# Web Clickstream Data Analysis Using a Dimensional Data Warehouse

by

Richard D. Li

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

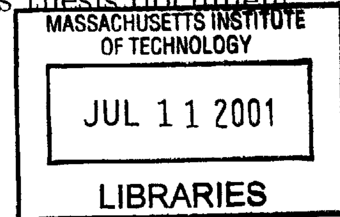
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

December 2000
[February 2001]

© Richard D. Li, MM. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.




BARKER

Author ..........................................................
Department of Electrical Engineering and Computer Science
December 15, 2000

Certified by ..........................................................
Harold Abelson
Class of 1922 Professor
Thesis Supervisor

Accepted by ..........................................................
Arthur C. Smith
Chairman, Department Committee on Graduate Students

# Web Clickstream Data Analysis Using a Dimensional Data Warehouse

by

Richard D. Li

Submitted to the Department of Electrical Engineering and Computer Science
on December 15, 2000, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

## Abstract

In this thesis, I designed, implemented, and tested an open, platform-independent system for analyzing the clickstream data of a web site. This system included a web server clickstream logging system, a dimensional data warehouse for storing clickstream data, software for populating the data warehouse, and a user interface for analyzing the clickstream data. Privacy, performance and scalability, transparency, platform independence, and extensibility were established as the design goals of the system. Early implementations of the clickstream system were deployed on three web sites, one of which received over four million HTTP requests a day. Data from these initial deployments contributed to the development of the system and was used to assess the extent to which the final clickstream design fulfilled the original design criteria. Final results from these deployments showed that the clickstream system was able to fulfill the design goals, providing accurate data that was unavailable through conventional web server log analysis.

Thesis Supervisor: Harold Abelson
Title: Class of 1922 Professor

# Acknowledgments

Many people have given me support, encouragement, and advice throughout the process of researching and writing this thesis. They include Nobuko Asakai, Wendy Chien, Randy Graebner, Samuel J. Klein, Vicki Kwok, Rajeev Surati, and my parents. A very special thanks also goes to Jon Salz, whose help was invaluable throughout the design and implementation of the clickstream system.

# Contents


**1 Introduction** **8**

1.1 The Clickstream System . . . . . . . . . . . . . . . . . . . . . . . . 8

1.2 A Clickstream User Scenario . . . . . . . . . . . . . . . . . . . . . . 10

**2 The Clickstream Architecture** **13**

2.1 The Life Cycle of a Click . . . . . . . . . . . . . . . . . . . . . . . . 14

2.2 The Clickstream Client . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.2.1 The ArsDigita Community System . . . . . . . . . . . . . . . . 16

2.3 The Clickstream Server . . . . . . . . . . . . . . . . . . . . . . . . . 17

2.3.1 Downloading and Loading Data . . . . . . . . . . . . . . . . . . 17

2.3.2 Populating the Warehouse . . . . . . . . . . . . . . . . . . . . . 17

2.3.3 Reports . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

2.4 The Clickstream Data Model . . . . . . . . . . . . . . . . . . . . . . 19

**3 Background** **22**

3.1 Data Warehousing . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

3.2 Gathering User Data . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

3.3 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

3.3.1 Accrue . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

3.3.2 Oracle . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

3.3.3 Other software providers . . . . . . . . . . . . . . . . . . . . . . 26

3.4 Definitions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

3.4.1 Session . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

3.4.2 Hit . . . . . . . . . . 28
3.4.3 Performance and Scalability . . . . . . . . . . 28

**4 The Clickstream Design** **29**

4.1 Privacy . . . . . . . . . . 29
4.1.1 Preserving Privacy . . . . . . . . . . 30
4.1.2 Aggregation . . . . . . . . . . 30
4.2 Performance and Scalability . . . . . . . . . . 30
4.2.1 Granularity . . . . . . . . . . 32
4.2.2 Minimizing Data Collection . . . . . . . . . . 35
4.2.3 Client-side Performance . . . . . . . . . . 35
4.3 Transparency . . . . . . . . . . 36
4.3.1 Design . . . . . . . . . . 36
4.4 Platform Independence . . . . . . . . . . 36
4.4.1 Design . . . . . . . . . . 37
4.5 Extensibility . . . . . . . . . . 37
4.5.1 Dimensional Data Warehouse . . . . . . . . . . 37
4.5.2 Reports . . . . . . . . . . 38

**5 Implementation** **39**

5.1 Server-Side Performance . . . . . . . . . . 39
5.1.1 Materialized Views and Query Rewrite . . . . . . . . . . 39
5.1.2 Direct Path Inserts . . . . . . . . . . 40
5.1.3 Disabling Constraints . . . . . . . . . . 41
5.1.4 Hints . . . . . . . . . . 41
5.1.5 Dynamic PL/SQL . . . . . . . . . . 42
5.1.6 Reducing the Data Warehouse Size . . . . . . . . . . 42
5.2 Client-Side Logging . . . . . . . . . . 44
5.3 Reports . . . . . . . . . . 46
5.3.1 The Dynamic Site Map . . . . . . . . . . 46
5.3.2 The Entry Pages Report . . . . . . . . . . 46

5.4 Additional Features . . . . . . . . . . . . . . . . . . . . . . . . . . 47
5.4.1 Multiple Tablespaces . . . . . . . . . . . . . . . . . . . . . . 47
5.4.2 ACS 4.0 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49

**6 Testing and Results** **50**
6.1 Privacy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
6.1.1 Methodology . . . . . . . . . . . . . . . . . . . . . . . . . . 50
6.1.2 ArfDigita . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
6.1.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
6.2 Performance and Scalability . . . . . . . . . . . . . . . . . . . . . 53
6.2.1 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53
6.3 Transparency . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
6.4 Platform Independence . . . . . . . . . . . . . . . . . . . . . . . . 54
6.5 Extensibility . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
6.6 Accuracy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56

**7 Conclusion** **59**
7.1 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
7.1.1 Performance and Scalability . . . . . . . . . . . . . . . . . . 60
7.1.2 Multiple Data Source Integration . . . . . . . . . . . . . . . 60
7.1.3 Portability . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
7.1.4 Evolutionary Improvements . . . . . . . . . . . . . . . . . . 61
7.2 Lessons Learned . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62

**A Clickstream Data Model** **63**

# List of Figures

1-1 A Navigation Report: The Dynamic Site Map . . . . . . . . . . . . . . 11

2-1 The Clickstream Architecture . . . . . . . . . . . . . . . . . . . . . . . . 13

2-2 Five Dimensions of the Clickstream Data Model . . . . . . . . . . . . . 20

5-1 A Navigation Report: The Entry Pages Report . . . . . . . . . . . . . 48

6-1 Total Daily Sessions, NetTracker vs. Clickstream . . . . . . . . . . . . 57

6-2 Total Daily Pageviews, NetTracker vs. Clickstream . . . . . . . . . . . 58

# Chapter 1

# Introduction

The growing sophistication of web services has created a need for a better understanding of how visitors use a web service. Such information can be used to improve the user experience, provide targeted marketing, personalize content, determine the return on investment of a marketing campaign, and more. However, this type of analysis technology is uncommon and undeveloped. The most common tools for analyzing a user experience are web server log analyzers, but these tools can only analyze a subset of the data available to a web site.

## 1.1 The Clickstream System

A clickstream analysis system was developed to address this problem of analyzing and aggregating user behavior patterns on a web site. Clickstreams are the streams of clicks generated as a user browses a web site. A clickstream not only includes the links clicked by a user, but also the interval between clicks. This data can be correlated with the user's identity, referrals, sessions, geographic location, purchase history, and other dimensions in a database. Thus clickstreams are the raw data that can show how specific user groups navigate and use a web site. With this information, a web site can personalize its navigation and content for a particular user or category of users with similar clickstream profiles. In addition, the information provided by clickstream logging can be immensely valuable in determining how a site is being

used, determining which user groups see what content, assessing the effectiveness of advertising campaigns, targeting advertising, and analyzing purchase patterns among different groups of people.

The system developed here was designed to be a simple, scalable, platform-independent, open clickstream analysis system that collected clickstream data without infringing on user privacy. The system consists of three distinct components: a client-side logging system, a server-side data warehouse and population facility, and a user interface for server-side reports. The client is the actual web site that is performing clickstream logging, while the server is a dedicated data warehouse system. Clickstreams are logged on each of the web server(s) of the client site and periodically transferred via HTTP to the server, where the data is loaded into a dimensional data warehouse for analysis. This approach is non-invasive and works transparently with all common web server architectures. A server-side reports interface provides a user interface to the data warehouse and allows analysts to query the data warehouse for information. The simplicity and open nature of this approach ensures its compatibility with most web site architectures. Moreover, this architecture allows for maximum integration with relational data such as user location or shopping cart contents collected by the web site itself, since the clickstream logging system is integrated with the web server. Less integrated systems are unable to access this relational data, since their capabilities are limited to analyzing externally observable data such as page load times, URLs requested, and the like.

A dimensional data warehouse was designed to store all clickstream data on the server [1]. A dimensional data warehouse (see Section 3.1) is a standard model for data warehouses that relates specific data dimensions to a set of basic facts. The clickstream data warehouse uses a single page request as the granularity of the basic fact, and relates these facts to data dimensions such as a users dimension, a browser dimension, and a sessions dimension. This granularity was chosen over a session-level granularity due to the greater amount of information that is retained in such a system. In addition to the data warehouse, software was developed to parse the clickstream logs and populate each of the dimensions of the data warehouse.

A server-side reports interface provides a flexible facility for creating additional reports which can be automatically generated and archived in HTML. In addition, the reports interface features a number of dynamic reports that display different data based on user input e.g., the dynamic site map reveals additional data when the user clicks on a particular link.

The rest of this paper begins with a background on data warehousing, an overview of the general techniques of gathering user data, an examination of common approaches to analyzing user data, and several definitions of common terms that are used throughout this paper in Chapter 3. Chapter 4 then introduces the design goals of the clickstream system and discusses the design decisions that were made to fulfill these goals. The specific implementation of the clickstream system is covered Chapter 5. This chapter discusses the implementation challenges and the solutions developed to meet those challenges. These challenges included optimizing server-side performance, the evolution of the client-side logging software, and the server-side reports interface. Chapter 6 discusses the testing methodology and results of the deployment of the clickstream system on three separate web sites, including a commercial high-volume web site serving over four million HTTP requests a day. The clickstream system is evaluated on the basis of each of the design goals outlined in Chapter 4, and the extent to which the system fulfills each of these goals is assessed in this chapter. Finally, Chapter 7 examines the final clickstream system, summarizes the capabilities of the system, and discusses what these results mean.

## 1.2 A Clickstream User Scenario

Suppose the Internet e-commerce startup foo.com wants to know how users view the web site to make the web site easier to use. The foo.com webmaster installs the clickstream logging system on the foo.com web site and accumulates clickstream data. Each day, the clickstream logs are downloaded by the clickstream data warehouse server and loaded into the data warehouse.

After a week, the webmaster generates a dynamic site map similar to the site map

depicted in Figure 1-1. The webmaster looks at the site map and gets a picture of how users navigate the web site. He notices that one of the key pages of the foo.com site, the "Hot Deals" page, is typically accessed by users clicking through a roundabout chain of six links instead of clicking directly on the link from the top-level index page. Moreover, he notices that the number of users who click through each link of the chain decreases as a user gets deeper into the chain. He realizes that the link to the "Hot Deals" page is not sufficiently prominent, and changes the placement of the "Hot Deals" link on the top-level foo.com page.

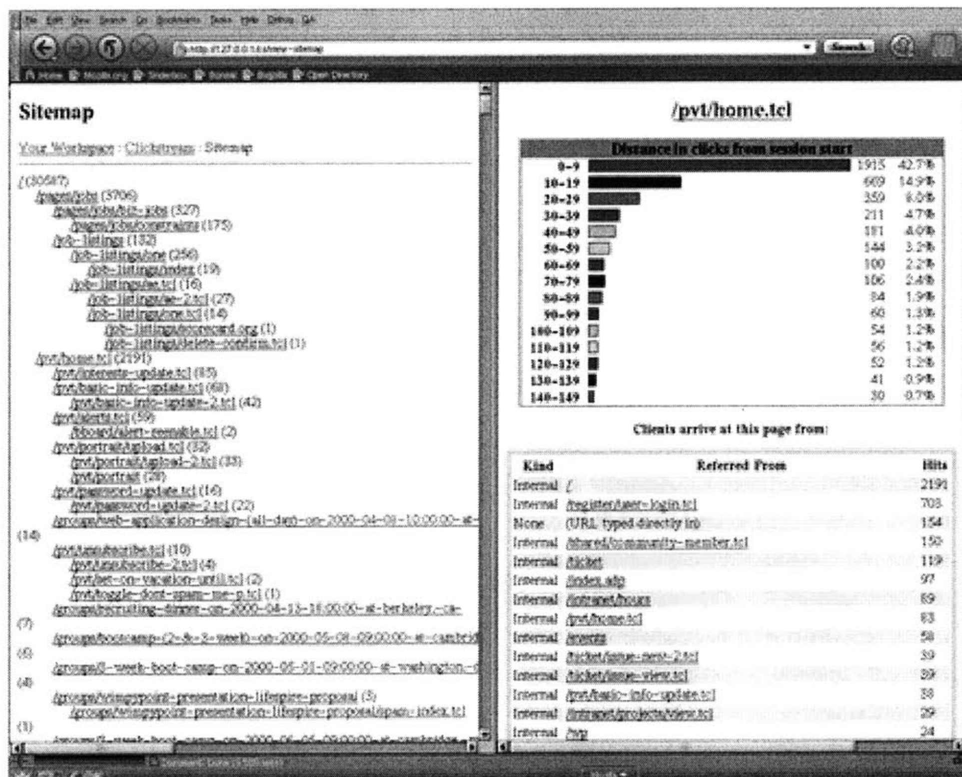



Figure 1-1: A Navigation Report: The Dynamic Site Map

A week later, he reviews the clickstream logs. He notes that the chain of clicks to the "Hot Deals" page is now one link long, and that most people who visit the site click through from the top-level foo.com page to the "Hot Deals" page. Checking the "Most Commonly Requested Pages" clickstream report, he is satisfied to see that the "Hot Deals" page is now the second-most requested page after the top-level foo.com

page.

The webmaster, as he becomes more familiar with the clickstream system, builds a set of reports that answer many of the questions that foo.com wants to know about the visitors to the web site. He adds these reports using the reports infrastructure available in the clickstream system, creating a set of reports that are automatically generated every night:

- What are the most popular pages on foo.com? What are the most popular pages for females between the ages of 25 and 35?
- How do people arrive at foo.com? Where do they go when they leave? How long do they stay?
- What percentage of sessions resulted in a sales transaction?
- Do an unusually large percentage of people leave the site with a loaded shopping cart on the same page?

The clickstream system developed in this project is a framework that web site maintainers can use and extend to analyze the clickstream data gathered by a web site.

# Chapter 2

# The Clickstream Architecture

The clickstream architecture relies upon a dedicated data warehouse server to perform computationally intensive processing and a lightweight client-side program to log clickstreams to a file. In the clickstream system, a user agent sends an HTTP request to a web server. The web server responds to the HTTP request and appends a record of the request to the clickstream log. The clickstream logs are periodically transferred over HTTP to the dedicated server, where they are loaded into the clickstream data warehouse. This process is illustrated in Figure 2-1.

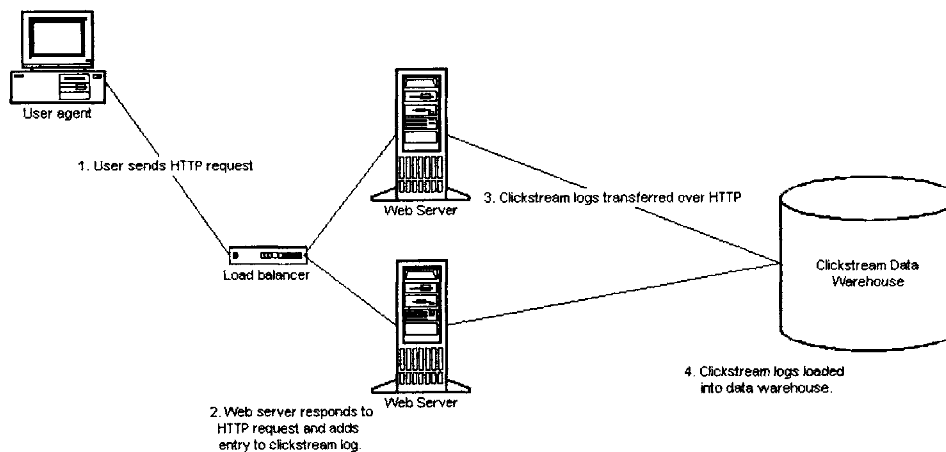



Figure 2-1: The Clickstream Architecture

## 2.1 The Life Cycle of a Click

The life cycle of a single page request in the clickstream system can be described in five discrete steps.

1. A user requests a particular page on a web site, either by clicking on a link or typing a URL into her browser. The server responds to the HTTP request for the page and adds a line to a log file containing detailed information about the request.
2. Every 24 hours, the data warehouse server downloads the compressed clickstream log for the previous 24 hour period via HTTP. The data warehouse server decompresses the log file and loads the clickstream data into the event log table in the database.
3. Entries in the event log table are parsed directly in Oracle by the data warehouse population software. This process populates the dimensions and, for each row in the event log table, inserts a row into the data warehousing fact table.
4. Standard reports are generated from the information in the data warehouse.
5. Clickstream software users can view the reports or execute additional queries against the data warehouse for more data.

## 2.2 The Clickstream Client

The clickstream client software is functionally identical to the logging functionality of any standard web server. The only difference between a typical web server logger and the clickstream logger is that the clickstream logger logs a superset of the data logged by a typical web server logger. Each line in the log file has the following fields:

- The time at which the page was requested, in seconds elapsed since January 1, 1970

- The time at which the download of the page was completed, in seconds elapsed since January 1, 1970
- The date of the request
- The URL requested
- The IP address of the request
- The user ID of the visitor, if known
- The query string in the request URL
- The content type of the page delivered
- The content length of the page, in bytes
- The session ID
- The browser ID
- The user agent string
- The accept language string
- The referring URL
- The HTTP method
- The HTTP status
- Whether or not the connection occurred over SSL

The line below is a typical line from a clickstream web log. The page was requested at 971064031. The second number is the time at which the server finished serving the page. The page was served on October 9, 2000, and the URL served was /, or the index page. The log also contains the user ID (7751), the browser (Microsoft Internet Explorer 5.5), and the referral URL (the etour.com web site).

```
971064031    971064032    20001009    /    24.112.25.125
↪ text/html; charset=iso-8859-1    7751    171794    171793
↪ Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)    en-us
↪ http://www.etour.com/member/start.asp?intD=Pets    GET    200
↪ f
```

Page requests are logged into files. Each file has a descriptive name such as *servername*-cs.log.2000-04-25.14:00 (this example is for the period of time between 2:00 and 3:00 p.m. on 25 April 2000). Every hour, a new clickstream log file is created. Every 24 hours, the hourly clickstream logs are concatenated into one large log file and compressed for download and the hourly logs are deleted. Hourly logs are kept so that up-to-the-hour reports can be generated if necessary.

### 2.2.1 The ArsDigita Community System

The clickstream reference implementation is integrated with the ArsDigita Community System (ACS), an open-source platform for web development [2]. The ACS provides much of the basic functionality and data models that are essential to any web service; these functions include user login and registration, the concept of users, session tracking, and user groups. In addition, the ACS provides its own cookie management mechanisms to track user logins and sessions. This data is used by the clickstream client system; the user ID and session ID data in the log files are obtained directly from this integration with the ACS.

The standard web server for the ACS is AOLserver, an open-source web server used and maintained by AOL. The AOLserver `ns_register_filter` API [3] is used to intercept requests to HTML pages so that information about each HTTP page request can be logged to a clickstream log file in the file system.

## 2.3 The Clickstream Server

The clickstream server is a dedicated data warehouse server. The server has several responsibilities. First, the server is responsible for periodically downloading the clickstream log data from the clickstream clients and loading the data into the database. Second, the server is responsible for populating the data warehouse. Finally, the server runs the software that allows users to analyze the data inside the clickstream warehouse.

A dedicated data warehouse machine is used because populating the data warehouse is computationally intensive and would substantially affect production database performance if they were performed on the production database. Moreover, database configurations are different for online transaction processing (OLTP) and data warehouse applications.

### 2.3.1 Downloading and Loading Data

HTTP is used to transfer the compressed clickstream log files from the production web server(s) to a dedicated data warehouse machine. The data warehouse server automatically downloads the clickstream log files from each of the production web servers. Once each log has been transferred, the log is decompressed, and a SQL*LOADER control file is dynamically generated with the log data. SQL*LOADER is a high-performance engine for loading external data into Oracle, and is capable of loading tens of thousands of rows per second into Oracle when in direct path mode, which bypasses the Oracle kernel and loads data directly into the database files [4]. In the clickstream system, SQL*LOADER is invoked in direct path mode to rapidly load the data into the event log table called `cs_event_log`.

### 2.3.2 Populating the Warehouse

Once data has been logged into the `cs_event_log` table, a set of stored procedures written in PL/SQL is run to populate the data warehouse. PL/SQL is a set of procedural language extensions to SQL available in Oracle databases. Since PL/SQL runs

natively inside the database, PL/SQL is very efficient at processing large amounts of data. A considerable amount of computation occurs while populating the data warehouse, so PL/SQL was used to maximize performance. The main data population routine performs several operations:

1. The routine calls PL/SQL procedures that populate the pages, referrers, and user agents dimensions of the warehouse. These procedures obtain a list of all entries in the appropriate dimension, and compare this list of entries to the incoming data set. If there is any data in the `cs_event_log` table that is not in the dimension, the procedures insert the additional data. For instance, there is one row in the referrer dimension for each unique referrer. The `cs_create_pages_and_referrers` procedure scans the `cs_event_log` table and inserts any new referrers into the `cs_dim_referrers` table.

2. The fact table is updated with any new facts. This step must occur after all the dimensions have been populated to ensure that the the foreign key references are correct.

3. The dynamic sitemap tree is created in the `cs_sitemap` table.

### 2.3.3 Reports

A set of standard reports are generated out of the data warehouse. These reports include the most requested pages report, the average session duration report, and the dynamic site map. Since each report requires queries against large amounts of data, each report is automatically generated, with the HTML saved directly into the file system. Thus, subsequent requests for the reports do not require repeated execution of the expensive queries.

The clickstream system provides a common set of reports as a starting point for users of the software. These reports fall into four basic categories:

- **Navigation reports** that show how the site is being navigated. The dynamic site map shown in Figure 1-1 is a navigation report.

- **Visitor reports** that show how many registered and unregistered visitors arrive during a particular time frame. This information is deduced from login information; users who have logged in to the system have a valid user ID logged to the clickstream log.

- **Session reports** that show the average session duration, the most common pages that visitors first see in a session, the most common content for a particular visitor demographic, and the most common ways users leave the web site.

- **Site analysis reports** that show the most requested pages, the most requested pages that don't exist, the most common ways people arrive at the web site (referrals), and the most common browsers used by site visitors.

The clickstream system also provides a reports infrastructure so that adding a new automatically generated report is simple. Adding a new report requires the user to add a new row into a single database table so that the clickstream system is aware of the report, write the actual query to perform the report, and format the results of the query in HTML using a few clickstream helper functions.

## 2.4 The Clickstream Data Model

The clickstream data model is a standard star schema dimensional data warehouse (see Section 4.5.1 for a discussion about the motivation for using this design). A large number of dimensions are joined to a single central fact table by artificially generated foreign keys. As illustrated in Figure 2-2, the set of dimensions referencing a single fact table creates a star-like pattern.

The clickstream data model defines a number of dimensions, joined to a central fact table called `cs_fact_table`. Each dimension contains data that answers different types of questions:

- **Page dimension** What was the name of the page requested? What type of page was requested? What was the URL? What was the content type?

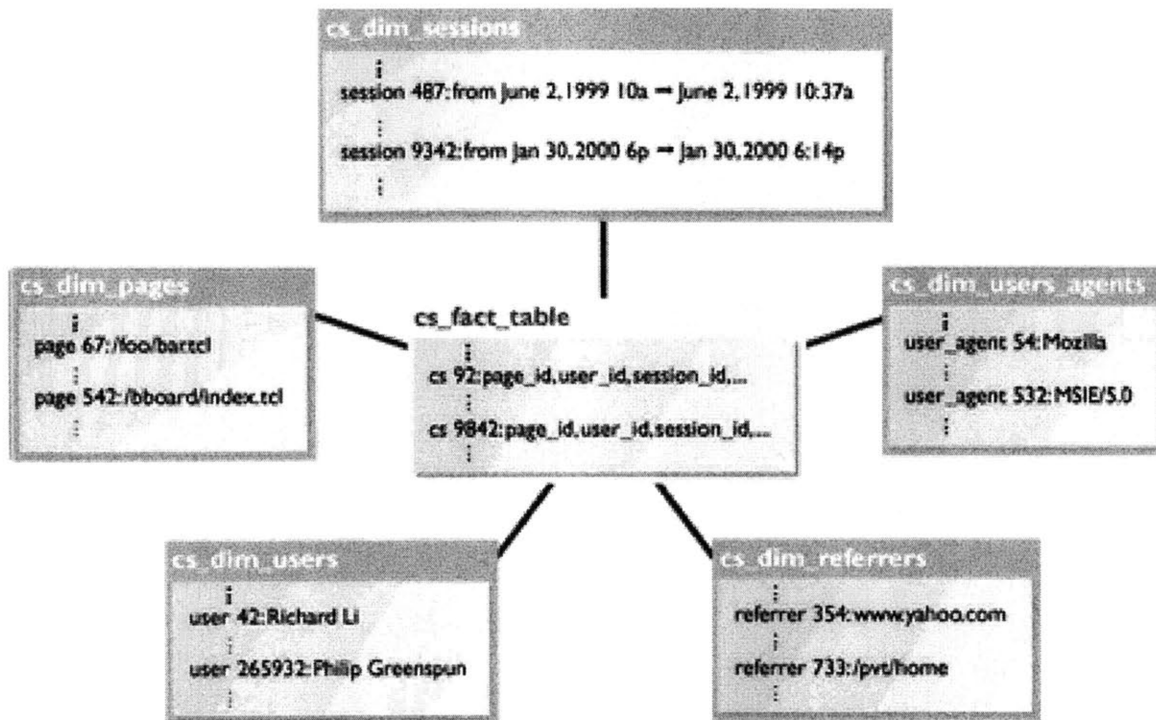



Figure 2-2: Five Dimensions of the Clickstream Data Model

- **Referrer dimension** What kind of referrer was it (e.g., search engine)? What was the referring URL? What search text, if any, was used? If it was from inside the site, what is the referring page ID?
- **Date dimension** Which day of the week was the request? Day in the month? Day in the year? Which week in the year?
- **User agent dimension** Which browser is the user using? Which version? Which operating system?
- **Session dimension** When did a particular session start? When did it end? Which visitor? How many clicks were part of the session? What was the total value of the sales in the session?
- **User dimension** What user segment does the given visitor belong to e.g., is the visitor in the "have seen privacy policy" segment? Note that the depiction of the

user dimension in the data model illustration labels user ID's with individual identities; this does not actually occur in the actual clickstream data model.

The core clickstream data model is provided in Appendix A.

# Chapter 3

# Background

## 3.1 Data Warehousing

A data warehouse is a relational database whose goal is the analysis of large amounts of data. This goal is distinct from the other common use of databases – transaction processing – whose goal is to maintain records of transactions. A data warehouse contains data derived from transaction data, but may also include data from other sources. A data warehouse is used to analyze data for patterns and to answer questions that may not be asked until after the data is actually available. In other words, a data warehouse must be a flexible structure for storing vast quantities of data so it can answer any question that may be posed to it – questions that are not anticipated at design time [5].

A star schema is the simplest, most basic design for a data warehouse. The heart of the star schema is the fact table, which has one row for each fact in the system. Choosing the level of detail in the information that goes into a particular row is known as determining the granularity of the data warehouse. Each column of the fact table is a foreign key that references a particular dimension; data about a particular dimension is stored in dimension tables that are joined with the fact table as needed. For instance, a data warehouse for a consumer products retailer company might have a granularity of one product sale per row. The dimensions of the data warehouse may include which product was sold, where it was sold, the price at which

it was sold, and the date when it was sold. When a query is asked that requires data from multiple dimensions, these dimensions are joined by the database to form what appears to be one large table with the necessary data.

The process of building a data warehouse can be broken down into three phases: gathering data, data transformation and data warehouse population, and querying the database [6]. In the example above, the data gathering process would collect data about every sale transaction, store location, and any other data required by the data warehouse. The data gathering process usually will require that data be gathered from multiple sources e.g., sales information is collected from each retailer, geographic information comes from a company database, and so forth. The data transformation phase takes all the data gathered in the previous step and loads the data into the warehouse. This process reconciles any inconsistencies in the data (for example, data from one source might use code BLU to indicate an item with the color blue, while another source may use the code BLUE to mark the same attribute), computes any necessary values from the raw data, and populates the data warehouse with the processed data. The third phase is targeted at the end user, and involves providing user interfaces that allow users to query and analyze the data in warehouse. Each of these phases is actually an ongoing process; new data is continually gathered, transformed, and analyzed.

## 3.2 Gathering User Data

Several common approaches exist today that can be used to gather clickstream data. The most common of these is web server logging. Modern web servers generate copious amounts of log data. A typical web server will log each HTTP request into an ASCII log file stored in the file system. The information usually contained in each log file entry includes IP address, time, the actual HTTP request, and the browser type and version; each entry is typically formatted according to the Common Log Format (CLF) specification. However, most log formats, including CLF, do not support logging individual visitor identity to the log files. In addition, correlating

log file entries with database information is complex and difficult. Thus analyzing standard web server logs is not an ideal means of gathering detailed clickstream data.

Most web sites also rely on the use of "cookies" to uniquely identify client browsers. The use of cookies allows a web site to associate arbitrary state with a single browser. Examples of state typically associated with cookies include user authentication information and the contents of the user shopping cart. Cookies are a useful mechanism to uniquely identify browsers (and generally users as well, although more than one user may be using the same browser). Clickstream systems can take advantage of cookie information to associate a particular set of page requests or actions to a particular browser and/or user identity.

User data can also be gathered directly by the web server. Many web servers support a filter mechanism whereby the same program is run after each HTTP request is fulfilled. This approach, in conjunction with some method of user tracking such as cookies, allows the greatest level of detail for collecting clickstream data, since different sources of information – user logins, user identity, sessions, demographics, and more – are simultaneously available and can be integrated into a single data warehouse for data mining and analysis [7].

## 3.3 Related Work

The field of analyzing clickstream data is still in its infancy. Many companies are planning products to address this market, but there is no commonly accepted solution or approach to addressing the issues involved. The market leader in this area is Accrue software, but other major software companies such as Microsoft and Oracle have recognized the importance of the clickstream market and have announced or shipped initial versions of clickstream analysis products.

### 3.3.1 Accrue

Accrue software is the leading commercial provider in the field of data warehousing clickstream data. The full Accrue Insight system uses a complex multi-tiered archi-

tecture that relies on a combination of dedicated servers that analyze the incoming network stream (packet sniffing), dedicated data warehouse servers, log analyzers, and custom applications that use pre-defined APIs to communicate with the remainder of the architecture. Accrue also provides a user interface to a standard set of queries against the data warehouse similar to the reports interface mentioned in Section 2.3.3 [8].

The clickstream system presented here will be fully integrated with the web server and database. This architecture has three advantages over the Accrue architecture: simplicity, portability, and integration. By residing inside a dedicated data warehouse server, no additions to any existing web site infrastructure are required in order to run the proposed clickstream system. In addition, the data model was designed to be portable across multiple databases and operating systems, which offers a more platform-independent abstraction layer. Finally, as discussed in Section 2.2, complete integration with the web server and database allows full access to all the HTTP information collected by the web server and the transaction information stored in the database. The packet sniffing component of the Accrue system cannot handle encrypted connections which are commonplace on ecommerce sites; by logging data directly at the web server level, the system here avoids this problem.

In addition, the clickstream system will be completely open. By using a standard, published data model, the system will be more flexible in adapting to the unique needs of a particular web site. The open source approach to development also means that sites using the system proposed here are not constrained by any user interface provided, as they are with the Accrue system.

### 3.3.2 Oracle

In October 2000, Oracle announced the release of its Clickstream Intelligence 1.0 software. The marketing literature [9] for this software package indicates that Clickstream Intelligence consists of a prepackaged data warehouse for clickstream analysis, graphical user interfaces for configuring the system, and a portal-based reports system.

In actuality, the software consists of a pre-built data warehouse repository built on Oracle's proprietary Oracle Warehouse Builder framework as well as a set of tools to build a clickstream data warehouse. Some of the tools and features that are provided by the Clickstream Intelligence package include Web server log parsing tools, reporting tools, and tight integration with Oracle 8*i*.

The Clickstream Intelligence system seems to be a capable clickstream system, but three factors distinguish the clickstream system presented in this paper from the Oracle Clickstream Intelligence system. First, the Clickstream Intelligence system relies upon Web server logs, parsing query and cookie string data for information. However, as discussed in Section 3.2, the CLF specification format does not include cookie data; any extension of the format to include all-important cookie data would require additional customization by the Clickstream Intelligence end user. Moreover, the customization would need to include mapping the meaning of the cookie information into a form that can be analyzed by the data warehouse e.g., a cookie that represents a user identity would need to be labeled as such by the data warehouse software. Second, the Clickstream Intelligence system by design is not database-independent, and is wholly tied to the Oracle RDBMS platform. Finally, the closed nature of the tools makes extending the platform difficult in certain respects.

### 3.3.3 Other software providers

Microsoft has announced a partnership with Syncsort, Inc. [10] that uses parsing and filtering techniques in conjunction with compression to reduce large amounts of log data to a smaller set of data which can then be stored for analysis inside Microsoft's SQL Server RDBMS. Microsoft's approach to handling clickstream data is to collect all the clickstream data and perform some preprocessing prior to populating the data warehouse, discarding less useful information. This data is then summarized and stored in a RDBMS. The process of discarded and aggregating this data reduces the 200GB of raw data that is accumulated daily into 1GB of data. These daily aggregates are stored for four months, which is then further aggregated into OLAP cubes that can be accessed by analysts [11].

Another company, Macromedia (formerly Andromedia), has a popular product called Aria that performs clickstream analysis. Aria performs some sophisticated web log analysis, such as a breakdown of external referrers, total hits, top visitor domains, and so forth. In addition, they perform some primitive clickstream analysis that provide reports of the most popular content, the number of repeat visitors, and an analysis of the entrance pages for visitors. The set of default reports in the clickstream system provides this functionality , and the clickstream data model allows significantly deeper analysis.

A different approach is used by DataSage, now a subsidiary of Vignette. DataSage's netCustomer product builds customer profiles based on clickstream data, and then stores the customer profile in a RDBMS. The actual clickstream data is not retained [11].

Other products on the market allow web sites to dynamically tailor content to users based on user profiles and rules. One such example of this technology is the Art Technology Group's Dynamo server suite. The Dynamo personalization server allows a web site to program business rules in an English-like syntax that dictate what content is shown to which types of user (e.g., "show Wealthy Individuals special one million dollar minimum investment options"). However, this technology still requires considerable effort to build user profiles and construct these rules. A true clickstream analysis product would allow user profiles to be built dynamically based on their clickstream behavior.

## 3.4 Definitions

There are no commonly accepted definitions of important traffic-related terms in use today. Instead, web server log analyzers each use its own definition of session, hit, and other words. The words that are used throughout this paper are defined here to eliminate any ambiguity.

### 3.4.1 Session

A session is defined as a series of page requests from the same unique browser ID, each of which must occur before a timeout occurs. The default value for this timeout is five minutes. To prevent the system from giving inaccurate numbers for browsers that do not accept cookies, all single page request sessions are ignored in session counts. Otherwise, a single browser that rejected cookies could be represented in dozens of sessions, each with a duration of a single page request.

### 3.4.2 Hit

The word "hit" is not used in this paper, since the word has two common meanings. The first common meaning is a single page request e.g., a site serving two million hits/day is serving two million pages a day. The second common meaning of the word "hit" is a single HTTP request. This second definition is significantly different from the first, since a graphics-laden site could easily serve over twenty hits on a single page request.

### 3.4.3 Performance and Scalability

Performance and scalability are two related but distinct characteristics. In this paper, performance applies to the ability of a single instance of the software to handle a particular load. Generally, a high-performance system can process a given set of data more efficiently when compared to a low-performance system. Efficiency can be measured by a variety of metrics such as time or additional overhead imposed.

A scalable system is a system that can handle large amounts of data, even as the size of the data set grows. A clickstream system may accumulate billions of data points, and a scalable system would be able to handle a large data set given the appropriate hardware resources. Scalability often refers to the ability of the software to handle these increased amounts of data by adding more instances of the software.

# Chapter 4

# The Clickstream Design

Five key considerations drove the design of the clickstream data warehouse architecture. These considerations were privacy, performance and scalability, transparency, platform independence, and extensibility.

## 4.1 Privacy

In the age of the Internet, privacy is an important question that raises many constitutional issues. These issues become even more important with the collection of clickstream data. Although the precedents set forth by previous Fourth Amendment rulings do not answer the question of whether or not people have a right to privacy of his or her clickstream data, many feel that the intent of the Fourth Amendment is to restrict the limits of governmental access to clickstream data. Some of the possible risks of clickstream data include abuse by law enforcement agents who randomly scan clickstream data for evidence of illegal activity or abuse by Internet Service Providers who build sophisticated marketing profiles matched to user identities [12].

Protecting user privacy and preserving user trust is an important goal for the design in the clickstream system. Although risks such as the ISP marketing risk do not apply to the clickstream system described here because this system focuses on the data for a single web site, care must be taken to prevent abuse of the clickstream system. Since the clickstream of a user on a particular web site can provide a large

amount of personal information, the information must be securely protected so that it is not abused.

### 4.1.1 Preserving Privacy

The clickstream system completely decouples user identities from the clickstream data warehouse. In the clickstream system, the data warehouse user ID is obtained from the ACS (see Section 2.2.1). Each visitor is referenced by this unique user ID, but the actual data that correlates the user ID with the user's identity is kept in the production database separate from the data warehouse server. Thus, any analyst who queries the clickstream data warehouse is unaware of the actual user identity of a given user in the clickstream warehouse.

### 4.1.2 Aggregation

The most common application of clickstream data is the aggregation of clickstream data to discern user behavior patterns. Thus, the entire reports interface is designed to provide a framework for efficient aggregation and grouping of clickstream data. To protect privacy, support for aggregation of clickstream data across multiple domains is not be provided, since the intent of this clickstream system is to provide targeted marketing and user experience information on a single web site. Finally, the granularity of the clickstream system can be increased to the session level; the loss of detail would correspond directly to an increase in privacy.

## 4.2 Performance and Scalability

On a heavily trafficked site, tens or hundreds of millions of clicks can occur every day. This enormous amount of data can quickly overwhelm even the most modern databases and storage systems available today if the design of system does not provide a robust and scalable mechanism for handling billions of rows of clickstream data. For instance, the Microsoft collection of web properties including hotmail.com, msn.com,

and microsoft.com, generate over 200GB of log files daily, representing two billion page requests by 25 million unique individuals [11]. In addition, populating the data warehouse is very computationally intensive; any population scheme must be scalable and efficient.

Performance in the clickstream system can be broken down into three categories: the client side clickstream logging code, the server side population code, and the server side queries. The performance goal of the client side clickstream logging code is to have negligible performance impact on the client site. This goal is important because a clickstream system with high overhead would force web sites to incur additional hardware and hardware maintenance costs proportional to the amount of overhead. Minimizing this cost is important in the adoption of the clickstream system.

The server side population code has to be able to analyze a single day of log data in less than a day. This requirement insures that the clickstream analyzer is not the bottleneck in delivering up-to-date clickstream data. The goal of the server side population code was to be able to process the log data of a site with ten million page visits on a Sun E450 (chosen because it was the most readily available hardware) in less than a day.

Data warehousing requires a considerable amount of systems engineering. A number of techniques have been developed to handle growing data warehouses. Hardware improvements in recent years include larger and faster disk arrays, faster CPUs, and 64-bit processors that allow more RAM to be addressed. In addition, software techniques such as aggregation, materialized views that pre-aggregate expensive aggregation operations, and preprocessing of data all help alleviate the issue of rapidly growing data warehouses [13].

A number of design decisions were made to improve the performance and scalability of the system. These included choosing an appropriate granularity for the system, determining the most appropriate types of data to log, and using the file system for client-side clickstream logging.

### 4.2.1 Granularity

One of the key decisions required in the design of the clickstream data warehouse is the granularity of the fact table. Two levels of granularity can be considered in the design of the clickstream data warehouse: session level and page-request level. The issues involved in the decision represent the classic tradeoff between limiting the amount of data in the warehouse (and thus improving scalability) and retaining as much data as possible so that all subsequent analyses are possible. In both levels of granularity, the input is the same: page-request level clickstream information. Thus, the fundamental difference between a page-request level of granularity and a session-level of granularity is the point at which analysis takes place. In a page request level system, all analysis is performed after the data is loaded into the data warehouse. In a session level system, some level of analysis is performed prior to loading data in the warehouse. Analysts can use either granularity to answer the same questions; the only difference is that the session level system forces analysts to determine the types of questions to be asked in the data warehouse design phase.

#### Page-request granularity

Data recorded at the granularity of one page request is a superset of the data recorded at the session-level granularity. Therefore, analysis of data recorded at the page level also allows the user to answer questions about specific user behavior that may not have been anticipated in the original design of the data warehouse in addition to the questions that can be answered by analyzing session-level clickstream data. The cost of such an implementation is the increased amount of data that must be stored, which is directly related to the average number of pages a visitor views on a web site.

#### Session level granularity

Session level granularity records one row in the fact table for each unique user session. This approach enables analysts to categorize customers by their clickstream profile, which allows for activities such as targeted marketing and personalized content. In

addition, the data can be analyzed for information such as the most common visitor patterns, the duration of user visits, and the peak hours for user visits for a given time zone. Storing data in a system with session-level granularity means that some level of precomputation must take place before the data is stored. This precomputation improves the scalability of the system since each row represents more data. The tradeoff is that future analyses need to be anticipated at load time; if a clickstream user wishes to execute a query that was not anticipated in the original implementation of the data warehouse, the analysis can not be performed.

### Compatibility between Granularities

The choice of granularity dictates the actual implementation of the data model and population code. While the fundamental design remains the same for both levels of granularity, a number of issues need to be addressed before code can be shared. Four components of the clickstream architecture must be customized:

- the data model: certain dimensions become "degenerate" under the page-level dimension. For instance, the session dimension has no real meaning at the page-level dimension.

- the client-side code: the actual client-side logging code can be the same for both systems. However, additional logic to compute various parameters for a particular session is required for the session-level system. The extra load from this computation is negligible, but is specific to the session-level system.

- the server code: since the data model is different for the two granularities, the code that analyzes the logged data to populate the clickstream data warehouse.

- the queries: the questions that can be answered by a clickstream data warehouse with session-level granularity is a subset of the questions that can be answered by a data warehouse with a page-request level of granularity. Queries that answer even the same question will differ in both warehouses since each row of information represents a fundamentally different unit in each warehouse. For

instance, a simple query that counts the total number of sessions in the sessions data warehouse would simply count all the rows in the sessions table. However, the same query in a data warehouse with page-request level granularity would require a GROUP BY clause based on the session identifier because a session would be represented by multiple rows in the database, since a session usually would consist of multiple page requests.

### Conclusion

The page-request granularity was chosen as the basis for the clickstream data warehouse. The tradeoff between the two levels of granularity is the classic tradeoff between greater and smaller granularity: depth of analysis versus scalability. The data accumulated by both methods is the same; the only difference is the level of data that is retained. In the session-level approach, preliminary analysis is performed on the data as it is gathered, and, once the analysis is complete, the data deemed irrelevant is thrown away. The amount of data that must be stored is reduced by the average number of pages that entails a session. Kimball [1] estimates that a typical user visit requests five pages in a particular session. Thus, the session-level approach reduces the amount of data that must be stored by approximately a factor of five for the web sites involved. The downside of this approach is greatly reduced flexibility when the session-level analyses performed to create the session-level data is deemed insufficient to fulfill some need that was not anticipated at design time.

The decision to base the initial implementation of the clickstream data warehouse on the page-level request granularity was based on three reasons. First, a session-level granularity system could leverage a significant portion of the amount of the server code written for the page request granularity system, since code that performed page-request type analysis and data warehouse population would be necessary. Second, the design of the system is simplified since determining the types of questions that the system should ask is unnecessary, since no information is thrown away. Finally, this decision is consistent with the general philosophy of data warehouses of retaining all data, since anticipating the usefulness of a particular dimension of data is difficult to

anticipate.

One of the difficulties that must be addressed by the session logging approach is that the intelligence to determine the definition of a session must be implemented on the client side. This requirement has two important implications. First, the definition of a session must be defined during implementation and cannot change once data has been collected, since changing the definition of a session will invalidate historically gathered data. In addition, the client-side code must contain complex logic to determine the start and end of a session, as well as record session-specific data that can be logged into the log file.

The obvious solution is to push the analysis of the data to the server side, and to use the same client-side logging code that a page-request granularity system would make. This solution pushes the complexity onto the server, where raw computational speed is less of a concern. However, the original two limitations remain, since the data used to make the session level decisions is thrown away once it has been processed.

### 4.2.2 Minimizing Data Collection

A web site that serves four million HTTP requests a day would accumulate approximately $4000000 * 1K = 400GB$ of space a day if each row in the database requires 1024 bytes of space in the database. The commodity database technology available today is incapable of handling a system that grows by 400GB daily, so the clickstream system uses a filter mechanism that only logs HTML and dynamic pages that are requested. On a four million HTTP requests/day site, data revealed that the site had served approximately 100,000 page loads. This figure gives us a storage requirement of $100000 * 1K = 100MB$ of space required a day for storage, an amount of data that is manageable using today's technology.

### 4.2.3 Client-side Performance

The client-side logging code logs data directly into the file system for maximum performance. This approach makes the client-side code virtually as scalable as the

native web server logging mechanism, since the client-side code can be implemented by simply augmenting the server log mechanism in most cases.

The requirement that the server code run on a separate, dedicated server minimizes the amount of computation that is required on the production web servers. Thus the total overhead imposed on client sites by the clickstream system is absolutely minimal – client sites only need to run an augmented logging system in order to support clickstreaming.

## 4.3 Transparency

Developer and visitor transparency are important criteria for the clickstream system. The clickstream system should not be intrusive or visible to either the developer or web site visitor. Web developers should not be required to make significant changes to existing coding practices or techniques. Significant changes to the status quo would raise additional training costs, slow development, and introduce additional areas for error. In addition, visitors should not be able to discern any difference between a site with clickstream logging and a site without clickstream logging.

### 4.3.1 Design

The goal of visitor and developer transparency was accomplished through the use of an efficient HTTP filter that analyzes each HTTP request and logs the request if the HTTP request is for an actual page. Visitors should see no noticeable performance penalty when clickstream logging is turned on, and developers can continue working as they did prior to the installation of clickstream logging.

## 4.4 Platform Independence

All components of the system should be platform independent in design. Both the client-side clickstream system, the data warehouse, and the server-side reports interface should be platform independent. Different web sites use different technologies;

the clickstream system should not preclude the use of any particular web technology. Moreover, integrating the clickstream system into an existing web site should not require rearchitecting substantial parts of the original web site.

### 4.4.1 Design

The fundamental data model of the clickstream system is platform-independent in the sense that it can be installed on any modern RDBMS. No part of the entire system design relies on a specific feature of a given component of the reference implementation, although the reference implementation does take advantage of optimizations that are made available by the choice of architecture.

## 4.5 Extensibility

The system should be extensible so that it can be customized to answer unanticipated questions. There are two aspects to extensibility. The first aspect involves integrating additional sources of information that were not included in the original design. The second aspect of extensibility is the ability to create new reports and queries that analyze the existing data in new ways.

### 4.5.1 Dimensional Data Warehouse

The dimensional data warehouse is a standard design that is the most common data model for a data warehouse. The ubiquity of the design insures that the data model is familiar to people with a background in data warehousing. In addition, adding new dimensions to the dimensional data warehouse model is straightforward. The current implementation of the data warehouse provides a set of common dimensions, as discussed in Section 2.4. Adding additional dimensions requires three steps:

1. Designing the dimension table and adding the table to the data model.

2. Adding a column referencing the dimension table to the main fact table.

3. Writing a PL/SQL procedure to populate the new dimension and adding a call to the new dimension in the main PL/SQL procedure `cs_populate_dw`.

### 4.5.2 Reports

A generic user interface and infrastructure is provided to automatically generate reports. Adding a new report into the system is straightforward with the clickstream reports infrastructure, as discussed in Section 2.3.3. In addition, the open nature of the system makes writing new user interfaces accessible to any programmer.

# Chapter 5

# Implementation

Many issues were raised during the implementation of the clickstream design. These issues included performance, usability, and reporting. The implementation of all parts of the system was continually refined as test data was accumulated.

## 5.1 Server-Side Performance

The server-side performance of the clickstream system was measured throughout the course of its development. Numerous techniques were adopted to eliminate bottlenecks in performance.

### 5.1.1 Materialized Views and Query Rewrite

A common technique used in data warehouses is the use of materialized views and query rewrite. Materialized views are views that precompute aggregates and joins, storing the results in tables. As new rows are inserted into the underlying tables, the materialized view can be incrementally refreshed (Oracle calls this feature "fast refresh"). Since aggregates and joins on large data sets are expensive operations, a materialized view can improve performance by several orders of magnitude. Query rewrite is a technique which dynamically rewrites queries to take advantage of existing materialized views. Query rewrite is particularly useful for clickstream users who are

exploring a data warehouse, since these users are typically analysts who are not skilled at optimizing SQL queries. A database administrator can analyze the most common types of queries and create materialized views to improve their performance; query rewrite would automatically rewrite subsequent queries to take advantage of the newly created materialized views.

Early implementations of the clickstream system used materialized views and query rewrite to pre-aggregate large tables. Unfortunately, restrictions imposed by Oracle on the use of materialized views precluded using materialized views for every aggregate and join needed for queries. For instance, a materialized view of a single table aggregate cannot have a where clause. In addition, the implementation of fast refresh in Oracle 8.1.6 did not work consistently on mid-range data sets, and periodically recreating materialized views from scratch was necessary.

Ultimately, the clickstream system used a code generator to dynamically create commands to store the results of aggregates and joins in regular tables instead of using materialized views. This approach proved to be much more reliable. Since a code generator is used for this functionality, updating the clickstream software to use materialized views will be a straightforward change to the implementation, and can take place without affecting any other components of the clickstream system.

### 5.1.2 Direct Path Inserts

Fast refresh for materialized views requires direct path inserts (in Oracle, this feature is achieved through the use of the `/*+ APPEND */` hint). A direct path insert is more efficient than a regular insert because the database does not scan the table for free space and instead appends the data directly to the end of the table. However, direct path inserts requires that inserts and updates are explicitly committed before selects on the data are performed. Periodic commits means that transactional safety must be explicitly maintained by the software, since aborting in the middle of the data population process will not result in a full rollback of the population transaction. Thus the data warehouse may be partially populated. To solve this problem, the `cs_jobs` table has an integer column called `progress`; the value of `progress` is updated prior

to each commit. Transactions that have already been committed to the database are not repeated, even when the populate PL/SQL is called on the same set of data.

Ultimately, this feature was removed due to its complexity after the base system transitioned to a "one day in database" model as discussed in Section 5.1.6. In addition, debugging the system with the periodic commit model was more tedious, since an error in the system would leave the data warehouse in an inconsistent state, necessitating rebuilding the entire warehouse from scratch.

### 5.1.3 Disabling Constraints

Referential integrity in a regular database is maintained through the use of foreign key constraints. Constraints insure the integrity of the data in the database, but also impose a performance burden on the database, since each insert into a table requires that the database verify referential integrity on the inserted data. In the clickstream architecture, the data warehouse population software has exclusive responsibility for populating the warehouse. Thus, all constraints were disabled to improve the speed of loading data into the database. The constraints are still listed in the data model so that Oracle can use the constraint information in optimizing query execution plans.

### 5.1.4 Hints

Hints such as the `/*+ STAR */` hint were used to optimize queries. The `/*+ STAR */` hint instructs Oracle that the query is being executed against a star schema data model. However, these optimizations did not significantly improve the performance of the system. The Unix `top` utility showed IO wait percentages as high as 99%, which suggested that the performance of these queries were IO bound. Carefully optimizing queries by analyzing execution plans and optimizing database configurations to reduce IO contention are areas for substantial improvement, as no sophisticated database or tuning was performed during the development of the clickstream system.

### 5.1.5 Dynamic PL/SQL

The data population routines written for the data warehouse were discovered to be very slow. These routines were written in PL/SQL, a proprietary procedural language extension to SQL implemented by Oracle. PL/SQL was chosen as the language for implementing the data population routines since the code runs inside the database, giving PL/SQL code direct access to the database kernel. Thus the performance of PL/SQL is substantially greater than the performance of languages that have to go through an extra layer to access the database.

However, the original implementation was still very slow, and populated the data warehouse on the order of 10-20 rows per second. Profiling the PL/SQL population code revealed that the code spent the majority of its time executing redundant SQL queries. For instance, one particular routine would loop through each row in the raw clickstream data and execute a SQL query for each row to determine whether or not the given referrer already existed in the database. Since the SQL query was being executed against a table that did not change while the population code was being executed, the population code was updated to use dynamic PL/SQL. The dynamic PL/SQL code queries the referrer keys table at initialization and dynamically builds and executes another PL/SQL procedure that contains each key in the referrer keys table in a large IF/ELSE IF/ELSE block. Thus, the data population code no longer has to perform the SQL query for each row; it merely has to execute some compiled IF/ELSE statements to determine whether or not the referrer key actually exists in the database. Testing of this new code revealed an order of magnitude improvement in performance to 200-300 rows/second, since the context-switching overhead of moving between SQL and PL/SQL is eliminated.

### 5.1.6 Reducing the Data Warehouse Size

Queries against the data warehouse grew increasingly slower as more rows were loaded into the warehouse. This slowdown can be attributed to the lack of hardware and the difficulty in properly configuring Oracle. In order to compensate for the lack of

hardware, a more lightweight version of the clickstream software was developed. This version retains only a single day of clickstream data. When the population code is run, all the data in the data warehouse is deleted on a daily basis, and clickstream data for the most recent 24-hour period is loaded. This data are then analyzed; the results of each analysis is stored in an HTML file for the given day. The price paid for the scalability, reliability, and performance that this approach offers is that aggregation across multiple days becomes more complex, and new reports cannot be performed on previously collected data. In order to perform aggregation of data across multiple days to be performed, certain types of daily aggregates are stored in aggregation tables. The `cs_historical_by_day` table is an example of a daily aggregate table:

```
create table cs_historical_by_day (
    date_id                  integer
                             constraint cs_hbd_date_id_fk
                             references cs_dim_dates disable,
    -- count(*) from cs_fact_table
    n_page_views             integer not null,
    -- count(user_id) from cs_fact_table
    n_user_page_views        integer not null,
    -- count(session_id) from cs_fact_table
    n_session_page_views     integer not null,
    -- count(*) from cs_dim_sessions
    n_sessions               integer not null,
    -- count(user_id) from cs_dim_sessions
    n_user_sessions          integer not null,
    -- count(distinct user_id) from cs_dim_sessions
    n_unique_users           integer not null,
    -- avg(session_end - session_start)
    -- from cs_dim_sessions where clicks > 1
    avg_session_length       number,
    -- avg(clicks) from cs_dim_sessions where clicks > 1
```

```
    avg_session_clicks        number,
    aggregate_group           varchar(3000),
    constraint cs_date_agg_grp_un unique(date_id, aggregate_group)
);
```

This table stores daily aggregates of a number of different variables. Each row of the table corresponds to a single day, and records information such as the number of page views, the number of sessions, and the number of unique users that arrived at the web site on the given day.

## 5.2 Client-Side Logging

The client-side code evolved based on feedback from early adopters of the client-side code and early results from these deployments. The primary requirement for the client-side code was scalability and efficiency; the client-side code was written to be as lightweight as possible to insure that the clickstream logging code would be as scalable as the client web site itself.

The initial implementation of the clickstream logging code used array DML inserts. An array DML insert simultaneously inserts an entire array of values into a table instead of a single value. Thus, multiple rows can be inserted in one database request. The AOLserver to Oracle database driver was extended to support array DML inserts, and client-side code was written to use array DML inserts. In the clickstream client implementation, page request information was cached in memory and periodically flushed to the database via a single array DML operation. Array DML operations are generally not used in OLTP environments since a server restart would mean the loss of the data cached in memory. However, in a heavily loaded server, the loss of a few hundred data points of clickstream data was deemed a worthwhile tradeoff, since the lost data would be insignificant in the context of the total data gathered.

Testing of the array DML approach raised two issues. First, the array DML approach was not as scalable as hoped. An array DML insert was an order of magnitude more efficient than standard DML operations, allowing tens of inserts per second.

However, this approach quickly overwhelmed the database for high volume sites, since a database connection for clickstreaming still constrained other transactions on the web site, as each HTTP request incurred an additional amount of overhead that needed to be processed by the database driver. A second, equally important consideration was the realization that a database table was not the ideal medium to transfer data to a dedicated data warehouse machine, since the data is stored in a proprietary binary format.

These two reasons led to an iteration of the design that directly logged clickstream information into the file system. Each hour, all clickstream data is logged to a new clickstream log file. This approach overcomes the disadvantages of the array DML approach. First, no database handles are used, so the performance overhead is minimal. Second, by logging to the file system, the ASCII log files are easily transferred using HTTP or FTP. Once transferred to the data warehouse system, the log files can be parsed with a data bulk loader; most relational databases provide a utility for loading in large quantities of text data into the database. Using Oracle's SQL*LOADER software, the clickstream system was able to load tens of thousands of rows per second into the system, an improvement of three orders of magnitude over array DML.

A few weeks of testing on heavily loaded sites showed that the new approach was much more scalable than the previous approach. However, the rapid growth in size of the log files presented an additional problem, since the clickstream log files required a considerable amount of disk space. Thus, the client side code was improved to consolidate all the hourly logs into one daily log that was then compressed prior to download. This approach maintained the benefit of hourly log files (up-to-the-hour clickstream information could be provided by the hourly log files) while older log files could be archived in a compressed format, minimizing the disk space and bandwidth required to store and download the logs. Since the log files contain a large amount of redundant information in text format, files could be compressed to as little as 5% of their original size.

## 5.3 Reports

### 5.3.1 The Dynamic Site Map

The dynamic site map illustrates the most popular paths visitors follow when navigating the site. For each page in the site, the site map determines the most common referrer for each page. If this "best referrer" is external, the page shows up as a root-level node in the sitemap (since people typically get to the page from some external source); if the best referrer is internal, the page shows up beneath that referrer in the sitemap tree.

The `cs_create_sitemap` procedure builds this tree breadth-first: first it inserts into the sitemap all pages which belong in the top level (i.e., have an external "best referrer"). Then it inserts into the second level of the sitemap any pages that have a top-level node as a best referrer; then it inserts any pages which have a second-level node as best referrer, and so forth. The algorithm loops until it can no longer insert any more nodes.

When this process is complete, there may still be some pages which have not been inserted into the tree: consider the case where two pages have each other as best referrers. When a cycle such as this occurs, the node $X$ in the cycle which is most commonly requested is chosen, and the potential parents of all other nodes in the cycle are deleted. The loop then continues as previously described. At this point, $X$ is guaranteed to be inserted somewhere in the site map since none of the nodes in the cycle can possibly be $X$'s best referrer anymore. This process is repeated until all cycles are resolved. The end result is a dynamically built sitemap illustrating the paths visitors most frequently take navigating the site (see Figure 1-1).

### 5.3.2 The Entry Pages Report

The Entry Pages report that details how visitors typically enter a web site is a good example of a typical clickstream report. Retrieving this data requires a single query against the data warehouse:

```
select    p.page_id, p.page_url name, count(*) value
from      cs_fact_table f, cs_dim_pages p, cs_dim_sessions s
where     f.page_id = p.page_id
and       f.session_id = s.session_id
and       f.cs_within_session = 1
and       p.exclude_p = 'f'
and       s.clicks > 1
group by  p.page_id, p.page_url
order by  3 desc
```

This query performs a join between the main fact table and two dimension tables, the `cs_dim_pages` table and the `cs_dim_sessions` table. The first two lines of the where clause are simply the join. The remaining clauses are explained below:

- The `f.cs_within_session = 1` clause ensures that the page_id retrieved is the first page clicked in a given session.
- The `p.exclude_p = 'f'` clause excludes requests that should not be counted (e.g., requests for images).
- `s.clicks > 1` ensures that only sessions that have registered more than one page request are analyzed. One-hit sessions are thrown away because the system is unable to distinguish between first time visitors who hit the web site once and visitors who have non-cookied browsers. If one-hit sessions were included, the data would be skewed by non-cookied visitors who are browsing the site, since every page request from these visitors would look like an entry page.

## 5.4 Additional Features

### 5.4.1 Multiple Tablespaces

In order to increase the usability of the clickstream system, the clickstream server software was extended to take advantage of multiple tablespaces. The multiple tablespace

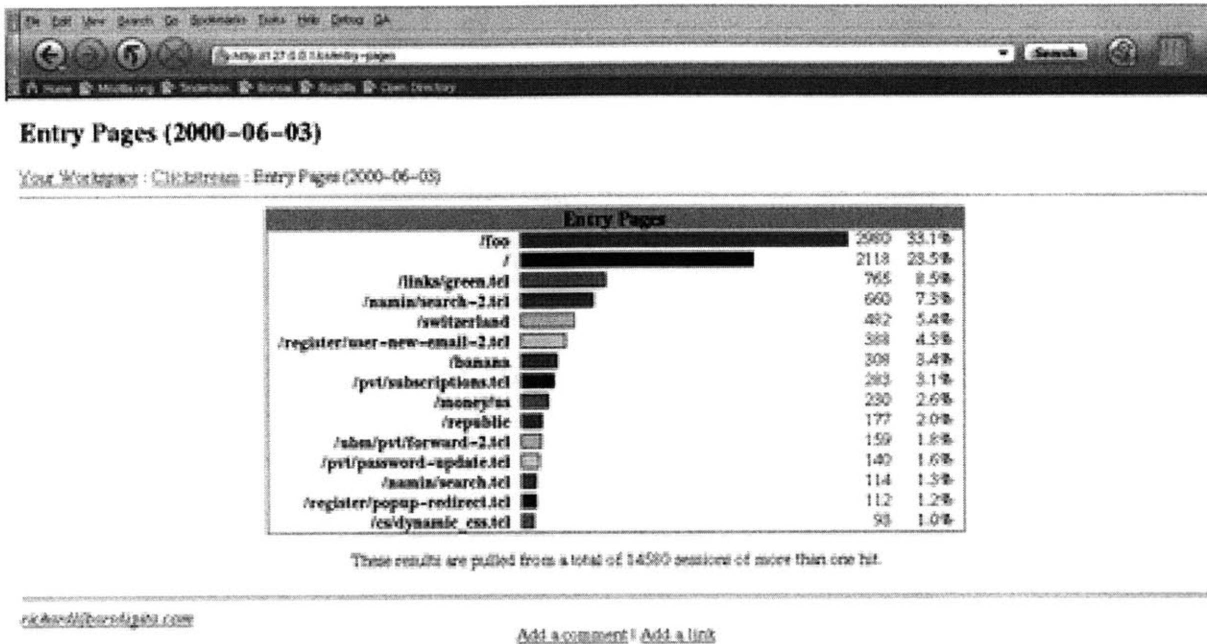



Figure 5-1: A Navigation Report: The Entry Pages Report

support that was implemented allowed a single clickstream instance to connect to the database as multiple users, with each user having its own tablespace. This approach allowed the clickstream data model to be loaded multiple times into separate tablespaces, with each tablespace representing the clickstream data of a separate web site. The motivation for this feature was to amortize the cost of providing a dedicated clickstream server across multiple web sites; multiple web sites could share the same clickstream system resources.

Three major issues arose after the implementation. First, the code was complex and difficult to maintain, since each page of the system had to be converted to determine the appropriate database pool to use and obtain handles. Second, testing

showed that there was no clear benefit by users of the multiple sites support, since users generally required a user interface that was customized to a particular site. These unique requirements included permissioning schemes that would prevent unauthorized users from seeing the data of other web sites and custom reporting queries. After this implementation, the scope of these additional features required to make the multiple sites feature usable became apparent, and the multiple clickstream application instance approach became clearly superior. Finally, the third issue that arose was that separating site data on the tablespace level was not the ideal mechanism; instead, the idea of using multiple schemas to manage multiple sites arose. This approach was deemed to be superior to the multiple tablespace implementation since tablespace management could be considerably simplified as the number of tablespaces could be reduced to a few common tablespaces that were shared by all instances of the clickstream application.

### 5.4.2 ACS 4.0

During the development cycle of the clickstream software, version 4.0 of the ArsDigita Community System (ACS) was released. Since the original software was built on top of the ACS, the clickstream software was upgraded to take advantage of 4.0 features. There are two benefits to a 4.0-compliant clickstream system. First, the clickstream system is available in one self-contained package. This self-contained package includes the clickstream data models, the population code, the client-side logging system, the server-side reports system, and all of the configuration parameters inside an XML file. A second benefit to the ACS 4.0 system is subsite support. Under the ACS 4.0 architecture, multiple instances of a package can be instantiated. This feature greatly simplifies the goal of supporting multiple clickstream sites on the same clickstream server.

# Chapter 6

# Testing and Results

In order to determine how well the completed system fulfilled the original design goals, a set of measures and criteria was established for each design goal. These criteria were used in evaluating the final clickstream system.

## 6.1 Privacy

### 6.1.1 Methodology

The clickstream logging system was used to measure visitor trust by comparing the aggregate behavior of visitors who read the privacy policy (and are presumably aware of clickstream logging) versus the behavior of visitors who did not read the policy. Since a web site visitor can only become aware of clickstream logging by reading the privacy statement, analyzing the clickstream logs allows visitors to be segmented into two user groups, visitors who read the privacy policy and visitors who did not. Discovering that visitors who read the privacy statement have shorter session durations would lead to the conclusion that the clickstream system may be strong negative influence on user trust.

### 6.1.2 ArfDigita

The complete clickstream system was installed on the ArfDigita web site (http://www.arfdigita.org). ArfDigita is a non-profit site designed to match animals in animal shelters with people who want to have pets. The ArfDigita site was deemed to be ideally suited to the role of initial clickstream deployment because its low level of traffic would enable expedient analysis of the data. The following privacy policy was published on ArfDigita:

```
We realize that much of the information you enter into the site is of
a personal nature and that privacy is a big concern. We have created
this privacy policy to assure you that ArfDigita will make its best
effort to ensure that your information is kept safe and confidential.

ArfDigita will not distribute or sell any personal information to
outside parties. Statistical and demographic information may be
reported, but we will not report any personally identifiable
information.

ArfDigita collects data about the clickstream behavior of its
users. Collecting clickstream data entails recording which links are
clicked by which user, as well as the interval between clicks; this
enables us to measure data such as which pages on the web site take
the longest to load, what are the most popular pages on the web site,
and so forth. The aggregate clickstream data is used to determine how
the site is being used in order for us to better tailor the site to
serve you, and is not used for any other purpose. Data on the
clickstream behavior of a single user is never used and is permanently
deleted once the data has been aggregated, a process that occurs every
24 hours. This clickstream data will not be distributed to any outside
parties.
```

```
All shelters using this system have agreed to a Shelters' Privacy
Policy whereby any information you enter is used only for their
records and will not be distributed to any outside parties. Shelters
are required to agree to the Shelters' Privacy Policy before they are
allowed to access any data.  However, we cannot guarantee the
cooperation of the shelters with this policy. If you think that a
shelter has acted in violation of this policy, please email
webmaster@arfdigita.org.

If you have any questions about our privacy policy, please contact
webmaster@arfdigita.org.
```

### 6.1.3 Results

Analysis of the ArfDigita data was inconclusive. Nearly 700,000 clickstream events collected over the course of three months were analyzed, representing the clickstream behavior of over 600 visitors. Less than twenty of those visitors had read the privacy policy. There was no discernible difference between the two sets of visitors in the average session duration.

Analyzing the ArfDigita clickstream data did reveal two factors that should be taken into account in future clickstream privacy analyses. First, the privacy policy should be prominent. The privacy policy was two clicks away from the top-level index page. Future tests that analyze the effect of clickstream logging on visitor behavior should have a more prominent privacy policy that is linked off the top-level index page to increase the probability that a visitor will click to the privacy policy. Second, the fraction of visitors who actually read privacy policies is very low. Thus, future analyses should use much larger data sets over longer period of times to produce more concrete results.

## 6.2 Performance and Scalability

The performance and scalability of the clickstream system were assessed separately. These measures are based on the definitions discussed in Section 4.2. The performance analysis determined the impact of the clickstream logging filter on web server performance. In addition, the performance analysis measured the number of rows/second that the server-side population code can process. The scalability analysis analyzed the upper limits of the scalability of a web server architecture with clickstream logging enabled and the maximum amount of data that the clickstream system can process in a given timeframe.

### 6.2.1 Results

The performance of the clickstream logging code was measured on a web site with multiple web servers. The clickstream filter was applied to one web server in the web server pool; the load on the web server was compared to that of the other web servers. The web server showed no discernible increase in load when compared to the other web servers.

The performance of the clickstream server population code was measured in rows/second on a Sun E450 server. Benchmarks on the population code revealed a speed ranging from 200 to 300 rows per second, or about one million rows per hour.

The standard architecture for a high-volume web site uses multiple front-end web servers that communicate to a smaller set of back-end database servers. Since the clickstream client-code runs only on the web server, the clickstream code is as scalable as the web server architecture itself. As traffic increases, new web servers are added; the negligible performance impact of the clickstream client-side logging code is evenly distributed across the additional web servers. Thus, the clickstream client-side logging code is highly scalable.

The current server-side clickstream architecture is not as scalable as the client-side logging system, and relies more upon its high-performance implementation optimizations than fundamental scalability work. Scaling the database across multiple servers

is difficult, and the technology to distribute the database is immature. The complexity of products such as Oracle Parallel Server, which allows a database to be distributed across multiple machines, led to the decision not to explore approaches to scale the clickstream system to multiple database servers.

## 6.3 Transparency

Developers deploying new applications or features onto a web site with clickstream logging do not need to perform any additional development work. In this sense, the system has completely achieved its goal of transparency for developers.

There are two ways that the clickstream system may be visible to visitors. First, the clickstream system may impact performance on a web site. A second way that visitors may notice the clickstream system is by reading the privacy policy and deciding that the system is too invasive.

The performance data discussed above indicates that the clickstream logging system does not adversely affect web site performance. In addition, a cursory examination of the download times for various pages of a system with clickstream logging reveals that the download times for the same pages with and without logging are comparable.

The issue of privacy and its effect on user behavior is more of an unknown. As discussed above, the data on how visitors view a web site that performs clickstream logging is inconclusive. More testing and analysis is needed to determine how well a clickstream system can satisfy user expectations of trust.

## 6.4 Platform Independence

A development snapshot of the server-side data warehouse and population facility was successfully ported to PostgreSQL, an open-source RDBMS. This effort took a single developer one month of part-time work to port. Most of the time was spent on converting Oracle-specific features to PostgreSQL functions, since the PostgreSQL database

offers a subset of the functionality provided by Oracle. The developer was unfamiliar with PL/SQL, and was only partially familiar with PL/PgSQL and PL/TCL (the target languages for the porting). There were no issues moving the data model between databases; all of the porting effort was spent in reconciling procedural language syntax differences. Virtually all modern relational databases provide facilities for stored procedures using a language similar to PL/SQL, so any porting effort between databases of the clickstream server software would require a similar process.

The porting effort was simplified by the fact that the web server was kept the same for the port to Postgres. Thus the server-side user and reports interface required minimal porting, an effort that only involved updating the SQL syntax of some of the user interface queries. The server-side user interface is a web-based user interface that does not use any advanced or esoteric features of Oracle or AOLserver, so porting the user interface to another web server, application server, or database platform should be a straightforward exercise for a programmer versed in the target platform technology.

The final component of the clickstream system is the client-side logging code. The code written in this project is web-server dependent. In particular, the client-side code is written in Tcl since AOLserver contains an embedded Tcl interpreter; the code also takes advantage of specific AOLserver API calls that are available to the programmer. Despite this web server dependence, the client-side logging code is very portable since it is very lightweight, consisting of less than four hundred lines of Tcl code. In addition, the API calls used are very standard across web servers. An analysis of the Netscape iPlanet web server API reveals that all of the AOLserver calls used in the clickstream client code have equivalents in the iPlanet server. One further point to note about the portability of the clickstream client code is that all web servers have logging capabilities, and the clickstream logging code is merely an extension of the existing logging functionality of any web server.

## 6.5 Extensibility

The clickstream system was deployed on three web sites, including a commercial high-volume web server environment. In the production environment, the basic reporting user interfaces and data population infrastructure was initially used. After a period of several months, the commercial web site decided to reduce the scope of the process for faster turnaround and used the clickstream logging and loading facilities only. A single developer was able to write simple code to replace the standard population code to fit their needs. Thus, two conclusions can be drawn from this case study. First, the componentized nature of the system allowed it to be easily adapted to fit the specialized needs of the client. Second, the version of data population code and hardware that the client site used was inadequate to handle the load of the web site.

## 6.6 Accuracy

The accuracy of the clickstream system is an implicit goal of the system. Knowledge that the clickstream system provides accurate data is a prerequisite for deployment of the clickstream system. In order to verify the accuracy of the clickstream system, clickstream data for a commercial high-traffic web site was compared to the data for a commercial log analyzer, NetTracker. NetTracker is a commonly used, well-regarded commercial log analyzer that parses standard web server logs [14].

Figure 6-1 shows the number of sessions as recorded by NetTracker and the clickstream system over a week. The top line shows the number of sessions as counted by the clickstream system; the bottom line shows the number of sessions according to NetTracker of the same time period. The data shows that there is 100% correlation between the NetTracker and clickstream data, and the numbers differ only by magnitude. The NetTracker data consistently shows a smaller number of sessions than the clickstream data. This phenomenon can be attributed to the fact that the clickstream system is more accurate and uses cookies for tracking. The NetTracker system is based on intelligent analysis of IP addresses e.g., all page requests from a

single IP address in a given time frame is considered "one session." However, this approach cannot account for traffic from proxy servers, where many web site visitors may appear to be coming from a single server. This case is particularly common for America Online (AOL) users, since AOL users form a large proportion of Internet users. The other reason for the discrepancy in numbers may be differences in the definition of a session. As discussed in Section 3.4, the clickstream system has a timeout interval of five minutes; the NetTracker documentation does not discuss what value it uses for timeouts, nor does it discuss its definition of a session.

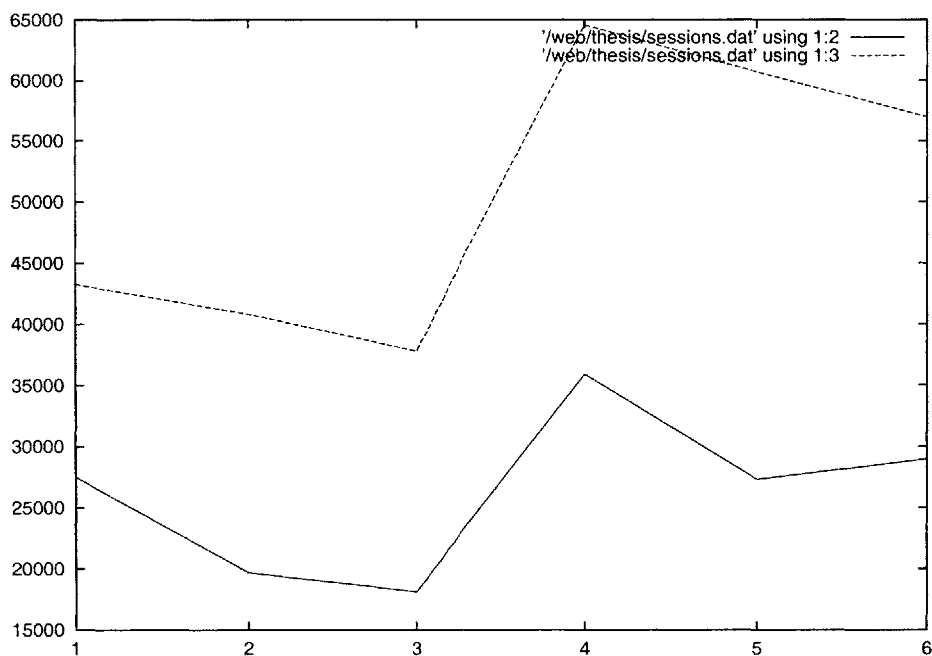



Figure 6-1: Total Daily Sessions, NetTracker vs. Clickstream

A similar correlation exists with the total daily pageviews reported by NetTracker and the clickstream system, as shown in Figure 6-2. The clickstream system reports a higher number of pages because the NetTracker report has been configured to exclude certain sections of the web site from the total pageview count; no page views are being excluded from the clickstream count.

The simplicity of the clickstream architecture allows a high degree of confidence in the accuracy of the overall system. A page request is translated into a line in

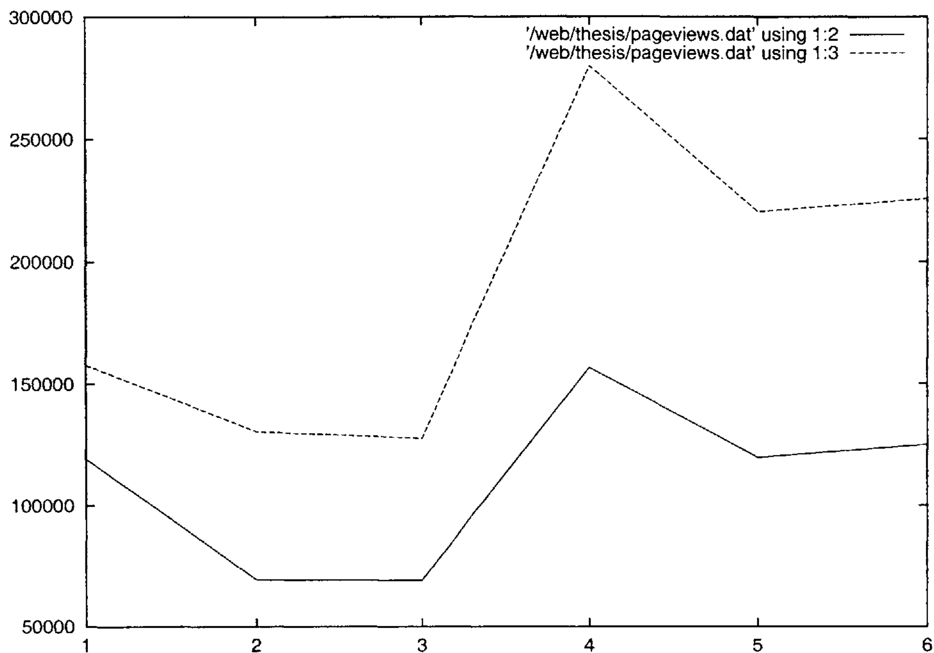



Figure 6-2: Total Daily Pageviews, NetTracker vs. Clickstream

the clickstream log; each line is loaded into the database. The modular nature of the clickstream architecture allows very concrete unit tests to be performed. For example, verifying that a given page request creates a line in the clickstream log is a simple matter. The correlation data presented above provides additional evidence that the clickstream system is accurate. This data, in conjunction with the unit tests performed, allows clickstream users to have a high degree of confidence in the correctness of the fundamental clickstream system.

# Chapter 7

# Conclusion

Building a data warehouse for a clickstream system requires a substantial investment in hardware, configuration, maintenance, and analysis. The clickstream system presented in this paper, while capable of revealing previously unknown data about how a web site is being used, is only a starting point for additional clickstream analysis and development.

The basic goal established in the original design of the clickstream system was to build a simple architecture that could be used as the basis for a clickstream system. The architecture and software presented here fulfills that goal. In addition, the system has been tested under a high-load environment to measure scalability. Data from the system has been analyzed to determine visitor behavioral patterns. The clickstream design is platform independent; as discussed above, an independent developer was able to port the data warehouse server software, the most complex component of the clickstream system, to an alternative database platform with minimal difficulty. Finally, components of the clickstream system have been customized by other developers in fulfilling specific needs.

## 7.1 Future Work

The lessons learned during development of the clickstream design and implementation make many of the limitations of the system apparent.

### 7.1.1 Performance and Scalability

Although a considerable amount of effort was made to optimize the performance and scalability of the server-side data population code, a considerable amount of profiling and optimization is still necessary before the system can scale linearly to handle arbitrary amounts of data. Areas that could be investigated include the use of distributing the database across multiple servers, additional performance optimizations, and the possibility of performing additional preprocessing in a high-performance compiled language prior to inserting the data into the database.

### 7.1.2 Multiple Data Source Integration

Integrating the clickstream data warehouse with other data warehouses can multiply the analytic power of the clickstream data warehouse. For instance, integrating a clickstream data warehouse for an ecommerce site with a data warehouse containing all sales data for the entire company can allow the company to analyze its clickstream data in even more novel ways. For instance, a clickstream data warehouse can answer questions such as "how much money did the average visitor who fits a particular user group spend on my web site?" However, an integrated system consisting of both a clickstream data warehouse and a sales data warehouse can further answer questions such as "what is the average profit margin for sales to a particular user group who buys things on my web site?" The ability of this clickstream approach to support the integration of website and other relational data with the clickstream data warehouse is an important difference between this system and commercial clickstream and web-log analyzers that exclusively rely on externally-visible characteristics (e.g., URL requested or page load time).

### 7.1.3 Portability

The advent of Java servlet technology presents an opportunity to increase the platform independence of the clickstream system. For instance, the client-side clickstream logging code could be written as a Java servlet that performs logging. Thus, the

clickstream client servlet would then be immediately portable across any web server that implements the Servlet specification.

The server side code could also benefit from a Java implementation, since a Java servlet implementation would also be portable across web servers. Programming the server-side code in Java would allow for a portable, web-based user interface to the data warehouse to be available to the user. However, a Java implementation of the server-side code would be limited to the user interface because the server side population code should be written in a procedural language that runs inside the database for maximum performance.

### 7.1.4 Evolutionary Improvements

In addition to the architectural improvements enumerated above, many evolutionary improvements could be undertaken to further improve the functionality of the clickstream system.

- Refining and developing a comprehensive set of queries that answer most of the questions that users want.
- Improving the reports user interface and infrastructure.
- Fully supporting the ACS 4.0 subsite architecture.
- Extending the data population code to include all dimensions in the data model and determining what additional dimensions should be added to the core system.
- Defining a standard way to extend the data model, and developing a mechanism to transfer the additional data necessary to populate these extra fields from the production web server database to the data warehouse.

## 7.2 Lessons Learned

Data warehousing clickstream data presents a unique challenge. A staggering amount of data is accumulated in a very short period of time. Each data point represents a very small event, so large numbers of events must be analyzed and mined to discern patterns.

Three major lessons can be learned from the development of the clickstream system. First, implementing a system to analyze clickstream data is hard. Despite the overt simplicity of the design and concept of clickstream analysis, the actual implementation of scalable, usable clickstream analysis software has been difficult. Second, server-side scalability and performance is a very challenging problem. Building a scalable server-side clickstream architecture requires deep understanding of a particular RDBMS platform, a large amount of hardware, and extensive load testing. Finally, clickstream data can be easily analyzed to produce logfile analyzer-type reports, but deeper, more insightful patterns are much more difficult to discern. The amount of development work required to design and test the dynamic site map algorithm was equivalent to the amount of effort required to implement all of the logfile analyzer-type reports. Developing the necessary algorithms and techniques for analyzing deep patterns in clickstream data may become the most challenging task for future users of clickstream analysis systems.

The clickstream system is a usable framework for future clickstream development. Developers, analysts, and others who wish to use clickstream analysis systems can build and extend this framework to meet their needs. This architecture has been refined through real-world deployments and user feedback; people should leverage the experience gained in developing this system by building and extending this framework instead of starting from scratch. This clickstream system is a beginning, and not an end.

# Appendix A

# Clickstream Data Model

This appendix contains the core SQL data models used in the clickstream system.

```
-- clickstream data model: HTTP-level granularity (although a more
-- normal and scalable configuration is a page-request level
-- granularity)
--
-- richardl@mit.edu
--
-- $Id: click-dw.tex,v 1.4 2000/12/17 21:28:50 richardl Exp $

set scan off

-- page dimension

create sequence cs_page_id_sequence;

-- a mapping table between short names of a page function and some
-- pretty names. for instance, we might say that a page has a function
-- of "admin", which translates into a "site-wide administration page"
```

```
create table cs_page_functions (
    page_function          varchar(20)
                           constraint cs_pf_page_function_pk primary key,
    pretty_name            varchar(100),
    exclude                char(1) default 'f'
                           constraint cs_pf_exclude_p_nn not null disable
                           constraint cs_pf_exclude_p_ck
                           check(exclude_p in ('t','f')) disable
);

-- mappings from a particular URL pattern to a page function.
create table cs_page_function_keys (
    try_order          integer
                       constraint cs_pk_try_order_pk primary key,
     -- must contain associated '%'s, if any
    key_string         varchar(500)
                       constraint cs_pk_key_string_nn not null disable,
    page_function      constraint cs_pk_page_function_fk
                       references cs_page_functions disable
                       constraint cs_pk_page_function_nn not null disable
);

-- we store clickthroughs here as well, with the /ct prefix
-- and a page_function of type ct
create table cs_dim_pages (
    page_id            integer
                       constraint cs_dp_page_id_pk primary key,
    page_title         varchar(3000),
    -- url (excludes query parameters!)
    page_url           varchar(3000)
```

```
                    constraint cs_dp_page_url_nn not null disable,
    url_stub        varchar(3000),
    instance_var    varchar(3000),
    -- kind of page, e.g., product information, contact form,
    -- about company, etc.
    page_function    constraint cs_dp_page_function_fk
                     references cs_page_functions disable,
    -- expected content type for this page
    content_type     varchar(100),
    -- exclude this page?
    exclude_p        char(1) default 'f'
                     constraint cs_dp_exclude_p_nn not null disable
                     constraint cs_dp_exclude_p_ck
                     check(exclude_p in ('t','f')) disable,
    -- what file serves this up? (null if none)
    local_file       varchar(3000)
);

create unique index cs_dim_pages_by_url on cs_dim_pages(page_url);
create index cs_dim_pages_by_url_stub on cs_dim_pages(url_stub);

-- referrer dimension

create table cs_local_host_names (
    host_name       varchar(100) primary key
);

create sequence cs_referrer_id_sequence;

create table cs_referrer_types (
```

```
    referrer_type          varchar(20)
                           constraint cs_rt_referrer_type_pk primary key,
    pretty_name            varchar(100)
);

create table cs_referrer_keys (
    try_order       integer primary key,
    kind            varchar(20)
                    constraint cs_rk_kind_ck
                    check(kind in ('host','domain','url')) disable
                    constraint cs_rk_kind_nn not null disable,
    -- must contain associated '%'s, if any
    key_string      varchar(500)
                    constraint cs_rk_key_string_nn not null disable,
    referrer_type   constraint cs_rk_referrer_type_fk
                    references cs_referrer_types disable
                    constraint cs_rk_referrer_type_nn not null disable
);

-- referring_page_id is null for external referrers

create table cs_dim_referrers (
    referrer_id            integer
                           constraint cs_dr_referrer_id_pk primary key,
    -- e.g., search engine, intra-site, remote-site
    referrer_type          constraint cs_dr_referrer_type_fk
                           references cs_referrer_types disable,
    referring_url          varchar(3000)
                           constraint cs_dr_referring_url_nn not null disable,
    -- local path (if not external)
```

```
    referring_local_path   varchar(3000),
    referring_site         varchar(500),
    referring_domain       varchar(500),
    -- if the person came from a search engine
    search_text            varchar(3000),
    -- if the person came from within the site
    referring_page_id      constraint cs_dr_referring_page_id_fk
                           references cs_dim_pages disable,
    -- same as referrer_id, *only* if the referral
    -- is external. this is so we can group by referring_page_id,
    -- external_referrer_id for sitemap purposes
    external_referrer_id   integer
);

create index cs_dim_referrers_by_page_id on
    cs_dim_referrers(referring_page_id, external_referrer_id);
create unique index cs_dim_referrers_by_url on
    cs_dim_referrers(referring_url);

-- calendar date dimension
create table cs_dim_dates (
    -- date_id of the form 20001231 (for 2000-12-31)
    date_id              integer
                         constraint cs_dd_date_id_pk primary key,
    sql_date             date
                         constraint cs_dd_sql_date_un unique disable
                         constraint cs_dd_sql_date_nn not null disable,
    day_of_week          integer not null, -- between 1 and 7
    day_number_of_month  integer not null, -- between 1 and 31
    day_number_in_year   integer not null, -- between 1 and 366
```

```
    week_number_in_year  integer not null, -- between 1 and 53
    month                integer not null, -- between 1 and 12
    quarter              integer not null, -- between 1 and 4
    year                 integer not null, -- use 4 digit years
    holiday_p            char(1) default 'f'
                         constraint cs_dd_holiday_p_ck
                         check (holiday_p in ('t', 'f')) disable,
    -- as opposed to weekend
    weekday_p            char(1) default 'f'
                         constraint cs_dd_weekday_p_ck
                         check (weekday_p in ('t', 'f')) disable
);

-- the user agent dimension
create sequence cs_user_agent_id_sequence;

create table cs_dim_user_agents (
    user_agent_id            integer
                             constraint cs_dua_user_agent_id_pk primary key,
    -- The string returned by the browser
    user_agent_string        varchar(3000) not null,
    -- Mozilla, Opera, IE
    browser_type             varchar(3000),
    browser_version          varchar(3000),
    -- The major part of the browser version. For MSIE
    -- and Mozilla, this is just the first three characters.
    browser_version_major    varchar(3000),
    -- MacOS, Win32, Unix
    operating_system         varchar(50),
    -- MacOS 8, Windows 95, Windows 98, Windows NT, Linux, Solaris, etc.
```

```
    operating_system_variant  varchar(50)
);

create unique index cs_dim_user_agents_by_string on
    cs_dim_user_agents(user_agent_string);

create table cs_operating_system_keys (
    try_order                integer
                             constraint cs_dos_try_order_pk primary key,
    key_string               varchar(100)
                             constraint cs_dos_key_string_nn not null disable,
    operating_system         varchar(100)
                             constraint cs_dos_operation_system_nn not null
                             disable,
    operating_system_variant varchar(100)
);

create sequence cs_event_id_seq cache 1000;

-- the event log
create table cs_event_log (
    event_id             integer
                         constraint cs_el_event_id_pk primary key,
    event_time           integer
                         constraint cs_et_event_time_nn not null disable,
    end_time             integer
                         constraint cs_et_end_time_nn not null disable,
    date_id              integer
                         constraint cs_et_date_id_nn not null disable,
    url                  varchar(4000)
```

```
                            constraint cs_et_url_nn not null disable,
    instance_id             varchar(4000),
    user_ip                 varchar(50)
                            constraint cs_et_user_ip_nn not null disable,
    user_id                 integer,
    query                   varchar(4000),
    bytes                   integer,
    content_type            varchar(200),
    session_id              integer,
    browser_id              integer,
    user_agent_string       varchar(4000),
    accept_language         varchar(10),
    referring_url           varchar(4000),
    method                  varchar(10)
                            constraint cs_et_method_nn not null disable,
    status                  integer
                            constraint cs_et_status_nn not null disable,
    secure_p                char(1)
                            constraint cs_et_secure_p_nn not null disable
                            constraint cs_et_secure_p_ck
                            check(secure_p in ('t','f'))
) nologging storage (
  initial 50m
  next 50m
  pctincrease 0);

-- user information

-- note that this table is not normalized because we want to prevent
-- snowflaking, and there can be more than one user_state per user.
```

```
create table cs_dim_users (
    user_id               integer,
    birthdate             date,
    sex                   char(1)
                          constraint cs_du_sex_ck
                          check (sex in ('m','f')) disable,
    postal_code           varchar(80),
    ha_country_code       char(2),
    affiliation           varchar(40),
    race                  varchar(100),
    income_level_lower    integer,
    income_level_upper    integer,
    -- these last two have to do with how the person
    -- became a member of the community
    how_acquired          varchar(40),
    -- will be non-NULL if they were referred by another user
    referred_by           integer,
    -- e.g., "seen privacy policy"
    user_state            varchar(3000),
                          constraint cs_du_user_id_state_un
                          unique(user_id, user_state)
);

create sequence cs_session_id_sequence increment by 1000;

create table cs_dim_sessions (
    session_id        integer
                      constraint cs_ds_session_id_pk primary key,
    session_start     integer,
    session_end       integer,
```

```
    date_id             integer,
    referrer_id         constraint cs_ds_referrer_id_fk
                        references cs_dim_referrers disable,
    user_id             integer,
    user_agent_id       constraint cs_ds_user_agent_id_fk
                        references cs_dim_user_agents disable,
    browser_id          integer,
    clicks              integer not null,
    last_fact           integer,
    -- e.g., repeat visitor, search engine visitor, complainer...?
    user_type           varchar(20)
);

-- old sessions lasting for a day or two that we keep around
-- to prevent double counts on day boundaries (e.g., someone logged
-- in from 11:30pm to 1am should be counted once, not twice).

create table cs_old_sessions (
        date_id                 integer,
        session_id              integer
);

-- the cs fact table, processed from the cs_event_log table.

create sequence cs_id_sequence;

create table cs_fact_table (
    cs_id                   integer primary key,
    -- load start/end, dwell time if able to determine
    load_start              integer,
```

```
load_time              integer,
dwell_time             integer,
page_id                integer constraint cs_ft_page_id_fk
                       references cs_dim_pages not null disable,
-- which instance of the page is it?
instance_id            varchar(3000),
instance_provided_p    char(1) default 'f' not null
                       check(instance_provided_p in ('t','f')) disable,
referrer_id            integer constraint cs_ft_referrer_id_fk
                       references cs_dim_referrers disable,
date_id                integer
                       constraint cs_ft_date_id_nn not null disable
                       constraint cs_ft_date_id_fk
                       references cs_dim_dates disable,
session_id             integer
                       constraint cs_ft_session_id_fk
                       references cs_dim_sessions disable,
user_agent_id          integer
                       constraint cs_ft_user_agent_id_fk
                       references cs_dim_user_agents disable,
user_ip                varchar(50)
                       constraint cs_ft_user_ip_nn not null disable,
browser_id             integer,
user_id                integer,
-- HTTP method (GET, POST, HEAD)
method                 varchar(10)
                       constraint cs_ft_method_nn not null disable,
-- HTTP status code
status                 integer
                       constraint cs_ft_status_nn not null disable,
```

```
    bytes                   integer,
    content_type            varchar(200),
    -- might want to turn accept_language into a dimension?
    accept_language         varchar(10),
    secure_p                char(1)
                            constraint cs_ft_secure_p_nn not null disable,
                            constraint cs_ft_secure_p_ck
                            check(secure_p in ('t','f')) disable,
    -- how many css into the session are we? 1, 2, 3...
    cs_within_session       integer
) nologging storage (
  initial 50m
  next 50m
  pctincrease 0);

create index cs_fact_table_by_session_id on cs_fact_table(session_id);

-- a bitmap star join, so we create bitmap indices on
-- columns with low cardinality
create bitmap index cft_page_id_idx on cs_fact_table(page_id);
create bitmap index cft_date_id_idx on cs_fact_table(date_id);
create bitmap index cft_user_agent_id_idx on cs_fact_table(user_agent_id);
create bitmap index cft_method_id_idx on cs_fact_table(method);

-- this table aggregates by page/instance/referrer pair
create table cs_fact_table_aggregate (
    cs_aggregate_id          integer primary key,
    -- we just sum the number of records that have the
    -- same page/referrer pair when we do the aggregation
    weight                   integer,
```

```
    -- avg load_start - load_time difference
    load_duration           integer,
    -- avg dwell time
    dwell_time              integer,
    page_id                 integer not null,
    -- which instance of the page is it?
    instance_id             varchar(3000),
    referrer_id             references cs_dim_referrers,
    -- avg bytes
    bytes                   integer,
    content_type            varchar(200),
    -- might want to turn accept_language into a dimension?
    accept_language         varchar(10),
    secure_p                char(1) not null check(secure_p in ('t','f')),
    -- how many css into the session are we? 1, 2, 3...
    -- this will be an average too
    cs_within_session       integer
);

-- for handling errors
create sequence cs_error_id_sequence;
create sequence cs_job_id_sequence;

-- for logging when click_dw_populate runs
create global temporary table cs_active_job (
    job_id          integer
) on commit preserve rows;

create table cs_jobs (
    job_id          integer
```

```
                constraint cs_jobs_job_id_pk primary key,
    start_stamp   date
                constraint cs_jobs_start_stamp_nn not null,
    end_stamp     date,
    event_count   integer
);

create table cs_errors (
    error_id      integer
                  constraint cs_errors_error_id_pk primary key,
    event_id      integer,
    job_id        integer
                  constraint cs_errors_job_id_fk references cs_jobs
                  constraint cs_errors_job_id_nn not null,
    error_number  number,
    error_message varchar(3000),
    location      number,
    timestamp     date
);

create table cs_sitemap (
    page_id         constraint cs_sitemap_page_id_fk
                    references cs_dim_pages disable
                    constraint cs_sitemap_page_id_pk primary key,
    parent_page_id  constraint cs_sitemap_parent_page_id_fk
                    references cs_dim_pages disable,
    loop_top_p      char(1) default 'f' not null
                    check(loop_top_p in ('t','f')),
    hits            integer,
    tree_level      integer,
```

```
    tree_order      varchar(400)
);

create index cs_sitemap_by_level on cs_sitemap(tree_level);

create sequence cs_processing_log_id_sequence;
create table cs_processing_log (
        log_id          integer primary key,
        job_id          constraint cs_pl_job_id_nn not null
                        constraint cs_pl_job_id_fk
                        references cs_jobs on delete cascade,
        log_msg         varchar(4000),
        stamp           date
);

create table cs_event_log_loads (
    chunk_start_time   integer
                       constraint cs_event_log_load_date_pk primary key,
    processed_p        char(1) default 'f'
                       constraint cs_event_log_processed_p_nn not null
                       constraint cs_event_log_processed_p_ck
                       check(processed_p in ('t','f'))
);

create table cs_best_referrers (
        page_id          integer,
        parent_page_id   integer,
        hits             integer
);
```

```
-- helper table
create table cs_historical_visits_temp (
        date_id                 integer,
        n_sessions_day          integer,
        member_p                integer,
        n_users                 integer
);

-- maps user_states to URL stubs
create sequence cs_url_user_states_seq;

create table cs_url_user_states (
       url_user_state_id    integer primary key,
       url                  varchar(3000) not null,
       user_state           varchar(3000) not null,
       -- is this a URL stub?
       stub_p               char(1) constraint cs_url_us_stub_ck
                            check(stub_p in ('t','f')) disable
);

-- clickstream historical data model

-- since we are throwing away data, we want to be able to aggregate
-- data in different ways. the approach we take here with the
-- aggregate_group column is conceptually similar to kimball's "level"
-- approach. each aggregate_group represents a particular aggregate
-- group e.g., people who have read the privacy policy v people who
-- have not read the privacy policy.

create table cs_historical_by_day (
```

```
    date_id                     integer
                                constraint cs_hbd_date_id_fk
                                references cs_dim_dates disable,
    -- count(*) from cs_fact_table
    n_page_views                integer not null,
    -- count(user_id) from cs_fact_table
    n_user_page_views           integer not null,
    -- count(session_id) from cs_fact_table
    n_session_page_views        integer not null,
    -- count(*) from cs_dim_sessions
    n_sessions                  integer not null,
    -- count(user_id) from cs_dim_sessions
    n_user_sessions             integer not null,
    -- count(distinct user_id) from cs_dim_sessions
    n_unique_users              integer not null,
    -- avg(session_end - session_start) from cs_dim_sessions where clicks > 1
    avg_session_length          number,
    -- avg(clicks) from cs_dim_sessions where clicks > 1
    avg_session_clicks          number,
    aggregate_group             varchar(3000),
    constraint cs_date_agg_grp_un unique(date_id, aggregate_group)
);

create table cs_historical_by_hour (
    date_id         constraint cs_hbh_date_id references cs_dim_dates disable,
    hour            integer
                    constraint cs_hbh_hour_ck check (hour >= 0 and hour < 24),
    n_page_views    integer not null,
                    constraint cs_historical_by_hour_pk primary key(date_id, hour)
);
```

```
create table cs_historical_visits (
    date_id                  integer
                             constraint cs_hv_date_id_fk
                             references cs_dim_dates disable
                             constraint cs_hv_date_id_nn not null,
    browser_id               integer,
    user_id                  integer,
    n_sessions_day           integer,
    unique(date_id, browser_id, user_id)
);

-- These are views are to be used in /admin/cs/vists-report-cumulative.tcl

-- Counts the number of members and non-members per number of sessions
-- in a day e.g., 500 members who had 50 sessions on this particular
-- day. count(browser_id) counts the number of browsers that have
-- visited the web site in this time period; we consider this number
-- to be the number of visitors to the web site. We do not do a count
-- distinct because we want to consider the case where there are
-- multiple users using the same browser. However, we are missing the
-- Nada case, where the same user may use different browsers to log on
-- to the system.

create or replace view cs_historical_visits_grouped
as
select date_id, n_sessions_day, decode(user_id,null,0,1) member_p,
        count(browser_id) n_users
            from cs_historical_visits
            group by date_id, n_sessions_day, decode(user_id,null,0,1);
```

```
create or replace view cs_n_sessions_day_user
as
select b.date_id, b.n_sessions_day, nvl(a.n_users, 0) as members,
            b.n_users as non_members
            from cs_historical_visits_grouped a, cs_historical_visits_grouped b
            where b.n_sessions_day = a.n_sessions_day(+)
              and b.date_id = a.date_id(+)
              and 1 = a.member_p(+)
              and b.member_p = 0
UNION
select a.date_id, a.n_sessions_day, a.n_users as members,
            nvl(b.n_users, 0) as non_members
            from cs_historical_visits_grouped a, cs_historical_visits_grouped b
            where a.n_sessions_day = b.n_sessions_day(+)
              and a.date_id = b.date_id(+)
              and a.member_p = 1
              and 0 = b.member_p(+);

create table cs_historical_page_views (
    date_id             constraint cs_hpv_date_id
                        references cs_dim_dates disable,
    page_id             constraint cs_hpvbd_page_id_fk
                        references cs_dim_pages disable
                        constraint cs_hpvbd_page_id_nn not null disable,
    n_page_views        integer not null,
    constraint cs_historical_page_views_pk primary key(date_id, page_id)
);
```

```
-- a list of available reports

create table cs_daily_reports (
    report_name     varchar(30) not null primary key,
    report_title    varchar(200) not null,
    description     varchar(4000),
    sort_key        integer
);
```

# Bibliography

[1] Ralph Kimball and R. Merz. *The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse.* John Wiley & Sons, 2000.

[2] ArsDigita, Inc., http://www.arsdigita.com/doc/. *ArsDigita Community System Documentation.*

[3] AOL, Inc., http://www.aolserver.com. *AOLserver 3.x Documentation.*

[4] Oracle, Inc., http://oradoc.photo.net. *Oracle8i Release 2 Documentation Library.*

[5] Ralph Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses.* John Wiley & Sons, 1996.

[6] Ralph Kimball et al. *The Data Warehouse Lifecycle Toolkit.* John Wiley & Sons, 1998.

[7] S. Rupley. Web profiling. *PC Magazine*, 23 November 1999.

[8] Accrue Software, Inc., http://www.accrue.com/pdf/webminingwhitepaper_0300.pdf. *Web Mining Whitepaper*, 2000.

[9] Oracle, Inc., http://technet.oracle.com/products/clickstream/pdf/click_ds.pdf. *Oracle Clickstream Intelligence 1.0*, October 2000.

[10] M. Hammond. Microsoft joins 'webhousing' crowd. *PC Week*, 15 October 1999.

[11] Richard Winter. More than you hoped for. *Scalable Systems*, 3(6), 10 April 2000.

[12] Gavin Skok. Establishing a legitimate expectation of privacy in clickstream data. *Michigan Telecommunications Technology Law Review*, 61, 22 May 2000. http://www.mttlr.org/volsix/skok.html.

[13] R. Armstrong. Data warehousing: dealing with the growing pains. In *Proceedings of the 13th International Conference on Data Engineering*, pages 199–205, 1997.

[14] Sane, Inc., http://www.sane.com.