Speeding Oracle Utility Operations

A Single-Pass Reorg and ETL Method for Large Oracle Tables

CoSort / IRI, Inc.

MIS White Paper Series, Vol. 5

Abstract:
According to a 2006 Gartner report, Oracle remains the world's most widely used database. At the same time, data growth continues unabated. As a result, database administrators (DBAs) and data warehouse architects face ever increasing volume, and shrinking processing windows. Strict service level agreement (SLA) commitments and business intelligence cycles demand that database operations such as reorganizations, and data warehouse extract, transform, and load (ETL) operations run as fast and efficiently as possible. The basic methods presented here show how to address these issues without impacting on-line operations or major budgets.

FACT for Oracle Version 2
CoSort FAst extraCT

Introduction

Oracle databases allow IT departments around the world to build robust and valuable data warehouses. With the decreasing cost of storage, companies are now, more than ever, able to store an increasing amount of data, and need to transform that data into actionable business intelligence. Successful data warehousing systems are those able to process increasingly larger volumes of data within the same, and sometimes smaller, production time frames.

The amount of digital data that companies have generated in the last five years is three times the amount generated in the last 35. A regional telecommunications provider can generate 200 million "call detail" records everyday. In the United States alone, more than 100 billion credit card transactions are made annually at an annual growth rate of 15%. Even larger sources of data lie in the on-line transactions of internet commerce. Turning this data into information quickly requires efficient Extract, Transform, and Load (ETL) operations.

Making Oracle-related ETL (and database reorg) processing more efficient can start with the simple premise of optimizing the performance of each step, and the movement of data between the steps. This document explains the importance of speed in each case, and recommends off-line (external) processing to improve speed. File-based staging unburdens the database, ETL and BI tools in which high volume transforms need not, and should not, occur. At the end of the document is a step-by-step example of IRI's methodology for faster extraction of data into this file system, combined with large transforms, advanced reporting, and pre-sorted bulk loads.

The Extraction Bottleneck

Most data warehousing projects are centered around solving the same fundamental business problem – taking data from disparate sources across the enterprise, aggregating and merging them into a single schema capable of supporting web access, reporting and business intelligence tools. This process must be automated, controlled, reliable – and fast.

Today's Oracle DBAs and data warehouse architects face increasing table volumes and processing bottlenecks. Service-level agreement (SLA) commitments and shrinking production windows require fast database reorg and data warehouse extract, transform, and load (ETL) solutions that deliver both high performance and database availability.

Obtaining timed and filtered extracts from relational databases can be prohibitively slow. The larger the source, and the more business logic applied to the extraction, the slower extracts become. You may have only a very limited operational 'downtime' during which you can run an extraction, and for synchronization reasons, all extracts may need to run simultaneously. Extraction speed is therefore a key issue in many data warehousing projects. With the growing popularity of ETL tools, the temptation is to simply plug one of these tools into the production database – using ODBC as the transport method. This would appear to be an excellent solution, as the developer can key all the business logic into SQL queries, and let the database report back the data shape required for the warehouse.

Unfortunately, open connectors like ODBC and JDBC, as well as the default database unload methods (like SQL SELECT to SPOOL) often take too much time to configure, and are inherently slow for high volumes (even with parallel hints, and other tuning methods). Beyond a million rows, a specialized, high-performance extraction tool can dramatically mitigate the unload bottleneck, and facilitate downstream processing of the data.

The Flat File Opportunity

Data warehouse experts like Ralph Kimball recommend staging large volumes of data in flat files[1]. That is because the fastest sort, join, convert, aggregation, report, and reload processing can occur through the file system, where unencumbered files (and the tools that manipulate them) have access to disk, memory, and thread resources unfettered by application layers. Data stored in proprietary formats and structured DBMS/ETL systems are optimized for queries and graphical mappings, not for bulk transformations or loads.

Database and data warehousing technology expert Chuck Kelly further concurs. "Being able to sort at a flat file level as opposed to at the ETL level, offers performance gains every time. Using a flat file, in some cases, will be faster than using relational structures." Furthermore, "flat files are one of the few forms [of data] that all ETL processes and database systems can read. This provides a de facto standard in moving data between different database systems."

Using flat files, a database administrator (DBA) or a data warehouse engineer bypasses the ETL tool overhead of database connections, network bandwidth, and stage-related I/Os. Some ETL processes read a row, run it though a single transformation process, and write a row to the next. However, quickly extracting data to flat files, and even pushing or pulling the file to be executed on a local machine, if necessary, can create a faster processing opportunity than more complex and expensive applications.

Extraction to Flat Files

Oracle's export and Data Pump utilities output data in binary format, which is ideal for moving, copying, or backing up database tables. Given the value of file system processing, however, a logical goal is to find the most efficient way to extract the data out of Oracle into flat files, perform all the required transformations, and send the data to its various targets.

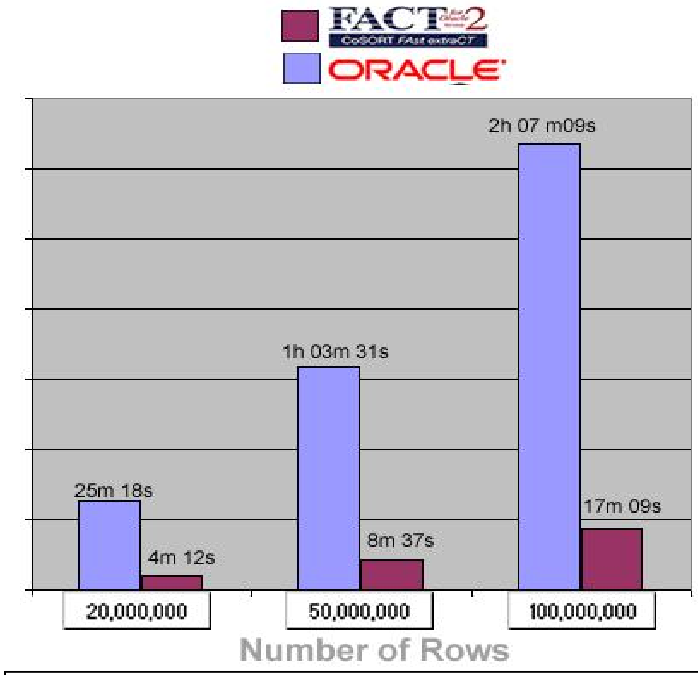
Oracle guidelines state that "the most basic technique for extracting data is to execute a SQL query in SQL*Plus and direct the output of the query to a file." The sample SQL script below will create a flat file called, country_city.log, with a pipe (|) delimiter between column values. The file will contain a list of the cities in the US from the tables 'countries' and 'customers'[2]:

```
SET echo off SET pagesize 0
SPOOL country_city.log
SELECT distinct t1.country_name ||'|'|| t2.cust_city
FROM countries t1, customers t2
WHERE t1.country_id = t2.country_id
AND t1.country_name= 'United States of America';
SPOOL off
```

Unfortunately, SQL*Plus is designed for access to, and manipulation of, small amounts of Oracle data, and as such, it carries overhead that makes the movement of bulk data inefficient.

Fortunately, the same SQL SELECT statement can be issued within a CoSort Fast Extract (FACT) configuration file (in text based .ini or .xml format). A side-by-side comparison of the Oracle Spool results against the OCI-based, parallel FACT tool shows the difference in unload speeds:

[1] Kimball, Ralph. "Is Data Staging Relational? Or does it have more to do with sequential processing?" DBMS Magazine, April 1998
[2] http://download-uk.oracle.com/docs/cd/B10501_01/server.920/a96520/extract.htm - 12623



HP-UX B.11.11, Oracle 9.2, 50-byte VARCHAR
HP9000 L2000-44. 4 PA 8500 CPUs @ 440MHz. 8GB RAM

*FAst extraCT (FACT) for Oracle, Unload-Only Benchmarks vs. Oracle SQL*Plus SPOOL*

Fast Extract (FACT) for Oracle

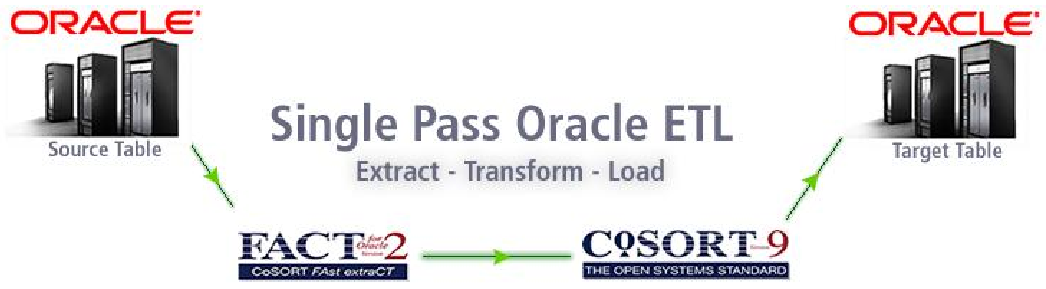
FACT rapidly extracts tables to in a variety of portable, flat file formats. It allows DBAs and ETL users to unload tables with more speed and reformatting functionality than other tools. Unlike the proprietary format of Oracle's export or Data Pump utilities (which require its import counterparts to load the data back), flat output files work with all databases and applications.

During unloads, FACT uses the table description to write the extract file's field layouts into the data definition file (DDF) format used by

- CoSort Sort Control Language, SortCL, for transformation, protection, and reporting
- The RowGen test data generation tool
- The Meta Integration Model Bridge (MIMB) for conversion into other tool metadata

The metadata for SortCL can be referenced in job specification files that define one or more manipulations (such as a reorg or pre-load sort on the longest index key) and reports.

FACT also creates the control file metadata for SQL*Loader re-load operations:



File System Transformations

Once the data is piped from the unload utility, it can be piped directly into a high-volume file processor like CoSort's Sort Control Language (SortCL) program to simultaneously perform a number of critical selection, manipulation, protection and/or reporting functions. The external file manipulation functions that SortCL can perform and combine include:

Filter	At the byte, field and record level
Segment	Conditional (include/omit) selection
Sort	Multiple keys, directions, sequences
Merge	Pre-sorted files
Join	Sorted or un-sorted files over many conditions
Re-map	Resize, reposition, and realign fields
Convert	Change data types (e.g. EBCDIC⇔ASCII, Packed⇔Numeric)
Re-format / Interchange	Convert between file formats (e.g. Text⇔XML, VS⇔RS, Micro Focus ISAM⇔ACUCOBOL-GT Vision, LDIF⇔CSV, MFVL⇔Text)
Aggregate	Parallel roll-up and drill-down sum, min, max, average, and count values. Accumulation. Ranking.
Calculate	Expressions and functions across detail and summary rows
Sub-string	Perl-compatible regular expression (PCRE) logic for pattern matching and other intra-field manipulations
Validate	Check and realign characters to specifications (e.g. "isdigit")
Sequence	For indexing and loading operations
Lookup	Discrete field substitutions, pseudonymization, etc. using "SET" file field dimensions
Protect	Encrypt data at the field level and audit data security measures, plus anonymization, de-identification, filtering, pseudonymization
Report	Custom-formatted, segmented detail, delta, and summary targets
Transform	Custom field-level user functions (e.g. data quality libraries)
Log	XML audit trail records job specs for compliance verification, etc.

In the case of Fast Extract (FACT) for Oracle, the data definitions for SortCL are created automatically, for use within one or more SortCL job specification files that accomplish the functions above as needed. In addition to the pipe or flat file extracts from Oracle, SortCL can integrate other data sources from other legacy index and flat files at the same time, mapping across the field names common to each input, and producing new, integrated views of data and meaningful reports that provide actionable business intelligence.

By piping the data directly from FACT into SortCL (which accepts standard input), you can save the I/O of intermediate transfer files. Similarly, SortCL can output sorted data into a named pipe, or flat file, to accelerate loads into SQL*Loader and other RDBMS load utilities.

Pre-Sorted Loads

Per Oracle's Server Utilities Guide, pre-sorting improves the performance of direct path loads and queries, and minimizes the temporary storage requirements during the load. Oracle's internal block management is vastly improved by pre-sorting; the sorted sustained rate is roughly twice that of the un-sorted sustained rate. In other words, SQL*Loader loads large data sets faster when they are pre-sorted.

According to Chuck Kelley, "having the data processed in sorted order with files being merged/compared will provide impressive performance gains. If the table that you are inserting into is clustered via a specific column or group of columns, then having them in sorted order will increase load performance. With many data warehouses using data partitioning, being able to break the file into multiple load streams based on the partitioning scheme will allow multiple streams of load to be executed simultaneously.[3]"

To speed loads into Oracle:

1. Extract the table(s) to flat file(s) using CoSort's FAst extraCT (FACT) tool for Oracle, SQL select, or another unload utility;

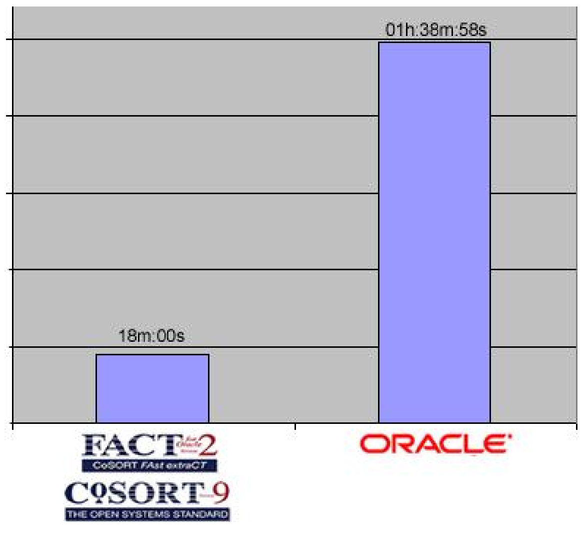
2. Sort the flat file (or the pipe from FACT) on the longest index field, using a high-performance sorting engine like CoSort's SortCL program;

*3. Use SQL*LOADER to load the sorted output file or named pipe*[4], with the argument DIRECT=TRUE; and,*

4. To create indexes during the load, use the clause SORTED INDEXES in the load control file. To create the indexes after the load, use SQL CREATE INDEX with the NOSORT option.

Combined ETL and Reorg Processing

By running the optimized extraction, transformation, and loading steps together, data warehouse ETL and database reorg operations can be improved dramatically.



Extract-Transform-Load Benchmark

`fact | sortcl | sqlldr`
`00h:18m:00s`

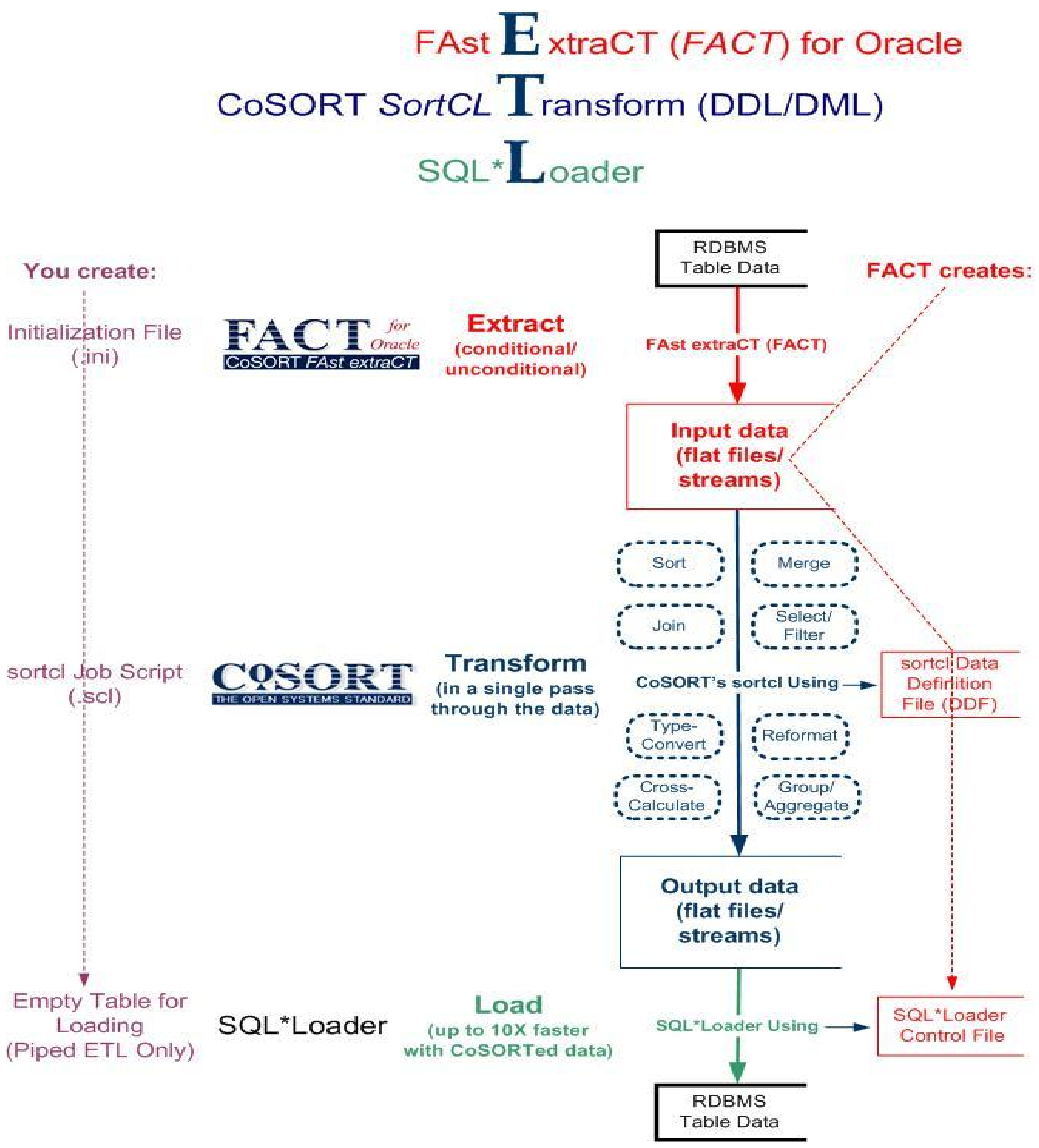
`vs.`

`Oracle insert into`
`01h:38m:58s`
`(select * ... order by)`

Software Versions: FACT v1.12, CoSort v8.1, and Oracle 9i SQL*Plus
Source Data: ~ 50 million, 50-byte rows (2.32 GB) sorted on 1 key
Test Hardware: ia64 hp server rx5670, 2 x1GhZ CPUs, 32GB RAM, HP-UX 11.23

[3] *Kelley has worked in some facet of the design and implementation phase of more than 50 data warehouses and data marts*
[4] *According to data warehousing expert Dan Linstedt, "loading through a pipe directly into an RDBMS bulk-load facility can be slower than staging to a flat file and blasting the bulk-load with buffering mechanisms" if the data flowing through that pipe is too large for O/S resources to handle. If your data is too "fat" for your pipe, direct the pre-sorted output to a flat file and load it in a separate step.*

This schematic explains the operational relationship between the elements discussed:


FAst ExtraCT (FACT) for Oracle
CoSORT SortCL Transform (DDL/DML)
SQL*Loader
You create:
Initialization File (.ini)
FACT for Oracle CoSORT FAst extraCT
Extract (conditional/ unconditional)
RDBMS Table Data
FACT creates:
FAst extraCT (FACT)
Input data (flat files/ streams)
Sort
Merge
Join
Select/ Filter
sortcl Job Script (.scl)
CoSORT THE OPEN SYSTEMS STANDARD
Transform (in a single pass through the data)
CoSORT's sortcl Using
sortcl Data Definition File (DDF)
Type-Convert
Reformat
Cross-Calculate
Group/ Aggregate
Output data (flat files/ streams)
Empty Table for Loading (Piped ETL Only)
SQL*Loader
Load (up to 10X faster with CoSORTed data)
SQL*Loader Using
SQL*Loader Control File
RDBMS Table Data

The operational sample that follows demonstrates these concepts in action.

Step-by-Step Command Line Example

{Ater starting Oracle on Linux, we preview a fact table called 'orders', in order by "ORDER"}

```
ORDER CUSTOMER                            EMPLOYEE          SHIP_DATE FREIGHT
----- ----------------------------------- ----------------- --------- ---------
11073 Pericles Comidas clacas             Fuller, Andrew              24.95
11074 Simons bistro                       King, Robert                18.44
11075 Richter Supermarkt                  Callahan, Laura             6.19
11076 Bon app                             Peacock, Margaret           38.28
11077 Rattlesnake Canyon Grocery          Davolio, Nancy               8.53
```

The purpose of this ETL example is to extract orders, sort it in customer order, and load it into another table called orders_sorted. Each optimized E, T, and L step is also combined into one I/O stream.

{We have created a single-pass ETL command, "script", that runs from the shell, batch, etc.

```
[cosort@demo example]$ cat script
rm -rd stdout.dat ; fact –c orders.ini ; mkfifo stdout.dat ; fact orders.ini |
sortcl /spec=sort_orders.scl & sqlldr scott/tiger control=stdout.ctl DIRECT=TRUE
```

where -c is a FACT execution option to create only the .ddf and .ctl (without extraction) based on the orders.ini config file. This is necessary so that when the extraction for the piped operation starts, sortcl can find its .ddf and SQL*Loader can find its .ctl immediately. }

```
[cosort@demo example]$ sh script
```

{Following is the screen output from the above; notice an inter-mixed process display}

```
                   ----------------------------------------------

                           FAst extraCT for Oracle v2.31 B8
                           Copyright 2007 CoSort Korea Ltd.

                   ----------------------------------------------

   Check License ......... OK

                   You are running in trial license mode

                   - ONLY legacy synchronous I/O mode applied (slower)
                   - Your license duration is only 1 month

   Initiating .................. OK

   SQL*Loader: Release 9.2.0.1.0 - Production on Tue Oct 9 16:14:02 2007

   Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.


   Connected to:
   Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
   With the Partitioning, OLAP and Oracle Data Mining options
   JServer Release 9.2.0.1.0 - Production

   Exporting To Standard Out ...
   CoSort Ver 9.1.1 D8040325-1715 (c) 1978-2007 IRI, Inc.   www.cosort.com
   EST 04:14:02 PM Tue Oct 9 16:14:02 2007. #07118.9101    Monitor Level 5
      <00:00:00.00> event (66): CoSort() process begins
      <00:00:00.13> event (59): stdin infile opened
      <00:00:00.14> event (63): ./CS00004145 workfile opened
      <00:00:00.14> event (63): ./CS01004145 workfile opened
   OK
   Releasing all resources ...... OK
   Reporting and Writing log .... OK
   Refer to report file : stdout.log
   Successfully Completed.

      <00:00:00.18> event (65): ¦0, 0 processed
      <00:00:00.18> event (60): stdin infile closed
      <00:00:01.06> event (64): ./CS00004145 workfile deleted
      <00:00:01.06> event (64): ./CS01004145 workfile deleted
      <00:00:01.06> event (67): CoSort() process ends

Load completed - logical record count 830.
```

**{And that's it … the entire E-T-L operation completed in 1 second …
with several additional, simultaneous Transforms/reports created too … see below}**

{Now, let's prove it worked … we log back into Oracle to see the new table, orders_sorted}

```
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

SQL> select * from orders_sorted where customer like '%Wols%';

ORDER CUSTOMER                                 EMPLOYEE          SHIP_DATE FREIGHT
----- ---------------------------------------- ----------------- --------- -------
10870 Wolski  Zajazd                           Buchanan, Steven  13-FEB-98  12.04
10998 Wolski  Zajazd                           Callahan, Laura   17-APR-98  20.31
10792 Wolski  Zajazd                           Davolio, Nancy    31-DEC-97  23.79
10374 Wolski  Zajazd                           Davolio, Nancy    09-DEC-96   3.94
11044 Wolski  Zajazd                           Peacock, Margaret 01-MAY-98   8.72
10906 Wolski  Zajazd                           Peacock, Margaret 03-MAR-98  26.29
10611 Wolski  Zajazd                           Suyama, Michael   01-AUG-97  80.65

7 rows selected.

SQL>exit
```

{So, a new (previously-created empty) table called "orders_sorted" has been loaded, in order now by "CUSTOMER" and "EMPLOYEE" rather than by "ORDER". This new index order will speed queries, thanks to CoSort/sortcl, the Transform part of the operation; see sort_orders.scl below.}

The next pages show what else was created, as well as the original ETL configuration files.

{Here is the list of everything created during this 1-second ETL operation}

```
[cosort@demo example]$ ls -lt | more
total 1412
-rw-rw-r--    1 cosort   cosort         2255 Oct   9 16:14 stdout.log
-rw-r--r--    1 cosort   cosort         7230 Oct   9 16:14 orders1.out
-rw-r--r--    1 cosort   cosort           16 Oct   9 16:14 orders2.out
-rw-r--r--    1 cosort   cosort        75030 Oct   9 16:14 orders3.html
-rw-r--r--    1 cosort   cosort        52893 Oct   9 16:14 orders.csv
prw-rw-r--    1 cosort   cosort            0 Oct   9 16:14 stdout.dat
-rw-rw-r--    1 cosort   cosort          363 Oct   9 16:14 stdout.ctl
-rw-rw-r--    1 cosort   cosort          195 Oct   9 16:14 stdout.ddf
-rw-rw-r--    1 cosort   cosort          710 Oct   4 14:27 sqlnet.log
-rwxrwxr-x    1 cosort   cosort          138 Oct   4 14:27 script
-rw-rw-r--    1 cosort   cosort          607 Oct   4 15:50 orders.ini
```

{Below is the FACT configuration file that extracted the data from the original "orders" table in Oracle, converted it to CSV (with a framed employee field), and created the metadata files for CoSort's sortcl and Oracle's SQL*Loader}.

```
[cosort@demo example]$ cat script
rm -rd stdout.dat; mkfifo stdout.dat ; fact -c orders.ini | sortcl /spec=sort_orders.scl
& sqlldr scott/tiger control=stdout.ctl DIRECT=TRUE
```

```
# "orders.ini"
# FACT initialization file for ETL example
# gets run with fact emp.ini

DATABASE=ORACLE
INSTANCE=test
USERID=FACT_MANAGER
PASSWORD=alias
QUERY=SELECT * FROM orders
LOADTABLE=orders_sorted
LOADTYPE=INSERT
# sorted records from table orders will be
# inserted into the newly-created table, orders_sorted
OUTFILE=stdout
# extracted output records to be streamed via unanamed
# pipe into CoSort; sortcl ddf /FILE name will be stdin
VARIABLE
DELIM=,
# FACT can also reformat output in CSV format, etc.
FRAMEFIELD=EMPLOYEE
# Specifes the name of the column to be framed
FRAMECHAR="
# Framed column to be enclosed in double quotes
FETCHSIZE=auto
DATAEXT=.dat
CTLEXT=.ctl
DDFEXT=.ddf
REPORTEXT=.log
```

{This is the CoSort Sort Control Language (SortCL) Data Definition File, automatically created by FACT via the above config file, and which will be incorporated by reference into the CoSort SortCL (transformation) job script that was separately created, sort_orders.scl (shown below). This once-created, but centralized metadata is available for re-use, making this, and another transformation or RowGen test data generation program, easier to code.}

```
[cosort@demo example]$ cat stdout.ddf

/FILE=stdin
        /FIELD=(ORDER_ID,POS=1,SEP=',')
        /FIELD=(CUSTOMER,POS=2,SEP=',')
        /FIELD=(EMPLOYEE,POS=3,SEP=',',FRAME='"')
        /FIELD=(SHIP_DATE,POS=4,SEP=',')
        /FIELD=(FREIGHT,POS=5,SEP=',',NUMERIC)
```

{Notice Oracle's column names and data types – VARCHAR to ASCII, for example. FACT has also output the records in CSV format and protected the EMPLOYEE field's column.}

```
[cosort@demo example]$ cat sort_orders.scl
```

{Below is the CoSort sort control language (sortcl) job specification (DML) file which was created in advance to handle the pre-load sort transform (as well as other transforms and reports) while data's passing through …)}

[cosort@demo example]$ cat script
rm -rd stdout.dat; mkfifo stdout.dat ; fact –c orders.ini | ***sortcl /spec=sort_orders.scl***
& sqlldr scott/tiger control=stdout.ctl DIRECT=TRUE

```
/SPEC=stdout.ddf                 # calls metadata created by FACT (orders.ini)
/INFILE=stdin                    # streamedrecords from FACT (unnamed pipe)
/SORT
         /KEY=CUSTOMER           # primary key field for new Oracle table
         /KEY=EMPLOYEE           # secondary sort key, ascending
/OUTFILE=stdout.dat              # streamed records to SQL*Loader (named pipe)
```

```
[cosort@demo example]$ cat stdout.ctl
```

{This is the Oracle SQL*Loader (sqlldr) control file automatically created by FACT via the above config file, and which runs as the final Load step.}

[cosort@demo example]$ cat script
rm -rd stdout.dat; ***mkfifo stdout.dat*** *; fact –c orders.ini* | *sortcl /spec=sort_orders.scl*
& sqlldr scott/tiger control=stdout.ctl DIRECT=TRUE

```
LOAD DATA
INFILE 'stdout.dat'

INTO TABLE orders_sorted
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' AND '"'
TRAILING NULLCOLS
( ORDER_ID                         char
, CUSTOMER                         char
, EMPLOYEE                         char
, SHIP_DATE                        DATE "SYYYYMMDDHH24MISS"
, FREIGHT                          DECIMAL EXTERNAL
)
```

{Note that FACT extracted the fixed table into CSV format; this output file is in the same format as the records loaded into Oracle via stdout.dat.}

```
[cosort@demo example]$ more orders1.out
```

{This is the first of several additional, optional transformations performed simultaneously with the ETL operation. This formatted report is in the same customer/employee order, with aggregation.}

```
Customer                           Employee             Max     Min     Avg     Ct
------------------------------------------------------------------------------
Alfreds Futterkiste                Davolio, Nancy       69.53   1.21    37.60   6
Ana Trujillo Emparedados y helados King, Robert         43.90   1.61    24.36   4
Antonio Moreno Taqueria            Davolio, Nancy       84.84   4.03    38.36   7
Around the Horn                    Callahan, Laura     146.32   3.04    36.30   13
Berglunds snabbkop                 Buchanan, Steven    244.79   3.50    86.64   18
Blauer See Delikatessen            Callahan, Laura      53.83   0.15    24.04   7
Blondel pe et fils                 Buchanan, Steven    156.66   5.74    56.70   11
Bon app                            Buchanan, Steven    350.64  10.19    79.87   17
Boo Comidas preparadas             Dodsworth, Anne      97.09  16.16    63.72   3
Bottom-Dollar Markets              Davolio, Nancy      243.73   2.40    56.71   14
Bs Beverages                       Davolio, Nancy      123.83   2.17    28.13   10
Cactus Comidas para llevar         Callahan, Laura      31.51   0.33    12.13   6
Centro comercial Moctezuma         Peacock, Margaret     3.25   3.25     3.25   1
Chop-suey Chinese                  Buchanan, Steven     96.65   1.17    45.91   8
Comeio Mineiro                     Callahan, Laura      79.70   0.21    37.56   5
Consolidated Holdings              Callahan, Laura      38.24   6.17    17.87   3
```

```
[cosort@demo example]$ more orders2.out
```

{This is the second of several more optional transformations performed simultaneously with the ETL operation. This target is a just static summary of all freight charges.}

```
Total Freight Charges: 64,942.69
```

```
[cosort@demo example]$ more orders3.html
```

{In this third output, HTML tags were inserted in the sort_orders.scl job specification file in order to produce this web-ready report. The next page shows it in a browser.}

```
<HTML><HEAD>
<TITLE>HTML produced by CoSort's SortCL Transform 4GL/Program</TITLE>
</HEAD>
<BODY><H2>Customer Freight Summary</H2>
<FONT COLOR='RED'>Employee transactions; charges above $100 are shown in
green.
</FONT>
<TABLE CELLPADDING=4 CELLSPACING=1 BORDER COLS=5>
<TR>
<TD><B>Davolio, Nancy</B></TD>
<TD align=right>         $69.53
<TR>
<TD><B>Davolio, Nancy</B></TD>
<TD align=right>         $40.42
<TR>
<TD><B>Leverling, Janet</B></TD>
<TD align=right>          $1.21
<TR>
<TD><B>Peacock, Margaret</B></TD>
<TD align=right>         $61.02
<TR>
<TD><B>Peacock, Margaret</B></TD>
<TD align=right>         $23.94
--More--(0%)
```

{When you open orders3.html in a web browser, here's what you see:}

Customer Freight Summary

Employee transactions; charges above $100 are shown in green.

Davolio, Nancy	$69.53
Davolio, Nancy	$40.42
Leverling, Janet	$1.21
Peacock, Margaret	$61.02
Peacock, Margaret	$23.94
Suyama, Michael	$29.46
Alfreds Futterkiste	**$225.58**
King, Robert	$1.61
Leverling, Janet	$11.99
Leverling, Janet	$43.90
Peacock, Margaret	$39.92
Ana Trujillo Emparedados y helados	**$97.42**

{And so on, to the end of the report …}

Wilman Kala	**$88.41**
Buchanan, Steven	$12.04
Callahan, Laura	$20.31
Davolio, Nancy	$23.79
Davolio, Nancy	$3.94
Peacock, Margaret	$8.72
Peacock, Margaret	$26.29
Suyama, Michael	$80.65
Wolski Zajazd	**$175.74**

By **cosort** as of Oct/09/2007 04:14:03 PM.

{The user name and timestamp were 'conversion specifiers' in sortcl's /FOOTREC statement.}

{Following is the final log from FACT}

```
[cosort@demo example]$ more stdout.log

SQL*Loader: Release 9.2.0.1.0 - Production on Tue Oct 9 16:14:02 2007

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

Control File:   stdout.ctl
Data File:      stdout.dat
  Bad File:     stdout.bad
  Discard File:  none specified

 (Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Continuation:   none specified
Path used:      Direct

Table ORDERS_SORTED, loaded from every logical record.
Insert option in effect for this table: INSERT
TRAILING NULLCOLS option in effect

   Column Name                  Position   Len  Term Encl Datatype
------------------------------ ---------- ----- ---- ---- ----------------
-----
ORDER_ID                            FIRST     *   ,  O(") CHARACTER O(")
CUSTOMER                             NEXT     *   ,  O(") CHARACTER O(")
EMPLOYEE                             NEXT     *   ,  O(") CHARACTER O(")
SHIP_DATE                            NEXT     *   ,  O(") DATE
SYYYYMMDDHH24MISS O(")
FREIGHT                              NEXT     *   ,  O(") CHARACTER O(")

Table ORDERS_SORTED:
  830 Rows successfully loaded.
  0 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.

  Date cache:
   Max Size:      1000
   Entries :       387
   Hits    :       422
   Misses  :         0

Bind array size not used in direct path.
Column array  rows :    5000
Stream buffer bytes:  256000
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:           830
  830 Rows successfully loaded.
  0 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.
```

```
Data cache:
   Max Size:      1000
   Entries :       387
   Hits    :       422
   Misses  :         0

Bind array size not used in direct path.
Column array  rows :    5000
Stream buffer bytes:  256000
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:           830
Total logical records rejected:         0
Total logical records discarded:        0
Total stream buffers loaded by SQL*Loader main thread:        1
Total stream buffers loaded by SQL*Loader load thread:        0

Run began on Tue Oct 09 16:14:02 2007
Run ended on Tue Oct 09 16:14:04 2007

Elapsed time was:     00:00:01.78
CPU time was:         00:00:00.08
```

INNOVATIVE ROUTINES INTERNATIONAL (IRI), INC.
Suite 303, Atlantis Center
2194 Highway A1A
Melbourne, FL 32937-4932 USA
Phone 321-777-8889
Fax 321-777-8886
http://www.cosort.com