Gossip Guy

Project Report

A gossip management application

Cory Eurom, Xing Xu, Katie Ho
6/7/2013

Codename: gg

Project Name: Gossip Guy
Team Members: Cory Eurom, Xing Xu, Katie Ho
Website: http://staff.washington.edu/cte13/gossipguy/
Github: https://github.com/ktho/gg

Report

Introduction

Gossip Guy was created as the final project for the class INFO 445 Spring 2013 Advanced Relational Database Design, Management, and Maintenance. The project uses the student web servers allocated to each student at the University of Washington. Students are required to create a 3-tier-architecture system that have to do with documents.

Application Area/Framing

The Gossip Guy application has the basic functionality that would be applicable to an online gossip blog or a tabloid website. In the following, we will describe the purpose of each of the modules; however, for more complete information about each of the backend and frontend functions, please see the Functions List and Data Dictionary.

Document Module
The document module includes the tables Reporter, Celebrity, Gossip, and Version. Users can add and update Reporter and Celebrity data. Users can also create gossip and save each version of gossip.

Bundle and Tag Module
The Bundle and Tag module are used for organizing Gossip. Gossip can be organized into tags, which are organized into bundles. The Bundle and Tag module is flexible in how Gossip is organized. Some examples of Bundles might be Relationships, Drug Use, Looks, or Seattle Celebrities. Some examples of Tags in the bundle 'Relationships' could be Divorce, Break-up, or Marriage. Tags could also be more specific and refer to couple names, such as the one for Brad Pitt and Angelina Jolie: 'Brangelina.' Thus, the Bundle and Tag module are fairly flexible and would be useful for navigation.

Workflow Module
Users would be able to change the status of the gossip by moving it along a workflow. Because the process for online publishing can be complex, the Gossip Guy application allows for flexibility in that process. As the organizations who use the Gossip Guy application grow from a one-man blogger to something akin to Huffington Post, their process can also change. For example, they may need to add a new stage in the process as such as 'factcheck' or 'legal review.'

Utility Module
Mechanisms are in place to allow for importing and exporting of reporters, celebrities, and gossip. In addition, user will be able to review recent changes in the document module, such as additions and changes to celebrities,

Requirements

We have implemented all necessary back-end functions such that the Gossip Guy application can perform the basic tasks such as add, delete and update. The functions will be called by a command prompt that follows strict syntax and command structure. The commands are identified by key words and must be followed by the correct amount and type of parameters, otherwise an error will be reported.

Performance and Scaling

The only area in which concern would arise with regards to scaling is as time progresses, there would be a need to remove workflows and gossip. Celebrities are not of much concern, because only a limited number of Celebrities exist and much more data will be going into the gossip related tables. Tag and bundle entries will rely mostly on the growth of gossip. Gossip and its bridge tables will be the most commonly updated tables, because they make up the core of our application. The main function will be to track gossip so this table will be updated and queried on a regular basis depending on the amount of users.

Strength and Opportunities

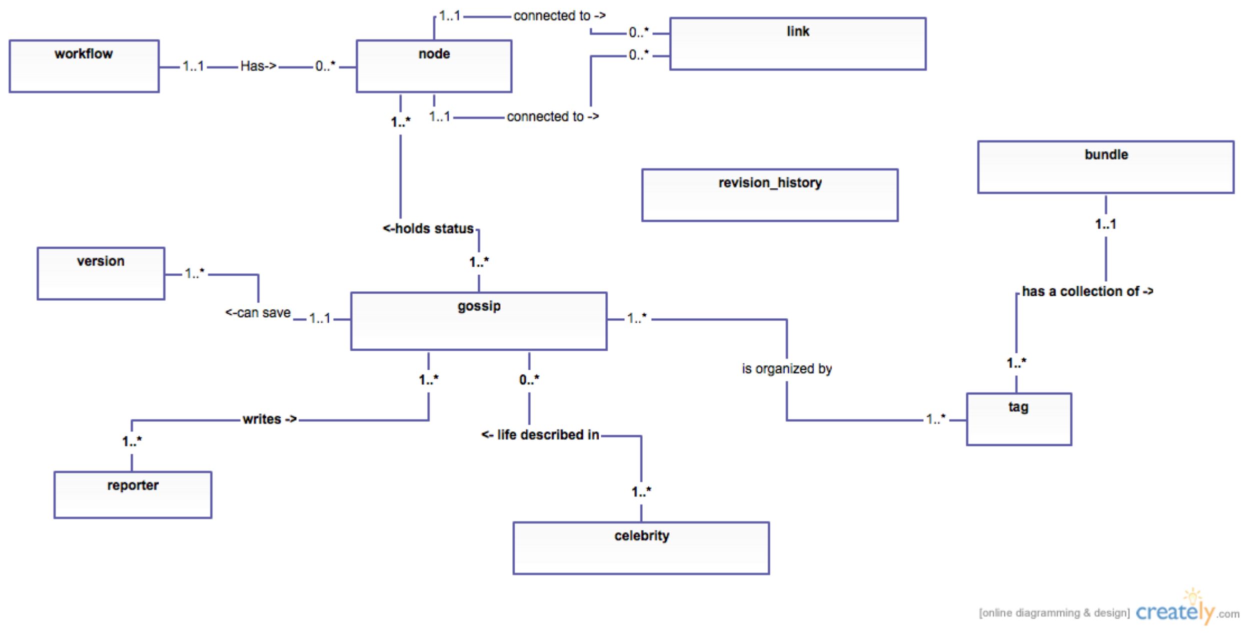
Strength

- The functions needed for adding and updating tables in all modules are implemented in the backend and frontend functions.
- Sites that might use Gossip Guy as the backend have flexibility in their workflows and in their organization of gossip through the Workflow Module and the Tagging/Bundle module.
- Users can save versions of the gossip body and the gossip title.
- Gossip can be categorized by tags to make particular topics easy to find.

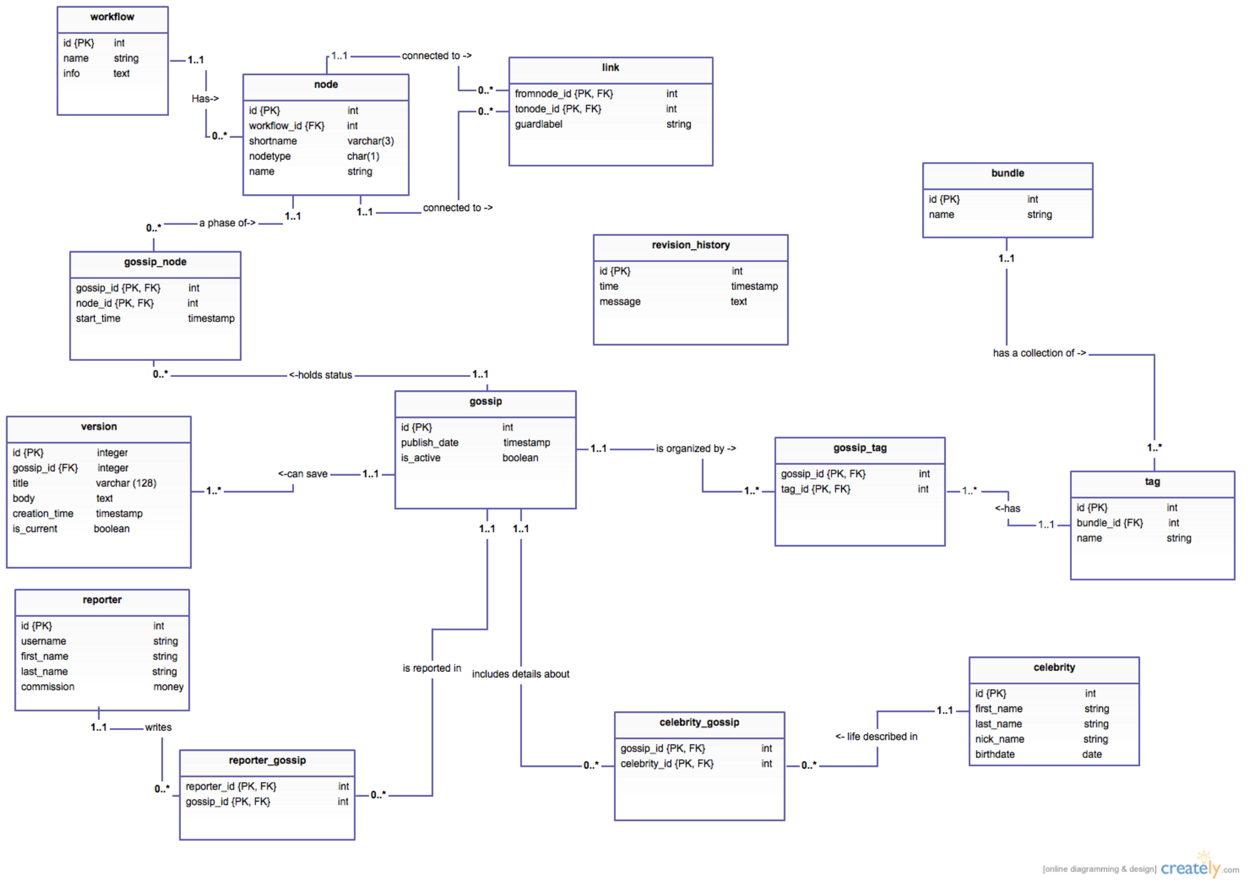
Opportunities

- Security can be improved, so that the application can limit access and changes depending on the user.
- Interface.
- Images would be helpful for the system.
- Error messages can be more informative.
- The system could improve on the handling of ‘deletions’ of data by avoiding deleting altogether by adding attributes that limit views and access. Thus, referential integrity cannot be violated, and organizations that use Gossip Guy will not lose data.

Conceptual Data Model



Logical Data Model



Workflow Module
The model allows for the creation of multiple workflows. One workflow can have zero or many nodes. Nodes belong to only one workflow.

A node can relate to another node through a link. There can be zero or many links for each node. Each link connects only two nodes together. Link has foreign keys fromnode_id and tonode_id that matches with node.id, the primary key for Node. Fromnode_id and Tonode_id are composite primary keys. Each pair of fromnode_id and tonode_id is unique.

Documentation Module
A workflow node can have many gossip, and gossip can be connected to one more more nodes. How gossip is related to node is the mechanism for tracking how the gossip progresses through the workflow.

Gossip has one one or more Version. If Gossip is updated, a new Version is created.

Many Reporters can work together write different Gossip. Gossip can be written by different reporters.

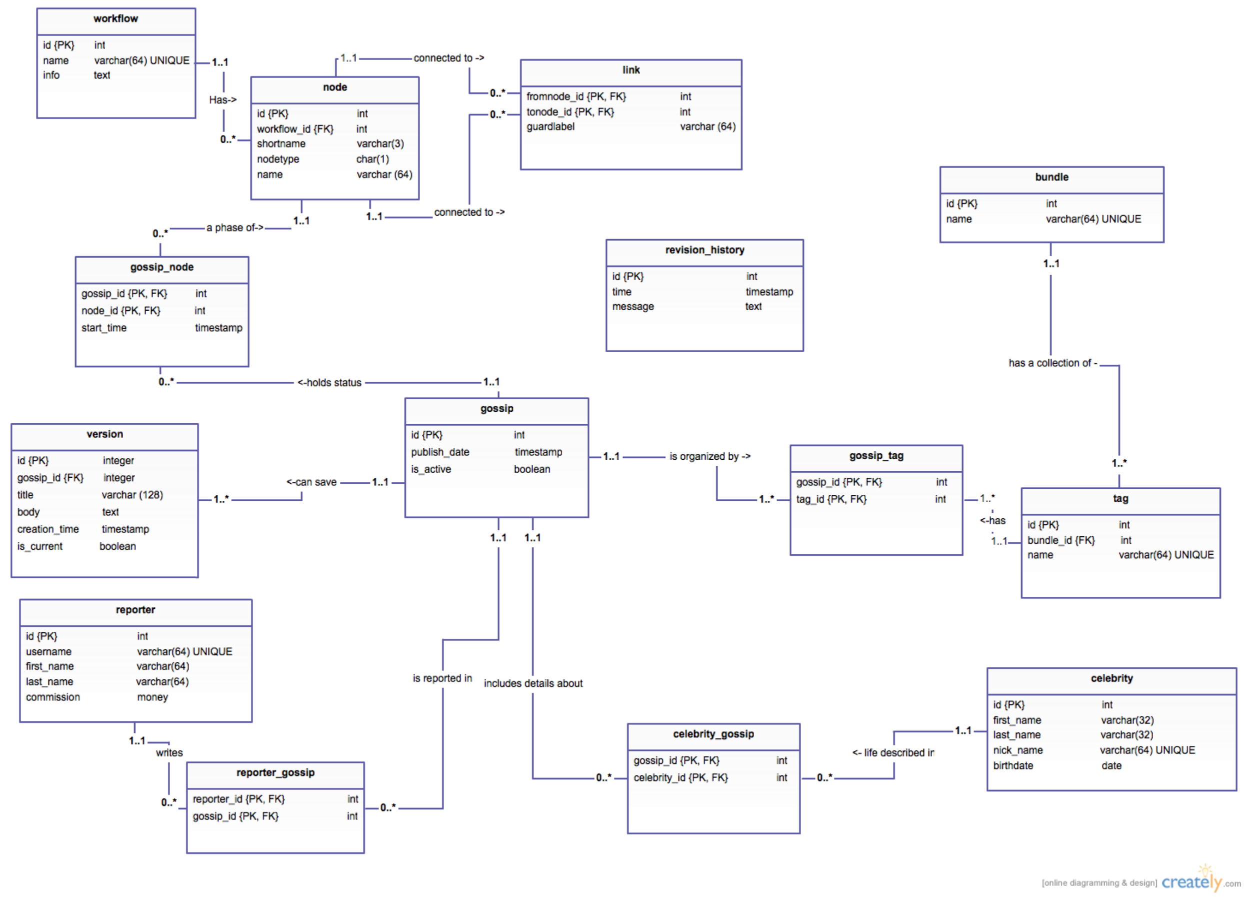
Gossip can be about multiple Celebrities. A Celebrity can have have multiple Gossip written about them.

Tagging Module
Each tag has only one bundle. A bundle can have many tags.

Each gossip can have many tags. Tags can be related to many bundles.

Utility Module
The table revision_history will allow technical super users to keep track of key additions, modifications, and changes in the Gossip Guy system. This table is not related to any other tables.

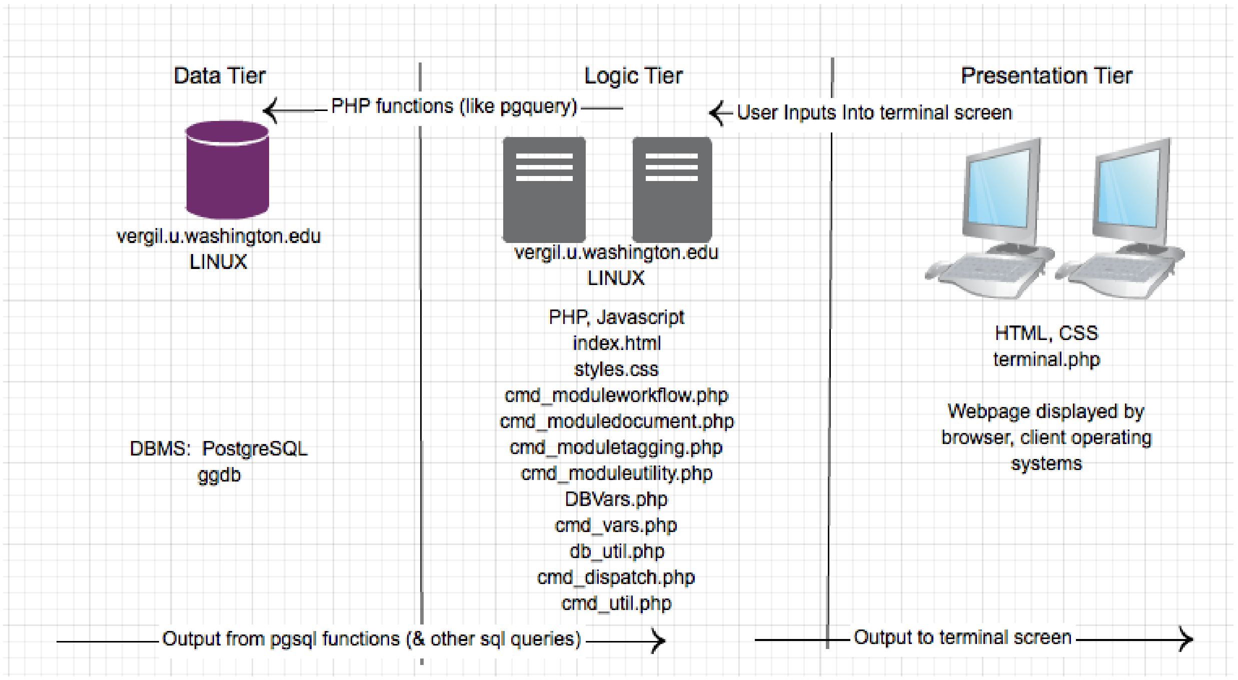
Physical Database Model



There is a GIN INDEX on ggdb.gossip.body on ggdb.reporter.first_name. The GIN Index help in the functions involved in best match searching of similar words for the reporter's first name and gossip body.

There are hash indexes on the unique attributes commonly used for identifying records in specific tables, such as through the command shell. These hash indexes are on the following tables and attributes: workflow.name, node.shortname, celebrity.nick_name, reporter.username, bundle.name, tag.name.

System Architecture Diagram



Summary of development tools and processes

The idea began as a simple reporting application where a reporter would have the ability to take notes on any celebrity. These notes were to be stored in a workflow so that their status could be tracked. Once we had the idea established, we began thinking of the variety of functions that this application might serve outside a simple reporting workflow. We began thinking about how the tagging and bundling attributes might fit. It was apparent that we could use those aspects to categorize gossip, similar to how Twitter uses tags for Tweets. Those tags could be contained within more general baskets called bundles. Tags and bundles would allow a contextual retrieval of gossip. We also thought of functions that involved finding all gossip related to a particular reporter or celebrity. At a certain point, we even got to the discussion of how public users might interact with this site and how their accounts might fit into our ER diagram. But with a group of three people we decided that should be out of our scope so that we could focus efforts on how gossip fits into the workflow. After all 48 functions were implemented in SQL and interpreted through the PHP powered terminal; we found that it was already the end of the quarter, leaving us without time to improve the usability of our system.

The system utilizes the following tools:

github SOCIAL CODING	Git and Github was used for source control
pgAdmin PostgreSQL Tools	pgAdmin used for postgreSQL development
Notepad++	Text editors Sublime Text 2 and Notepad++ for PHP, HTML, and CSS editing.

The Gossip Guy project team separated most of the coding work according to functions. Thus, the people who created the backend function in pgsql also write the PHP function that calls it.

Each team member had their own student web server, which they uploaded synced copies of the Gossip Guy application, differing only in DB Vars.php, which holds the connection details to the specific server. Thus, the team relied heavily on github for collaboration.

Reflections

The system meets all functionality requirements that we decided were necessary for the application to operate. We encountered many obstacles to overcome in our ER diagram, SQL, and PHP. We managed to overcome those obstacles by iteratively improving our system to meet the needs of Gossip Guy. There are many opportunities present, while our system possesses strengths that make it a unique, useful application. The core of the application exists in the set up such that the system can be applied to any type of gossip blog. Usability can be enhanced through the development of a user interface along with other application features, such as user accounts. We hoped to develop the application with a functional interface after we had fully implemented Gossip Guy operations through the terminal, but this proved to be a difficult task with the amount of resources and time that we had allocated.

The experience proved to be insightful both in terms of collaboration and application development. Our group was able to work together on code and share ideas through github and google drive, which enhanced our collaborative abilities and expanded our tool set. The development process started from scratch and turned into a complex application, only to leave more opportunity to add to its complexity. We learned about how the design of an ER can change a lot as you begin to write functions to allow the application to perform particular operations.

Appendix

GossipGuy Project Data Dictionary
Updated On: 6/7/2013

<Workflow>						
Data Member Name	**Type**	**Additional Type Information**	**Definition**	**Defult Value**	**Mandatory?**	**Unique?**
<id>	SERIAL	PRIMARY KEY		N/A	Yes	Yes
<name>	VARCHAR(64)	not case sensitive, periods and spaces allowed.	name of workflow	N/A	Yes	Yes
<info>	TEXT		description of workflow	N/A	No	No

<Node>						
Data Member Name	**Type**	**Additional Type Information**	**Definition**	**Defult Value**	**Mandatory?**	**Unique?**
<id>	SERIAL	PRIMARY KEY		N/A	Yes	Yes
<workflow_id>	INT	FOREIGN KEY		N/A	Yes	Yes
<shortname>	VARCHAR(3)	Unique to workflow	short name of node, unique to workflow	N/A	Yes	Yes
<name>	VARCHAR(64)	not case sensitive, periods and	descriptive long name for node	N/A	Yes	No

		spaces allowed.				
<nodetype>	TEXT	A, F, J, S, E	type of node (Activity, Fork, Join, Start, End)	N/A	No	No

<LINK>

Data Member Name	Type	Additional Type Information	Definition	Defult Value	Mandatory ?	Unique?
<fromnode_id>	INT	PRIMARY KEY, FOREIGN KEY	preceeding node	N/A	Yes	Yes
<tonode_id>	INT	PRIMARY KEY, FOREIGN KEY	succeeding node	N/A	Yes	Yes
<guardlabel>	VARCHAR(64)		description of link	N/A	Yes	No

<GOSSIP_NODE>

Data Member Name	Type	Additional Type Information	Definition	Defult Value	Mandatory ?	Unique?
<gossip_id>	INT	PRIMARY KEY, FOREIGN KEY		N/A	Yes	Yes
<node_id>	INT	PRIMARY KEY, FOREIGN KEY		N/A	Yes	No
<start_time>	timestamp		time when gossip reached that phase, represented by node in workflow	N/A	Yes	No

<GOSSIP>

Data Member Name	Type	Additional Type Information	Definition	Defult Value	Mandatory ?	Unique?
<id>	SERIAL	PRIMARY		N/A	Yes	Yes

		KEY				
<publish_date>	timestamp		date that gossip is published and viewableto public	N/A	Yes	No
<is_active	boolean		boolean controlling when gossip is viewable to public	TRUE	Yes	No

<VERSION>						
Data Member Name	**Type**	**Additional Type Informatio n**	**Definition**	**Defult Value**	**Mandatory ?**	**Unique?**
<id>	SERIAL	PRIMARY KEY, FOREIGN KEY		N/A	Yes	Yes
<title>	varchar(128)		title of gossip	N/A	Yes	No
<body>	text		body of gossip	N/A	Yes	No
<creation_time>	timestamp		date that version is created	N/A	Yes	No
<is_current>	timestamp		boolean indicating whether this is the current version of the gossip	N/A	Yes	No

<REPORTER>						
Data Member Name	**Type**	**Additional Type Informatio n**	**Definition**	**Defult Value**	**Mandatory ?**	**Unique?**
<id>	SERIAL	PRIMARY KEY		N/A	Yes	Yes
<username>	VARCHAR(6 4)	not case sensitive, periods and spaces allowed.	reporter username	N/A	Yes	Yes
<first_name>	VARCHAR(6 4)	not case sensitive, periods	first name of reprorter	N/A	Yes	No

		and spaces allowed.				
<last_name>	VARCHAR(64)	not case sensitive, periods and spaces allowed.	last name of reporter	N/A	Yes	No
<commission>	money		commission reporter receives	N/A	Yes	No

<REPORTER_GOSSIP>						
Data Member Name	**Type**	**Additional Type Information**	**Definition**	**Defult Value**	**Mandatory?**	**Unique?**
<reporter_id>	INT	PRIMARY KEY, FOREIGN KEY		N/A	Yes	Yes
<gossip_id>	INT	PRIMARY KEY, FOREIGN KEY		N/A	Yes	Yes

<CELEBRITY>						
Data Member Name	**Type**	**Additional Type Information**	**Definition**	**Defult Value**	**Mandatory?**	**Unique?**
<id>	SERIAL	PRIMARY KEY		N/A	Yes	Yes
<nick_name>	VARCHAR(64)	not case sensitive, periods and spaces allowed.	name that public uses to refer to the celebrity, such as 'Madonna'	N/A	Yes	Yes
<first_name>	VARCHAR(32)	not case sensitive, periods and spaces allowed.	first name of celebrity	N/A	Yes	No
<last_name>	VARCHAR(32)	not case sensitive, periods and spaces	last name of celebrity	N/A	Yes	No

		allowed.				
birth_date	date		birthday	N/A	Yes	No

<CELEBBRITY_GOSSIP>						
Data Member Name	**Type**	**Additional Type Information**	**Definition**	**Defult Value**	**Mandatory ?**	**Unique?**
<celebrity_id>	INT	PRIMARY KEY, FOREIGN KEY		N/A	Yes	Yes
<gossip_id>	INT	PRIMARY KEY, FOREIGN KEY		N/A	Yes	Yes

<BUNDLE>						
Data Member Name	**Type**	**Additional Type Information**	**Definition**	**Defult Value**	**Mandatory ?**	**Unique?**
<id>	SERIAL	PRIMARY KEY		N/A	Yes	Yes
<name>	VARCHAR(64)	not case sensitive, periods and spaces allowed.	name of bundle	N/A	Yes	No

<TAG>						
Data Member Name	**Type**	**Additional Type Information**	**Definition**	**Defult Value**	**Mandatory ?**	**Unique?**
<id>	SERIAL	PRIMARY KEY		N/A	Yes	Yes
<bundle_id>	INT	FOREIGN KEY		N/A	Yes	No
<name>	VARCHAR(64)	not case sensitive, periods and spaces allowed.	name of tag	N/A	Yes	No

<GOSSIP_TAG>	

Data Member Name	Type	Additional Type Information	Definition	Defult Value	Mandatory?	Unique?
<gossip_id>	INT	FOREIGN KEY		N/A	Yes	No
<tag_id>	INT	FOREIGN KEY		N/A	Yes	No

<REVISION_HISTORY>						
Data Member Name	**Type**	**Additional Type Information**	**Definition**	**Defult Value**	**Mandatory?**	**Unique?**
<id>	SERIAL	PRIMARY KEY		N/A	Yes	Yes
<TIME>	TIMESTAMP		timestamp of change	N/A	Yes	No
<MESSAGE>	TEXT		text about the type of change	N/A	Yes	No

GGDB Function Lists

Meaning	Back End Function	Description	Front End Function
workflow	create_workflow()	Function creates a workflow	workflow create -n<workflow.name> -i<info>
workflow	drop_workflow	Function deletes workflow and all of its associated nodes and links	workflow delete -n<workflow.name>
workflow	get_workflows	Function returns all workflow information in database.	workflow list
workflow	add_node	Function adds nodes to a workflow	node add -wf<workflow> -sn<node.shortname> -t<type> -n<node.name>
workflow	link_from_start	Function links node to starting node for that workflow	link start -wf<workflow> -to<tonode> -g<>
workflow	link_between	Function links node to other nodes	link between -wf<workflow> -from<fromnode>-to<tonode> -g<>
workflow	link_to_finish	Function links node to ending node for that workflow	link finish -wf<workflow> -from<fromnode> -g<>

workflow	get_children	Function returns information on children of given node.	link children -wf<workflow> -sn<shortname>
workflow	get_node_by_id	Function returns node information when given id of the node	NONE
workflow	get_nodes	Function returns node information for a workflow	node list -wf<workflow>
workflow	find_loose_nodes	Function returns node information for nodes without links	node loose -wf<workflow>
workflow/document	get_gossip_status	show state of the gossip with respect to workflow	gossip getstatus -gid "1"
workflow/document	change_gossip_status	move gossip from one workflow node stage to another workflow node stage	gossip changestatus -gid "1" -n "pub" -ac "false"
document	add_reporter	adds the reporter	reporter add -id <username>-f <first_name> -l<last_name> -c <commission>
document	update_reporter	make changes to reporter	reporter update -id <username> -f <first_name> -l <last_name>-c <commission>
document	add_celebrity	adds the celebrity	celebrity add -f<first_name> -l<last_name> -n<nick_name> -b<bday>
document	update_celebrity	update celebrity information	celebrity update -f<first_name> -l<last_name> -n<nick_name> -b<bday>
document	create_gossip	reporter posts the gossip about a celebrity	'gossip create -wf <workflow> -sn <nodeshortname> -r <reporterusername> -c<celebritynickname> -t <title> -b <body>
document	add_reporter_to_gossip	include additional reporter to the gossip	gossip add -gid <gossip.id> -r <reporter.username
document	add_celebrity_to_gossip	include additional celebrity to the gossip	gossip add -gid <gossip.id> -c <celebrity.nick_name>

document	add_tag_to_gossip	include additional tags to the gossip	gossip add -gid <gossip.id> -t <tag.name>
document	update_gossip	make updates to current gossip	gossip update -gid<gossip.id> -t <version.title> -b <version.body> -ac <gossip.is_active>
document	get_reporter_by_comm	Get list of reporters based off of commission	reporter get -id <username> -c<commission>
document	get_reporter_by_lname	Get list of reporters based off of last name	reporter get -id <username> -l<lastname>
document	get_reporter_by_fname	Get list of reporters based off of first name	reporter get -id <username> -f<firstname>
document	get_reporter_by_id	Get list of reporters based off of username	reporter get -id <username>
document	delete_gossip	delete gossip	gossip del -gid <gossip.id>
document	get_gossip_by_id	gets all versions of gossip	gossip list -gid <gossip.id>
document	get_gossip_by_reporter	gets latest version of gossip that a reporter wrote with -ac parameter indicating acive or inactive gossip	gossip listby -r <reporter.username> -ac <gossip.is_active>
document	get_gossip_by_celebrity	gets latest version gossip about a celebrity with -ac parameter indicating acive or inactive gossip	gossip listby -c <celebrity.nick_name> -ac <gossip.is_active>
document	get_gossip_by_tag	gets latest version of gossip by tag with -ac parameter indicating acive or inactive gossip	gossip listby -t <tag.name> -ac <gossip.is_active>
document	get_gossip_by_bundle	gets latest version gossip connected to a bundle with -ac parameter	gossip listby -b <bundle.name> -ac <gossip.is_active>

		indicating acive or inactive gossip	
document	get_reporter	gets list of reporters based off of criteria, such as commission	reporter get -f <firstname>; reporter get -l <lastname>; reporter get -c <commission>
document	get_celebrity	gets list of celebrity based off of criteria, such as birthdate	celebrity get -f <firstname>; celebrity get -l <lastname>; celebrity get -b <birthdate>
document	get_reporter_by_id	get specific reporter information based off of id	reporter get -id <username>
document	get_celebrity_by_id	get specific celebrity information based off of id	celebrity get -id <nickname>
document	search_reporter	use best match search to find reporter	reporter bestmatch -f<firstname>
document	search_gossip	use best match search to find gossip	gossip bestmatch -k<keyword>
tagging	create_tag	creates a new tag	tag create -b <bundle> -n <tag name>
tagging	update_tag	update tag information	tag update -n <current tag name> -nb <new bundle name> -nt <new tag name>
tagging	delete_tag	delete the tag	tag remove -n <tag name>
tagging	create_bundle	creates a new bundle	bundle create -n <bundle name>
tagging	update_bundle	update bundle	bundle update -n <current bundle name> -nb <new bundle name>
tagging	delete_bundle	delete the bundle	bundle remove -n <bundle name>
utility	update_revision_history	show last updates to the system	NONE
utility	get_revision_history	show last updates to the system after time	history get -t <time>