A Privacy-Aware Database Interface
NCSU TR # TR-2003-05

Hal Haygood
Qingfeng He
Shane Smith
Jodi Snare

December 10, 2002
Executive Summary

Firms have long collected and used various types of data about their current and potential customers. The explosive growth of Internet-enabled technologies presents a significant opportunity in this arena; however, both consumer sentiment and governmental regulation are quickly increasing the need for tools to manage the privacy of this data. The World Wide Web Consortium (W3C) has developed the Platform for Privacy Preferences Project (P3P) to address this need.

One of the most serious gaps in P3P regards the enforcement of the privacy policies set forth by firms and agreed to by consumers. Few automated enforcement solutions exist, and their auditability is limited. The Privacy-Aware Database Interface (PADI) provides a platform with which firms can ensure that the use of data collected by their Internet-enabled systems is compliant with the policies under which it was collected.

PADI acts as a gatekeeper which requires that programs requesting data provide the purpose for which they will use that data. It then allows or denies access to the data according to the P3P privacy policy under which the data was collected. By presenting a uniform application programming interface (API) for this access, PADI provides auditing capability at the source code level.

Primarily a research project, PADI has been implemented in limited prototype form, using Java Database Connectivity (JDBC) and Oracle technologies. While the prototype clearly has shortcomings, it effectively demonstrates how such a system could be successfully implemented. There are also several exciting potential enhancements that future projects may yet explore.

Privacy management is a rapidly developing field of study, with great potential for tightening the relationships between firms and their customers while protecting personal information. PADI can be an effective solution for one part of that puzzle.
Introduction

The Need for Privacy Technology

The collection, aggregation, and analysis of information about people’s attributes and behaviors are an important part of doing business. Today’s Internet-enabled technologies provide an unprecedented channel for gathering this information, and thus present organizations with tremendous opportunities.

These opportunities must be approached carefully. Consumers are becoming more aware of the implications that this data collection holds for their privacy and anonymity. Their personal information may be used in ways that they do not approve; for example, the shipping address for their online purchase might become an entry in a direct-mail database, or their clickstream might be captured so that a merchant can determine their attitudes for marketing purposes.

Furthermore, many industries, such as health care, are becoming more heavily regulated. HIPAA, the Health Insurance Portability and Accountability Act, has strict guidelines addressing how certain kinds of information can be used. Firms in this industry must be careful to ensure that they remain in compliance with these privacy guidelines.

P3P

Because of these and other factors, technologies to manage the privacy of personal information are beginning to gain more widespread attention. At the forefront of these technologies is the Platform for Privacy Preferences Project (P3P), a standard developed by the World Wide Web Consortium (W3C).

The W3C describes P3P as follows:

At its most basic level, P3P is a standardized set of multiple-choice questions, covering all the major aspects of a Web site's privacy policies. Taken together, they present a clear snapshot of how a site handles personal information about its users. P3P-enabled Web sites make this information available in a standard, machine-readable format. P3P enabled browsers can "read" this snapshot automatically and compare it to the consumer's own set of privacy preferences. P3P enhances user control by putting privacy policies where users can find them, in a form users can understand, and, most importantly, enables users to act on what they see.

Approaches for Privacy Policy Enforcement

Organizations implementing P3P technologies for their websites can approach enforcement in three primary ways: Ad hoc, or manual, enforcement; privacy-aware database storage; or privacy-aware programming interfaces.
Ad Hoc
Most firms currently implementing P3P technologies appear to be using ad hoc enforcement methodologies. Essentially, ad hoc enforcement implies that the privacy policies are developed and managed separately from the data collected about users. Furthermore, this means that programmers who access the data must rely on manual tools to ensure that they are using the data in ways that are compliant with the policy under which it was collected.

This clearly presents a problem for both users and firms. The user has no assurance that the firm is actually complying with the terms under which the data was collected, and the firm has no way to audit their software to ensure that they are complying with their own policies or government regulation.

Privacy-aware Database Storage
Privacy-aware database storage, a concept presented by Paul Miotti in his master’s thesis, “Conception and Implementation of a Privacy-Supportive Database”, provides the data storage medium itself with intelligence about privacy.

Privacy-aware Programming Interfaces
Integrating privacy awareness into the API used to develop the software for both data collection and eventual data use provides a solution for these issues. Middleware provides for strict access controls on the data, while still allowing firms and developers an easy method to audit their source code for compliance.

An API which manages this privacy information also provides portability across database platforms and programming languages. The middleware can be designed to use only standard relational database features, which allows for multiple backend database platforms. The conceptual features of the API can also be easily implemented in many programming languages (e.g. JDBC for Java or ODBC for C).

PADI
The rest of this paper explores the design and architecture of such an API, called the Privacy-Aware Database Interface (PADI). The prototype implementation is explored in detail; where compromises were made during development, they are explained and alternatives provided. Some possible future enhancements to the design are also discussed.

PADI Features and Limitations
PADI provides two primary features for controlling access to consumer data: purpose-based and recipient-based access control. It also provides logging of requests for which access was denied.
**Purpose-based Access Control**

The P3P specification provides the *PURPOSE* element, which states one or more purposes for which the collected data will be used. For example, the contact purpose states that:

> Information may be used to contact the individual, through a communications channel other than voice telephone, for the promotion of a product or service. This includes notifying visitors about updates to the Web site. This does not include a direct reply to a question or comment or customer service for a single transaction … In addition, this does not include marketing via customized Web content or banner advertisements embedded in sites the user is visiting.[iv]

When using PADI to access data collected under a P3P policy, the program requesting the data must first provide the purpose for which the data will be used. If the purpose provided does not match with an allowed purpose under which the data was collected, access to the data is denied and an exception or error is returned to the program. The invalid access attempt can also be logged to the database for auditing purposes.

**Recipient-based Access Control**

The P3P specification also provides the *RECIPIENT* element, which states one or more parties to whom the data may be transferred. For example, the delivery recipient means that:

> Legal entities *performing delivery services* that may use data for purposes other than completion of the stated purpose. This should also be used for delivery services whose data practices are unknown.[v]

PADI provides similar functionality surrounding the recipient as it does for the purpose, with the exception that if no recipient is supplied, PADI defaults to the ours recipient. If the recipient for the data does not match with an allowed recipient under which the data was collected, access to the data is denied. This type of invalid access attempt can also be logged.

**Internal Error Logging**

When invalid access is attempted, these errors are logged internally to the database for future audit purposes.

**PADI Limitations**

PADI’s raison d’etre is to provide assurance to an organization that they are using their customers’ data in a way that is compliant with their stated policies. Ensuring that all data access is via PADI is the responsibility of the organization itself; there is no provision for mandatory, externally-enforceable restrictions.
This limitation can be mitigated by implementing a network architecture in which only a single secondary server, acting as a gatekeeper, is allowed direct access to the database. The prototype has not been implemented in this manner, primarily due to time and resource constraints.

**PADI Architecture**

PADI consists of two primary toolsets, described in the prototype design documentation as Subsystem 1 and Subsystem 2. Subsystem 1 provides a toolset for creating a database schema capable of storing P3P policies and for populating policies into that schema. Subsystem 2 provides the API for developers to use when accessing user data stored in the database.

It is interesting to note that there is no requirement that user data and P3P policy data be stored in the same database, or even on the same platform. For the purposes of this paper, however, the prototype was developed with a single database instance storing both P3P policies and user data.

For the prototype accompanying this paper, the subsystems have been implemented as follows.

**Subsystem 1: XML-DBMS**

Subsystem 1 is based on a freely-available Java-based tool called XML-DBMS. This tool provides methods to translate XML Document Type Definitions (DTDs) into database schemas and to insert XML documents into those schemas. The P3P DTD is fed to a program named GenerateMap.java, which create a set of SQL statements that describe the tables and fields necessary to store P3P policies. This set of SQL statements is then used to actually create the necessary tables. (See Appendix A.)

GenerateMap.java also creates a map file. Another XML-DBMS-based program, Transfer.java, uses this map file to take an actual P3P policy and create SQL statements that will insert that policy into the database.

**Subsystem 2: PADI**

Subsystem 2 consists of two conceptually distinct pieces: a set of prerequisites for storing user data, and the PADI object, which provides the API for programmers wishing to retrieve user data from the database.

---

1 This is actually somewhat simplified. During prototype development, it was discovered that the official P3P DTD contains ambiguities which prevent it from being translated exactly into a coherent database schema. Resolving these ambiguities requires making edits to the official DTD. (See Appendix C.) While these changes do affect the DTD semantics, they address areas unrelated to the purpose of the PADI. Also, the resulting SQL must be edited to replace hyphens with underscores and to prepend “MY” to names such as “PUBLIC” or “DATE” which are Oracle keywords.
The prerequisites for storing user data into the database are:

- The P3P policy which will govern access to the data must be already populated in the database.
- The database must have a P3P_MAP table to link table fields to specific P3P “DATA” elements. (See Appendix B.)
- The P3P_MAP table must contain a record for each table field to which access should be controlled.
- The database must have an ERRORS table to store access violation records. (See Appendix C.)

The PADI object is a wrapper around the built-in java.sql.statement class. It is extended by two methods: setPurpose and setRecipient. The user of the class must set the purpose; the recipient defaults to ours (a special recipient that indicates the firm itself).

A simple example of a PADI session looks like this:

```java
// Note that conn represents an already open database connection
PADI query = new PADI(conn.createStatement());
query.setPurpose(query.TELEMARKETING);
query.setRecipient(query.OURS);
ResultSet rs = query.executeQuery("select ccnum from customer");
while (rs.next()) {
   // Do something with results
}
```

Assuming that telemarketing is not allowed to receive credit card numbers, this example, without a try/catch block, would halt with a fatal exception at the `executeQuery` statement.

Behind the scenes, PADI takes the query string passed to it – in this case, `select ccnum from customer` – and parses out the list of tables and columns being accessed. It then queries the database to determine if the provided purpose and recipient are compliant with those tables and columns. If the access is allowed, it runs the provided query string and returns the result set; if the access is disallowed, it throws an exception. (Obviously, in production programs, there should be a try/catch block around the `executeQuery` statement to deal with such failures.)

**Prototype Limitations**

Time and development resources have been limited, and this is reflected in the feature set that was actually implemented in the prototype. The following list details the areas in which the prototype is limited with regards to the described functionality, as well as ways that the team has determined that these limitations can be overcome in the future.

**Limited SQL Syntax**

Queries passed through the PADI prototype must conform to a very limited subset of SQL. In particular, the only allowed SQL statement type is a simple SELECT from a single table which conforms to the following stub:
SELECT field [, field ...] [AS name [, name ...]]
FROM table [WHERE conditional]

This is a limitation of the simple SQL parser included in the PADI object. This can be resolved by enhancing this parser or replacing it with an external parser.

Future enhancers should take note that once the parser has been enhanced to allow selecting from multiple tables, care must be taken that the purpose and recipient restrictions on fields in the WHERE clause are honored, since fields in the WHERE clause may not necessarily be included in the SELECT list.

**Reliance on Oracle**

While there is nothing in the PADI design at this time which restricts it to a particular vendor’s product, the prototype has been developed against Oracle 9.2i. This has necessarily caused some vendor-specific syntax to be included in the source code, which will necessitate some rework if PADI is to be used with a different DBMS.

**Incomplete Handling of REQUIRED attributes**

Both PURPOSE and RECIPIENT policy elements can contain an attribute called REQUIRED. This attribute denotes whether the purpose is always required (that is, users cannot opt in or out of the use of their data for this purpose), or can be opted in or out. The PADI prototype always acts as if this attribute is set to always (the default, if none is specified). Future enhancements of PADI should address this; one possible solution is to add a field to each data table which specifies the opt-in and/or opt-out preference for that particular record.

**Potential Future Enhancements**

The limited time and resources provided to the research team have led to considerable scope contraction over the lifetime of the project. There are several enhancements to both the PADI design and the prototype implementation that the team was forced to abandon during development. Some of these enhancements are enumerated below.

**Design Enhancements**

**Direct ties between data and policy**

As it currently stands, PADI restricts access to data based solely on the table and column names. This is adequate protection for a small database, but may require compromises to data normalization rules. For example, if a firm collects customer data under two different policies – say, an order page and a mailing list subscription page – this data must be stored in two different tables, even though many of the fields are duplicated between the tables.

One potential resolution for this shortcoming is to take advantage of some of the more sophisticated object-oriented features provided with databases such as Oracle.
allows for the creation of user-specified data object types. These objects can contain data fields, references to other objects, and methods. One might create a data type for a customer’s last name which contains a string field to store the name, a reference to a DATA element in a P3P policy, and a method which selects the proper DATA element to which to link at the time the object is inserted into the database. This would allow conceptually homogenous data, such as customers’ last names, to be stored in a single table, while still allowing each individual data entry to be linked to a different P3P policy.

**Prototype Enhancements**

**Database creation and policy mapping tooling**

There are still many manual steps necessary to create a P3P database. Many of these steps, such as renaming tables and fields that conflict with Oracle keywords, could be automated.

Also, there is currently no tooling in place for creating the records in the P3P_MAP table that link table and column names to DATA elements; they must be created by hand. Development of a tool to ease this linking should be a fairly straightforward programming exercise.

**Separation of policy and data storage**

As previously mentioned, there is nothing in the PADI design as it stands that would prevent the storage of P3P policies and actual collected data from being separated. This would allow organizations implementing PADI to provide a single storehouse for P3P policy data, while allowing collected data to be stored in several different databases.

Note that this enhancement is fundamentally in conflict with the described design change that would include object references within stored data items. Oracle currently has no functionality that allows for references between database instances to be created.

**Lessons Learned**

Through the course of the project, the team learned several important lessons. In no particular order, these are:

**The Importance of Clearly-Defined Goals**

In the initial stages of the project, the goals were nebulous – the best statement of the project’s intent that we had was to “ensure that practice matches policy.” This led to considerable confusion during the project’s initiation as the team struggled to define more concrete goals. It is not clear how this could have been avoided, however, since this project is primarily a research-oriented project addressing new areas of privacy thought.

**Standards Can Be Vague**

The P3P specification 1.0, while important in its intent, is clearly lacking in some key areas. For example, the RETENTION element has no specification for how data retention
should be defined; it is essentially freeform text. The purpose of P3P is to provide privacy policy information in a machine-readable format that can easily be automated; this “feature” runs counter to that purpose, since it makes parsing and interpreting the field problematic at best and simply impossible at worst.

The project, in fact, had originally intended to tackle RETENTION as part of the enforcement that PADI provides. This gap in the standard, however, prohibited the team from doing so.

**Scope Contraction is Sometimes Necessary**

The original design of PADI was quite ambitious, including enforcement of data retention policies and allowing the direct data-to-policy ties described under “Design Enhancements.” Clearly, this scope was too ambitious for the time and resources available to the project team, and had to be contracted somewhat.

At several points during design and development, the team was faced with choices about what features could be implemented and which direction the design could be allowed to take. At these decision points, the team informally discussed the ramifications of the necessary scope contraction with the project sponsors and kept everyone informed of the evolving nature of the project. Without this strong communication, the project may have failed entirely to produce a workable specification or prototype.
Appendix A: Database Creation SQL
This is the script used to create the schema for the database which stores the P3P policies.

-- create table to store P3P policies
CREATE TABLE "XMLDBMSKEY" ("HIGHKEY" INTEGER)
/
INSERT INTO XMLDBMSKEY VALUES (0)
/
CREATE TABLE "DATA_STRUCT" ("DATA_STRUCTORDER" INTEGER, "STRUCTREF" VARCHAR(255), "DATASCHEMA FK" INTEGER, "DATA_STRUCT PK" INTEGER, "LONG_DESCRIPTION" VARCHAR(255), "SHORT_DESCRIPTION" VARCHAR(255), "LONG_DESCRIPTIONORDER" INTEGER, "NAME" VARCHAR(255))
/
CREATE TABLE "CATEGORIES" ("DATA DEFFK" INTEGER, "CATEGORIESORDER" INTEGER, "DATA FK" INTEGER, "CATEGORIESSPK" INTEGER, "DATA_STRUCTFK" INTEGER)
/
CREATE TABLE "POLICY" ("DISCURI" VARCHAR(255), "POLICYPK" INTEGER, "TEST" VARCHAR(255), "POLICIESFK" INTEGER, "TESTORDER" INTEGER, "OPTURI" VARCHAR(255), "POLICYORDER" INTEGER, "NAME" VARCHAR(255))
/
CREATE TABLE "IMG" ("WIDTH" VARCHAR(255), "IMGORDER" INTEGER, "DISPUTESFK" INTEGER, "HEIGHT" VARCHAR(255), "ALT" VARCHAR(255), "SRC" VARCHAR(255))
/
CREATE TABLE "DATA_DEF" ("STRUCTREF" VARCHAR(255), "DATASCHEMA FK" INTEGER, "DATA_DEFIN ORDER" INTEGER, "LONG_DESCRIPTION" VARCHAR(255), "SHORT_DESCRIPTION" VARCHAR(255), "LONG_DESCRIPTIONORDER" INTEGER, "DATA_DEF PK" INTEGER, "NAME" VARCHAR(255))
/
CREATE TABLE "META" ("META PK" INTEGER, "METAORDER" INTEGER)
/
CREATE TABLE "CUSTOMIZATION" ("CUSTOMIZATIONORDER" INTEGER, "PURPOSEFK" INTEGER, "REQUIRED" VARCHAR(255))
/
CREATE TABLE "SAME" ("SAMEPCDATAORDER" INTEGER, "SAMEPCDATA" VARCHAR(255), "RECIPIENTFK" INTEGER, "SAMEORDER" INTEGER, "REQUIRED" VARCHAR(255))
/
CREATE TABLE "INDIVIDUAL_ANALYSIS" ("PURPOSEFK" INTEGER, "INDIVIDUAL ANALYSIS ORDER" INTEGER, "REQUIRED" VARCHAR(255))
/
CREATE TABLE "INDIVIDUAL DECISION" ("INDIVIDUAL DECISION ORDER" INTEGER, "PURPOSEFK" INTEGER, "REQUIRED" VARCHAR(255))
/
CREATE TABLE "CONTACT" ("PURPOSEFK" INTEGER, "CONTACTORDER" INTEGER, "REQUIRED" VARCHAR(255))
/
CREATE TABLE "OTHER_PURPOSE" ("OTHER PURPOSEPCDATA" VARCHAR(255), "PURPOSEFK" INTEGER, "OTHER PURPOSEORDER" INTEGER, "OTHER PURPOSEPCDATAORDER" INTEGER, "REQUIRED" VARCHAR(255))
CREATE TABLE "PSEUDO_DECISION" ("PURPOSEFK" INTEGER, "PSEUDO_DECISIONORDER" INTEGER, "REQUIRED" VARCHAR(255))

CREATE TABLE "EXPIRY" ("POLICY_REFERENCESFK" INTEGER, "MYDATE" VARCHAR(255), "POLICYFK" INTEGER, "MAX_AGE" VARCHAR(255), "EXPIRYORDER" INTEGER)

CREATE TABLE "MYPUBLIC" ("PUBLICPCDATAORDER" INTEGER, "PUBLICORDER" INTEGER, "PUBLICPCDATA" VARCHAR(255), "RECIPIENTFK" INTEGER, "REQUIRED" VARCHAR(255))

CREATE TABLE "OTHER_RECIPIENT" ("OTHER_RECIPIENTORDER" INTEGER, "OTHER_RECIPIENTPCDATA" VARCHAR(255), "RECIPIENTFK" INTEGER, "REQUIRED" VARCHAR(255))

CREATE TABLE "TAILORING" ("TAILORINGORDER" INTEGER, "PURPOSEFK" INTEGER, "REQUIRED" VARCHAR(255))

CREATE TABLE "DATA" ("REF" VARCHAR(255), "DATA_GROUPFK" INTEGER, "OPTIONAL" VARCHAR(255), "DATAPK" INTEGER, "DATAORDER" INTEGER)

CREATE TABLE "PROFILING" ("PROFILINGORDER" INTEGER, "PROFILINGFK" INTEGER, "REQUIRED" VARCHAR(255))

CREATE TABLE "PURPOSE" ("PURPOSEORDER" INTEGER, "STATEMENTFK" INTEGER, "PURPOSEFK" INTEGER)

CREATE TABLE "UNRELATED" ("UNRELATEDORDER" INTEGER, "UNRELATEDPCDATAORDER" INTEGER, "RECIPIENTFK" INTEGER, "REQUIRED" VARCHAR(255), "UNRELATEDPCDATA" VARCHAR(255))

CREATE TABLE "TELEMARKETING" ("TELEMARKETINGORDER" INTEGER, "PURPOSEFK" INTEGER, "REQUIRED" VARCHAR(255))

CREATE TABLE "HISTORICAL" ("PURPOSEFK" INTEGER, "HISTORICALORDER" INTEGER, "REQUIRED" VARCHAR(255))

CREATE TABLE "ENTITY" ("ENTITYORDER" INTEGER, "ENTITYFK" INTEGER, "POLICYFK" INTEGER)

CREATE TABLE "DATA_GROUP" ("DATA_GROUPORDER" INTEGER, "DATAGROUPFK" INTEGER, "ENTITYFK" INTEGER, "BASE" VARCHAR(255), "STATEMENTFK" INTEGER)

CREATE TABLE "MYACCESS" ("OTHER_IDENT" VARCHAR(255), "OTHER_IDENTORDER" INTEGER, "IDENT_CONTACT" VARCHAR(255), "NONIDENTORDER" INTEGER, "NONEORDER" INTEGER, "CONTACT_AND_OTHER" VARCHAR(255), "ACCESSORDER" INTEGER, "ALLORDER" INTEGER, "NONE" VARCHAR(255), "IDENT_CONTACTORDER" INTEGER, "POLICYFK" INTEGER, "MYALL" VARCHAR(255), "NONIDENT" VARCHAR(255), "CONTACT_AND_OTHERORDER" INTEGER, "ACCESSFK" INTEGER)

CREATE TABLE "MYCURRENT" ("CURRENTORDER" INTEGER, "PURPOSEFK" INTEGER, "REQUIRED" VARCHAR(255))

CREATE TABLE "POLICIES" ("METAFK" INTEGER, "POLICIESFK" INTEGER, "POLICIESORDER" INTEGER)
CREATE TABLE "DATASHEMA" ("DATASCHEMAPK" INTEGER, "DATASHEMAORDER" INTEGER, "POLICYFK" INTEGER)
/
CREATE TABLE "EXCLUDE" ("EXCLUDE" VARCHAR(255), "EXCLUDEORDER" INTEGER, "POLICY_REFFK" INTEGER)
/
CREATE TABLE "OURS" ("OURS" VARCHAR(255), "RECIPIENTFK" INTEGER, "OURSORDER" INTEGER)
/
CREATE TABLE "PHYSICAL" ("PHYSICAL" VARCHAR(255), "CATEGORIESFK" INTEGER, "PHYSICALORDER" INTEGER)
/
CREATE TABLE "HEALTH" ("HEALTH" VARCHAR(255), "CATEGORIESFK" INTEGER, "HEALTHORDER" INTEGER)
/
CREATE TABLE "FINANCIAL" ("FINANCIAL" VARCHAR(255), "FINANCIALORDER" INTEGER, "CATEGORIESFK" INTEGER)
/
CREATE TABLE "POLITICAL" ("POLITICAL" VARCHAR(255), "POLITICALORDER" INTEGER, "CATEGORIESFK" INTEGER)
/
CREATE TABLE "OTHER" ("OTHER" VARCHAR(255), "OTHERORDER" INTEGER, "CATEGORIESFK" INTEGER)
/
CREATE TABLE "PURCHASE" ("PURCHASE" VARCHAR(255), "PURCHASEORDER" INTEGER, "CATEGORIESFK" INTEGER)
/
CREATE TABLE "STATE" ("STATE" VARCHAR(255), "STATEORDER" INTEGER, "CATEGORIESFK" INTEGER)
/
CREATE TABLE "INTERACTIVE" ("INTERACTIVEORDER" INTEGER, "INTERACTIVE" VARCHAR(255), "CATEGORIESFK" INTEGER)
/
CREATE TABLE "MONEY" ("REMEDIESFK" INTEGER, "MONEY" VARCHAR(255), "MONEYORDER" INTEGER)
/
CREATE TABLE "MYONLINE" ("MYONLINE" VARCHAR(255), "CATEGORIESFK" INTEGER, "ONLINEORDER" INTEGER)
/
CREATE TABLE "LAW" ("REMEDIESFK" INTEGER, "LAW" VARCHAR(255), "LAWORDER" INTEGER)
/
CREATE TABLE "INCLUDE" ("INCLUDE" VARCHAR(255), "INCLUDEORDER" INTEGER, "POLICY_REFFK" INTEGER)
/
CREATE TABLE "UNIQUEID" ("UNIQUEID" VARCHAR(255), "UNIQUEIDORDER" INTEGER, "CATEGORIESFK" INTEGER)
/
CREATE TABLE "GOVERNMENT" ("GOVERNMENTORDER" INTEGER, "GOVERNMENT" VARCHAR(255), "CATEGORIESFK" INTEGER)
/
CREATE TABLE "EMBEDDED_EXCLUDE" ("EMBEDDED_EXCLUDE" VARCHAR(255), "POLICY_REFFK" INTEGER, "EMBEDDED_EXCLUDEORDER" INTEGER)
/
CREATE TABLE "DATAPCDATA" ("DATAFK" INTEGER, "DATAPCDATAORDER" INTEGER, "DATAPCDATA" VARCHAR(255))
/
CREATE TABLE "METAPCDATA" ("METAFK" INTEGER, "METAFCDATAORDER" INTEGER, "METAFCDATA" VARCHAR(255))
/
CREATE TABLE "COMPUTER" ("CATEGORIESFK" INTEGER, "COMPUTER" VARCHAR(255), "COMPUTERORDER" INTEGER)
/
CREATE TABLE "DEMOGRAPHIC" ("DEMOGRAPHICORDER" INTEGER, "CATEGORIESFK" INTEGER, "DEMOGRAPHIC" VARCHAR(255))
/
CREATE TABLE "NAVIGATION" ("NAVIGATIONORDER" INTEGER, "CATEGORIESFK" INTEGER, "NAVIGATION" VARCHAR(255))
/
CREATE TABLE "PREFERENCE" ("CATEGORIESFK" INTEGER, "PREFERENCE" VARCHAR(255), "PREFERENCEORDER" INTEGER)
/
CREATE TABLE "CONTENT" ("CONTENTORDER" INTEGER, "CATEGORIESFK" INTEGER, "CONTENT" VARCHAR(255))
/
CREATE TABLE "METHOD" ("POLICY_REFFK" INTEGER, "METHODORDER" INTEGER, "METHOD" VARCHAR(255))
/
CREATE TABLE "EMBEDDED_INCLUDE" ("POLICY_REFFK" INTEGER, "EMBEDDED_INCLUDEORDER" INTEGER, "EMBEDDED_INCLUDE" VARCHAR(255))
/
CREATE TABLE "CORRECT" ("REMEDIESTFK" INTEGER, "CORRECTORDER" INTEGER, "CORRECT" VARCHAR(255))
/
commit
Appendix B: P3P_MAP Table Creation SQL

This is the SQL script used to create the P3P_MAP table.

```sql
-- create p3p_map table
drop table p3p_map;
create table p3p_map (  
p3p_mappk number not null primary key,  
tablename varchar(255),  
columnname varchar(255),  
datafk number  
);

-- create p3p_mapsequence autoincrementer for creating PKs
drop sequence p3p_mapsequence;
create sequence p3p_mapsequence  
    increment by 1  
    start with 1  
    nomaxvalue  
    minvalue 1  
    nocycle  
    nocache  
    noorder  
;

-- create p3p_mapincrement trigger to populate PKs
create or replace trigger p3p_mapincrement  
    before insert on p3p_map  
    for each row  
begin  
    select p3p_mapsequence.nextval  
    into :new.p3p_mappk  
    from dual;  
end;  
/

-- uncomment following to test autoincrement trigger

-- insert into P3P_map  
-- (table_name, column_name, dataref_id)  
-- values  
-- ('test table', 'test column', '42')  
-- ;
```
Appendix C: ERRORS Table Creation SQL

This is the SQL script used to create the ERRORS table.

```sql
-- create errors table
drop table errors;
create table errors
  (id number(6) not null primary key,
   code number(6) not null,
   text varchar(1024),
   time date
  );

-- create errorssequence autoincrementer for creating PKs
drop sequence errorssequence;
create sequence errorssequence
  increment by 1
  start with 1
  nomaxvalue
  minvalue 1
  nocycle
  nocache
  noorder
  ;

-- create errorsincrement trigger to populate PKs
create or replace trigger errorsincrement
before insert on errors
for each row
begin
  select errorssequence.nextval
  into :new.id
  from dual;
end;
/

-- create errordate trigger to populate date
create or replace trigger errordate
before insert on errors
for each row
begin
  select sysdate
  into :new.time
  from dual;
end;
/

-- uncomment following to test triggers

-- insert into errors
-- (code, text)
-- values
-- (402, 'access denied')
-- ;
```
References


v Ibid. Available online at http://www.w3.org/TR/P3P/#RECPNT.