:ind3 = 1) then set :CODE_1 = NULL; else set :CODE_1 = ?; end if;
set :ind4 = ?; if (:ind4 = 1) then set :CODE_2 = NULL; else set :CODE_2 = ?; end if;
set :ind5 = ?; if (:ind5 = 1) then set :AMT = NULL; else set :AMT = ?; end if;
trace 'action = ',:act;
trace 'ACCOUNT_ID = ',nvl(:ACCOUNT_ID,'NULL');
trace 'SEQ_NUM = ',nvl(:SEQ_NUM,'NULL');
trace 'CODE_1 = ',nvl(:CODE_1,'NULL');
trace 'CODE_2 = ',nvl(:CODE_2,'NULL');
trace 'AMT = ',nvl(:AMT,'NULL');
```

# A “Simple” Example

```
set :t_dbkey = (select dbkey from update_db.T2 where ACCOUNT_ID = :ACCOUNT_ID and SEQ_NUM = :SEQ_NUM );
if (:act = 'M') then
if (:t_dbkey is Not Null) then
update update_db.T2 set CODE_1 = :CODE_1, CODE_2 = :CODE_2, AMT = :AMT where dbkey = :t_dbkey;
else
insert into update_db.T2 (ACCOUNT_ID, SEQ_NUM, CODE_1, CODE_2, AMT) values
(:ACCOUNT_ID, :SEQ_NUM, :CODE_1, :CODE_2, :AMT);
end if;
elseif (:act = 'D' and :t_dbkey is Not Null) then
delete from update_db.T2 where dbkey = :t.dbkey ;
end if;
```

# Order of Application of Changes

- If target database is to be slaved to source database
- If application code performs updates as:
  - Delete
  - Insert
- The data being read by the loader is in unpredictable order and the insert could occur before the delete
- If a unique index is placed on the target database table for, e.g. hash placement reasons, this will cause the loader to fail
- The loader stores all input for a transaction in memory and applies all delete operations prior to update operations

# Target Database Structure

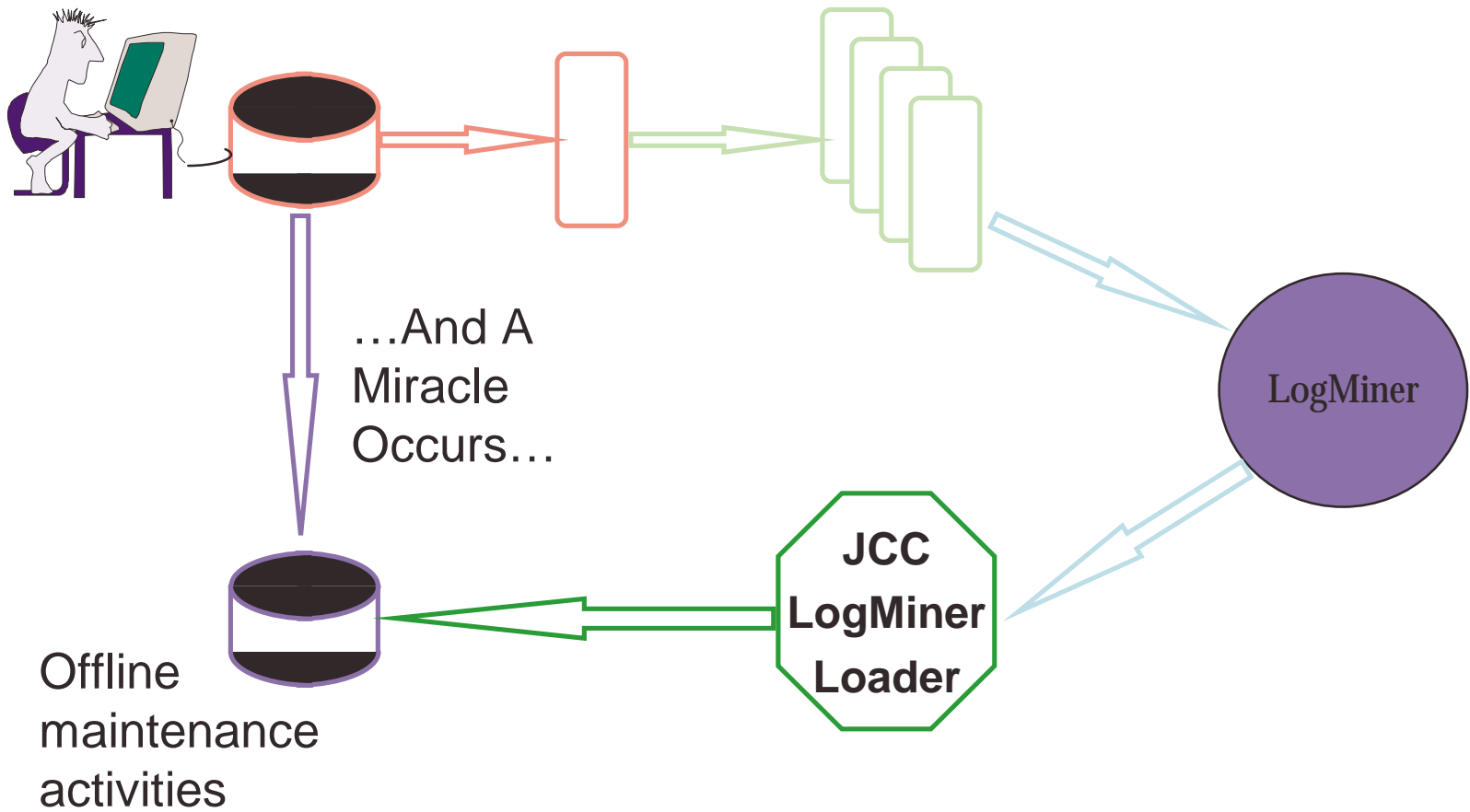
- Constraints
  - Can't guarantee records in same order as original transaction
  - Must relax foreign key constraints and check constraints that perform a similar function (enforced on source database)
- Triggers
  - Effects of triggered actions are represented in AIJ
  - Drop triggers and replace prior to production release
- Indices
  - Unique columns must be honored
  - Anything you want
  - Indices to support original dbkey column (where needed)
- Column data types can be anything that is compatible

# How To Use the Tools

- Must specify only a single output for LogMiner
  - Need to have all records in a transaction in the same input stream (to maintain transactional consistency)
  - For a complete rebuild can create parallel streams each with subsets of the tables
- Target database can be local or remote
- Will automatically detect if last execution failed and will restart with the record after the last successful checkpoint

```
$ jcc_logminer_loader [-i input_name] [-o output_name  
-t output_type] "-I" control_file_name
```

# Minimal Downtime



# The Eightfold Way of Database Restructuring

1. Restore a backup of the production database
2. Create target empty database with only hashed placement indexes. Add necessary DBKey columns if needed
3. Unload views containing DBKeys if necessary and load target database
4. AIJ backup(s) on production database, load data changes using LogMiner for Rdb and JCC LogMiner Loader in tandem
5. Repeat step 4 till downtime is reduced to acceptable value
6. Add remaining indexes
7. Repeat step 5
8. Add constraints and triggers

# Other Applications of the Technology

- Maintaining a reporting database
  - Differing indexing
- Loading data into a Data Warehouse
  - Delete records can be transformed into updates
- Maintaining multiple warm standby databases in remote locations
  - Same as the minimal downtime scenario
  - Could be Oracle 8i
- Maintain down level versions of database
  - Rdb 6.0 minimum for multi-statement SQL

# Current Status

- All testing to date has been with
  - Rdb V7.0-61 (with latest LogMiner patches) as the target database
  - File input
- Performance is much better than the application that wrote the data
  - No rolled back transactions
  - Only deal with data being changed (no extraneous selects)
  - No application logic to enforce
  - Restricted subset of tables for each stream possible
  - Large commit interval (more updates/commit)

# Futures for JCC LogMiner Loader

- Client/Server
  - Persistent server, transient client
  - VMS pipe input
- Oracle 8 target database
  - Uses SQL\*net interface from VMS
  - Will support redefinition of table and column names
  - Will support subset of columns
- External callouts to support application specific requirements for external data store
- Multiple target databases
- Handle multiple record versions
- Access database directly for metadata description
- Suggestions?

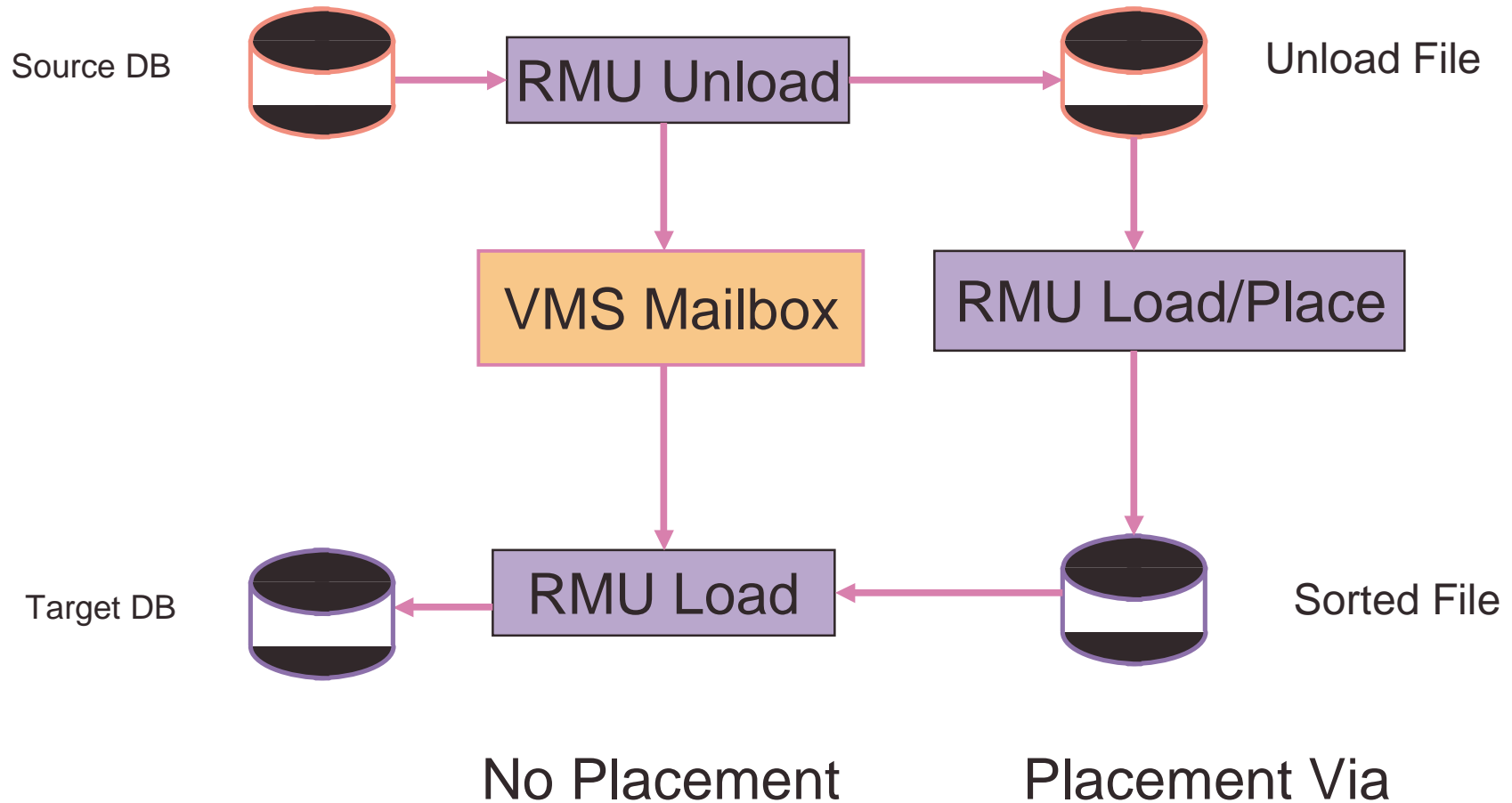
# Case Study

- 50 Gb database supporting a ½ million customer business
- 23 tables are placed in mixed format areas. Other 150 tables are in uniform areas
- Largest placed table is over 60 million rows, multiple 20-million row tables
- Number of customers has increased much faster than projected
  - RMU/MOVE area resulted in excessively large database areas and buffer sizes
    - Database buffer size is now 24 blocks.
    - Don't feel that increasing that is reasonable anymore
    - Would prefer 12 block buffers for this TP application

# New Target Database

- Sized for 1 million customers
  - Many tables are horizontally partitioned
  - One mixed area sized for full customer load
- Buffer size back to standard 12 blocks
- Rebuild is highly parallelized
  - Keep the CPUs busy
- For rebuild uses local buffers
  - All streams perform orthogonal work
- Will be switched back to system space global buffers for production

# Unload Load Optimization



# Case Study — Optimized Computing Environment

- Hardware is a dual-processor ES40 with 2 CPUs
- 5 Gb memory, all available for this activity
  - Create local memory disk for root file and RUJ writes during unload-load activity
- Locally attached fast wide SCSI disks on 6 controllers
  - Formed host-based RAID sets across controllers
- Reorganized database on two separate stripe sets
- Source for unload on separate stripe set
- Sort work files on 4 separate spindles

# Sizes of AIJ and Recovery

- AIJ File size was 997,866 blocks long
- Unloaded a total of 649,014 modify records and 55,160 records
- Recover job indicated that it recovered 329,825 transactions
- LogMiner unload job ran for 7:19.20 and used 1:19.64
- JCC LogMiner Loader ran in
  - 13 parallel streams
    - Partitioned large tables into their own streams
  - 60 minutes for longest stream
  - Entire process was CPU bound

# Parallel Rebuild

- Unload-load jobs run in 5 parallel streams
  - One table per stream
  - System becomes CPU bound
  - Required 7 ½ hours to perform unload-load on all tables with only hash placement indexes as necessary
- LogMiner Loader ran in 13 parallel streams
  - Used originating db-key methodology because application undocumented
  - Required addition of indexes on originating DBKey columns
    - Index build ran CPU bound in 14 parallel streams
    - Required 106 minutes to complete

# Timing for Traditional Export/Import

Export Database	150 Minutes
Import Database	1,050 Minutes
Release to Production requires about 20 hours	

# Timing for Traditional Unload-Reload

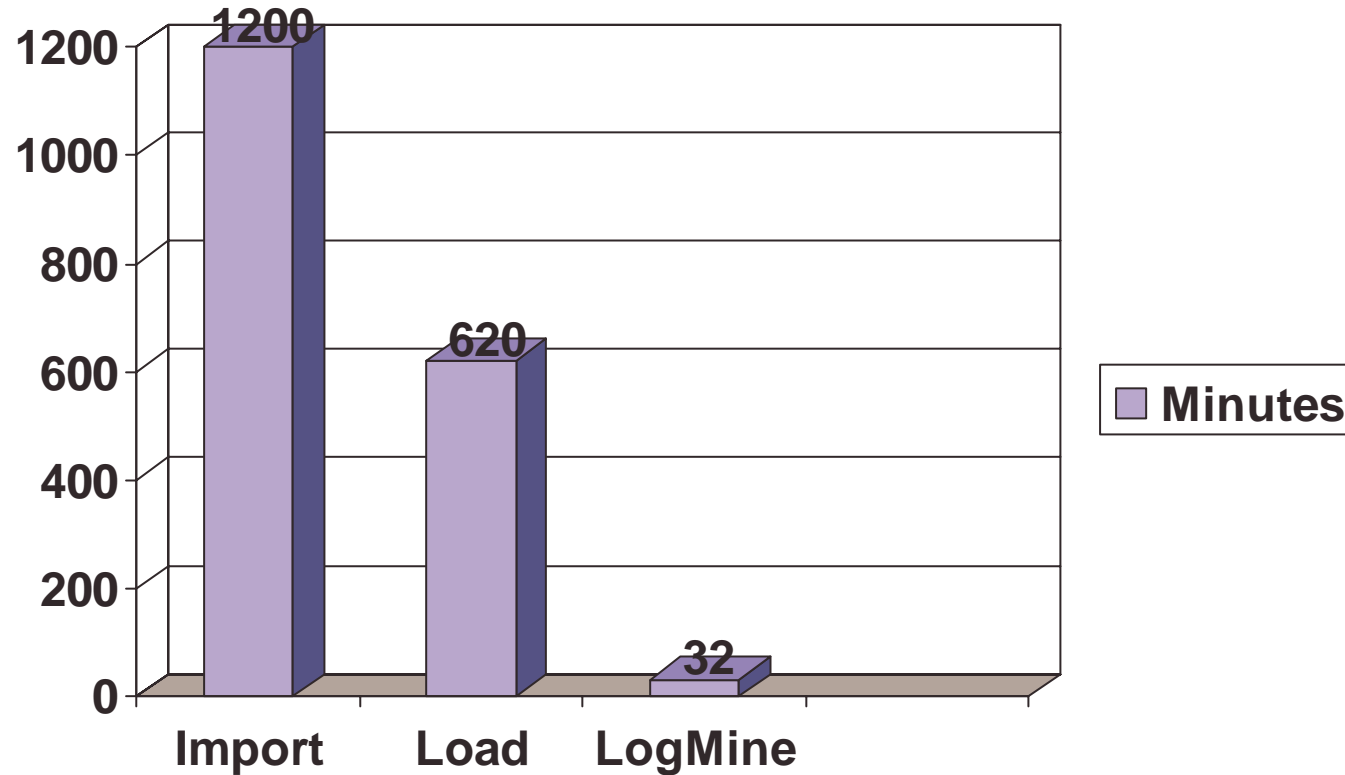
Import Empty database	25 Minutes
Restore Backup DB	120 Minutes
Unload-Load All Tables	440 Minutes
Add Normal Indexes	180 Minutes
Release to Production requires about 10 1/2 hours	

# Timing for LogMiner Loader Approach

Import Empty database	25 Minutes
Restore backup DB	120 Minutes
Unload Load all tables	420 Minutes
Add DBKey Indices	106 Minutes
LogMine 1 day of AIJ	7 Minutes
Run JCC Loader	60 Minutes
Apply indexes	180 Minutes
Mine and rerun loader	30 Minutes
Triggers and constraints, DBKey indexes	2 Minutes

production  
is  
down

# Downtime Comparison Between Methods



# Verifying the Load

- Proof of correctness done by comparing all rows of all tables
  - Loaded database
  - Recovered database
- Use view which sorts table by all columns
- VMS differences of unload files (text format with special NULL string)
- Ran through 2 VMS mailboxes, one for each database
- Required about 20 wall-clock (and about 35 CPU) hours to verify all tables

# Observations

- Reduced down time makes reorganization of database possible. Downtime well within interval for normal database maintenance
- Highly parallel operations allow system throughput to be maximized
- Elimination of file writes to disk improves system utilization
- Use of striped disks improves throughput of rebuild
- Use of memory disk for root file and small RUJ writes improves throughput
- Comparing all the data via VMS difference utility is very expensive.
  - We may write our own code to see whether something specialized may do better

# Conclusions

- LogMiner for Rdb is a valuable tool that gives us access to the data stored in AIJ format
- JCC LogMiner Loader enhances the capabilities of the LogMiner
  - To support database reorganizations with minimal disruption and
  - Replicate changes to other database(s)
  - Replicate changes to other data stores
- Provides new dimensions for deploying applications

# Field Test

- The JCC LogMiner Loader is in field test
- We are soliciting a limited number of sites who will actively test this product
- Final production release is expected this summer
- Please direct inquiries to me at the E-mail address

[Jeff@jcc.com](mailto:Jeff@jcc.com)

# Rdb List Server

- Join the international discussion group about Rdb.
- Send E-mail to:

[oraclerdb-request@jcc.com](mailto:oraclerdb-request@jcc.com)

- In the body of the message include the command:

**Subscribe OracleRdb**

# Thanks Rdb Engineering!

The authors want to thank Rdb engineering for their close cooperation and support while we were trying the new LogMiner code and for adding features we all need...



# Questions

