AN ALTERNATIVE READ-ONLY TABLE

REPLICATION SYSTEM IN ORACLE DATABASE

YÜKSEKÖĞRÜF‹T‹M HUKUKU
DÖ-dialoguyon Merkezi

MASTER OF SCIENCE THESIS

Comp. Eng. Kenan Ç‹FTÇ‹
(504981187)

Date of submission: 11 June 2001

Date of defence examination: 25 June 2001

Supervisor (Chairman) Assoc. Prof. Dr. Nadia ERDOĞAN

Members of the Examining Committee Assoc. Prof. Dr. Coşkun SÖNMEZ (İTÜ.)

Assoc. Prof. Dr. Tevfik AKGÜN (YTÜ.)

JUNE 2001
ORACLE VERİTABANINDA ALTERNATİF
SALT-OKUNABİLİR TABLO
KOPYALAMA SİSTEMİ

YÜKSEK LİSANS TEZİ
Kenan ÇİFTÇİ
(504981187)

Tezin Enstitüye Verildiği Tarih : 11 Haziran 2001
Tezin Savunulduğu Tarih: 25 Haziran 2001

Tez Danışmanı  Doç. Dr. Nadia ERDOĞAN
Diğer Jüri Üyeleri Doç. Dr. Coşkun SÖNMEZ (İTÜ.)
Doç. Dr. Tevfik AKGÜN (YTÜ.)

HAZİRAN 2001
PREFACE

I express sincere appreciation to Assoc. Prof. Dr. Nadia ERDOĞAN for her guidance throughout the study. To my wife, Arzu, I offer sincere thanks for her unshakable faith in me and her willingness to endure with me the vicissitudes of my endeavors. To my children, Nihal and Didar, I thank them for understanding my frequent absences.

June 2001

Kenan ÇİFTÇİ
CONTENTS

PREFACE .................................................................................................................. ii

LIST OF TABLES ........................................................................................................ v

ÖZET ............................................................................................................................. vi

ABSTRACT .................................................................................................................. vii

1. INTRODUCTION ..................................................................................................... 1

2. REPLICATION CONCEPT ...................................................................................... 2
  2.1. What Is Replication ............................................................................................ 2
    2.1.1. Basic Replication .......................................................................................... 2
    2.1.2. Advanced Replication ................................................................................ 3
  2.2. Basic Replication Concepts .............................................................................. 4
    2.2.1. Read-Only Table Snapshots ......................................................................... 5
    2.2.2. Read-Only Snapshot Architecture ............................................................... 5
    2.2.3. Snapshot Refreshes ..................................................................................... 7
    2.2.4. Snapshot Logs ............................................................................................. 8
    2.2.5. Snapshot Refresh Groups ........................................................................... 8
    2.2.6. Automatic Snapshot Refreshes ..................................................................... 9
    2.2.7. Automatic Refresh Intervals ....................................................................... 9
    2.2.8. Refresh Types ............................................................................................ 9
    2.2.9. SNP Background Processes ......................................................................... 10
    2.2.10. Manual Snapshot Refreshes ....................................................................... 10
    2.2.11. Other Basic Replication Options ................................................................ 10
  2.3. Advanced Replication Concepts ...................................................................... 11
    2.3.1. Advanced Replication Configurations ....................................................... 12
    2.3.2. Hybrid Configurations .............................................................................. 14
    2.3.3. Advanced Replication Objects, Groups, Sites, and Catalogs ..................... 15
    2.3.4. Oracle's Advanced Replication Architecture ............................................. 17
    2.3.5. Replication Conflicts ................................................................................ 21
    2.3.6. Replicated Data Models and Conflicts ....................................................... 22
    2.3.7. Conflict Avoidance Techniques ................................................................ 23

3. DATA WAREHOUSE CONCEPTS ........................................................................... 26
  3.1. What is Data Warehouse ................................................................................... 26
    3.1.1. Subject Oriented ......................................................................................... 26
    3.1.2. Integrated .................................................................................................. 26
    3.1.3. Nonvolatile ................................................................................................ 27
    3.1.4. Time Variant ............................................................................................ 27
  3.2. Contrasting a Data Warehouse with an OLTP System ................................... 27
  3.3. Typical Data Warehouse Architecture ........................................................... 28
  3.4. ETT Overview ................................................................................................ 30
    3.4.1. Extraction .................................................................................................. 30
    3.4.2. Transportation ......................................................................................... 34
    3.4.3. Transformation ....................................................................................... 34
4. AN ALTERNATIVE READ-ONLY REPLICATION SYSTEM IMPLEMENTATION ................................................................. 35
   4.1. Capabilities of the new system .................................................................................................................. 35
   4.2. Why do we need an alternative read-only replication system? ............................................................ 36
   4.3. Design of the system ............................................................................................................................... 37
      4.3.1. Master Site Tables and Objects ...................................................................................................... 37
      4.3.2. Replication Site Tables and Objects .............................................................................................. 43
   4.4. Security Problems of the Implementation ............................................................................................ 51
   4.5. Installation of the System .................................................................................................................... 52

5. CONCLUSIONS .............................................................................................................................................. 53

REFERENCES .......................................................................................................................................................... 54

VITA ....................................................................................................................................................................... 56
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2-1</td>
<td>Basic, Read-Only Replication</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2-2</td>
<td>Advanced Replication</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2-3</td>
<td>Information Distribution</td>
<td>4</td>
</tr>
<tr>
<td>Figure 2-4</td>
<td>Information Off-Loading</td>
<td>4</td>
</tr>
<tr>
<td>Figure 2-5</td>
<td>Basic Replication Components: read-only snapshots, master tables,</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2-6</td>
<td>Advanced Replication System Supporting Multiple Points of Update Access</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2-7</td>
<td>Multimaster Replication System</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2-8</td>
<td>Advanced Replication System with Updatable Snapshots</td>
<td>14</td>
</tr>
<tr>
<td>Figure 2-9</td>
<td>Hybrid Configuration</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2-10</td>
<td>Asynchronous Data Replication Mechanisms</td>
<td>18</td>
</tr>
<tr>
<td>Figure 2-11</td>
<td>Dynamic Ownership in an Order Processing System</td>
<td>23</td>
</tr>
<tr>
<td>Figure 3-1</td>
<td>Typical Data Warehouse Architecture</td>
<td>29</td>
</tr>
<tr>
<td>Figure 3-2</td>
<td>Typical Architecture for a Complex Data Warehouse</td>
<td>29</td>
</tr>
<tr>
<td>Figure 4-1</td>
<td>Front-end tool for administering master site specifications</td>
<td>39</td>
</tr>
<tr>
<td>Figure 4-2</td>
<td>Front-end tool for administering replication site specifications</td>
<td>46</td>
</tr>
</tbody>
</table>
ÖZET

ORACLE VERİTABANINDA ALTERNATİF SALT-OKUNABİLİR KOPYALAMA SİSTEMİ

Kenan ÇİFTÇİ


Anahtar Kelimeler: Tablo kopyalama, Veri ambarı
AN ALTERNATIVE READ-ONLY TABLE REPLICATION SYSTEM IN ORACLE DATABASE

Kenan ÇİFTÇİ

In this study using Oracle database an alternative read-only table replication system from one database site to another database site has been developed. Aim of this study is to remove the restrictions in Oracle table replication system to help data warehouse administrator in Extraction, Transportation and translation (ETT) process. In Oracle replication system, user of the system cannot modify the internal objects of the replication but in the new developed system user can easily modify the internal objects to ease ETT process.

Keywords: Table Replication, Data warehouse ETT
1. INTRODUCTION

For years, operational applications and database technology have improved productivity by managing business transactions. Now leading enterprises are taking the next step, using advanced query, reporting, online analytical processing (OLAP) and data mining solutions to apply operational information to decision making. Facts now supplement intuition as analysts, managers and executives use these technologies to make more informed, effective decisions.

An enterprise decision support solution (DSS) integrates cross-departmental data into a coherent whole, enabling all users to work from a common information base. It gives users the ability to work with all of the information in a data warehouse or a subset of information in a datamart. Enterprise DSS can also go beyond the warehouse and integrate data from disparate systems such as relational, legacy or external. This integration enables new applications, such as fact-based selling, activity-based costing and product profitability analysis, to transcend the boundaries of a single department.

To build an enterprise DSS, you need to Extract, Transport and Translate (ETT) operational data into a data warehouse or datamart. Building and continuing to feed DSS with current operational data are time-consuming process. So that we need an automation tool to feed the DSS system. There are some methods to do ETT process which are explained in chapter 3 in detail. Oracle replication system is an alternative for this methods which is explained in chapter 2 in detail. In this thesis, an alternative read-only replication method is developed. The main difference of this method from Oracle replication is that it is an open source-code system and users of this system can change the internal objects of this replication method. A DSS database administrator can use our tool for extraction, transportation and partially transformation process. So this tool makes the ETT process easy. Details of this new alternative read-only replication system is explained in chapter 4 in detail.
2. REPLICATION CONCEPT

In this chapter, replication concept, terminology and architecture are explained.

2.1. What Is Replication

Replication is the process of copying and maintaining schema objects in multiple databases that make up a distributed database system. Replication can improve the performance and protect the availability of applications because alternate data access options exist. For example, an application can normally access a local database rather than a remote server to minimize network traffic and achieve the best performance. Furthermore, the application can continue to function if the local server fails while other servers with replicated data remain accessible. In many commercial databases there are two forms of replication: basic and advanced replication. In this chapter, basic and advanced replication concept of Oracle 8.1.7 is explained detailly.

2.1.1. Basic Replication

With basic replication, data replicas provide read-only access to the table data that originates from a primary or "master" site. Applications can query data from local data replicas to avoid network access regardless of network availability. However, applications throughout the system must access data at the primary site when updates are necessary. Figure 2-1 illustrates basic replication. Oracle Server can support basic, read-only replication environments using read-only table snapshots. Later read-only snapshots in oracle will be explained in this chapter [1].
2.1.2. Advanced Replication

Advanced replication features extend the capabilities of basic read-only replication by allowing applications to update table replicas throughout a replicated database system. With advanced replication, data replicas anywhere in the system can provide both read and update access to a table's data. Participating Oracle database servers automatically work to converge the data of all table replicas, and ensure global transaction consistency and data integrity. Figure 2-2 illustrates advanced replication [1].
2.2. Basic Replication Concepts

Basic replication environments support applications requiring read-only access to table data originating from a primary site.

For example, consider the operations of a large consumer department store chain. In this case, it is critical to ensure that product price information is always available and relatively current and consistent at retail outlets. To achieve these goals, each retail store can have its own copy of product price data that it refreshes nightly from a primary price table. Figure 2-3 illustrates information distribution.

![Figure 2-3 Information Distribution](image)

Basic replication is also useful as a way to replicate entire databases or off-load information. For example, when the performance of high-volume transaction processing systems is critical, it can be advantageous to maintain a duplicate database to isolate the demanding queries of decision support applications like in data warehouses or datamarts. Figure 2-4 illustrates information off-loading from an online transaction processing database to a data warehouse or datamart database. In chapter 3, data warehouse concept and information off-loading are explained detailly [1].

![Figure 2-4 Information Off-Loading](image)
2.2.1. Read-Only Table Snapshots

A read-only table snapshot is a local copy of table data originating from one or more remote master tables. An application can query the data in a read-only table snapshot, but cannot insert, update, or delete rows in the snapshot. Figure 2-5 illustrates the architecture of read-only table snapshots and basic replication [1, 2].

![Diagram of read-only table snapshots and basic replication](image)

Figure 2-5 Basic Replication Components: read-only snapshots, master tables, and snapshot logs

2.2.2. Read-Only Snapshot Architecture

Oracle supports basic data replication with its table snapshot mechanism. The following sections explain the architecture of simple read-only table snapshots.

a) A Snapshot's Defining Query

The logical data structure of table snapshots is defined by a query that references data in one or more remote master tables. A snapshot's defining query determines what data the snapshot will contain.

A snapshot's defining query should be such that each row in the snapshot corresponds directly to a row or a part of a row in a single master table. Specifically,
the defining query of a snapshot should not contain a distinct or aggregate function, a GROUP BY or CONNECT BY clause, join, restricted types of subqueries, or a set operation. The following example shows a simple table snapshot definition.

CREATE SNAPSHOT sales.customers AS
    SELECT * FROM sales.customers@hq.acme.com

Note that in all cases, the defining query of the snapshot must reference all of the primary key columns in the master table.

A snapshot's defining query can include restricted types of subqueries that reference multiple tables to filter rows from the snapshot's master table. A subquery snapshot can be used to create snapshots that "walk up" the many-to-one references from child to parent tables that may involve multiple levels. The following example creates a simple subquery snapshot.

CREATE SNAPSHOT sales.orders AS
    SELECT * FROM sales.orders@hq.acme.com o
    WHERE EXISTS
        ( SELECT c_id FROM sales.customers@hq.acme.com c
            WHERE o.c_id = c.c_id AND zip = 19555);

b) Internal Snapshot Objects

When you create a new, read-only table snapshot, Oracle creates several underlying database objects to support the snapshot. User can not change or access these internal objects structure or semantics. But user can change storage characteristics of the internal objects such as initial extent, next extent sizes or pctfree, pctused parameters of storage.

- Oracle always creates a base table to store snapshot data. The name of a snapshot's base table is SNAPS_snapshotname. User can change only the storage characteristics of the table.
- Oracle creates an index for the primary key in the snapshot base table. The name of the index is the name of the index used to enforce the corresponding PRIMARY KEY constraint at the master site. User can also
change storage characteristics of the index or add additional indexes to improve select command performance of the snapshot.

- Oracle always creates a view for each snapshot. The view takes the name of the snapshot and provides read-only access to the snapshot.
- Oracle might create additional indexes for a snapshot defined with a subquery [1].

2.2.3. Snapshot Refreshes

A snapshot's data does not necessarily match the current data of its master tables. A table snapshot is a transaction-consistent reflection of its master data as that data existed at a specific point in time. To keep a snapshot's data relatively current with the data of its master, Oracle periodically refreshes the snapshot. A snapshot refresh is an efficient batch operation that makes that snapshot reflect a more current state of its master.

You must decide how and when to refresh each snapshot to make it a more current. For example, snapshots stemming from master tables that applications often update usually require frequent refreshes. In contrast, snapshots depending on relatively static master tables usually require infrequent refreshes. In summary, you must analyze application characteristics and requirements to determine appropriate snapshot refresh intervals.

To refresh snapshots, Oracle supports different types of refreshes, "complete" and "fast" snapshot refresh groups, as well as "manual" and "automatic" refreshes.

a) Complete Refreshes

To perform a complete refresh of a snapshot, the server that manages the snapshot executes the snapshot's defining query. The result set of the query replaces the existing snapshot data to refresh the snapshot. Oracle can perform a complete refresh for any snapshot.

b) Fast Refreshes

To perform a fast refresh, the server that manages the snapshot first identifies the changes that occurred in the master since the most recent refresh of the snapshot and then applies them to the snapshot. Fast refreshes are more efficient than
complete refreshes when there are few changes to the master because participating servers and networks replicate less data. Fast refreshes are available for snapshots only when the master table has a snapshot log [1, 3].

2.2.4. Snapshot Logs

When a master table corresponds to one or more snapshots, create a snapshot log for the table so that fast refreshes of the snapshots are an option. A master table’s snapshot log tracks fast refresh data for all corresponding snapshots—only one snapshot log is possible per master table. When a server performs a fast refresh for a snapshot, it uses data in its master table’s snapshot log to refresh the snapshot efficiently. Oracle automatically purges specific refresh data from a snapshot log after all snapshots perform refreshes such that the log data is no longer needed.

When you create a snapshot log for a master table, Oracle creates an underlying table to support the snapshot log. A snapshot log’s table holds the primary keys, timestamps, and optionally the ROWIDs of rows that applications update in the master table. A snapshot log can also contain filter columns to support fast refreshes for snapshots with subqueries. The name of a snapshot log’s table is MLOG$\_master\_table\_name.

2.2.5. Snapshot Refresh Groups

To preserve referential integrity and transaction consistency among the snapshots of several related master tables, Oracle organizes and refreshes each snapshot as part of a refresh group. Oracle refreshes all snapshots in a group as a single operation. After refreshing all snapshots in a refresh group, the snapshot data in the group corresponds to the same transaction-consistent point in time.
2.2.6. Automatic Snapshot Refreshes

When creating a snapshot refresh group, administrators usually configure the group so Oracle automatically refreshes its snapshots. Otherwise, administrators would have to manually refresh the group whenever necessary.

When configuring a refresh group for automatic refreshes, you must:

- Specify a refresh interval for the group.
- Configure the server managing the snapshots with one or more SNP background processes to periodically refresh snapshots that are due for refreshing. (Oracle database management system consists of many background processes. According to Oracle usage, you can configure to start these processes)

2.2.7. Automatic Refresh Intervals

When you create a snapshot refresh group, you can specify an automatic refresh interval for the group. When setting a group's refresh interval, consider the following characteristics:

- The dates or date expressions specifying the refresh interval must evaluate to a future point in time.
- The refresh interval must be greater than the length of time necessary to perform a refresh.
- Relative date expressions evaluate to a point in time relative to the most recent refresh date. If a network or system failure interferes with a scheduled group refresh, the evaluation of a relative date expression could change accordingly.
- Explicit date expressions evaluate to specific points in time, regardless of the most recent refresh date.

2.2.8. Refresh Types

By default, Oracle attempts to perform a fast refresh of each snapshot in a refresh group. If Oracle cannot perform a fast refresh for an individual snapshot, for
example when a master table has no snapshot log, the server performs a complete refresh for the snapshot.

### 2.2.9. SNP Background Processes

Oracle Server's automatic snapshot refresh facility functions by using job queues to schedule the periodic execution of internal system procedures. Job queues require that at least one SNP background process be running. An SNP background process periodically checks the job queue and executes any outstanding jobs. To configure Oracle to start SNP background processes, `job_queue_processes` and `job_queue_interval` initialization parameters must be set in `init.ora`. (In Oracle 7, instead of these parameters `snapshot_refresh_interval` and `snapshot_background_processes` parameters must be used) [4].

### 2.2.10. Manual Snapshot Refreshes

Scheduled, automatic snapshot refreshes may not always be adequate. For example, immediately following a bulk data load into a master table, dependent snapshots will no longer represent the master table's data. Rather than wait for the next scheduled automatic group refreshes, you might want to manually refresh dependent snapshot groups to immediately propagate the new rows of the master table to associated snapshots [5].

### 2.2.11. Other Basic Replication Options

Oracle supports additional basic replication features that can be useful in certain situations:

- Complex Snapshots.
- ROWID Snapshots.

Complex Snapshots:

When the defining query of a snapshot contains a distinct or aggregate function, a `GROUP BY` or `CONNECT BY` clause, join, restricted types of subqueries, or a set
operation, the snapshot is a complex snapshot. The following example is a complex
table snapshot definition.

CREATE SNAPSHOT scott.emp AS
SELECT ename, dname
FROM scott.emp@hq.acme.com a, scott.dept@hq.acme.com b
WHERE a.deptno = b.deptno
ORDER BY dname

The primary disadvantage of a complex snapshot is that Oracle cannot perform a
fast refresh of the snapshot; Oracle only performs complete refreshes of a complex
snapshot. Consequently, use of complex snapshots can affect network performance
during complete snapshot refreshes.

ROWID Snapshots:

Primary key snapshots, as discussed in earlier sections of this chapter, are the
default for Oracle. Oracle bases a primary key snapshot on the primary key of its
master table. Because of this structure, you can:

- Reorganize the master tables of a snapshot without completing a full refresh
  of the snapshot.
- Create a snapshot with a defining query that includes a restricted type of
  subquery.

For backward compatibility only, Oracle also supports ROWID snapshots based on
the physical row identifiers or "ROWIDs" of rows in the master table. Only use
ROWID snapshots for snapshots of master tables in an Oracle7.3 database, and not
when creating new snapshots of master tables in Oracle8 databases [2].

2.3. Advanced Replication Concepts

In advanced replication environments, data replicas anywhere in the system can
provide both read and update access to a table data. Advanced replication is useful
for the deployment of transaction processing applications that operate using
disconnected components. For example, consider the typical sales force automation
system for a life insurance company. Each salesperson must visit customers regularly with a laptop computer and record orders in a personal database while disconnected from the corporate computer network and centralized database system. Upon returning to the office, each salesperson must forward all orders to a centralized, corporate database.

Advanced replication can also be useful to protect the availability of a mission critical database. For example, an advanced replication system can replicate an entire database to establish a failover site should the primary site become unavailable due to system or network outages. In contrast with Oracle's standby database feature, such a failover site can also serve as a fully functional database to support application access when the primary site is concurrently operational.

Another usage of advanced replication is for transaction processing applications that require multiple points of access to database information for the purposes of distributing a heavy application load, ensuring continuous availability, or providing more localized data access like in Figure 2-6. Applications that have such requirements commonly include customer service oriented applications [1].

![Advanced Replication System Supporting Multiple Points of Update Access](image)

2.3.1. Advanced Replication Configurations

Oracle supports the requirements of advanced replication environments using multimaster replication as well as snapshot sites.
Multimaster Replication:

Oracle's multimaster replication allows multiple sites, acting as equal peers, to manage groups of replicated database objects. Applications can update any replicated table at any site in a multimaster configuration. Figure 2-7 illustrates a multimaster advanced replication system.

![Multimaster Replication System](image)

Figure 2-7 Multimaster Replication System

Snapshot Sites and Updatable Snapshots:

Master sites in an advanced replication system can consolidate information that applications update at remote snapshot sites. Oracle's advanced replication facility allows applications to insert, update, and delete table rows through updatable snapshots. Figure 2-8 illustrates an advanced replication environment with updatable snapshots.
Figure 2-8 Advanced Replication System with Updatable Snapshots

Updatable snapshots have the following properties.

- Updatable snapshots are always simple, fast-refreshable table snapshots.
- Oracle propagates the changes made through an updatable snapshot to the snapshot's remote master table. If necessary, the updates then cascade to all other master sites.
- Oracle refreshes an updatable snapshot as part of a refresh group identical to read-only snapshots.
- Updatable snapshots have the same underlying objects, such as base table, indexes, and views, as read-only snapshots. Additionally, Oracle creates the table USLOG$\_snapshotname to support updatable snapshots.

2.3.2. Hybrid Configurations

Multimaster replication and updatable snapshots can be combined in hybrid or "mixed" configurations to meet different application requirements. Mixed configurations can have any number of master sites and multiple snapshot sites for each master.

For example, as shown in Figure 2-9, n-way replication between two masters can support full-table replication between the databases that support two geographic regions. Snapshots can be defined on the masters to replicate full tables or table subsets to sites within each region.
Figure 2-9 Hybrid Configuration

Key differences between updatable snapshots and replicated masters include the following:

- Replicated masters must contain data for the full table being replicated, whereas snapshots can replicate subsets of master table data.
- Multimaster replication allows you to replicate changes for each transaction as the changes occur. Snapshot refreshes are set-oriented, propagating changes from multiple transactions in a more efficient, batch-oriented operation, but at less frequent intervals.
- If conflicts occur from changes made to multiple copies of the same data, master sites detect and resolve the conflicts.

2.3.3. Advanced Replication Objects, Groups, Sites, and Catalogs

The following sections explain the basic components of an advanced replication system, including replication objects, groups, sites, and catalogs.

Replication Objects:

A replication object is a database object existing on multiple servers in a distributed database system. Oracle's advanced replication facility enables you to replicate tables and supporting objects such as views, database triggers, packages, indexes, and synonyms.
Replication Groups:

In an advanced replication environment, Oracle manages replication objects using replication groups. By organizing related database objects within a replication group, it is easier to administer many objects together. Typically, you create and use a replication group to organize the schema objects necessary to support a particular database application. That is not to say that replication groups and schemas must correspond with one another. Objects in a replication group can originate from several database schemas and a schema can contain objects that are members of different replication groups. The restriction is that a replication object can be a member of only one group.

Replication Sites:

A replication group can exist at multiple replication sites. Advanced replication environments support two basic types of sites: master sites and snapshot sites.

- A master site maintains a complete copy of all objects in a replication group.
- All master sites in a multi-master, advanced replication environment communicate directly with one another to propagate data and schema changes in the replication group. A replication group at a master site is more specifically referred to as a master group.
- Additionally, every replication group has one and only one master definition site. A replication group's master definition site is a master site serving as the control point for managing the replication group and objects in the group.
- A snapshot site supports simple read-only and updatable snapshots of the table data at an associated master site. A snapshot site's table snapshots can contain all or a subset of the table data within a replication group. However, these must be simple snapshots with a one-to-one correspondence to tables at the master site. For example, a snapshot site may contain snapshots for only selected tables in a replication group. And a particular snapshot might be just a selected portion of a certain replicated table. A replication group at a snapshot site is more specifically referred to as a snapshot group. A snapshot group can also contain other replication objects.
Replication Catalog:

Every master and snapshot site in an advanced replication environment has a replication catalog. A site's replication catalog is a distinct set of data dictionary tables and views that maintain administrative information about replication objects and replication groups at the site. Every server participating in an advanced replication environment can automate the replication of objects in replication groups using the information in its replication catalog.

2.3.4. Oracle's Advanced Replication Architecture

Oracle converges data from typical advanced replication configurations using row-level replication with asynchronous data propagation. The following sections explain how these mechanisms function.

Row-Level Replication:

Typical transaction processing applications modify small numbers of rows per transaction. Such applications at work in an advanced replication environment will usually depend on Oracle's row-level replication mechanism. With row-level replication, applications use standard DML statements to modify the data of local data replicas. When transactions change local data, the server automatically captures information about the modifications and queues corresponding deferred transactions to forward local changes to remote sites.

Generated Replication Objects:

To support the replication of transactions in an advanced replication environment, you must generate one or more internal system objects to support each replicated table, package, or procedure.

- When you replicate a table, you can generate two corresponding packages. Oracle uses a replicated table's tablename$RP package to replicate
transactions that involve the table. Oracle uses a replicated table's tablename$RR package to resolve replication conflicts that involve the table.

- When you replicate a package specification and package body to support procedural replication, you can generate a corresponding wrapper package specification and package body. By default, Oracle names the wrapper for a package specification and package body using the name of the object with the prefix "defer_".

Asynchronous (Store-and-Forward) Data Propagation:

Typical advanced replication configurations that rely on row-level replication propagate data level changes using asynchronous data replication. Asynchronous data replication occurs when an application updates a local replica of a table, stores replication information in a local queue, and then forwards the replication information to other replication sites at a later time. Consequently, asynchronous data replication is also called store-and-forward data replication.

![Asynchronous Data Replication Mechanisms](image)

Figure 2-10 Asynchronous Data Replication Mechanisms

As Figure 2-10 shows, Oracle uses its internal system of triggers, deferred transactions, deferred transaction queues, and job queues to propagate data-level...
changes asynchronously among master sites in an advanced replication system, as well as from an updatable snapshot to its master table.

- When applications work in an advanced replication environment, Oracle uses internal triggers to capture and store information about updates to replicated data. Internal triggers build remote procedure calls (RPCs) to reproduce data changes made at the local site to remote replication sites. The internal triggers supporting data replication are essentially components within the Oracle Server executable. Therefore, Oracle can capture and store updates to replicated data very quickly with minimal use of system resources.

- Oracle stores RPCs produced by the internal triggers in a site's deferred transaction queue for later propagation. Oracle also records information about initiating transactions so that all RPCs from a transaction can also be propagated and applied remotely as a transaction. Oracle's advanced replication facility implements the deferred transaction queue using Oracle's advanced queueing mechanism.

- Oracle manages the propagation process using Oracle's job queue mechanism and deferred transactions. Each server participating in an advanced replication system has a local job queue. A server's job queue is a database table storing information about local jobs such as the PL/SQL call to execute for a job, when to run a job, and so on. Typical jobs in an advanced replication environment include jobs to push deferred transactions to remote master sites, jobs to purge applied transactions from the deferred transaction queue, and jobs to refresh snapshot refresh groups.

- Oracle forwards data replication information by executing RPCs as part of deferred transactions. Oracle uses distributed transaction protocols to protect global database integrity automatically and ensure data survivability.

Serial Propagation:

With serial propagation, Oracle asynchronously propagates replicated transactions, one at a time, in the same order of commit as on the originating site.

Parallel Propagation:

With parallel propagation, Oracle asynchronously propagates replicated transactions using multiple, parallel transit streams for higher throughput. When necessary,
Oracle orders the execution of dependent transactions to ensure global database integrity.

Parallel propagation uses the same execution mechanism Oracle uses for parallel query, load, recovery, and other parallel operations. Each server process propagates transactions through a single stream. A parallel coordinator process controls these server processes. The coordinator tracks transaction dependencies, allocates work to the server processes, and tracks their progress.

Purging of the Deferred Transaction Queue:

After a site pushes a deferred transaction to its destination, the transaction remains in the deferred transaction queue until another job purges the applied transaction from the queue.

Snapshot Propagation Mechanisms:

Updatable snapshots in an advanced replication environment can both "push" and "pull" data to and from its master table, respectively.

Master Table Updates:

Updates to an updatable snapshot are asynchronously pushed to its master table using Oracle's row-level, asynchronous data propagation mechanisms (RPCs, deferred transactions, and job queues).

Snapshot Refresh:

Identical to basic replication environments, advanced replication systems use Oracle's snapshot refresh mechanism to pull changes asynchronously from a master table to associated updatable (and read-only) snapshots.

Other Considerations:

An updatable snapshot's push and pull tasks are independent operations that you can configure associatively or separately.
• Snapshot sites can configure refresh groups to automatically push changes made to the member snapshots to the master site, and then refresh the snapshots.
• Snapshot sites can configure updatable snapshots to push changes to the master site and refresh snapshots at different times and intervals.

For example, an advanced replication environment that consolidates information at a master site might configure updatable snapshots to push changes to the master site every hour but refresh updatable snapshots infrequently, if ever.

Replication Administrators, Propagators, and Receivers:

An Oracle advanced replication environment requires several unique database user accounts to function properly, including replication administrators, propagators, and receivers.

• Every site in an Oracle advanced replication system requires at least one replication administrator, a user responsible for configuring and maintaining replicated database objects.
• Each replication site in an Oracle advanced replication system requires special user accounts to propagate and apply changes to replicated data.

2.3.5. Replication Conflicts

Advanced replication systems supporting an update-anywhere model of data replicas must address the possibility of replication conflicts. The following sections explain the different types of replication conflicts, when they can occur, and how Oracle detects and resolves replication conflicts.

Uniqueness Conflicts:

A uniqueness conflict occurs when the replication of a row attempts to violate entity integrity (a PRIMARY KEY or UNIQUE constraint). For example, consider what happens when two transactions originating from two different sites each insert a row
into a respective table replica with the same primary key value. In this case, replication of the transactions will cause a uniqueness conflict.

Update Conflicts:

An update conflict occurs when the replication of an update to a row conflicts with another update to the same row. Update conflicts occur when two different transactions originating from different sites update the same row at nearly the same time.

Delete Conflicts:

A delete conflict occurs when two transactions originate from different sites, with one transaction deleting a row that the other transaction updates or deletes.

2.3.6. Replicated Data Models and Conflicts

When designing applications to work on top of a database system using advanced replication, you must consider the possibility of replication conflicts. In such a case, your applications must use one of several different replicated data ownership models that ensure global database integrity by avoiding or resolving replication conflicts.

Primary Site, Static Ownership:

Primary ownership, also called static ownership, is the replicated data model that basic read-only replication environments support. Primary ownership prevents all replication conflicts, because only a single server permits update access to a set of replicated data.

Rather than control ownership of data at the table level, applications can use horizontal and vertical partitioning to establish more granular static ownership of data. For example, applications might have update access to specific columns or rows in a replicated table on a site-by-site basis.
Dynamic Ownership:

The dynamic ownership replicated data model is less restrictive than primary site ownership. With dynamic ownership, the capability to update a data replica moves from site to site, still ensuring that only one site provides update access to specific data at any given point in time. A workflow system clearly illustrates the concept of a dynamic ownership. For example, related departmental applications can read the status code of a product order to determine when they can and cannot update the order. Figure 2-11 illustrates an application that uses a dynamic ownership model.

![Figure 2-11 Dynamic Ownership in an Order Processing System](image)

Shared Ownership:

Primary site ownership and dynamic ownership replication data models that promote conflict avoidance are often too restrictive to implement for some database applications. Some applications must operate using a shared ownership replicated data model in which applications can update the data of any table replica at any time.

When a shared data ownership system replicates changes asynchronously (store-and-forward replication), corresponding applications must avoid or detect and resolve replication conflicts if and when they occur.

2.3.7. Conflict Avoidance Techniques

Typically, you can design an advanced replication system to avoid all or a large percentage of replication conflicts, especially uniqueness and delete conflicts.
Conflict Detection:

Although conflict avoidance is preferable, it is not always possible. When an application uses a shared ownership data model with asynchronous row-level replication and replication conflicts are possible, Oracle automatically detects uniqueness, update, and delete conflicts. To detect conflicts during replication, Oracle compares a minimal amount of row data from the originating site with the corresponding row information at the receiving site. When there are differences, Oracle detects the conflict.

To detect replication conflicts accurately, Oracle must be able to uniquely identify and match corresponding rows at different sites during data replication. Typically, Oracle's advanced replication facility uses the primary key of a table to uniquely identify rows in the table. When a table does not have a primary key, you must designate an alternate key. This key serves as a column or set of columns that Oracle uses to identify rows in the table during data replication. In either case, you should not allow applications to update the identity columns of a table. This ensures that Oracle can identify rows and preserve the integrity of replicated data.

Conflict Resolution:

When a receiving site in an advanced replication system is using asynchronous row-level replication and it detects a conflict in a transaction, the default behavior is to log the conflict and the entire transaction and leave the local version of the data intact. In most cases, you should use Oracle's advanced replication facility to automate the resolution of replication conflicts.

Oracle uses column groups to detect and resolve update conflicts during asynchronous, row-level advanced replication. A column group is a logical grouping of one or more columns in a table. Every column in a replicated table is part of a single column group. When configuring replicated tables, you can create column groups and then assign columns and corresponding update conflict resolution methods to each group.

Each column group in a replicated table can have a list of one or more update conflict resolution methods. Indicating multiple methods for a group allows Oracle to resolve an update conflict in different ways should other methods fail to resolve the
conflict. When trying to resolve an update conflict for a group, Oracle executes the group's resolution methods in the order listed for the group.

By default, every replicated table has a shadow column group. A table's shadow column group contains all columns that are not within a specific column group. You cannot assign conflict resolution methods to a table's shadow group.

When designing column groups you can choose from among many built-in conflict resolution methods. For example, to resolve update conflicts, you might choose to have Oracle overwrite the column values at the destination site with the column values from the originating site. Oracle offers many other update conflict resolution methods.

Oracle also allows you to assign uniqueness conflict resolution methods to PRIMARY KEY and UNIQUE constraints. However, Oracle offers no delete conflict resolution methods. Consequently, applications that operate within an asynchronous, shared ownership data model should avoid delete conflicts by not using DELETE statements to delete rows. Instead, applications can mark rows for deletion and configure the system to periodically purge deleted rows [1,2,4,6].
3. DATA WAREHOUSE CONCEPTS

In this chapter, data warehouse characteristics, typical data warehouse architectures and Extraction, Transportation, Transformation (ETT) methods of data warehouse are explained detaitely.

3.1. What is Data Warehouse

A data warehouse is a relational database that is designed for query and analysis rather than transaction processing. It usually contains historical data that is derived from transaction data, but it can include data from other sources. It separates analysis workload from transaction workload and enables an organization to consolidate data from several sources.

Inmon describes the data warehouse as a "subject oriented, integrated, nonvolatile, time variant collection of data in support of management decisions" [7].

3.1.1. Subject Oriented

Data warehouses are designed to help you analyze your data. For example, you might want to learn more about your company's sales data. To do this, you could build a warehouse concentrating on sales. In this warehouse, you could answer questions like "Who was our best customer for this item last year?" This kind of focus on a topic, sales in this case, is what is meant by subject oriented [7].

3.1.2. Integrated

Integration is closely related to subject orientation. Data warehouses need to have the data from disparate sources put into a consistent format. This means that naming conflicts have to be resolved and problems like data being in different units of measure must be resolved [8].
3.1.3. Nonvolatile

Nonvolatile means that the data should not change once entered into the warehouse. This is logical because the purpose of a warehouse is to analyze what has occurred [9].

3.1.4. Time Variant

Most business analysis requires analyzing trends. Because of this, analysts tend to need large amounts of data. This is very much in contrast to OLTP systems, where performance requirements demand that historical data be moved to an archive [10].

3.2. Contrasting a Data Warehouse with an OLTP System

One major difference between the types of system is that data warehouses are not usually in third-normal form.

Data warehouses and OLTP systems have vastly different requirements. Here are some examples of the notable differences between typical data warehouses and OLTP systems:

- **Workload:** Data warehouses are designed to accommodate ad hoc queries. The workload of a data warehouse may not be completely understood in advance, and the data warehouse is optimized to perform well for a wide variety of possible query operations. OLTP systems support only predefined operations. The application may be specifically tuned or designed to support only these operations.

- **Data Modifications:** The data in a data warehouse is updated on a regular basis by the ETT process (often, every night or every week) using bulk data-modification techniques. The end users of a data warehouse do not directly update the data warehouse. In an OLTP system, end users routinely issue individual data-modification statements in the database. The OLTP database
is always up-to-date, and reflects the current state of each business transaction.

- **Schema Design**: Data warehouses often use denormalized or partially denormalized schemas (such as a star schema) to optimize query performance. OLTP systems often use fully normalized schemas to optimize update/insert/delete performance, and guarantee data consistency.

- **Typical Operations**: A typical data warehouse query may scan thousands or millions of rows. For example, "Find the total sales for all customers last month." A typical OLTP operation may access only a handful of records. For example, "Retrieve the current order for a given customer."

- **Historical Data**: Data warehouses usually store many months or years of historical data. This is to support historical analysis of business data. OLTP systems usually store only a few weeks' or months' worth of data. The OLTP system only stores as much historical data as is necessary to successfully meet the current transactional requirements [7].

3.3. **Typical Data Warehouse Architecture**

Data warehouses and their architectures can vary depending upon the specifics of each organization's situation. Figure 3-1 shows the most basic architecture for a data warehouse. In it, a data warehouse is fed from one or more source systems, and end users directly access the data warehouse.
A more complex data warehouse environment is illustrated in Figure 3-2. In addition to a central database, there is a staging system used to cleanse and integrate data, as well as multiple datamarts, which are systems designed for a particular line of business [10].
3.4. ETT Overview

You need to load your data warehouse regularly so that it can serve its purpose of allowing business analysis. To do this, however, data from one or more operational systems needs to be extracted and copied into the warehouse. This process of reading and preparing the data is relatively difficult, and needs to be performed on a regular basis.

The process of extracting data from source systems and bringing it into the data warehouse is commonly called ETT, which stands for Extraction, Transformation, and Transportation. Indeed, the acronym ETT is perhaps too simplistic, since it omits one important phase, the loading of the data warehouse, and implies that each of other phases of the process is distinct. Rather than introduce new terminology, we will instead refer to the entire process as ETT. You should understand that ETT refers to a broad process, and not three well-defined steps.

3.4.1. Extraction

Extraction is the operation of copying data from a database into a file or onto a network connection. This is the first step of the ETT process: data must be extracted from the source system(s) so that this data may be subsequently transformed and loaded into the data warehouse.

The source systems for a data warehouse are typically transaction-processing database applications. For example, one of the source systems for a sales-analysis data warehouse may be the order-entry system which records all of the current order activities.

Designing and creating the extraction process is often one of the most time-consuming tasks in the ETT process and, indeed, in the entire data warehousing process. The source systems may be very complex, and thus determining which data needs to be extracted can be difficult. Moreover, the source system typically cannot be modified, nor can its performance or availability be impacted, to accommodate the needs of the data warehouse extraction process. These are very important considerations for extraction, and ETT in general.
This chapter, however, focuses on the technical considerations for extracting data. It assumes that the data warehouse team has already identified the data that will be extracted, and discusses common techniques used for extracting data from source databases. The techniques for extraction fall into two broad categories:

- Techniques which extract data from an operational system and place the data into a file. Examples are data-unloads and exports.

- Techniques which extract data from an operational system and directly transport data into the target database (a data warehouse or a staging database). Examples are gateways and distributed queries. Method proposed in this thesis is an example of distributed operations.

These techniques are broad concepts. Therefore only the second technique is explained here in detail.

Using distributed-query technology, one Oracle database can directly query tables located in another Oracle database. Specifically, a data warehouse or staging database could directly access tables and data located in an Oracle-based source system. This is perhaps the simplest method for moving data between two Oracle databases because it combines the extraction and transformation into a single step, and, moreover, requires minimal programming.

Continuing our example from above, suppose that we wanted to extract a list of employee names with department names from a source database, and store it into our data warehouse. Using a Net8 connection, and distributed-query technology, this can be achieved using a single SQL statement:

```
CREATE TABLE empdept
AS
SELECT ename, dname FROM emp@source_db, dept@source_db
WHERE emp.deptno = dept.deptno;
```

This statement creates a local table in the data warehouse, EMPDEPT, and populates it with data from the EMP and DEPT tables on the source system.
This technique is ideal for moving small volumes of data. However, the data is transported from the source system to the data warehouse via a single Net8 connection. Thus, the scalability of this technique is limited. For larger data volumes, file-based data extraction and transportation techniques are often more scalable and thus more appropriate.

Gateways are another form of distributed-query technology, except that gateways allow an Oracle database (such as a data warehouse) to access database tables stored in remote, non-Oracle databases. Like distributed queries, gateways are very easy to set up and use, but also lack scalability for very large data volumes.

An important consideration for extraction is incremental extraction, also called change data capture. If a data warehouse extracts data from an operational system on a nightly basis, then the only data which that data warehouse requires is the data that has changed since the last extraction (that is, the data that has been modified in the last 24 hours).

If it was possible to efficiently identify and extract only the most recently-changed data, the extraction process (as well as all downstream operations in the ETT process) could be much more efficient since it would only need to extract a much smaller volume of data. Unfortunately, for many source systems, identifying the recently modified data may be difficult or intrusive to the operation of the system. Change data capture is typically the most challenging technical issue in data extraction.

Many data warehouses do not use any change-capture techniques as part of the extraction process. Instead, entire tables from the source systems are extracted to the data warehouse or staging area, and these tables are compared with a previous extract from the source system to identify the changed data. While this approach may not have significant impact on the source systems, it clearly can place a considerable burden on the data-warehouse processes particularly if the data volumes are large.

Change data capture as part of the extraction process is often desirable and there are three techniques for implementing change data capture on Oracle based source systems.
• Timestamps

The tables in some operational systems have timestamp columns. The timestamp specifies the time and date that a given row was last modified. If the tables in an operational system have columns containing timestamps, then the latest data can easily be identified using the timestamp columns. For example, the following query might be useful for extracting today's data from an ORDERS table:

```
SELECT * FROM orders WHERE TIMESTAMP = TO_DATE(sysdate, 'mm-dd-yyyy');
```

If timestamps are not available in an operational system, the system may be able to be modified to include timestamps. This would require, first, modifying the operational system's tables to include a new timestamp column, and second, creating a trigger to update the timestamp column following every operation which modifies a given row.

• Partitioning

Some source systems may utilize Oracle's range-partitioning, such that the source tables are partitioned along a date key, which allows for easy identification of new data. For example, if you are extracting from an ORDERS table, and the ORDERS table is partitioned by week, then it is easy to identify the current week's data. But range partitioning is only available on Oracle version 8 or later.

• Triggers

Triggers can be created in operational systems to keep track of recently updated records. They can then be used in conjunction with timestamp columns to allow you to identify the exact time and date when a given row was last modified. You do this by creating a trigger on each source table that requires change-data-capture. Following each DML statement that is executed on the source table, this trigger updates the timestamp column with the current time. Thus, the timestamp column provides the exact time and date when a given row was last modified.

In this thesis trigger based method is used in change data capture. This technique is explained detailly in chapter 4.
3.4.2. Transportation

Transportation is, literally, the act of moving data from one system to another system. In a data warehouse environment, the most common requirements for transportation are in moving data from a source system to a staging database or a data warehouse database; from a staging database to a data warehouse; or from a data warehouse to a datamart.

Transportation is often one of the simplest portions of the ETT process, and is commonly integrated with other portions of the process. For example, as shown in extraction part before, distributed query technology provides a mechanism for both extracting and transporting data. So that transportation is not explained much.

3.4.3. Transformation

Data transformations are often the most complex and, in terms of processing time, the most costly part of the ETT process. Many, if not all, data transformations can occur within an Oracle database, although transformations are also often implemented outside of the database (for example, on flat files) as well.

In data warehouse environment, there may be many source systems and you should consolidate these systems. So that some data type conversions are needed. For example, date format of systems may be different, in that case you must change all the date type data into same format. Data warehouse databases is usually unnormalized. But the source systems are OLTP systems and these systems have normalized data. To unnormalize the data warehouse database you need some transformations. Transformations usually done in staging area.

Another transformation issue is integration of data. For example in one table, gender is encoded as m/f but in another table it is encoded as 1/0. It should be done consistently, whatever the source application. If application data is encoded as X/Y, it is converted as it is moved to the warehouse. The same consideration of consistency goes for all of the application design issues, such as naming conventions, key structure, measurement of attributes, and physical characteristics of data [10].
4. AN ALTERNATIVE READ-ONLY REPLICATION SYSTEM IMPLEMENTATION

In this chapter, an alternative read-only replication system implementation will be explained in detail. Why do we need an alternative read-only replication mechanism instead of the oracle replication system? Also, what is the additional capabilities of this system according to Oracle read-only replication? In the implementation, Oracle database version 8.0.5, 8.1.5, 8.1.6 and 8.1.7 are used and tested. But the replication code generated by the system can be used for table replication between all Oracle version 7 and version 8 databases. All of these issues will be explained in detail in this chapter.

4.1. Capabilities of the new system

This system replicates a table from one site (master site) to another site (snapshot site) and periodically synchronizes the replicated table automatically without user intervention.

In the snapshot site, the replicated table is used read-only. In other words, the rows of the replicated table can not be updated, deleted or new rows can not be inserted into replicated table. In fact, DBA can do these kind of operations in the replicated table considering the replication mechanism. This can be considered either the power or the weakness of the new system. Weakness, because if the DBA does not know the internal mechanism of the new replication system then this operation causes some inconsistencies in the replication. For example, after a mass insert operation to master table, if DBA inserts these rows into the replicated table to synchronize a replication environment without deleting the related log records from the master log table, then this results in inconsistencies in replication system. But if DBA knows the internal mechanism and deletes the log records then this will be the power of this system, because this operation speeds up the synchronization process.

The user must not replicate all of the columns of the master table except for the primary key columns. So that a subset of master table columns can be replicated.
Also user can translate data type or data of a column except primary key columns according to the design of replicated table. For example;

- User can decode data according to design criteria. For example; if in the master table sex field holds data MALE and FEMALE, but in the replicated table only M and F is held then in the definition step of the system, user can make necessary translation

  \[ \text{DECODE(sex,'MALE','M','FEMALE','F',null)} \]

- User can change measurements of data according to design criteria. For example; if the master table holds measurement in inches and replicated table holds it in centimeters then user can apply user-defined functions to that column or simply multiply, divide, subtract or add some value.

- If master table data type is different than replicated table data type then user can change this data type in definition step of the system by using built-in data type conversion functions such as \text{TO_CHAR}, \text{TO_DATE}, \text{TO_NUMBER}...

- Also user can use any function to convert a column or add a new column to replicated table. For example instead of using Department_no in the replicated table you can use Department_name and you can use a function that finds the department name from department no. But you should consider the performance of the synchronization process. Instead of doing this operation, you can do that in the staging area of the data warehouse [3].

4.2. Why do we need an alternative read-only replication system?

In data warehouse, ETT process is very complicated in many cases. So that there is really no automated ETT systems. The oracle replication architecture, in some cases is sufficient for ETT process. But in most cases, it is not sufficient and user intervention is required. But in Oracle replication, user can not change the internal mechanism of the replication. Therefore, we need a system similar to the oracle replication system so that we interfere the internal mechanism according to needs
of ETT process in data warehouse design. An alternative replication system that is implemented in this thesis is the result of this need.

Additionally, to use oracle replication system, user must pay additional licence fee. Because replication is an option in Oracle. When you buy an Oracle database management system software, you must buy also replication option of database to use replication.

In this thesis, objects necessary for replication are automatically produced and saved in the database according to user input. These objects are easily customized by any database administrator, because there are many oracle tools that are used to customize these objects. For example Oracle Schema Builder for tables and indexes, Oracle Procedure Builder for procedures and triggers. An experienced database administrator does not need these kind of tools. Instead of these kind of tools, PL/SQL can be used in a text editor [3].

4.3. Design of the system

For the implementation of the replication system, a front-end is designed. User inputs related to the system are saved in the database tables using this front-end. Front-end is developed by Oracle Developer 2000 tool. According to these tables, replication system objects are generated. Table structures and replication objects generated by the replication system are explained in detail later in this chapter. In master site and replication site, SYSTEM user is used as a schema owner of the definition tables, because SYSTEM user is found in every Oracle database as a DBA [4].

4.3.1. Master Site Tables and Objects

In master site, there is a table whose name is snap_log where master site definitions are saved. The structure of this table is as the following:

- **OWNER**: Data type of this column is varchar2(30). Owner of the master table is saved in this column. Data type of this column is consistent with
Oracle. It is same with OWNER column of DBA_TABLES dictionary. Any user name length in Oracle is maximum 30 characters.

- **TABLE_NAME**: This column is used for master table name. Also this column has data type of varchar2(30). It is same with TABLE_NAME column of DBA_TABLES dictionary. Any table name length in Oracle is maximum 30 characters.

- **LOG_TABLE_NAME**: This column is used to store log table name. Also this column has the data type of varchar2(30). According to master table name, for each master table, system generates a log table that holds the changes to the master table. With this name a table is automatically created by the system.

- **LOG_TRIGGER_NAME**: This column is used to store trigger name. For each master table a trigger is created automatically to store the changes to the log table generated before automatically. This column has the datatype of varchar2(30). It is same with TRIGGER_NAME column of DBA_TRIGGERS dictionary. Trigger is a program code that is activated after each DML operation on master table to capture the changes on the master table and to store these changes into the log table.

- **TABLESPACE_NAME**: This column is used to store the tablespace name where the log table is stored in the database. Data type of this column is varchar2(30). This column is also consistent with database dictionary. Corresponding column in the dictionary is TABLESPACE_NAME column in DBA_TABLESPACES.

- **INITIAL_EXTENT**: This column is used to store the storage characteristics of the log table. Data type of this column is NUMBER. For each table, user can determine initial storage size. This column is used for initial storage size of the log table.

- **NEXT_EXTENT**: This column is used also to store the storage characteristics of the log table. Data type of this column is also NUMBER.

The primary key of this table is a composite primary key and it consists of owner and table_name fields.

Data Definition Language (DDL) command of this table and its primary key is below.
Create table snap_log

(Owner varchar2(30) not null,
 table_name varchar2(30) not null,
 log_table_name varchar2(30) not null,
 log_trigger_name varchar2(30) not null,
 tablespace_name varchar2(30),
 initial_extent number,
 next_extent number);

alter table snap_log
add constraint pk_snap_log
primary key (owner,table_name);

Figure 4-1 Front-end tool for administering master site specifications

Using front-end tool which is shown in Figure 4-1, user enters the specifications of the master table into SNAP_TABLE. After that a stored procedure whose name is TRG_SNAP_LOG is called to create log_table and log_trigger according to these specifications.

In the TRG_SNAP_LOG stored procedure, first, all of the primary key columns of the master table and data type of this columns are determined using internal Oracle definitions. Then, according to this primary key definition, a log table whose name is
stored in log_table_name column of snap_log table is created. This log table has
columns that are primary key columns of the master table, operation code column
whose name is oper and operation date whose name is op_date. Parameters of the
TRG_SNAP_LOG procedure is owner, table_name and operations which is either I
or D. In case of “I” a new log table and trigger is created and in case of “D” existing
log table and trigger is dropped from the database.

For example, if we use SYSTEM schema’s table SINAV as a master table and we
enter log_table_name as LOG_SINAV and log_trigger_name as TRG_SINAV then
the procedure creates a table whose name is LOG_SINAV and a trigger whose
name is TRG_SINAV. Definition of these objects are below.

DESC SYSTEM.LOG_SINAV
OP_DATE DATE
YIL1 NUMBER(4)
YIL2 NUMBER(4)
DONEM NUMBER(1)
BILNO VARCHAR2(7)
DERS_KOD VARCHAR2(6)
UYE_KOD VARCHAR2(4)
OPER VARCHAR2(1)

As SYSTEM.LOG_SINAV table definition was seen above it has a composite
primary key whose columns are YIL1, YIL2, DONEM, BILNO, DERS_KOD and
UYE_KOD. Two additional columns OP_DATE and OPER are also added to the log
table definition from replication procedure. This table is used to save change data
capture.

TRG_SINAV trigger is also created by the replication procedure and it’s definition is
below. This trigger is actually a program unit that is fired after every DML operation
on SYSTEM.SINAV table. Source of this trigger is exported from Oracle Procedure
Builder for readability and some comments are added for explanation.

CREATE OR REPLACE TRIGGER TRG_SINAV
AFTER INSERT OR DELETE OR UPDATE ON SYSTEM.SINAV
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
begin
  if deleting then
  
/* "deleting" means that the operation is delete and as was seen below operation 
code is 'D' and oper_date is sysdate function which returns current date-time */
  insert into LOG_SINA
      values (sysdate,:old.YIL1,:old.YIL2,
             :old.DONEM,:old.BILNO,:old.DERS_KOD,
             :old.UYE_KOD,'D');
  elseif updating then
  
/* "updating" means that the operation is update and operation code is 'U' */
  insert into LOG_SINA
      values (sysdate,:new.YIL1,:new.YIL2,:new.DONEM,
             :new.BILNO,:new.DERS_KOD,:new.UYE_KOD,'U');
  elseif inserting then
  
/* "inserting" means that the operation is insert and operation code is 'I' */
  insert into LOG_SINA
      values (sysdate,:new.YIL1,:new.YIL2,:new.DONEM,
             :new.BILNO,:new.DERS_KOD,:new.UYE_KOD,'I');
  end if;
end;

The most important part of the TRG_SNAP_LOG procedure which creates log table 
and trigger, is to find the primary key definition of the master table. According to 
primary key definition, log table and log trigger are generated. Primary key 
definitions are found in Oracle dictionary. These dictionaries are 
DBA_TAB_COLUMNS, DBA_CONS_COLUMNS, DBA_CONSTRAINTS.

- DBA_TAB_COLUMNS describes the columns of all tables, views, and 
  clusters in the database.
- DBA_CONSTRAINTS describes all constraint definitions in the database.
- DBA_CONS_COLUMNS describes all columns in the database that are 
specified in constraints.

Using these dictionaries in a query, primary key definitions are retrieved. This 
query is below:
select c.column_name, data_type||
DECODE(DATA_TYPE,
'VARCHAR2', '(||TO_CHAR(data_length))''),
'NUMBER', '(||TO_CHAR(data_precision))'||'(||TO_CHAR(data_scale))''),
'DATE', ' ',
'CHAR', '(||TO_CHAR(data_length))'') AS TYPE
from sys.dba_tab_columns c, sys.dba_cons_columns a, sys.dba_constraints b
where a.table_name = b.table_name
and a.owner = b.owner
and a.constraint_name = b.constraint_name
and b.constraint_type = 'P'
and a.owner = c.owner
and a.table_name = c.table_name
and a.column_name = c.column_name
and a.table_name = P_TABLE
and a.owner = P_OWNER
order by a.position;

In this query, according to our example P_TABLE is SINAV that is the name of table and P_OWNER is SYSTEM that is the owner of the SINAV table. This query returns the following rows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>YIL1</td>
<td>NUMBER(4,0)</td>
</tr>
<tr>
<td>YIL2</td>
<td>NUMBER(4,0)</td>
</tr>
<tr>
<td>DONEM</td>
<td>NUMBER(1,0)</td>
</tr>
<tr>
<td>BILNO</td>
<td>VARCHAR2(7)</td>
</tr>
<tr>
<td>DERS_KOD</td>
<td>VARCHAR2(6)</td>
</tr>
<tr>
<td>UYE_KOD</td>
<td>VARCHAR2(4)</td>
</tr>
</tbody>
</table>

Using these rows in a loop log table whose name is saved in “log_table_name” field of snap_log table and a trigger whose name is saved in “log_trigger_name” field of snap_log table are created by means of TRG_SNAP_LOG stored procedure. TRG_SNAP_LOG procedure creates a string of “CREATE TABLE” statement for log table and a string of “CREATE TRIGGER” statement for log trigger and then executes these statements.
TRG_SNAP_LOG stored procedure whose file name trg_snap_log.pls is in disk attached to the thesis [3,5,11].

4.3.2. Replication Site Tables and Objects

In replication site there are two tables used to hold replication object definition. These tables are SNAP_TABLE and SNAP_COLUMN_MAP. In snap_table, master table and replication table information, database link information, refresh procedure name and refresh interval information are stored. In snap_column_map, replicated table's column and master table's column relation are stored. Using these tables, TRG_SNAP_TABLE procedure creates the necessary refresh procedure and schedules refresh procedure to synchronize replicated table and master table.

Snap_table and snap_column_map table's DDL commands are below:

Create table snap_table
(sourc_owner varchar2(30) not null,
 source_table_name varchar2(30) not null,
 dest_owner varchar2(30) not null,
 dest_table_name varchar2(30) not null,
 link varchar2(128) not null,
 refresh_procedure varchar2(30),
 transfer_data varchar2(1),
 next varchar2(200) not null,
 start_with date default sysdate,
 last_refresh date);

alter table snap_table
add constraint pk_snap_table
primary key (source_owner,sourc_table_name);

create table snap_column_map
(sourc_owner varchar2(30) not null,
 sourc_table_name varchar2(30) not null,
 sourc_column_id number not null,
source_column_name  varchar2(30),
dest_column        varchar2(200));

alter table snap_column_map
add constraint pk_snap_column_map
primary key (source_owner,source_table_name,source_column_id);

As shown, the primary key of snap_table consists of source_owner and
source_table_name columns. Primary key of snap_column_map is source_owner,
source_table_name and source_column_id.

Snap_table's column definitions are below:

- SOURCE_OWNER: Replicated table owner. Data type is varchar2(30).
- SOURCE_TABLE_NAME: Replicated table name. Data type is varchar2(30).
- DEST_OWNER: Master table owner. Data type is varchar2(30).
- DEST_TABLE_NAME: Master table name. Data type is varchar2(30).
- LINK: Master table database link name. Data type is varchar2(128). It is
  same with Oracle dictionary DBA_DB_LINKS.DB_LINK column. Database
  link is used to identify the destination database name and protocol. The user
  of the distributed environment should create database link before. Database
  link name for a database must be same with its global name. In the database
  initialization parameters file, GLOBAL_NAMES parameter must be set as
  TRUE. To create a database link in a database, the following command must
  be used.

    CREATE PUBLIC DATABASE LINK <remote databases global name>
    USING <network alias>
    CONNECT TO SYSTEM
    IDENTIFIED BY <remote database system user password>;

<network alias> must be the master database alias in the servers
TNSNAMES.ora network initialization file.

- REFRESH_PROCEDURE: This column is used to store refresh procedure
  name. This procedure is created by the TRG_SNAP_TABLE procedure.
Refresh procedure is used to synchronize the replicated table with the master table.

- **TRANSFER_DATA**: This column holds the information whether all the master table data transferred in the initialization of replication or not. Sometimes master table has millions of rows and to transport these rows online is meaningless. So that this column is an indicator of transportation in create time of replicated table.

- **NEXT**: This column holds the refresh interval calculation method. For example "SYSDATE+1" means once in a day, "SYSDATE+1/24" means once in an hour.

- **START_WITH**: This is a date type column that means the start date of next refresh.

- **LAST_REFRESH**: This is also a date column that means the last refresh date of the replication.

Snap_column_map table's column definitions are below:

- **SOURCE_OWNER**: Replicated table schema owner.
- **SOURCE_TABLE_NAME**: Replicated table name.
- **SOURCE_COLUMN_ID**: This column is used to hold replicated table internal column id. This field corresponds to DBA_TAB_COLUMNS database dictionary COLUMN_ID column and its data type is number.
- **SOURCE_COLUMN_NAME**: Replicated table column name.
- **DEST_COLUMN**: This column is used to hold corresponding master table columns and the operations on that columns. For example 'to_char(hiredate, 'YYYYMMDD')' means that hiredate column of the master table whose data type is date, is converted into VARCHAR2 data type.
Figure 4-2 Front-end tool for administering replication site specifications

According to the definitions of the replication in these tables, a refresh procedure whose name is stored in REFRESH_PROCEDURE column is created and scheduled to run every refresh interval. In our example that is shown in Figure 4-2, REF_SINA1V procedure is created to replicate SYSTEM.SINA1V table in site HHOBS.WORLD to OBS.SINA1V table. CREATE REPLICATION button in Figure 4-2 is used to create REF_SINA1V procedure. This procedure is used to synchronize replicated table with the master table.

In our example, the refresh procedure REF_SINA1V is created for SINA1V table. This procedure is created by TRG_SNAP_TABLE procedure. REF_SINA1V procedure is following:

```sql
procedure REF_SINA1V is
    err_msg varchar2(500);
    v_last_refresh date;
    cur_date date;

    /*The most important part of this procedure is the following statement. This
    statement joins the master table and log_table to get the recent changes. This query
```
has two parts: One of them is to retrieve update and insert statements the other one is to retrieve delete statements. In the delete statement part, outer join is used because deleted row information is found in the log table but it is not found in the master table. If these tables are normally joined then the delete information cannot be retrieved. */

cursor c_change is
select oper,b.YIL1 as a1
 ,b.YIL2 as a2
 ,b.DONEM as a3
 ,b.BILNO as a4
 ,b.DERS_KOD as a5
 ,b.UYE_KOD as a6
 ,A1_NOT as a7
 ,A2_NOT as a8
 ,F_NOT as a9
 ,BUT_NOT as a10
 ,YUK_NOT as a11
 ,TD_NOT as a12
 ,ONAY as a13
 ,EK_YUK_NOT as a14
 ,SINIF as a15
from SYSTEM.SINAV@HHOBS.WORLD a,
SYSTEM.LOG_SINAV@HHOBS.WORLD b
/* The outer join part and oper='D' for delete operation. (+) shows that this join operation is an outer join operation */
where op_date > v_last_refresh and op_date <= cur_date and oper='D'
 and a.YIL1(+) = b.YIL1 and a.YIL2(+) = b.YIL2 and a.DONEM(+) = b.DONEM
 and a.BILNO(+) = b.BILNO and a.DERS_KOD(+) = b.DERS_KOD
 and a.UYE_KOD(+) = b.UYE_KOD
union all
select oper,b.YIL1 as a1
 ,b.YIL2 as a2
 ,b.DONEM as a3
 ,b.BILNO as a4
 ,b.DERS_KOD as a5
 ,b.UYE_KOD as a6
 ,A1_NOT as a7
A2_NOT as a8
F_NOT as a9
BUT_NOT as a10
YUK_NOT as a11
TD_NOT as a12
ONAY as a13
EK_YUK_NOT as a14
SINIF as a15

from SYSTEM.SINAV@HHOBS.WORLD a,
SYSTEM.LOG_SINAV@HHOBS.WORLD b

where op_date > v_last_refresh and op_date <= cur_date and oper in ('I','U')
and a.YIL1 = b.YIL1 and a.YIL2 = b.YIL2 and a.DONEM = b.DONEM
and a.BILNO = b.BILNO and a.DERS_KOD = b.DERS_KOD
and a.UYE_KOD = b.UYE_KOD

order by 1;

begin

cur_date := sysdate; /* The current date is determined here */
select last_refresh into v_last_refresh /* The last refresh information selected */
from snap_table

where source_owner='OBS' and source_table_name='SINAV';
update snap_table /* the new refresh information is updated as current date */
set last_refresh = cur_date

where source_owner='OBS' and source_table_name='SINAV';

/* Now for every record in the c_change cursor which is a query that returns more
than one row, the changes is reflected to the replicated table */

for dest_rec in c_change loop
if dest_rec.oper = 'I' then /* insert operation part */
insert into OBS.SINAV
values (dest_rec.a1,dest_rec.a2,dest_rec.a3,dest_rec.a4,dest_rec.a5,
dest_rec.a6,dest_rec.a7,dest_rec.a8,dest_rec.a9,dest_rec.a10,
dest_rec.a11,dest_rec.a12,dest_rec.a13,dest_rec.a14,dest_rec.a15);

elsif dest_rec.oper = 'U' then /* update operation part */
update OBS.SINAV
set YIL1 = dest_rec.a1,YIL2 = dest_rec.a2,DONEM = dest_rec.a3,
BILNO = dest_rec.a4, DERS_KOD = dest_rec.a5,
UYE_KOD = dest_rec.a6,A1_NOT = dest_rec.a7,
A2_NOT = dest_rec.a8,F_NOT = dest_rec.a9,
BUT_NOT = dest_rec.a10, YUK_NOT = dest_rec.a11,  
TD_NOT = dest_rec.a12, ONAY = dest_rec.a13,  
EK_YUK_NOT = dest_rec.a14, SINIF = dest_rec.a15  
where YIL1 = dest_rec.a1 and YIL2 = dest_rec.a2 and DONEM = dest_rec.a3  
and BILNO = dest_rec.a4 and DERS_KOD = dest_rec.a5  
and UYE_KOD = dest_rec.a6;  
elsif dest_rec.oper = 'D' then /* Delete operation part */  
delete from OBS.SINAV  
where YIL1 = dest_rec.a1 and YIL2 = dest_rec.a2 and DONEM = dest_rec.a3  
and BILNO = dest_rec.a4 and DERS_KOD = dest_rec.a5  
and UYE_KOD = dest_rec.a6;  
end if;  
end loop;  
commit;  
end;

Listing of the TRG_SNAP_PROCEDURE is in file trg_snap_procedure.pls in the disk attached. To create a refresh procedure, to get the changed records after the last refresh is important. We need a query that joins the log table and the master table in the master site. The WHERE clause of this query is very important. Because joining the log_table and master table does not give the deleted records. To get the deleted records primary key field, outer join must be used. To make an outer join, the primary key of columns of the master table must be found and these columns must be selected from log table in the column list of the select clause. To find the primary key columns of the master table the same method is used like the log_table and trigger creation in master site. The other important part is to generate the update, delete and insert statements. To generate these statements, the SNAP_COLUMN_MAP table is used. The statement is following:

select source_column_name, dest_column, source_column_id  
from snap_column_map  
where source_table_name = pin_table_name  
and source_owner = pin_owner  
order by source_column_id;

In this query, pin_table_name is “SINAV” which is replicated table name and pin_owner is “OBS” which is replicated table owner. This query returns the column

49
mapping information as rows of the master table and replicated table's columns which are entered in Figure 4-2 in the order of replicated table column sequence. The output of this statement is following:

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YIL1</td>
<td>YIL1</td>
<td>1</td>
</tr>
<tr>
<td>YIL2</td>
<td>YIL2</td>
<td>2</td>
</tr>
<tr>
<td>DONEM</td>
<td>DONEM</td>
<td>3</td>
</tr>
<tr>
<td>BILNO</td>
<td>BILNO</td>
<td>4</td>
</tr>
<tr>
<td>DERS_KOD</td>
<td>DERS_KOD</td>
<td>5</td>
</tr>
<tr>
<td>UYE_KOD</td>
<td>UYE_KOD</td>
<td>6</td>
</tr>
<tr>
<td>A1_NOT</td>
<td>A1_NOT</td>
<td>7</td>
</tr>
<tr>
<td>A2_NOT</td>
<td>A2_NOT</td>
<td>8</td>
</tr>
<tr>
<td>F_NOT</td>
<td>F_NOT</td>
<td>9</td>
</tr>
<tr>
<td>BUT_NOT</td>
<td>BUT_NOT</td>
<td>10</td>
</tr>
<tr>
<td>YUK_NOT</td>
<td>YUK_NOT</td>
<td>11</td>
</tr>
<tr>
<td>TD_NOT</td>
<td>TD_NOT</td>
<td>12</td>
</tr>
<tr>
<td>ONAY</td>
<td>ONAY</td>
<td>13</td>
</tr>
<tr>
<td>EK_YUK_NOT</td>
<td>EK_YUK_NOT</td>
<td>14</td>
</tr>
<tr>
<td>SINIF</td>
<td>SINIF</td>
<td>15</td>
</tr>
</tbody>
</table>

Using these rows in a loop C_CHANGE cursor and all the statements are generated. Also primary key fields are determined by dynamically generating the primary key statement for the master site. Because the link information cannot be dynamically written in the statements. In the from part of the query for example dba_cons_columns@HHOBS.WORLD is used and HHOBS.WORLD is the link information which is saved in the LINK column of SNAP_TABLE, and this information is not static but in the query this must be static. So that in the TRG_SNAP_TABLE procedure primary key query is dynamically generated. Then running this dynamic statement, the primary key fields are determined and the WHERE clause of the all statements are generated.

To run refresh procedure periodically for a given interval in replication site Oracle Database Management System must be configured to start SNP background process. SNP background process is actually a scheduler that periodically checks the job queue and runs the scheduled procedures. To start SNP background processes job_queue_processes and job_queue_interval initialization parameters must be set in initialization file of the database. In Oracle 7 these parameters are
snapshot_queue_processes and snapshot_refresh_interval. To periodically run the refresh procedure DBMS_JOB built in package is used. To submit a job to the database scheduler the following simple PL/SQL block is used in general [5].

Declare
    Jobno number;
Begin
    DBMS_JOB.SUBMIT(jobno, 'procedure name', start_date, 'interval');
End;

Package specification of submit procedure is following:

DBMS_JOB.SUBMIT(job out binary_integer,
                  What in varchar2,
                  Next_date in date,
                  Interval in varchar2);

4.4. Security Problems of the Implementation

In master site or in replication site, there are some security issues. In both sites we are are using the SYSTEM user who has DBA privilege. In Oracle DBA role is the top level role and a DBA in the database can do every kind of database operations, because DBA privilege is a role that covers every privilege in the database. But in TRG_SNAP_LOG and TRG_SNAP_TABLE procedures we are using dynamic sql statements, in dynamic sql roles are not considered and therefore any authorization that we need must be granted explicitly. So that the privileges following are granted explicitly to the SYSTEM user.

- create any table
- create any index
- create any procedure
- create any trigger
- select any table
- insert any table
- update any table
- delete any table
• drop any table
• drop any trigger
• drop any index

This security problem is not well-documented in the Oracle documentation and it seems to be a bug but it is not a bug and this is an additional security for dynamic sql usage [4, 12].

4.5. Installation of the System

To install the system, first you should install an Oracle database in two machines then;

• Create snap_log table in the master site
• Create trg_snap_log procedure in the master site
• Grant privileges mentioned above to the SYSTEM user in both master and replication site
• Create snap_table and snap_column_map table in replication site
• Create trg_snap_table procedure in replication site
• Create a public database link to master site database in replication site database

To use front-end tool you can use any client computer in the network. In client PC you should install Developer 2000 client and run replication.fmx with the related forms driver.

Replication system can be used in any computer or in any operating system. Because this system is portable to any system where Oracle runs.
5. CONCLUSIONS

In this thesis, an alternative read-only table replication system is developed for ETT process of data warehouse systems. This alternative system has some restrictions such as primary key columns of the master table must be replicated as is. But this kind of restrictions are found in similar systems. Our aim for this alternative system is to use it in ETT process of data warehousing and this goal is achieved.

In new studies, restrictions of this system can be removed and add some additional capabilities such as replicating only subset of a master table rows may be added. In this thesis, we are replicating subset of columns of master table but this subset must include all of the primary key columns and in some cases replication of subset of rows are required. To do this log table must contain all of the columns that are used in the "WHERE" clause of the defining query of the master table. But usually this subsetting is not needed in ETT process.

Comparing the other ETT methods, this method minimizes the user intervention and using this tool does not require deep understanding of database tools. For example using unload utility for ETT needs experience on database administration.

Nowadays, database users usually use visual tools and do not have the information behind that visual capabilities. In this study, Oracle database dictionary is used to retrieve information related to table's primary key, columns and database links between distributed databases. Anybody who wants to learn the internal structure of Oracle can find the source codes in disk as a good example.

In this thesis, development language which is PL/SQL is not explained in detail. Because it is explained in Oracle Application Developers Guides and anybody who wants more information can find these documents from Oracle web pages easily.
REFERENCES


Kenan Çiftçi was born in Yavuzeli, Gaziantep in 1966. He received his B.S. degree in Computer Engineering from the Middle East Technical University in June 1989. He worked in Turkish Air Force Headquarter in Ankara as a DB2 and Oracle DBA from 1989 to 1996. Since then he has been a faculty member in Turkish Air Force Academy and DBA in Data Processing Department of Turkish Air Force Academy. His main areas of interest are relational database management systems, distributed databases, data warehousing, data mining and e-commerce.