

Database Systems Project

Phase V: GIS Attribute Set Template Website

Eric England

Richard Dominguez

Jason "Jay Jay" Williams

<http://cmps342.psychopup.net/>

Table of Contents

Phase I: GIS Attribute Set Template Sharing	5
1.1 Techniques Used	5
1.3 Introduction to Enterprise and Organization	5
1.4 Structure of the Enterprise	5
1.5 Itemized Description of Major Objects	6
1.6 Data Views and Operations for User Groups	6
2.1 Entity Set Description	7
User.....	7
AttributeSet.....	7
Attribute	8
AttributeType.....	9
Category.....	9
DownloadTracking.....	10
2.2 Relationship Set Description	10
HasAttributeSets.....	10
HasDownloaded	10
HasDownloadTrackings.....	10
IsOfCategory	11
HasAttributes	11
IsOfAttributeType.....	11
2.4 Entity Relational Diagram:	12
Phase II: Relational Database Model	13
2.1 ER Model And Relational Model	13
Description	13
Comparison	13
Conversion from ER Model to Relational Model.....	13
Constraints	14
2.2 ER Database to Relational Database Conversion Overview	15
User.....	15
AttributeSet.....	16
Attribute	17
AttributeType.....	18
Category.....	19
DownloadTracking.....	20
2.3 ER Database to Relational Database Conversion Data	21
User.....	21
AttributeSet.....	22
Attribute	23
AttributeType.....	24
Category.....	25
DownloadTracking.....	26
2.4 Queries Used	27
2.5 Query Representation	28
Phase III: Implementation of the Relational Database	30
3.0 Relation Normalization	30
When a table is in 1NF:.....	30

When a table is in 2NF:.....	30
When a table is in 3NF:.....	30
When a table is in BCNF:.....	30
3.1 About SQL*PLUS	31
3.2 Oracle Schema Objects	31
Tables.....	31
Indexes	31
Stored Procedures / Functions.....	31
Synonyms.....	32
Views.....	32
Dimensions.....	32
Database Linking	32
Sequence Generation	32
Packages	32
3.3 Instances and Relation Schemas.....	34
3.4 SQL Queries.....	41
Phase IV: Implementation of the Relational Database (Continued)	44
4.1 Common Features in Oracle PL/SQL and MS Trans-SQL	44
Components which consist of PL/SQL and MS Trans-SQL	44
Purposes of Stored subprograms	44
Benefits of subprograms over Dynamic SQL	44
4.2 Oracle PL/SQL	45
General Format of a PL/SQL Program:.....	45
Control Statements:.....	45
Stored Procedures:.....	46
Stored Functions:.....	47
Triggers:.....	47
Cursors:.....	48
Packages:.....	48
4.3 Oracle PL/SQL Sub-Programs.....	49
Stored Procedures.....	49
Stored Functions.....	51
Triggers.....	52
Phase V: Implementation of the Relational Database.....	53
5.1 Daily User Activities.....	53
Anonymous Users.....	53
Registered Users	53
Administrative Users	53
5.2 Relations, Views, and Subprograms	54
Relations.....	54
Views.....	54
Subprograms	54
5.3 Application Screenshots and Descriptions	55
Home Screen.....	55
Login Screen.....	56
Registration Screen.....	57
User Home Screen	58
Editing My Account.....	59
Editing My Attribute Sets	60
Editing Attributes For An Attribute Set	61
Searching Attribute Sets	62
Previewing Attributes for an Attribute Set.....	63

Downloading JSON or XML 64
Reports for Administrators 65
Latest Site Activity..... 66
5.4 Code Description..... 67
 Data Access Layer (DAL) 67
 Presentation Layer..... 68
5.5 Development Process..... 70
 Phase 1: Information Gathering 70
 Phase 2: Data Access Layer – DTO..... 70
 Phase 3: Data Access Layer – Repository and Extension Methods 70
 Phase 4: Presentation Layer – Design and Views 70
 Phase 5: Presentation Layer – Putting It All Together 71
5.6 Conclusion..... 72

Phase I: GIS Attribute Set Template Sharing

1. Fact-Finding Techniques

1.1 Techniques Used

The interview process for this project involved interviewing myself (Richard Dominguez). Talking through exactly what Richard wanted helped us develop the model for the project. Part of this interviewing helped Eric and Jason learn the complexity of the application suite, what exactly they are designing the database for, and which servers and related technologies were involved. After listing out exactly what types of data we were serializing, it was much easier to conceptually design the data needed and ultimately the ERM.

1.3 Introduction to Enterprise and Organization

Summer 2011 Richard Dominguez developed a mobile application to run on iOS, PsychoGIS Scratchpad. The application allows users to create Data Sets to collect GIS data. One of the applications features is the ability to create “Attribute Sets”. Attribute sets are nothing more than a template of common attributes that would be associated with a feature on earth. For example, let’s say you’re collecting coordinate and attribute data for the trees on school campus. Trees may have the following attributes: height, age, and species, etc. After creating this attribute set for trees, wouldn’t it be nice to share them with other organizations that would like to collect their own tree data?

This is where “GIS Scratchpad Template Sharing” website project (and related database) steps in. Registered users will be able to upload, create, modify, publish, and share their Attribute Set templates with other registered users.

1.4 Structure of the Enterprise

Richard Dominguez is the creator of the original program used on iOS platform. There is only one developer, publisher, and company involved. There are several servers needed that form the bulk of the application suite:

- SVN Server: Server that holds the versioned projects. The purpose is to share the project data between future developers / students.
- Web Server: Server that sits on the Internet. The purpose is to be the “Company” web server, as well as the application server (web services) to serve JSON and XML.
- Backup Server: Server that sits at home and makes incremental backups of the SVN and Web server in case of server-failure.

1.5 Itemized Description of Major Objects

The two main entities are the User and AttributeSet. A user will be able to create many Attribute Sets. An Attribute Set will belong to a single Category. Balancing Attribute Sets, the user will be able to add many related Attributes. Each Attribute will have a single AttributeType. The DownloadTracking entity will keep track of downloads of AttributeSets by Users.

1.6 Data Views and Operations for User Groups

There will only be two types of users: Admins and Registered Users.

- Users: will only have access to editing their own respective AttributeSet and Attribute entities. They can request to publish their AttributeSets, assign a Category, as well as un-publish their AttributeSets should they not wish to share the data.
- Administrators will have access to deactivate users (via the User entity), publish and un-publish Attribute Sets, and modify Category data. An Administrator will also be able to delete ANY information on the website they deem inappropriate or redundant.

2. Conceptual Database Design

2.1 Entity Set Description

User

- This entity type describes all the users that will have access to the site. This entity holds login, contact and access information for each user to the website.
- Candidate keys: UserId, UserName.
- Primary key: UserId.
- Strong/Weak Entity: Strong.
- Fields to be indexed: UserId, UserName
- Attributes:

Name:	UserId	UserName	Password	EmailAddress	DateCreated	DateCancelled
Description:	Primary key.	User's login for logging in.	Password for login.	Email address to send notifications to.	Date account opened.	Date account cancelled.
Domain/Type:	Unsigned Integer	String	String	String	DateTime	DateTime
Value Range:	0 ... 2^{32}	Any	Any	Any	Any	Any
Default Value:	None	None	None	None	None	None
Nullable?	No	No	No	No	Yes	Yes
Unique?	Yes	Yes	No	No	No	No
Single / Multiple:	Single	Single	Single	Single	Single	Single
Simple or Composite:	Simple	Simple	Simple	Simple	Simple	Simple

AttributeSet

- This entity type describes is one of many user's Attribute Sets. An example of an Attribute Set would be "Building Attributes", which would relate to things that describe a building.
- Candidate keys: AttributeSetId.
- Primary key: AttributeSetId.
- Strong/Weak Entity: Weak.
- Fields to be indexed: AttributeSetId, Description, IsPublished, Name
- Attributes:

Name:	AttributeSetId	Description	DateLastModified	IsPublished	Name	DateCreated	CategoryId	UserId
Description:	Primary key.	What the AttributeSet is for.	Date last edited.	Is set published (public?)	Name of AttributeSet	Date created.	Category this set belongs to.	Owner of set.
Domain/Type:	Unsigned Integer	String	DateTime	Boolean	String	DateTime	Unsigned Integer	Unsigned Integer
Value Range:	0 ... 2 ³²	Any	Any	0, 1	Any	Any	0 ... 2 ³²	0 ... 2 ³²
Default Value:	None	None	None	None	None	None	None	None
Nullable?	No	No	No	No	No	No	No	No
Unique?	Yes	Yes	No	No	No	No	Yes	Yes
Single / Multiple:	Single	Single	Single	Single	Single	Single	Single	Single
Simple or Composite:	Simple	Simple	Simple	Simple	Simple	Simple	Simple	Simple

Attribute

- This entity type describes is one attribute of an AttributeSet. I.e., one attribute of a “Buildings” AttributeSet may be “Square Footage”.
- Candidate keys: AttributeId, AttributeSetId
- Primary key: AttributeId.
- Strong/Weak Entity: Weak.
- Fields to be indexed: AttributeId, Name
- Attributes:

Name:	AttributeId	Description	DateLastModified	Name	AttributeSetId	AttributeTypeId
Description:	Primary key.	Description of this attribute for the template.	Date last edited.	Name of Attribute.	AttributeSet this Attribute belongs to.	Attribute type of this Attribute.
Domain/Type:	Unsigned Integer	String	DateTime	String	Unsigned Integer	Unsigned Integer
Value Range:	0 ... 2 ³²	Any	Any	Any	0 ... 2 ³²	0 ... 2 ³²
Default Value:	None	None	None	None	None	None
Nullable?	No	Yes	No	No	No	No
Unique?	Yes	No	No	No	Yes	Yes
Single / Multiple:	Single	Single	Single	Single	Single	Single
Simple or Composite:	Simple	Simple	Simple	Simple	Simple	Simple

AttributeType

- This entity type describes an attribute's type. I.e., String, Integer, Double.
- Candidate keys: AttributeTypeId
- Primary key: AttributeTypeId.
- Strong/Weak Entity: Strong.
- Fields to be indexed: AttributeTypeId, Name
- Attributes:

Name:	AttributeTypeId	DateLastModified	Name
Description:	Primary key.	Date last edited.	Name of Attribute type.
Domain/Type:	Unsigned Integer	DateTime	String
Value Range:	0 ... 2 ³²	Any	Any
Default Value:	None	None	None
Nullable?	No	No	No
Unique?	Yes	No	No
Single / Multiple:	Single	Single	Single
Simple or Composite:	Simple	Simple	Simple

Category

- This entity type describes a category for Attribute Sets. I.e., "Flora" and "Fauna" are examples of categories.
- Candidate keys: CategoryId
- Primary key: CategoryId.
- Strong/Weak Entity: Strong.
- Fields to be indexed: CategoryId, Name
- Attributes:

Name:	CategoryId	Name	DateCreated	DateLastModified
Description:	Primary key.	Category name.	Date category was added.	Date category was last modified.
Domain/Type:	Unsigned Integer	String	DateTime	DateTime
Value Range:	0 ... 2 ³²	Any	Any	Any
Default Value:	None	None	None	None
Nullable?	No	No	No	No
Unique?	Yes	Yes	No	No

Single / Multiple:	Single	Single	Single	Single
Simple or Composite:	Simple	Simple	Simple	Simple

DownloadTracking

- This entity type describes a download that occurred by a user for a particular attribute set. This data used for analytical purposes both public and private. Candidate keys: DownloadTrackingId, AttributeSetId
- Primary key: DownloadTrackingId.
- Strong/Weak Entity: Weak.
- Fields to be indexed: DownloadTrackingId, AttributeSetId
- Attributes:

Name:	DownloadTrackingId	DateDownloaded	UserId	AttributeSetId
Description:	Primary key.	Date downloaded.	User who downloaded the Attribute Set.	Attribute set downloaded.
Domain/Type:	Unsigned Integer	DateTime	Unsigned Integer	Unsigned Integer
Value Range:	0 ... 2 ³²	Any	0 ... 2 ³²	0 ... 2 ³²
Default Value:	None	None	None	None
Nullable?	No	No	No	No
Unique?	Yes	No	Yes	Yes
Single / Multiple:	Single	Single	Single	Single
Simple or Composite:	Simple	Simple	Single	Simple

2.2 Relationship Set Description

HasAttributeSets

- Mapping Cardinality: 1..M
- Description Field: A user can create many AttributeSets. Only one user owns an AttributeSet.
- Participation Constraint: Optional for User, Mandatory for AttributeSet

HasDownloaded

- Mapping Cardinality: 1..M
- Description Field: A user can download many AttributeSets. A DownloadTracking can only have one user.
- Participation Constraint: Since this is for historical purposes, it is optional for both User and DownloadTracking. This is up to the database manager whether this information is kept.

HasDownloadTrackings

- Mapping Cardinality: 1..M

- Description Field: An AttributeSet can be downloaded many times. A DownloadTracking can only have one AttributeSet.
- Participation Constraint: Since this is for historical purposes, it is optional for both AttributeSet and DownloadTracking. This is up to the database manager whether this information is kept.

IsOfCategory

- Mapping Cardinality: 1..M
- Description Field: An AttributeSet can only have one Category. A category can have many related AttributeSets.
- Participation Constraint: Optional for Category, Mandatory for AttributeSet

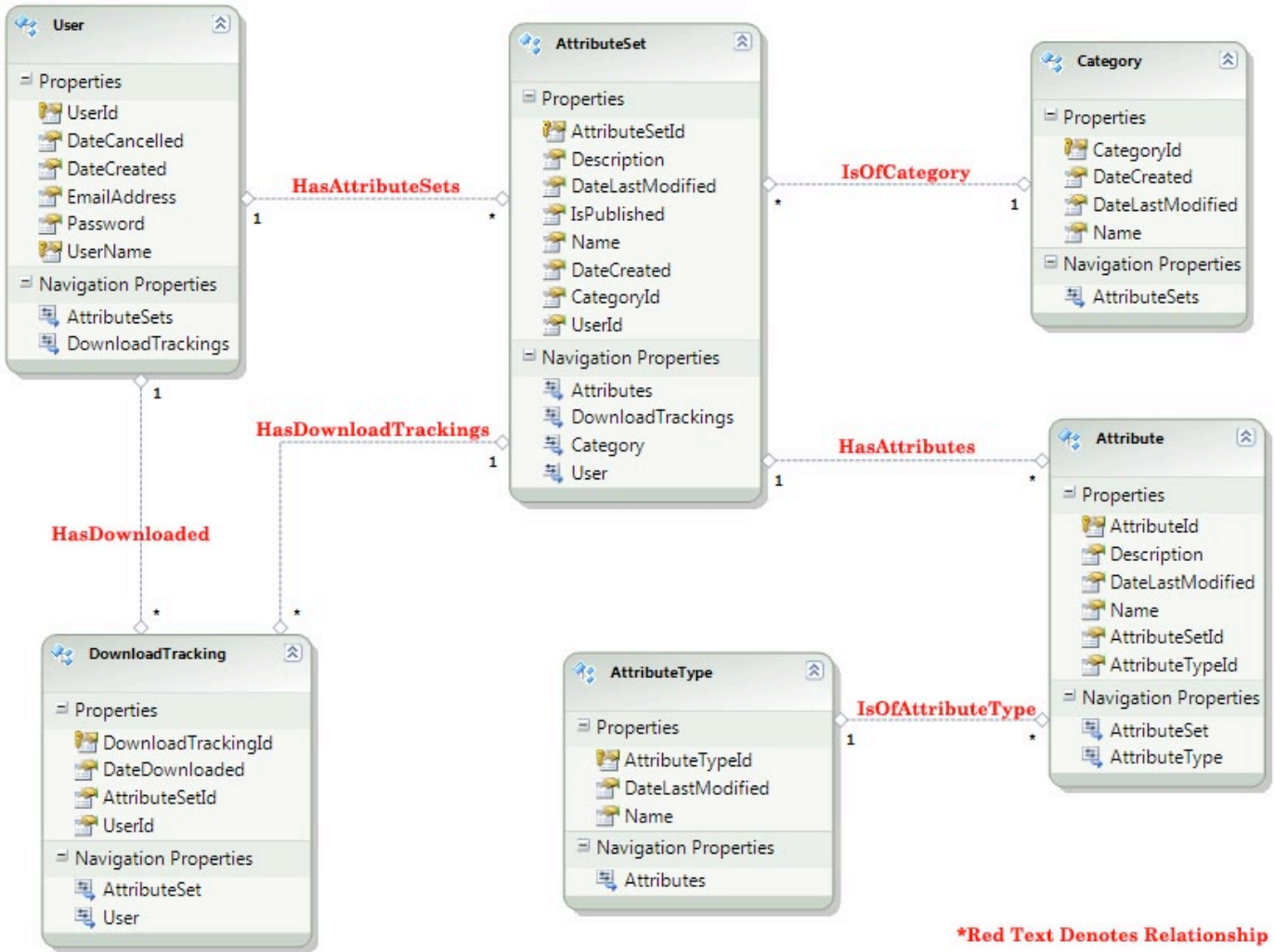
HasAttributes

- Mapping Cardinality: 1..M
- Description Field: An AttributeSet can have many Attributes. An Attribute can only have one AttributeSet.
- Participation Constraint: Mandatory for both AttributeSet and Attribute

IsOfAttributeType

- Mapping Cardinality: 1..M
- Description Field: An Attribute can have one AttributeType. An AttributeType can have many related Attributes.
- Participation Constraint: Mandatory for both AttributeType and Attribute

2.4 Entity Relational Diagram:



Phase II: Relational Database Model

2.1 ER Model and Relational Model

2.1 ER Model And Relational Model

Description

The Relational Model was invented and popularized by E.F. (Ted) Codd of IBM as a general model of data. This model started to be used commercially due to its blatant simplicity and mathematical background. It is so popular in fact that it is still widely used since its inception back in the 1970s known as System R. The model revolves around and provides a declarative method in creating data and querying the database for data.

Simply put, the designer or user can command directly in what information the database will contain and what information they want from it. In a relation, data grabbed is known as “tuples” which contain the data needed. The data in these tuples are labeled as attributes. All-in-all, the Relational Model will be the detailed blueprint to read from when the database is created.

Comparison

During the conceptual design period of a database, the Entity-Relationship model is used to paint the idea. This high level design is most useful because of its visual aspect. This will make it easier for anybody else, especially non-technical personal to get an idea of the database. The closer the concept of the database gets to creation, the relational data model is the logical next step. All of the relationships in the Entity-Relationship Model gets made into a relation. These will contain the attributes of each exact piece of data, and each instance of the relations is a tuple. Each tuple will contain the attributes of any needed tuple. The Entity-Relationship Model will be easier to visualize, but the the Relational Data Model will be the closest thing to the final database product.

Conversion from ER Model to Relational Model

When creating a database, it is entirely possible to skip the conversion from an E-R Model to a Relational Model. It is even entirely possible to skip the E-R Model in general. However, this is very bad practice and there is a need for any database designer to follow the flow from a high level conceptualization such as the E-R Model to the lower lever Relational Model and then finally to the database. It is through this process where a designer or any onlooker can fine detail and understand the structure of the database so it can be as perfect as possible.

One of the pertinent conversions is the conversion of strong entity types. One primary key will be used and composite attributes will be broken into simple components. Weak entity types will similarly be dealt with. The biggest difference is that the weak entity will contain the primary key of its owner entity and it will be labeled as a foreign key for use in relations. Although the use of foreign keys is easiest and most popular, there are still other ways of going about this conversion.

- **Merged Relation Approach:** This merges two entity types into one relation so it will include all of the attributes.

- **Cross-Reference Approach:** This creates a relation that will contain the two entity keys as attributes.

As mentioned before, the use of a foreign key is the simplest way. The approach will make use of the 1 side primary key in a 1:1 or 1:M and add it to the other side. Now each relationship instance will be a tuple containing both of the primary keys for both entities. The other approaches can be used, but are a little more demanding on memory. If one wishes to do so, they can multiply the number of records by the memory size to see if memory permits. There is one exception, do to cardinality constraints, binary M:N relationship types must use the Cross-Reference Approach.

Constraints

The constraints that are to be put upon a database are to make sure that the functionality of the database is consistent through all operations and no unexpected anomalies will occur. One of the main reasons constraints are put into place is that so no two tuples will be duplicated. A database can achieve this by use of a primary for each tuple in a relation. This primary key must be unique and it cannot be null. The use of a primary key will come into play when multiple tuples are selected and compared. Similar to a primary key, the database will also use foreign keys. These foreign keys must have the same domain of the primary key in order for the correct tuple to be referred. The foreign key will also be the primary key in a reference relation. All values must fall in line with these constraints and the business rules in order for the application of its use to work as intended.

2.2 ER Database to Relational Database Conversion

2.2 ER Database to Relational Database Conversion Overview

User

UserId:	Domain: Unsigned int 0 to 2³²-1. Must not be NULL.
DateCancelled:	Domain: Datetime. Any range. Can be NULL.
DateCreated:	Domain: Datetime. Any range. Can be NULL.
EmailAddress:	Domain: String. Must contain a valid email format with “@”. Must not be NULL.
Password:	Domain: String. Any value between 6 and 12. Must not be NULL.
UserName:	Domain: String. Any value between 6 and 12. Must not be NULL
Constraint – Primary Keys:	UserID. Must be unique and not NULL.
Constraint – Candidate Keys:	UserID, UserName
Business Rule:	User must fill out non-null able fields as well as be unique. Must pass email verification after signing up via the signup screen.

AttributeSet

AttributeSetId:	Domain: Unsigned int 0 to 2^32-1. Must not be NULL.
DateLastModified:	Domain: Datetime. Any range. Can be NULL.
DateCreated:	Domain: Datetime. Any range. Can be NULL.
CategoryId:	Domain: Unsigned int 0 to 2^32-1. Must not be NULL.
Name:	Domain: String. Any value between 3 and 50. Must not be NULL.
UserId:	Domain: Unsigned int 0 to 2^32-1. Must not be NULL.
IsPublished:	Domain: Bit. Must not be NULL.
Description:	Domain: String. Any value between 3 and 255. Must not be NULL.
Constraint – Primary Keys:	AttributeSetId. Must be unique and not NULL.
Constraint – Candidate Keys:	AttributeSetId.
Business Rule:	All fields are required as this information will be searchable and viewable by the public. The Name must be unique. Must select category to place AttributeSet in as well as (eventually) publish the AttributeSet.

Attribute

AttributeId:	Domain: Unsigned int 0 to 2³²-1. Must not be NULL.
DateLastModified:	Domain: Datetime. Any range. Can be NULL.
AttributeSetId:	Domain: Unsigned int 0 to 2 ³² -1. Must not be NULL.
Name:	Domain: String. Any value between 10 and 50. Must not be NULL.
AttributeTypeId:	Domain: Unsigned int 0 to 2 ³² -1. Must not be NULL.
Description:	Domain: String. Any value between 3 and 255. Must not be NULL.
Constraint – Primary Keys:	AttributeId. Must be unique and not NULL.
Constraint – Candidate Keys:	AttributeTypeId, AttributeId
Business Rule:	No fields may be null. All the user will be able to fill out is the Attribute Name, Attribute Description, and AttributeTypeId.

AttributeType

AttributeTypeId:	Domain: Unsigned int 0 to 2³²-1. Must not be NULL.
DateLastModified:	Domain: Datetime. Any range. Can be NULL.
Name:	Domain: String. Any value between 10 and 50. Must not be NULL.
Constraint – Primary Keys:	AttributeTypeId. Must be unique and not NULL.
Constraint – Candidate Keys:	AttributeTypeId
Business Rule:	These rows are not editable by any user and are standard to GIS data. There will only every be: String, Integer, and Double as values of data.

Category

CategoryId:	Domain: Unsigned int 0 to 2³²-1. Must not be NULL.
DateLastModified:	Domain: Datetime. Any range. Can be NULL.
DateCreated:	Domain: Datetime. Any range. Can be NULL.
Name:	Domain: String. Any value between 10 and 50. Must not be NULL.
Constraint – Primary Keys:	CategoryId. Must be unique and not NULL.
Constraint – Candidate Keys:	CategoryId.
Business Rule:	Categories are not manageable by normal users. Only Admins have the ability to delete and modify this data. .

DownloadTracking

DownloadTrackingId:	Domain: Unsigned int 0 to 2^32-1. Must not be NULL.
DateDownloaded:	Domain: Datetime. Any range. Can be NULL.
AttributeSetId:	Domain: Unsigned int 0 to 2^32-1. Must not be NULL.
UserId:	Domain: Unsigned int 0 to 2^32-1. Must not be NULL.
Constraint – Primary Keys:	DownloadTrackingId. Must be unique and not NULL.
Constraint – Candidate Keys:	DownloadTrackingId, AttributeSetId.
Business Rule:	Generated by the system when a user downloads someone else's AttributeSet. This information will be view statistically by users when viewing categories and actual AttributeSets.

2.3 ER Database to Relational Database Conversion

2.3 ER Database to Relational Database Conversion Data

User

(UserId, UserName, Password, EmailAddress, DateCreated, DateCancelled)

UserId	UserName	Password	EmailAddress	DateCreated	DateCancelled
1	Domiguezr	1337U2	cyclebiff@gmail.com	2011-10-01	NULL
2	Pinkerhook	1234567	chuckyyou@yahoo.com	2011-09-02	2011-10-03
3	Erice	Ifailatlife	pitselee@gmail.com	2011-08-03	NULL
4	Jjwilliams	Ba8y10nsux	jjwilliams@gmail.com	2011-07-02	NULL
5	vanmatreT	9876553	tvanmatre@kern.co.ca.us	2011-10-01	NULL
6	Mystique	Xmen4lfe	mystique@gmail.com	2011-10-07	NULL
7	Sjobs	Applerxz0r	sjobs@apple.com	2011-10-06	2011-10-08
8	Bgates	whyamlh3r3	bgates@microsoft.com	2011-06-05	NULL
9	Ahugandkiss	123123j2	amandah@yahoo.com	2011-10-04	NULL
10	Janiced	1l8vk3j	ldominguez@yahoo.com	2011-10-03	2011-10-04
11	Icanhackthis	Haxz0r245	icanhackthis@gmail.com	2011-05-02	2011-10-04
12	Smellycat	Whatrthe8	komakazeee@apple.com	2011-10-01	2011-10-04
13	Jlo	Iownthewor1d	jlo@shesucks.com	2011-10-02	NULL
14	Britneyspears	Tilltheworldends	bspears@sellout.com	2011-10-03	NULL
15	Amozart	1101110111	amozart@gmail.com	2011-07-07	NULL
16	Schoolrocks	Inmyo23232	schoollyperson@gmail.com	2011-10-08	NULL
17	Chrisatwork	1234623gj1	chrisperson@kern.co.ca.us	2011-03-09	NULL
18	Jayz	Jayzisthes810	jayz@jayz.com	2011-02-02	NULL
19	Theamerican	Wkofit8	Theamerican@yahoo.com	2011-01-01	NULL
20	Puertovallarta	Icantspellit	mexico@rocks.com	2011-08-09	NULL

AttributeSet

(AttributeSetId, Description, DateLastModified, IsPublished, Name, DateCreated, CategoryId, UserId)

Attribute Set Id	Description	Date Last Modified	Is Published	Name	Date Created	Category Id	UserId
1	Building attributes.	2011-10-01	1	CSUB - Buildings	2011-10-01	3	10
2	Plants.	2011-09-02	1	CSUB - Plants	2011-09-02	1	2
3	Walkways.	2011-08-03	1	CSUB - Walkways	2011-08-03	2	5
4	Parking lots.	2011-07-02	1	BC – Parking Lots	2011-07-02	4	5
5	Plants at BC.	2011-10-01	1	BC - Plants	2011-10-01	1	12
6	Buildings at BC.	2011-10-07	0	BC - Buildings	2011-10-07	3	1
7	CSUB walkways and bike paths.	2011-10-06	0	CSUB - Walkways	2011-10-06	2	3
8	Normal Area	2011-06-05	1	Cabin - Buildings	2011-06-05	3	
9	Normal plants	2011-10-04	1	Cabin - Plants	2011-10-04	1	
10	Normal Sightings of rattlers.	2011-10-03	0	Cabin – Snake Sightings	2011-10-03	5	

Attribute

(AttributeId, Description, DateLastModified, Name, AttributeSetId, AttributeTypeId)

Attribute Id	Description	Name	DateLastModified	AttributeSetId	AttributeTypeId
1	No. Of Floors	NoOfFloors	2011-10-01	1	1
2	No. Of Rooms	NoOfRooms	2011-09-02	1	1
3	Total Square Footage	SqFtTotal	2011-08-03	1	2
4	Building Number	BldgNo	2011-07-02	1	1
5	Date Building Opened its doors.	DateOpened	2011-10-01	1	3
6	Species of plant.	Species	2011-10-07	2	3
7	Date planted.	DatePlanted	2011-10-06	2	3
8	Watering frequency (days).	WaterFreq	2011-06-05	2	1
9	Length of path.	LengthOfPath	2011-10-04	3	2
10	Type of pathway material	TypeOfMaterial	2011-10-03	3	3
11	Date installed / set.	DateSet	2011-05-02	3	3
12	Name of path.	Name	2011-10-01	3	3
13	Name of school	LotSchool	2011-10-02	4	3
14	Name of lot.	LotName	2011-10-03	4	3
15	Number of lot.	LotNumber	2011-07-07	4	1
16	Number of cars fittable.	LotCapacity	2011-10-08	4	1
17	Date lot opened.	DateOpened	2011-03-09	4	3
18	Date planted.	DatePlanted	2011-02-02	5	3
19	Plant species.	Species	2011-01-01	5	3
20	Watering Freq.	WaterFreq	2011-08-09	5	1

AttributeType

(AttributeTypeId, DateLastModified, Name)

AttributeTypeId	DateLastModified	Name
1	2010-01-01	Integer
2	2010-01-01	Double
3	2010-01-01	String

Category

(CategoryId, Name, DateCreated, DateLastModified)

CategoryId	Name	DateCreated	DateLastModified
1	Flora	2011-10-01	2011-10-01
2	Walkways and Paths	2011-09-02	2011-09-02
3	Buildings	2011-08-03	2011-08-03
4	Parking Lots	2011-07-02	2011-07-02
5	Fauna	2011-10-01	2011-10-01

DownloadTracking

(DownloadTrackingId, DateDownloaded, UserId, AttributeSetId)

DownloadTrackingId	DateDownloaded	UserId	AttributeSetId
1	2011-10-01	1	1
2	2011-09-02	1	2
3	2011-08-03	3	3
4	2011-07-02	2	4
5	2011-10-01	10	5
6	2011-10-07	20	4
7	2011-10-06	17	3
8	2011-06-05	1	2
9	2011-10-04	5	1
10	2011-10-03	5	3

2.4 Queries

2.4 Queries Used

1. Select Users that were created after 2011-10-09.
2. Select Users that have downloaded the highest number of AttributeSets.
3. Select Users that have never downloaded any AttributeSets.
4. Select Users that have more than one AttributeSets.
5. Select Categories that have the highest number of AttributeSets.
6. Select Categories that have no Attributes.
7. Select Attribute that has the largest number of Attributes.
8. Select AttributeSets that are not published.
9. Select AttributeSets for a specific Category with Name = "Flora".
10. Select AttributeSet that has a name like "CSUB".
11. Select AttributeSets for User where UserName = "dominguezr"

2.5 Query Representation

2.5 Query Representation

Where “a” is Relational Algebra and “b” is domain calculus.

1. Select Users that were created after 2011-10-09
 - a. $\sigma_{\text{DateCreated} = 2011-10-09}(\text{User})$
 - b. $\{u.\text{UserName} \mid \text{User}(u) \wedge u.\text{DateCreated} = 2011-10-09\}$
2. Select Users that have downloaded the highest number of AttributeSets
 - a. $\text{User} * (\mathcal{F}_{\max(\text{No_Of_Downloads})}(\text{UserId}, \text{No_Of_Downloads} \mathcal{F}_{\text{count}(*)} \text{DownloadTracking}))$
 - b. $\{u.\text{UserName} \mid \text{User}(u) \wedge \text{AttributeSets}(s1) \wedge \text{AttributeSets}(s2) \wedge s1.\text{UserId}_1, \dots, s1.\text{UserId}_n \wedge s2.\text{UserId}_1, \dots, s2.\text{UserId}_m \wedge m > n \wedge s1.\text{UserId} \neq s2.\text{UserId} \wedge s1.\text{UserId} = u.\text{UserId}\}$
3. Select Users that have never downloaded any AttributeSets
 - a. $\rho_{\text{UserName}}((\text{User} \bowtie \text{DownloadTracking}) - (\text{User} * \text{DownloadTracking}))$
 - b. $\{u.\text{UserName} \mid \text{User}(u) \wedge \neg (\exists t)(\text{DownloadTracking}(t) \wedge t.\text{DownloadTrackingId} = u.\text{DownloadTrackingId})\}$
4. Select Users that have more than one AttributeSets
 - a. $\rho_{u.\text{UserName}}(\text{User}(u) * (\text{DownloadTracking } d1 \bowtie d1.\text{UserId} = d2.\text{UserId} \text{ AND } d1.\text{DownloadTrackingId} \neq d2.\text{DownloadTrackingId} \text{ DownloadTracking } d2))$
 - b. $\{u.\text{UserName} \mid \text{User}(u) \wedge \text{DownloadTracking}(t1) \wedge \text{DownloadTracking}(t2) \wedge t1.\text{UserId} = t2.\text{UserId} \wedge d1.\text{DownloadTrackingId} \neq d2.\text{DownloadTrackingId}\}$
5. Select Categories that have the highest number of AttributeSets
 - a. $\rho_{c.\text{Name}}(\text{Categories}(c) * (\mathcal{F}_{\max(\text{No_Of_Sets})}(\text{CategoryId}, \text{No_Of_Sets} \mathcal{F}_{\text{count}(*)} \text{AttributeSets}))))$
 - b. $\{c.\text{Name} \mid \text{Categories}(c) \wedge \text{AttributeSets}(s1) \wedge \text{AttributeSets}(s2) \wedge s1.\text{CategoryId}_1, \dots, s1.\text{CategoryId}_n \wedge s2.\text{CategoryId}_1, \dots, s2.\text{CategoryId}_m \wedge m > n \wedge s1.\text{CategoryId} \neq s2.\text{CategoryId} \wedge s1.\text{CategoryId} = c.\text{CategoryId}\}$
6. Select Categories that have no Attributes
 - a. $\rho_{c.\text{Name}}(\text{Category}(c) * ((\text{AttributeSet} \bowtie \text{Attribute}) - (\text{AttributeSet} * \text{Attribute})))$
 - b. $\{c.\text{Name} \mid \text{Category}(c) \wedge \neg (\exists s)(\text{AttributeSets}(s) \wedge c.\text{CategoryId} = s.\text{CategoryId})\}$
7. Select AttributeSet that has the largest number of Attributes
 - a. $\rho_{s.\text{Name}}(\text{AttributeSet}(s) * (\mathcal{F}_{\max(\text{No_Of_Attributes})}(\text{AttributeSetId}, \text{No_Of_Attributes} \mathcal{F}_{\text{count}(*)} \text{Attribute})))$
 - b. $\{s.\text{Name} \mid \text{AttributeSets}(s) \wedge \text{Attribute}(a1) \wedge \text{Attribute}(a2) \wedge a1.\text{AttributeId}_1, \dots, a1.\text{AttributeId}_n \wedge a2.\text{AttributeId}_1, \dots, a2.\text{AttributeId}_m \wedge m > n \wedge a1.\text{AttributeId} \neq a2.\text{AttributeId} \wedge a1.\text{AttributeId} = s.\text{AttributeId}\}$
8. Select AttributeSets that are not published
 - a. $\rho_{s.\text{Name}}(\sigma_{\text{IsPublished} = \text{no}}(\text{AttributeSet}(s)))$

- b. $\{s.Name \mid \text{AttributeSet}(s) \wedge s.IsPublished = \text{no}\}$
9. Select AttributeSets for a specific Category with Name = "Flora"
- $\rho_{s.Name}(\text{AttributeSet}(s) * (\sigma_{c.Name = \text{"Flora"}}(\text{Category}(c))))$
 - $\{s.Name \mid \text{AttributeSet}(s) \wedge \text{Category}(c) \wedge c.Name = \text{"Flora"} \wedge s.CategoryId = c.CategoryId\}$
10. Select AttributeSet that has a name like "CSUB Buildings"
- $\Sigma_{s.Name = \text{"CSUB Buildings"}}(\text{AttributeSet}(s))$
 - $\{s \mid \text{AttributeSet}(s) \wedge s.Name = \text{"CSUB Buildings"}\}$
11. Select AttributeSets for User where UserName = "Buffy"
- $\rho_{s.Name}(\text{AttributeSet}(s) * (\sigma_{u.UserName = \text{"Buffy"}}(\text{User}(u))))$
 - $\{s.Name \mid \text{AttributeSet}(s) \wedge \text{User}(u) \wedge u.UserName = \text{"Buffy"}\}$

Phase III: Implementation of the Relational Database

3.0 Relation Normalization

3.0 Relation Normalization

When a table is in 1NF:

A table is in 1NF when all the key attributes are defined (no repeating groups in the table) and when all remaining attributes are dependent on the primary key. However, a table in 1NF still may contain partial dependencies, i.e., dependencies based on only part of the primary key and/or transitive dependencies that are based on a non-key attribute.

When a table is in 2NF:

A table is in 2NF when it is in 1NF and it includes no partial dependencies. However, a table in 2NF may still have transitive dependencies, i.e., dependencies based on attributes that are not part of the primary key.

When a table is in 3NF:

A table is in 3NF when it is in 2NF and it contains no transitive dependencies.

When a table is in BCNF:

A table is in Boyce-Codd Normal Form (BCNF) when it is in 3NF and every determinant in the table is a candidate key. For example, if the table is in 3NF and it contains a nonprime attribute that determines a prime attribute, the BCNF requirements are not met. (Reference the text's Figure 6.8 to support this discussion.) This description clearly yields the following conclusions:

- If a table is in 3NF and it contains only one candidate key, 3NF and BCNF are equivalent.
- BCNF can be violated only if the table contains more than one candidate key. Putting it another way, there is no way that the BCNF requirement can be violated if there is only one candidate key.

3.1 SQL with SQL*PLUS

3.1 About SQL*PLUS

When interacting with a database management system, the language of choice is Structured Query Language – SQL for short. The generalized form of SQL is a quick and efficient language to know that can be used with Oracle, SQL, MySQL, PostgreSQL, etc. The more popular DBMS', Oracle and Microsoft SQL Server, utilize the language to interface with the database. Microsoft SQL uses Enterprise Management Studio, while Oracle uses SQL*PLUS to execute SQL scripts/run queries. This is, albeit dangerous, an effective way to manage the DBMS and related databases.

3.2 Oracle / Schema Objects

A schema is a logical set of logical data structures. They are stored within an oracle table space, which can exist within one or more physical files (data files). Utilizing mainly Table objects, oracle has the following data structures available:

3.2 Oracle Schema Objects

Tables

The most basic schema object is the table. Referring back to relational algebra/calculus, tuples make up rows, while attributes make up columns. An attribute/column consists of a unique identifier, data type, and a width/size. After creating a table's columns and attributes, you can apply constraints to limit acceptable values that certain columns may contain.

Indexes

Indexes, while optional, are EXTREMELY useful in shortening access time to frequently accessed tables. A database index stores associations between columns based on specified logical indexing schemes. Indexes are best used on tables where you frequently retrieve data, and not tables that are used for transactional purposes (ie, tables that have data frequently inserted).

Stored Procedures / Functions

Functions are much like C++ functions; they accept arguments and return scalar values. Like functions, stored procedures also accept arguments, however, they can return actual result sets (much like a SQL SELECT statement can).

Synonyms

Oracle uses synonyms as aliases for other schema objects. A synonym itself is a schema object so you can create synonyms of synonyms. Like views, only a synonym's definition is stored in the database so very little space is used. Synonyms are useful when you are trying to hide certain schema objects or need a shorter (or longer) representation of said object.

Views

Views are customizable structures of data contained in tables or other views. They can be made very granular. You can interact with a view as you would a Table, but certain limitations may apply. Only the view's definition is stored in the DBMS, which is efficient in terms of disk space and database storage.

Dimensions

A dimension is a hierarchical relationship between columns. This relationship does not have data-storage associated with it.

Database Linking

Database links are read-only links that allow users to see data, but not manipulate data on another server. They are constants that do not change and are best used to quickly see data on another DBMS without needing to authenticate as a user in that database.

Sequence Generation

Sequence generators allow efficient throughput when multiple users are utilizing the database. It works by avoiding unnecessary serialization when two transactions are accessing the same data at the same time.

Packages

A package groups logically related schema types, items and subprograms. They contain a specification, declares data types, variables, subprograms, and exceptions. Packages also, although sometimes unnecessary, contain a body – which defines cursors, subprograms, and an implementation of the specification.

3.3 Instances and Relation Schemas

3.3 Instances and Relation Schemas

Our project's tables were created using the following relation as an example (Our Users table):

```
CREATE TABLE RDEEJW_USERS
(
  USERID NUMBER NOT NULL
, DATECANCELLED DATE
, DATECREATED DATE NOT NULL
, EMAILADDRESS VARCHAR2(50) NOT NULL
, USERNAME VARCHAR2(25) NOT NULL
, PASSWORD VARCHAR2(50) NOT NULL
, CONSTRAINT RDEEJW_USERS_PK PRIMARY KEY
  (
    USERID
  )
  using index TABLESPACE cs342index
  ENABLE
);

COMMIT;

ALTER TABLE RDEEJW_USERS
ADD CONSTRAINT RDEEJW_USERS_UK1 UNIQUE
(
  EMAILADDRESS
, USERNAME
)
  using index TABLESPACE cs342index
  ENABLE;

COMMIT;
```

Our Project's relation names were in the format RDEEJW_RelationName. Below you will find a description and select * from for each relation.

RDEEJW_USERS:

```
DESC RDEEJW_USERS
Name          Null      Type
-----
USERID        NOT NULL  NUMBER
DATECANCELLED          DATE
DATECREATED   NOT NULL  DATE
EMAILADDRESS  NOT NULL  VARCHAR2(50)
USERNAME      NOT NULL  VARCHAR2(25)
PASSWORD      NOT NULL  VARCHAR2(50)
```

USERID	DATECANCELLED	DATECREATED	EMAILADDRESS	USERNAME	PASSWORD
2		08-NOV-11	cyclebiff@gmail.com	user1	12345
3		08-NOV-11	user2@gmail.com	user2	12345
4		08-NOV-11	user3@gmail.com	user3	12345
5		08-NOV-11	user4@gmail.com	user4	12345
6		08-NOV-11	user5@gmail.com	user5	12345
7		08-NOV-11	user6@gmail.com	user6	12345
8		08-NOV-11	user7@gmail.com	user7	12345
9		08-NOV-11	user8@gmail.com	user8	12345
10		08-NOV-11	user9@gmail.com	user9	12345
11		08-NOV-11	user10@gmail.com	user10	12345
12		08-NOV-11	user11@gmail.com	user11	12345
13		08-NOV-11	user12@gmail.com	user12	12345
14		08-NOV-11	user13@gmail.com	user13	12345
15		08-NOV-11	user14@gmail.com	user14	12345
16		08-NOV-11	user15@gmail.com	user15	12345
17		08-NOV-11	user16@gmail.com	user16	12345
18		08-NOV-11	user17@gmail.com	user17	12345
19		08-NOV-11	user18@gmail.com	user18	12345
20		08-NOV-11	user19@gmail.com	user19	12345
21		08-NOV-11	user20@gmail.com	user20	12345

RDEEJW_CATEGORIES:

DESC	RDEEJW_CATEGORIES		
Name	Null	Type	
CATEGORYID	NOT NULL	NUMBER	
DATECREATED		DATE	
DATELASTMODIFIED		DATE	
NAME	NOT NULL	VARCHAR2(50)	

CATEGORYID	DATECREATED	DATELASTMODIFIED	NAME
3	08-NOV-11	08-NOV-11	Flora
4	08-NOV-11	08-NOV-11	Walkways and Paths
5	08-NOV-11	08-NOV-11	Buildings
6	08-NOV-11	08-NOV-11	Parking Lots
7	08-NOV-11	08-NOV-11	Fauna

RDEEJW_ATTRIBUTETYPES:

DESC RDEEJW_ATTRIBUTETYPES

Name	Null	Type
ATTRIBUTETYPEID	NOT NULL	NUMBER
DATELASTMODIFIED	NOT NULL	DATE
NAME	NOT NULL	VARCHAR2(50)

ATTRIBUTETYPEID	DATELASTMODIFIED	NAME
2	08-NOV-11	Integer
3	08-NOV-11	Double
4	08-NOV-11	String

RDEEJW_ATTRIBUTESETS:

DESC RDEEJW_ATTRIBUTESETS

Name	Null	Type
ATTRIBUTESETID	NOT NULL	NUMBER
DESCRIPTION	NOT NULL	VARCHAR2(50)
DATELASTMODIFIED	NOT NULL	DATE
ISPUBLISHED	NOT NULL	CHAR(1)
NAME	NOT NULL	VARCHAR2(50)
DATECREATED	NOT NULL	DATE
CATEGORYID	NOT NULL	NUMBER
USERID	NOT NULL	NUMBER

ATTRIBUTESETID	DESCRIPTION	DATELASTMODIFIED
2	Building Attributes.	08-NOV-11
3	Plants.	08-NOV-11
4	Walkways.	08-NOV-11
5	Parking Lots.	08-NOV-11
6	Plants at BC.	08-NOV-11
7	Buildings at BC.	08-NOV-11
8	CSUB Walkways and bike paths.	08-NOV-11
9	Normal Area.	08-NOV-11
10	Normal Plants.	08-NOV-11
11	Normal sightings of rattlers	08-NOV-11

10 rows selected

ATTRIBUTESETID	ISPUBLISHED	NAME	DATECREATED	CATEGORYID	USERID
2	1	CSUB - Buildings	08-NOV-11	5	11
3	1	CSUB - Plants	08-NOV-11	3	3
4	1	CSUB - Walkways	08-NOV-11	4	6
5	1	BC - Parking Lots	08-NOV-11	6	6
6	1	BC - Plants	08-NOV-11	3	13
7	0	BC - Buildings	08-NOV-11	5	2
8	0	CSUB - Walkways	08-NOV-11	4	4
9	1	Cabin - Buildings	08-NOV-11	5	8
10	1	Cabin - Plants	08-NOV-11	3	15
11	0	Cabin - Snake Sightings	08-NOV-11	7	16

10 rows selected

RDEEJW_ATTRIBUTES:

```
DESC RDEEJW_ATTRIBUTES
Name          Null      Type
-----
ATTRIBUTEID   NOT NULL  NUMBER
DESCRIPTION   NOT NULL  VARCHAR2(50)
NAME          NOT NULL  VARCHAR2(50)
DATELASTMODIFIED NOT NULL  DATE
ATTRIBUTESETID NOT NULL  NUMBER
ATTRIBUTETYPEID NOT NULL  NUMBER
```

```
ATTRIBUTEID DESCRIPTION NAME
-----
1 No. Of Floors NoOfFloors
2 No. Of Rooms NoOfRooms
3 Total Square Footage SqFtTotal
4 Building Number BldgNo
5 Date Building Opened DateOpened
6 Species of Plant. Species
7 Date Planted DatePlanted
8 Watering freq (days) WaterFreq
9 Length Of Path LengthOfPath
10 Type Of Pathway Material TypeOfMaterial
11 Date installed / set DateSet
12 Name of path. Name
13 Name of school LotSchool
14 Name of Lot LotName
15 Number Of Lot LotNumber
16 Number of cars max LotCapacity
17 Date lot opened DateOpened
18 Date Planted DatePlanted
19 Plant Species Species
20 Watering Freq. WaterFreq
21 Age Age
```

21 rows selected

```
ATTRIBUTEID DATELASTMODIFIED ATTRIBUTESETID ATTRIBUTETYPEID
-----
1 08-NOV-11 2 2
2 08-NOV-11 2 2
3 08-NOV-11 2 3
4 08-NOV-11 2 2
5 08-NOV-11 2 4
6 08-NOV-11 3 4
7 08-NOV-11 3 4
8 08-NOV-11 3 2
9 08-NOV-11 4 3
10 08-NOV-11 4 4
11 08-NOV-11 4 4
12 08-NOV-11 4 4
13 08-NOV-11 5 4
14 08-NOV-11 5 4
15 08-NOV-11 5 2
16 08-NOV-11 5 2
17 08-NOV-11 5 4
18 08-NOV-11 6 4
19 08-NOV-11 6 4
20 08-NOV-11 6 2
21 08-NOV-11 6 2
```

21 rows selected

RDEEJW_DOWNLOADTRACKINGS:

DESC RDEEJW_DOWNLOADTRACKINGS

Name	Null	Type
DOWNLOADTRACKINGID	NOT NULL	NUMBER
DATEDOWNLOADED	NOT NULL	DATE
ATTRIBUTESETID	NOT NULL	NUMBER
USERID	NOT NULL	NUMBER

DOWNLOADTRACKINGID	DATEDOWNLOADED	ATTRIBUTESETID	USERID
1	08-NOV-11	2	2
2	08-NOV-11	3	2
3	08-NOV-11	4	4
4	08-NOV-11	5	3
5	08-NOV-11	6	11
6	08-NOV-11	5	21
7	08-NOV-11	4	18
8	08-NOV-11	3	2
9	08-NOV-11	2	6
10	08-NOV-11	4	6

10 rows selected

3.4 SQL Queries

3.4 SQL Queries

1. Select Users that were created after 2011-10-09

```
-- Select Users created after 2011-10-09
select u.USERID, u.USERNAME, u.DATECREATED from RDEEJW_Users u
WHERE u.DateCreated >= '09-OCT-11';
```

USERID	USERNAME	DATECREATED
2	user1	08-NOV-11
3	user2	08-NOV-11
4	user3	08-NOV-11
5	user4	08-NOV-11
6	user5	08-NOV-11
7	user6	08-NOV-11
8	user7	08-NOV-11
9	user8	08-NOV-11
10	user9	08-NOV-11
11	user10	08-NOV-11
12	user11	08-NOV-11
13	user12	08-NOV-11
14	user13	08-NOV-11
15	user14	08-NOV-11
16	user15	08-NOV-11
17	user16	08-NOV-11
18	user17	08-NOV-11
19	user18	08-NOV-11
20	user19	08-NOV-11
21	user20	08-NOV-11

20 rows selected

2. Select Users That have never downloaded any AttributeSets

```
-- Select Users that have never downloaded any AttributeSets
select U.UserId, U.UserName
FROM RDEEJW_Users U
WHERE NOT EXISTS (
  SELECT *
  FROM RDEEJW_DownloadTrackings D
  WHERE U.UserId = D.UserId
);
```

USERID	USERNAME
5	user4
8	user7
15	user14
12	user11
19	user18
10	user9
17	user16
7	user6
14	user13
16	user15
9	user8
13	user12
20	user19

13 rows selected

3. Select Users that have more than one AttributeSet

```
-- Select Users that have more than one AttributeSet
SELECT U.UserId, U.UserName, count(*) as "Number Of AttributeSets"
FROM RDEEJW_Users U inner join RDEEJW_AttributeSets A1 ON U.UserId = A1.UserId
GROUP BY U.UserId, U.UserName
HAVING count(*) > 1;
```

USERID	USERNAME	Number Of AttributeSets
6	user5	2

4. Select Categories that have the highest number of AttributeSets

```
-- Select Categories that have the highest number of AttributeSets
SELECT Data.CategoryName, count(Data.CategoryName) count
FROM
(
  SELECT A.Name, C.Name as CategoryName, A.CategoryId as CategoryId
  FROM RDEEJW_AttributeSets A INNER JOIN RDEEJW_Categories C on A.CategoryId = C.CategoryId
) Data
GROUP BY Data.CategoryName
ORDER BY count DESC;
```

CATEGORYNAME	COUNT
Flora	3
Buildings	3
Walkways and Paths	2
Fauna	1
Parking Lots	1

5. Select Categories that have no AttributeSets

```
-- Select Categories that have no AttributeSets
SELECT C.CategoryId, C.Name
FROM RDEEJW_Categories C
WHERE NOT EXISTS (
  SELECT *
  FROM RDEEJW_AttributeSets A
  WHERE C.CategoryId = A.CategoryId
);
```

CATEGORYID	NAME
8	An Unused Category

6. Select AttributeSet that has the largest number of Attributes

```
-- Select AttributeSets that have the largest number of Attributes
SELECT Data.AttributeSetName, count(Data.AttributeSetName) count
FROM
(
  SELECT A1.Name, A2.Name as AttributeSetName, A1.AttributeSetId as AttributeSetId
  FROM RDEEJW_Attributes A1 INNER JOIN RDEEJW_AttributeSets A2 on A1.AttributeSetId = A2.AttributeSetId
) Data
GROUP BY Data.AttributeSetName
ORDER BY count DESC;
```

ATTRIBUTESETNAME	COUNT
BC - Parking Lots	5
CSUB - Buildings	5
CSUB - Walkways	4
BC - Plants	4
CSUB - Plants	3

7. Select AttributeSets that are not published

```
-- AttributeSets that are not published
SELECT A.AttributeSetId, A.Name, A.IsPublished
FROM RDEEJW_AttributeSets A
WHERE A.IsPublished <> '1';
```

ATTRIBUTESETID	NAME	ISPUBLISHED
7	BC - Buildings	0
8	CSUB - Walkways	0
11	Cabin - Snake Sightings	0

8. Select AttributeSets for a specific Category with Name = "Flora"

```
-- AttributeSets for a specific Category with Category Name = "Flora"
SELECT A.AttributeSetId, A.Name as "Attribute Set Name", C.Name as "Category Name"
FROM RDEEJW_AttributeSets A INNER JOIN RDEEJW_Categories C on A.CategoryId = C.CategoryId
WHERE C.Name = 'Flora';
```

ATTRIBUTESETID	Attribute Set Name	Category Name
3	CSUB - Plants	Flora
6	BC - Plants	Flora
10	Cabin - Plants	Flora

9. Select AttributeSet that has a name like "CSUB Buildings"

```
-- AttributeSets that have a name like "CSUB Buildings"
SELECT A.AttributeSetId, A.Name as "Attribute Set Name"
FROM RDEEJW_AttributeSets A
WHERE A.Name like '%CSUB%Buildings%';
```

ATTRIBUTESETID	Attribute Set Name
2	CSUB - Buildings

10. Select AttributeSets for User where UserName = "Buffy"

```
-- AttributeSets for Users Where UserName = "Buffy"
SELECT A.AttributeSetId, A.Name as "Attribute Set Name", U.UserName as "Owned By User"
FROM RDEEJW_AttributeSets A INNER JOIN RDEEJW_Users U on A.UserId = U.UserId
WHERE U.UserName = 'Buffy';
```

ATTRIBUTESETID	Attribute Set Name	Owned By User
7	BC - Buildings	Buffy

Phase IV: Implementation of the Relational Database (Continued)

4.1 Common Features in Oracle PL/SQL and MS Trans-SQL

4.1 Common Features in Oracle PL/SQL and MS Trans-SQL

Components which consist of PL/SQL and MS Trans-SQL

PL/SQL and MS Trans-SQL, while syntactically different, share many of the same features as procedural languages. Both refer to their schema objects with the same keywords: procedures, functions, tables, and triggers. Also, both are used in stored procedures and subprograms. Both extend the functionality of SQL itself by creating re-usable code and using additional functionality.

Purposes of Stored subprograms

A stored subprogram's purpose is to efficiently run/execute a query without needing to create the entire query for each execution. Much like a function in object-oriented function/method, a stored subprogram accepts arguments (parameters), which it utilizes along with an actual query. Such queries can be INSERT, UPDATE, or DELETE.

Also similar to object-oriented methods, it hides the implementation from the user. This means that the user can see the name of the subprogram, its parameters, and return types but not the actual query being used.

Benefits of subprograms over Dynamic SQL

One advantage of a subprogram over Dynamic SQL is that it utilizes the server's resources by building the actual SQL query. The advantage here is that the user doesn't need to know the tables and columns needed or waste their own resources creating the query.

Another advantage to subprograms over Dynamic SQL is that subprograms promote reusability and maintainability. By re-using subprograms you don't need to re-create the wheel every time you need to run a specific function or procedure. Also, it's easier to maintain code at the DBMS level if you're the DBA.

4.2 Oracle PL/SQL

4.2 Oracle PL/SQL

To the casual C++ developer, the basic structure of a PL/SQL program can be generalized into the following three sections:

1. **Declaration**, keywords: *“AS” or “IS”*, This part of the program follows any arguments the program may take, as well as declare cursors, local variables, or user defined expressions.
2. **Execution**, keywords: *“BEGIN” and “END”*, This part of the program contains queries, logical operators, and utilizes the variables and arguments defined in the declaration section.
3. **Exception**, keyword *“EXCEPTION”*, This part of the program allows the user to handle any errors that may arise in the event of an exception.

General Format of a PL/SQL Program:

Referring to the previous section, the general format of a PL/SQL Program is as follows:

```
<TYPE> <Name> IS
BEGIN -- executable part starts here
    [EXCEPTION]
END;
```

Control Statements:

PL/SQL includes control statements that can theoretically be used to program/handle any situation. The following control statements can be found in most other languages (such as C++, C#, Java, Objective-C, etc). Below is the syntax of PL/SQL control statements:

```
IF    --true/false condition
THEN  -- statement
ELSEIF -- true/false condition
THEN  -- statement
ENDIF;

[<<label_name>>]
CASE selector
    WHEN expression1 THEN sequence_of_statements1;
    WHEN expression2 THEN sequence_of_statements2;
    ...
    WHEN expressionN THEN sequence_of_statementsN;
    [ELSE sequence_of_statementsN+1;]
END CASE [label_name];

LOOP
```

```

EXIT WHEN -- can be used similar to a "break" command
END LOOP;

WHILE condition LOOP
    sequence_of_statements
END LOOP;

FOR counter IN [REVERSE] lower_bound..higher_bound LOOP
    sequence_of_statements
END LOOP;

```

Stored Procedures:

Stored procedures allow the execution of commands that aid in the usage of DELETE, UPDATE, SELECT and INSERT SQL statements. They are excellent for extending out the functionality of SQL. The syntax is as follows:

```

CREATE [OR REPLACE] PROCEDURE <NAME>

[parametername] [datatype]

IS

    -- Declare constants and variables in this section.
    -- Example: <Variable Identifier> <DATATYPE>
    --           <Variable Identifier> CONSTANT <DATATYPE>
    --           varEname VARCHAR2(40);
    --           varComm REAL;
    --           varSalary CONSTANT NUMBER:=1000;
    --           comm_missing EXCEPTION;

BEGIN -- executable part starts here

    -- Write PL/SQL and SQL statements to implement the processing logic
    -- of subprogram. Example:
    --     SELECT ENAME,
    --           COMM
    --     INTO   varEname,
    --           varComm
    --     FROM   EMP
    --     WHERE  EMPNO = 7369;
    --
    --     IF varComm IS NULL THEN
    --         RAISE comm_missing;
    --     END IF;

    [EXCEPTION] -- exception-handling part starts here
    -- WHEN comm_missing THEN
    --     dbms_output.put_line('Commision is NULL');

END;

```

Stored Functions:

Stored functions act identically to stored procedures. One major difference is that stored functions will always return scalar variables, which are returned at the end of a function.

```
CREATE [OR REPLACE] FUNCTION <NAME> (  
  [parametername] IN [datatype]) RETURN [datatype] IS  
  
  -- Declare constants and variables in this section.  
  -- Example: <Variable Identifier> <DATATYPE>  
  --           <Variable Identifier> CONSTANT <DATATYPE>  
  --           varEname VARCHAR2(40);  
  --           varComm REAL;  
  --           varSalary CONSTANT NUMBER:=1000;  
  --           comm_missing EXCEPTION;  
  
BEGIN -- executable part starts here  
  
  -- Write PL/SQL and SQL statements to implement the processing logic  
  -- of subprogram. Example:  
  --   SELECT ENAME,  
  --         COMM  
  --   INTO   varEname,  
  --         varComm  
  --   FROM   EMP  
  --   WHERE  EMPNO = 7369;  
  --  
  --   IF varComm IS NULL THEN  
  --     RAISE comm_missing;  
  --   END IF;  
  
  RETURN <returnvalue>;  
  
  -- EXCEPTION -- exception-handling part starts here  
  -- WHEN comm_missing THEN  
  --   dbms_output.put_line('Commision is NULL');  
  
END;
```

Triggers:

Triggers are extremely helpful schema objects that run when “triggered” by an event that occurs on specific tables. There are many ways this can be useful, such as determining what to do just after an insert has occurred on a table. We can also validate data by calling a function or stored procedure.

```
CREATE [OR REPLACE] TRIGGER <Name>  
  <BEFORE,AFTER> <INSERT,[OR]UPDATE,[OR]DELETE> ON <tablename>  
  FOR EACH ROW  
  [WHEN] [condition]  
  DECLARE  
    [variable(s)]  
  BEGIN  
    <code>  
  END;
```

Cursors:

A cursor can be thought of like pointers in C++ or Objective-C – they point to rows in a SQL SELECT statement.

```
CURSOR cursor_name (parameter_list)
IS
SELECT_statement;
```

Packages:

A package groups together similar procedures, functions, variables tables, etc. As with procedures and functions, re-use of code is encouraged using OOP techniques, such as function/method overloading.

```
CREATE [OR REPLACE] PACKAGE package_name
[AUTHID {CURRENT_USER | DEFINER}]
{IS | AS}
[PRAGMA SERIALLY_REUSABLE;]
[collection_type_definition ...]
[record_type_definition ...]
[subtype_definition ...]
[collection_declaration ...]
[constant_declaration ...]
[exception_declaration ...]
[object_declaration ...]
[record_declaration ...]
[variable_declaration ...]
[cursor_spec ...]
[function_spec ...]
[procedure_spec ...]
[call_spec ...]
[PRAGMA RESTRICT_REFERENCES(assertions) ...]
END [package_name];

[CREATE [OR REPLACE] PACKAGE BODY package_name {IS | AS}
[PRAGMA SERIALLY_REUSABLE;]
[collection_type_definition ...]
[record_type_definition ...]
[subtype_definition ...]
[collection_declaration ...]
[constant_declaration ...]
[exception_declaration ...]
[object_declaration ...]
[record_declaration ...]
[variable_declaration ...]
[cursor_body ...]
[function_spec ...]
[procedure_spec ...]
[call_spec ...]
[BEGIN
sequence_of_statements]
END [package_name];]
```


4.3 Oracle PL/SQL Sub-Programs

4.3 Oracle PL/SQL Sub-Programs

Below are our stored procedures, stored functions, and triggers used in our project. We have placed additional convenience PL/SQL subprograms on the DELPHI server, which can be viewed on the server.

Stored Procedures

RDEEJW_INSERTUSER

This stored procedure takes are required fields needed to create a user record. The USERID is generated at insertion time, as the trigger TRIGGER_RDEEJW_USERS creates the id from sequence SEQ_RDEEJW_USERS. The stored procedure also utilizes the stored function RDEEJW_ISUSERUNIQUE to determine whether to insert the record or not.

```
-- PROCEDURE RDEEJW_INSERTUSER
CREATE OR REPLACE PROCEDURE RDEEJW_INSERTUSER
(
  ARG_USERNAME IN VARCHAR2
, ARG_PASSWORD IN VARCHAR2
, ARG_DATECREATED IN DATE
, ARG_EMAILADDRESS IN VARCHAR2
) AS
  USERCOUNT NUMBER;

BEGIN
  USERCOUNT := RDEEJW_ISUSERUNIQUE(ARG_USERNAME, ARG_EMAILADDRESS);

  IF USERCOUNT = 0 THEN
    INSERT INTO RDEEJW_USERS
    (USERNAME, PASSWORD, EMAILADDRESS, DATECREATED)
    VALUES
    (ARG_USERNAME, ARG_PASSWORD, ARG_EMAILADDRESS, ARG_DATECREATED);
    COMMIT;
  END IF;

END RDEEJW_INSERTUSER;
```

RDEEJW_DELETEATTRIBUTESET

This stored procedure takes the primary key of the RDEEJW_ATTRIBUTESETS table (ATTRIBUTESETID). Note that the table RDEEJW_ATTRIBUTES has a foreign key constraint on the ATTRIBUTESETID and is set to CASCADE delete. In short, this helps us delete all child RDEEJW_ATTRIBUTES for a given record in RDEEJW_ATTRIBUTESETS

```
|- PROCEDURE RDEEJW_DELETEATTRIBUTESET  
CREATE OR REPLACE PROCEDURE RDEEJW_DELETEATTRIBUTESET  
(  
  ARG_ATTRIBUTESETID IN NUMBER  
) AS  
  
BEGIN  
  
  DELETE FROM RDEEJW_ATTRIBUTESETS  
  WHERE ATTRIBUTESETID = ARG_ATTRIBUTESETID;  
  
  COMMIT;  
  
END RDEEJW_DELETEATTRIBUTESET;
```

Stored Functions

RDEEJW_AVGATTRFORALLSETS

This function accepts a CategoryID as an argument and returns the average number of attributes for all sets within that category. This is useful for reporting purposes of the admins of the site by providing insight to lower-attribute sets.

```
-- Function RDEEJW_AVGATTRFORCATEGORY
CREATE OR REPLACE FUNCTION RDEEJW_AVGATTRFORCATEGORY
(
  ARG_CATEGORYID IN NUMBER
) RETURN VARCHAR2 AS

  FINALAVERAGE NUMBER;

BEGIN

  SELECT AVG(COUNT)
  INTO FINALAVERAGE
  FROM
  (
    SELECT COUNT(*) AS COUNT, A1.ATTRIBUTESETID
    FROM RDEEJW_ATTRIBUTES A1 INNER JOIN RDEEJW_ATTRIBUTES A2
    ON A1.ATTRIBUTESETID = A2.ATTRIBUTESETID
    WHERE A1.CATEGORYID=ARG_CATEGORYID
    GROUP BY A1.ATTRIBUTESETID
  );

  RETURN FINALAVERAGE;

END RDEEJW_AVGATTRFORCATEGORY;
```

RDEEJW_ISUSERUNIQUE

This function accepts a USERNAME and EMAILADDRESS as arguments. It returns a scalar variable of COUNT(*). Basically, this function returns the number of records that match USERNAME OR EMAILADDRESS.

```
-- RDEEJW_ISUSERUNIQUE
CREATE OR REPLACE FUNCTION RDEEJW_ISUSERUNIQUE
(
  ARG_USERNAME IN VARCHAR2
, ARG_EMAILADDRESS IN VARCHAR2
) RETURN NUMBER AS
  RETURNVALUE NUMBER;

BEGIN

  SELECT count(*) INTO RETURNVALUE
  FROM RDEEJW_USERS
  WHERE USERNAME=ARG_USERNAME OR EMAILADDRESS=ARG_EMAILADDRESS;

  RETURN RETURNVALUE;

END RDEEJW_ISUSERUNIQUE;
```

Triggers

TRIGGER_RDEEJW_USERS_UPDATE

This trigger executes whenever a RDEEJW_USERS record is update or deleted. If the record has an update EmailAddress or UserName, these records are reflected in the log entry. This is useful for keeping a history of the user's emailaddress' and usernames in the past.

```
CREATE OR REPLACE TRIGGER TRIGGER_RDEEJW_USERS_UPDATE
AFTER
  UPDATE OF "USERNAME", "EMAILADDRESS" OR DELETE
  ON RDEEJW_USERS
  FOR EACH ROW
DECLARE

BEGIN

  INSERT INTO RDEEJW_LOG_USERS
  (OLDUSERNAME, NEWUSERNAME, OLDEMAILADDRESS, NEWEMAILADDRESS)
  VALUES
  (:old.USERNAME, :new.USERNAME, :old.EMAILADDRESS, :new.EMAILADDRESS);

END;
```

TRIGGER_RDEEJW_TABLENAME

These triggers are used when inserting records into any table that requires an auto-generated primary key. It utilizes the SEQ_RDEEJW_TABLENAME to grab the next available value in the sequence. In the example below, the trigger is made on RDEEJW_USERS and is executed when an insert is performed.

```
-- TRIGGER_RDEEJW_USERS
create or replace
TRIGGER TRIGGER_RDEEJW_USERS
before insert on RDEEJW_USERS
for each row
begin
select SEQ_RDEEJW_USERS.nextval into :new.USERID from dual;
end;
```

Phase V: Implementation of the Relational Database

5.1 GUI Design and Implementation

5.1 Daily User Activities

There are two user groups for our web-based application. All login-based users data is stored in the table RDEEJW_Users. “Anonymous Users” have limited access to site functionality, while “Registered Users” have access to other functionality more pertinent to the web site. There is a third group that has a yet-to-be-implemented interface, the “Administrative Users”.

Anonymous Users

The anonymous users are anybody who wants to view the site. They are potential GIS Administrators, GIS Hobbyists, and Computer Science professors. They are able to search and filter published Attribute Sets as well as view the tutorials and non-user information.

Registered Users

Registered users are users that, in addition to what Anonymous Users can do, however they are allowed to download XML, JSON, Excel, as well as connect to and consume web services.

Administrative Users

For brevity of this presentation and additional coding, Administrative User features are shown to all as an example of our knowledge of the product. There is currently no validation enabled (purposely), but as this project picks up we will review membership roles. Administrators will be able to edit Categories, create Sub-Categories, enable / disable users, etc.

5.2 Relations, Views, and Subprograms

5.2 Relations, Views, and Subprograms

Relations

The common daily activities will involve most of the tables in the database. Our database keeps limited historical data on tables, mainly RDEEJW_Users. The following tables are unlikely to be operated on as they contain referential data:

1. RDEEJW_CATEGORIES
2. RDEEJW_ATTRIBUTESETYPES

The following tables are very likely to be operated on daily:

1. RDEEJW_USERS
2. RDEEJW_ATTRIBUTES
3. RDEEJW_ATTRIBUTES
4. RDEEJW_DOWNLOADTRACKINGS

Views

Views are extremely helpful for storing frequently ran queries in a database. Because our database is normalized, it is a pain to frequently create queries with JOINS. This is where views come in – they create a comprehensive set of de-normalized data that a user or program can work with on a daily basis. The views we created for our database are used mainly in reporting or searching from the website, and for brief bits of statistical data on the site. The following list of views are used in our own website:

1. RDEEJW_VW_CATEGORYATTR
2. RDEEJW_VW_USERATTR
3. RDEEJW_VW_LATESTUPDATED

Subprograms

Our project's subprograms were an incredible help when creating, updating and deleting data. It allowed us to safely avoid any database insertion anomalies by wrapping the INSERT, UPDATE, and DELETE commands into Functions or Procedures. Below is the list of subprograms used in our database:

1. RDEEJW_DELETEATTRIBUTE
2. RDEEJW_DELETEATTRIBUTESET
3. RDEEJW_DELETEUSER
4. RDEEJW_INSERTATTRIBUTE
5. RDEEJW_INSERTATTRIBUTESET
6. RDEEJW_INSERTDOWNLOAD
7. RDEEJW_INSERTUSER

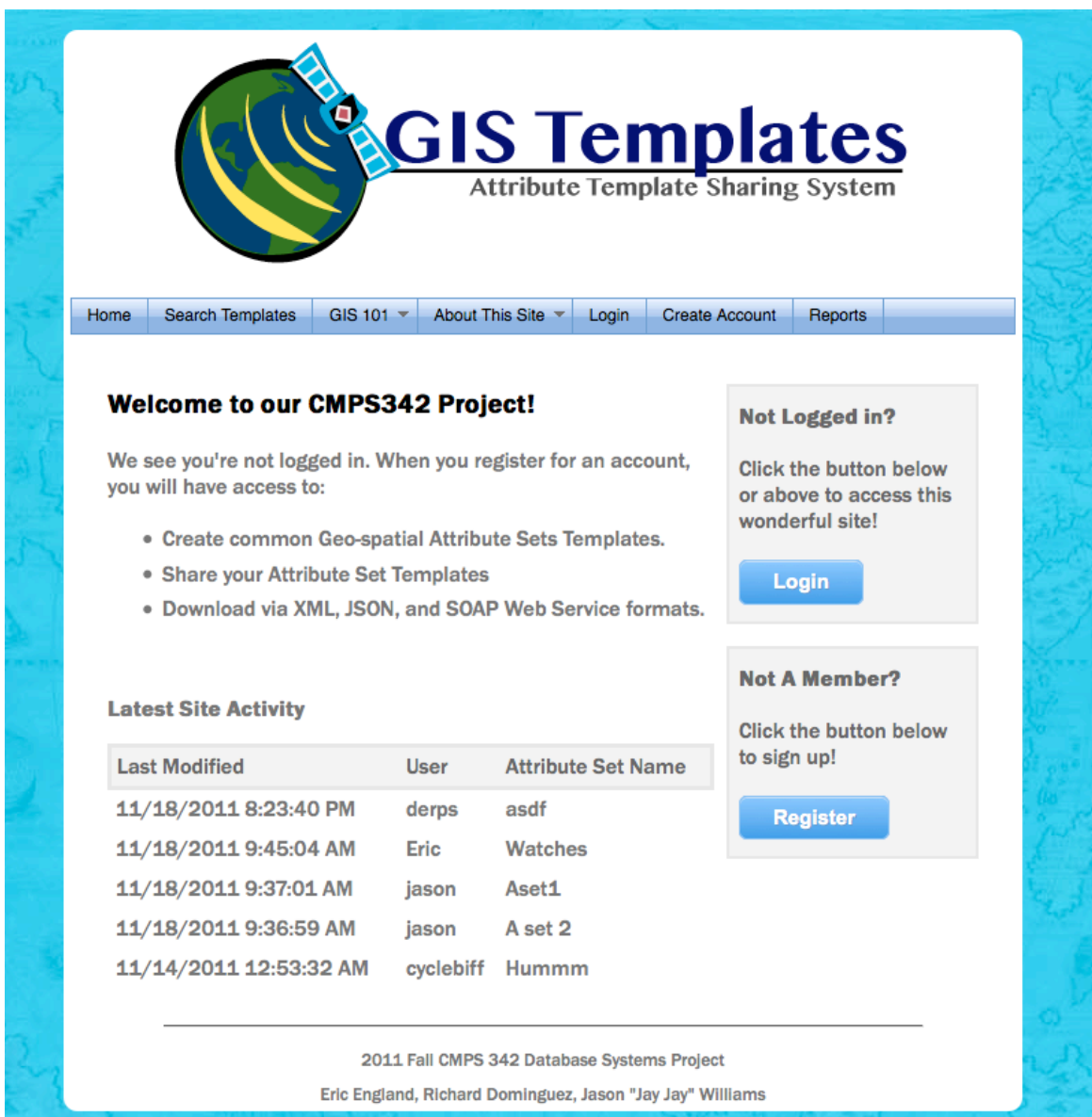
5.3 Application Screenshots and Descriptions

5.3 Application Screenshots and Descriptions

The design of this is not as professional as it could have been – No one in the group is a graphic designer or UI experience designer. That being said, the following screen shots and descriptions are part of our project.

Home Screen

The home page gives a brief introduction to the site. It allows you to traverse the site with or without logging in. Note below that the site recognizes that the user is not logged in, and adds emphasis to have the user LOGIN or REGISTER.



GIS Templates
Attribute Template Sharing System

Home Search Templates GIS 101 About This Site Login Create Account Reports

Welcome to our CMPS342 Project!

We see you're not logged in. When you register for an account, you will have access to:

- Create common Geo-spatial Attribute Sets Templates.
- Share your Attribute Set Templates
- Download via XML, JSON, and SOAP Web Service formats.

Not Logged in?

Click the button below or above to access this wonderful site!

Login

Latest Site Activity

Last Modified	User	Attribute Set Name
11/18/2011 8:23:40 PM	derps	asdf
11/18/2011 9:45:04 AM	Eric	Watches
11/18/2011 9:37:01 AM	jason	Aset1
11/18/2011 9:36:59 AM	jason	A set 2
11/14/2011 12:53:32 AM	cyclebiff	Hummm

Not A Member?

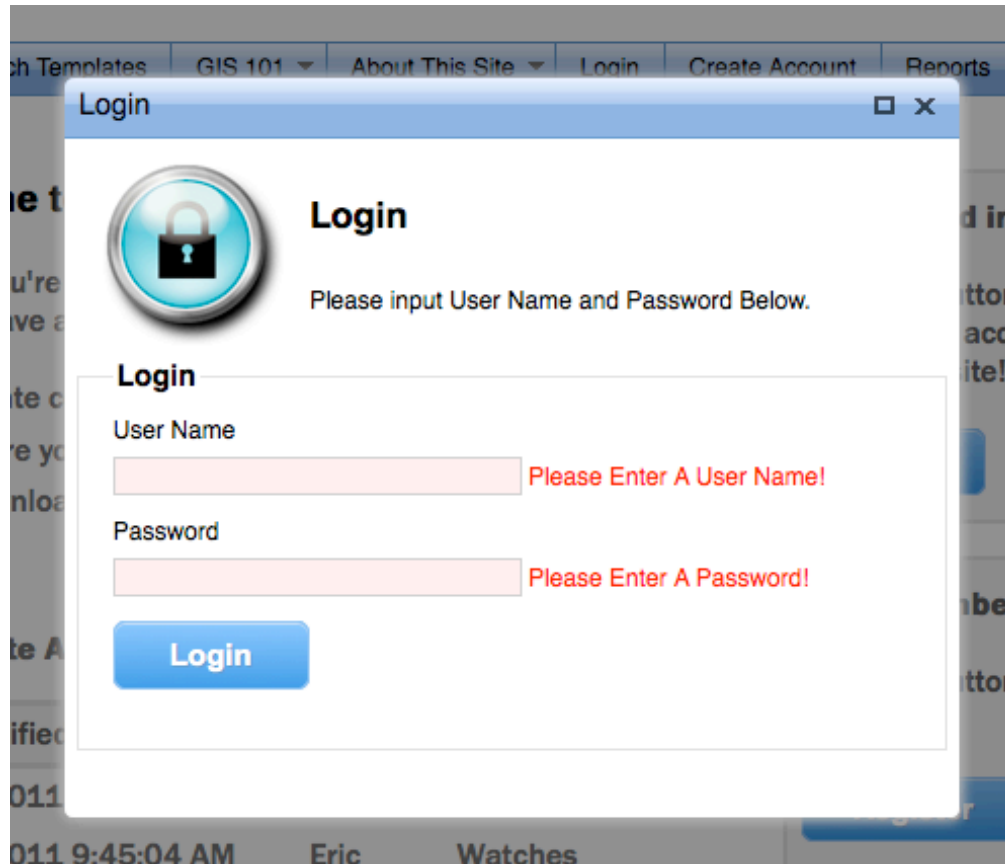
Click the button below to sign up!

Register

2011 Fall CMPS 342 Database Systems Project
Eric England, Richard Dominguez, Jason "Jay Jay" Williams

Login Screen

The login screen is a JavaScript popup page that is accessible from the “Login” button at the top of the page. Note that client-side validation is used to ensure the user types in something and not waste an HTTPPOST server transaction.




The image shows a JavaScript popup window titled "Login" with a blue header and standard window controls. Inside the popup, there is a circular icon with a padlock. Below the icon, the word "Login" is displayed in bold, followed by the instruction "Please input User Name and Password Below." The form contains two input fields: "User Name" and "Password". Both fields are currently empty and have red error messages next to them: "Please Enter A User Name!" and "Please Enter A Password!". A blue "Login" button is positioned below the input fields. The background of the page behind the popup is dimmed and shows a navigation menu with items like "GIS 101", "About This Site", "Login", "Create Account", and "Reports". At the bottom of the page, there is a status bar showing "011 9:45:04 AM", "Eric", and "Watches".

Registration Screen

Our registration popup allows a user to create an account for the site. Like the Login screen, the Registration screen, the registration screen requires the user to input data for all fields. The information from this form is submitted to an MVC controller and is verified on the server side. If the username exists and error is raised and presented back to the user

Register Account

 **Register**

Please enter the information below.

Registered User Form

User Name
 UserName Required!

Confirm User Name

Email Address
 Please Enter An Email Address!

Confirm Email Address

Password
 Password Required!

Confirm Password

Create

User Home Screen

The user's home screen allows the user access their own admin functions, to download JSON/XML, etc. Note the test-Admin dropdown that was not initially available to Anonymous Users.

The screenshot shows the user home screen for the Attribute Template Sharing System. At the top left is a globe icon. The title "Attribute Template Sharing System" is centered at the top. Below the title is a navigation bar with the following items: Home, Search Templates, GIS 101 (with a dropdown arrow), About This Site (with a dropdown arrow), test Admin. (with a dropdown arrow), Log Out, and Reports. The "test Admin." dropdown menu is open, showing three options: My Attribute Sets, My Download History, and My User Settings. Below the navigation bar, the main content area features a welcome message: "Welcome back, test!" followed by "You are now logged in. Please navigate using the 'Admin' dropdown above, or follow these handy links below." There are four blue buttons arranged in a 2x2 grid: My Attribute Sets, My Download History, Account Settings, and Find Attribute Sets. To the right of the buttons, there are two grey callout boxes. The first says "are logged in." and the second says "To logout, click the logout button above." At the bottom, there is a section titled "Latest Site Activity" with a table header containing "Last Modified", "User", and "Attribute Set Name".

Attribute Template Sharing System

Home Search Templates GIS 101 About This Site test Admin. Log Out Reports

My Attribute Sets
My Download History
My User Settings

Welcome back, test!

You are now logged in. Please navigate using the "Admin" dropdown above, or follow these handy links below.

My Attribute Sets My Download History
Account Settings Find Attribute Sets

are logged in.
To logout, click the logout button above.

Latest Site Activity

Last Modified	User	Attribute Set Name
---------------	------	--------------------

Editing My Account

The Edit Account for user screen allows a user to edit his or her settings (previously entered in the Registration Screen). It allows for a user to change his/her username, email address, and password. If another user already has the email address or username, an error is returned to the client.

Edit Account For test

Editing User

User Name

Confirm User Name

Email Address

Confirm Email Address

Password

Confirm Password

Save

Editing My Attribute Sets



















After logging in, click “My Admin” → “My Attribute Sets” from any page of the site. This page allows you to create, publish, un-publish, edit, and delete any of the current user’s attribute sets. It’s the main starting point for forming a user’s Attribute Set to create Attributes.

Home Search Templates GIS 101 About This Site test Admin. Log Out Reports

Attribute Sets For test

[Add new record](#)


Drag a column header and drop it here to group by that column

Attributes	Name	Description	Category	Published	Commands
	<input type="text" value="asfasfa"/>	<input type="text" value="safasfds"/>	Buildings	<input type="checkbox"/>	 
	CSUB Roads	Roads	Walkways and Paths	true	 
	dsfs	sdfs	Fauna	false	 
	jj	kk	An Unused Category	false	 
	Test	safasdfas	Flora	false	 
	Test	asdf	Flora	false	 





1





Displaying items 1 - 9 of 9

Editing Attributes For An Attribute Set

By clicking the  icon, you can begin editing Attributes for an Attribute Set. Note that, like editing Attribute Sets, editing Attributes is identical in every way.


CSUB Roads :: Editing Attributes

Add new record			
Drag a column header and drop it here to group by that column			
Name	Description	Data Type	Commands
Hello!	Testing	Double	 
Walkways	General Walkways	String	 

Navigation:   **1**  













Displaying items 1 - 2 of 2

Searching Attribute Sets

One of the advantages of the system is searching other user's published Attribute Sets. This screen allows users (logged in or not) to search and filter the data below. By clicking the  icon, you can filter results and submit a new search. This makes for a quicker and nicer user experience by not having a HTTPPOST.


Search Attribute Sets

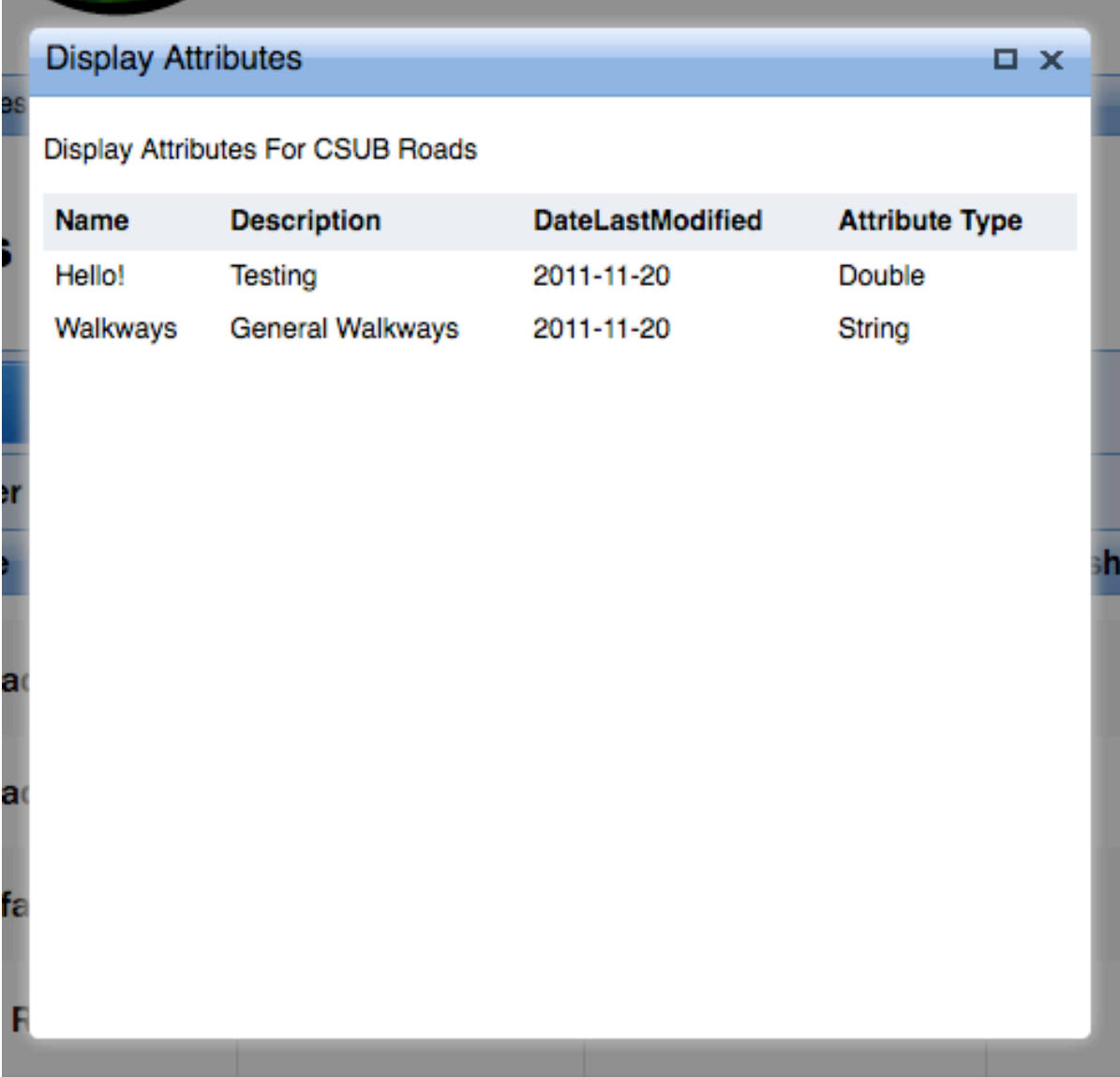
Drag a column header and drop it here to group by that column

	Name	Description	Category	User Name
  	asdf	asdf	Fauna	derps
  	Aset1	Desc	Flora	jason
  	BC - Plants	Plants at BC.	Flora	user12
  	CSUB Roads	Roads	Walkways and Paths	test

⏪ ⏩ 1 ⏪ ⏩ Displaying items 1 - 7 of 7

Previewing Attributes for an Attribute Set

Previewing Attributes can occur from the “Search Templates” and “My Attribute Sets” screens. You can preview the underlying Attributes by clicking the  button. It engages a popup via an AJAX request for an improved user experience.




The screenshot shows a popup window titled "Display Attributes" with a close button (X) in the top right corner. Below the title bar, the text "Display Attributes For CSUB Roads" is displayed. A table with four columns is shown: "Name", "Description", "DateLastModified", and "Attribute Type". The table contains two rows of data.

Name	Description	DateLastModified	Attribute Type
Hello!	Testing	2011-11-20	Double
Walkways	General Walkways	2011-11-20	String

Downloading JSON or XML

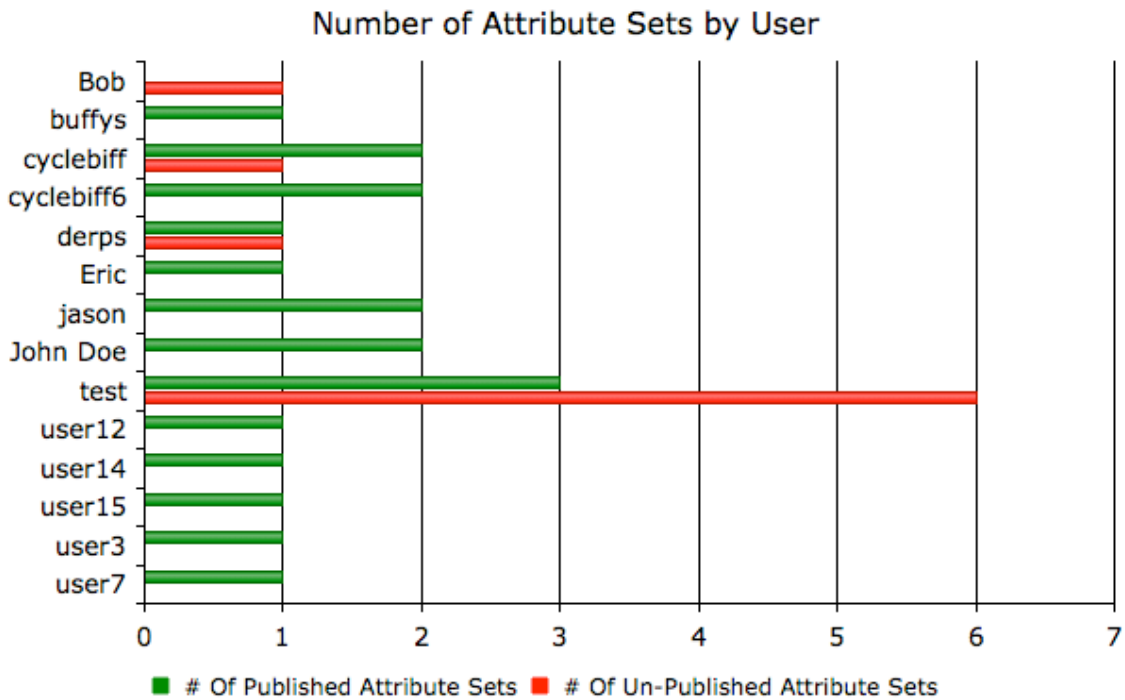
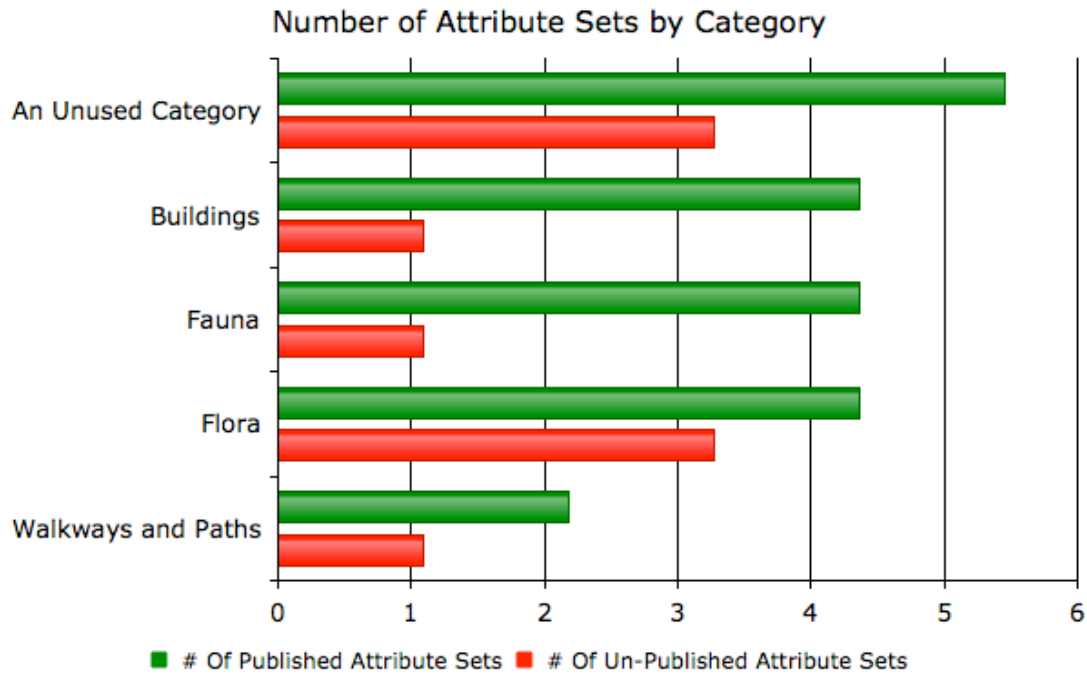
After being logged in, a user can download JSON or XML via the Search Templates screen. The

icons that are used . These downloads serialize Attribute Set data to the client for their own usage. Below is an example of serialized JSON from a created Attribute Set.

```
{
  AttributeSetId: 6,
  Name: "BC - Plants",
  Description: "Plants at BC.",
  IsPublished: true,
  DateCreated: "/Date(1320739200000)/",
  DateLastModified: "/Date(1320739200000)/",
  CategoryId: 3,
  UserId: 13,
  - Attributes: [
    - {
      AttributeId: 21,
      Name: "Age",
      Description: "Age",
      DateLastModified: "/Date(1320739200000)/",
      AttributeSetId: 6,
      AttributeTypeId: 2,
      - AttributeType: {
        AttributeTypeId: 2,
        Name: "Integer",
        DateLastModified: "/Date(1320739200000)/"
      }
    },
    - {
      AttributeId: 18,
      Name: "DatePlanted",
      Description: "Date Planted",
      DateLastModified: "/Date(1320739200000)/",
      AttributeSetId: 6,
      AttributeTypeId: 4,
      - AttributeType: {
        AttributeTypeId: 4,
        Name: "String",
        DateLastModified: "/Date(1320739200000)/"
      }
    },
    - {
      AttributeId: 19,
      Name: "Species",
      Description: "Plant Species",
      DateLastModified: "/Date(1320739200000)/",
      AttributeSetId: 6,
      AttributeTypeId: 4,
      - AttributeType: {
        AttributeTypeId: 4,
        Name: "String",
        DateLastModified: "/Date(1320739200000)/"
      }
    }
  ],
}
```


Reports for Administrators

The following reports are an example of real-time querying via views from the database. It allows administrators to see reports based data from the database. Clicking the raw "Reports" link at the top of any screen can access these reports.



Latest Site Activity

The “Latest Site Activity” area can be seen from the “Home” page near the bottom. It utilizes a view from the database to show the last 5 published Attribute Sets updated in the database.

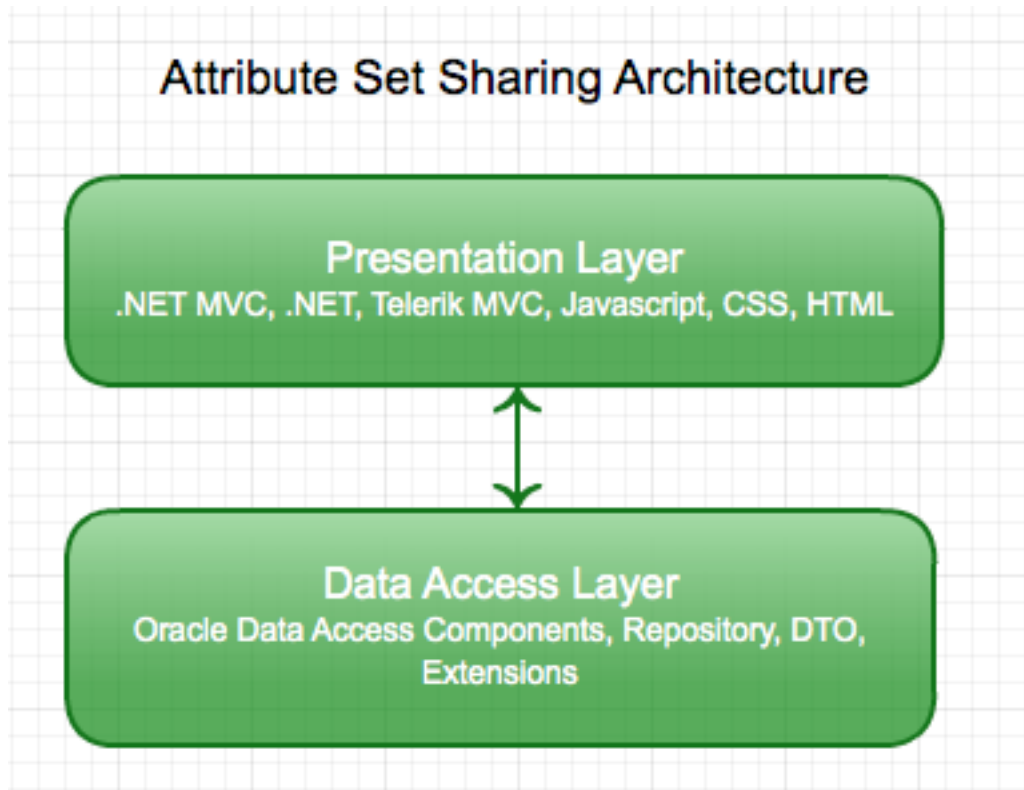
Latest Site Activity

Last Modified	User	Attribute Set Name
11/20/2011 4:01:50 PM	John Doe	plants
11/20/2011 4:01:40 PM	John Doe	Farm
11/20/2011 3:44:56 PM	test	CSUB Roads
11/20/2011 3:29:40 PM	test	asdfsadf
11/20/2011 3:26:25 PM	test	asdfsad

5.4 Code Description

5.4 Code Description

The website and related DLL's for our project was created with Microsoft Visual Studio, .NET Framework, .NET MVC Framework, and Telerik MVC Framework. We utilized a two-tier approach to developing the interface – a Data Access Layer (DAL), and a Presentation Layer (Microsoft MVC). We also utilized the C# and JavaScript languages. Below is the diagram of how our project's web project is architecture:



Data Access Layer (DAL)

When deciding on how to organize the classes we were going to use to INSERT, EDIT, SELECT, and DELETE data, we chose to create a DLL project within our Visual Studio. Our namespaces for the DAL include PsychoGISScratchPad.Data (the root of DAL project), PsychoGISScratchPad.Data.DTO (The data transfer objects used to send to other applications), PsychoGISScratchPad.Data.Repositories (The repository-pattern used to retrieve DTO's), and PsychoGISScratchPad.Data.Extensions (The extensions that help convert the Typed data sets generated by Microsoft to our DTOs.

Data Transfer Objects (DTO's) are sent to the web project. They are simplified objects that represent the database relations:

- DTOUser: This class represents the RDEEJW_USERS relation of the database.
- DTOCategory: This class represents the RDEEJW_CATEGORIES relation of the database.
- DTOAttributeType: This class represents the RDEEJW_ATTRIBUTETYPES relation.
- DTOAttributeSet: This is a representation of the RDEEJW_ATTRIBUTES relation.
- DTOAttribute: This class represents the RDEEJW_ATTRIBUTES relation of the database.

- **DTODownloadTrackings:** This class represents the RDEEJW_DOWNLOADTRACKINGS relation.

Extension Static Classes/Static Methods are the bridge used in converting between Typed Data Set objects (which are converted from Oracle Data Access Components) to our DTO objects. They include:

- ExtensionAttribute
- ExtensionAttributeSet
- ExtensionAttributeType
- ExtensionCategory
- ExtensionDownloadTracking
- ExtensionUser

Data Repository Objects are classes with created convenience methods used to SELECT, INSERT, UPDATE and DELETE DTO objects. The Repositories query our Typed Datasets (XSD), which interfaces with Oracle Data Access Components. They also convert between our DTO and Typed Datasets objects using the extension methods mentioned previously. This level of abstraction is necessary to avoid headaches at the presentation layer. Our repository objects include:

- RepositoryAttributes
- RepositoryAttributesSets
- RepositoryAttributeTypes
- RepositoryCategories
- RepositoryDownloadTrackings
- RepositoryUsers

Presentation Layer

After designing our DAL, we decided to use Microsoft MVC as the project to create our website. We also decided to use the open-source Telerik MVC toolset for professional, state-of-the-art controllers.

.NET MVC Razor: Microsoft MVC is an open source extension framework to the ASP.NET framework. MVC stands for Model, View, and Controller. The controllers control what gets sent and routed to which views and enforce business rules (we do not have a business layer). The views receive models from the controllers for display purposes. The models are referenced mainly from our DAL as DTOs. Razor syntax is the syntax used in the views to render HTML. It is very different from ASP.NET in that it is less complicated and cleans up the UI by not using expensive ASP.NET controls.

Telerik MVC: Used alongside .NET, the Telerik MVC toolset is also open source framework. It contains controls for the RAZOR syntax that assist in making a clean, professional design. Below is a list of the controls that we used:

- Telerik Menu: Our site's usage of the Menu control can be seen on every page – it's the blue menu on every screen.
- Telerik Grid: We utilize a grid in the Template Search, My Attribute Sets, and Edit Attributes pages. This grid is bound to an AJAX-method from their respective MVC controllers. Data bound to the Telerik Grid are respective lists of DTO objects.
- Telerik Window: These controllers are used in many pages. They are the modal pop-ups seen to Login, Register, or preview certain data.
- Telerik Chart: These controllers are used in the "Reports" page for demonstration purposes of displaying our data.

HTML, JavaScript, and CSS are all used in the MVC Views. They, along with the MS MVC Controllers and Telerik MVC Controllers, are rendered together at run-time.

Major Features:

One major feature of the site is the use of the default Membership Provider that ships with MVC Applications. For our purposes, we utilized storing the logged in DTOUser object as a session variable. This helps with security so users cannot update other user's Attribute Sets or Attributes.

Another excellent feature of this site is the dynamic, real-time usage of the Telerik Grids to display, sort, filter, update, delete and insert data. The fact that the site uses AJAX contributes to an enjoyable user experience.

The reporting functionality is also a great feature for Administrators. It would allow statistical analysis and help organize Categories better, or perhaps de-activate users who haven't been logged in after lengthy time has passed.

5.5 Development Process

5.5 Development Process

For our development process, we used an incremental approach to all aspects of the site as a whole. These closely followed the 5-Phase process that this paper represents. As such, we decided to follow Dr. Wang's 5-Phase database project development as a guide to developing the user interface.

Phase 1: Information Gathering

For Phase 1, we decided what we wanted the actual site to do. This included brainstorming what major functionality we wanted the site to accomplish. This also included compiling a list of user groups and what each user group was allowed to do. Being that we already had an idea of what we wanted to accomplish, this phase was trivial.

Phase 2: Data Access Layer – DTO

For Phase 2, we began creating the Visual Studio Project's DAL. We started by creating simple object-representations of the database relations as DTO objects. This phase of development was extremely easy, as we followed our Entity-Relationship model to determine how to model our DTO classes. These classes would later be used to pass data to the Presentation Layer and back to the Data Access Layer.

Phase 3: Data Access Layer – Repository and Extension Methods

In Phase 3, we added to the DAL by learning how to "talk" to the Oracle Database. This was EXTREMELY difficult to figure out. We learned how to install the Oracle Client as well as install and utilize the Oracle Data Access Client objects.

We then developed C# Extension Class/Methods (methods to extend functionality to a class that you don't own). These classes were designed to convert between the typed data sets the ODAC creates, and our DTO objects.

Our final part of Phase 3 was creating "Repositories" – Classes that establish connection to the database, SELECT, INSERT, UPDATE, OR DELETE database columns based on input. These classes are the main access point of data interaction from the Presentation Layer and Web Services.

Phase 4: Presentation Layer – Design and Views

For Phase 4, we created a Microsoft MVC Project as our GUI interface. This involved learning what MVC is, how it works, and what its strengths and weaknesses are. For this phase we simply designed the CSS, some HTML, and learned the Telerik MVC components. We also designed all the views (which wouldn't completely work, as we had not tied it together to the DAL yet). We also opened up Adobe Photoshop to design some of the images used throughout the site.

Phase 5: Presentation Layer – Putting It All Together

For our final phase, we imported the DAL Repositories and DTO namespaces to each Controller and View that required them. We then had to do rigorous testing to try to break each controller and screen (which was easy to do, hey we were / are learning!).

5.6 Conclusion

5.6 Conclusion

This project allowed our group to learn a great deal about everything involved. While individually we have A LOT to learn about databases in general, as a group we prevailed to design a unique concept and functional implementation that is very abstract. Below is a list of the concepts we dived into during the Quarter of 2011 in CMPS 342:

- Relational Algebra and Tuple/Domain Calculus.
- SQL – Practical and Theoretical.
- Oracle database objects.
- Oracle Functions, Procedures, Views, Triggers, and Sequences.
- C# Language.
- .NET Framework, .NET MVC Framework.
- Oracle Data Access Client.
- Oracle SQLPlus.
- Telerik Controls.
- CSS, HTML, JavaScript.