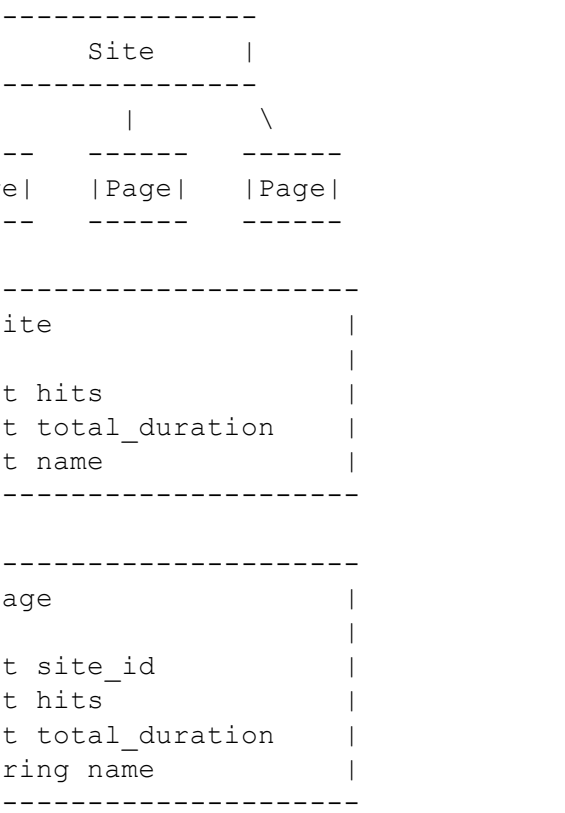
Object Model:

We need to be able to track the hits to each individual page within a site. The object model therefore looks like this:



Each Site has many Page objects that belong to it. This relationship goes as such: a Site has many Pages, and a Page belongs to one Site. This model allows us to record the hits to each individual page within a site and keep a counter for each of them, and also allows us to keep track of the aggregate number of hits to a Site. For a web analytics tool, both pieces of information are valuable (we get to see where users like to visit). If we also keep track of the amount of time spent on each Page, that allows the total amount of time spent on a Site to be calculated, and we can see which pages that users spend the most time.

Hits are not modeled as objects because the granularity of individual visits is not necessary at this point. We are looking for *aggregated* information, meaning that we do not need to know the number of visits from each IP address or the average duration of visits originating in North America (this could be an extension in later versions). Since the hits are not modeled as objects, processing a hit requires only updating the necessary tables dynamically.

This model also allows new Pages to be added to each Site easily. If the controller registers a Page hit to a location that is not already in the Pages table, the Site to which the new Page object belongs can be interpreted from the Page address. Note that Pages have a string name rather than an int. This is so that a Page can have an address like "lol/i/love/cats", not just "18".

When the summary information for each Site is requested, the average visit duration, number of hits, number of tracked Pages, etc, can be calculated "on the fly". This has an upfront cost of requesting the analytics information for a Site, but saves

computation time on the server when each Page is viewed. Since the number of Page views *should* (assuming a normal web traffic pattern) be much higher than the number of requests for analytic information, we would like to optimize for fast registering of Page views and push the computation time to the summary information requests; this is an acceptable trade-off.

Event Model:

In order to keep track of the amount of time spent on each page during a visit, we need to be able to keep track of when users Enter and Exit each page. Luckily, Javascript has a way for us to do this. The javascript functions "onload" and "onunload" tell us when the page has been opened or closed. There is also the possibility of looking at "blur" events, which tell us when the user has looked away from the screen (the window has lost focus). This functionality is not used for this version of the server code because the developers find the "amount of time on a page" to be just as valuable for the administrators (consider Wikipedia or Gmail, which run almost constantly in the background on many personal computers).

Because of these decisions, we have a simple simple event model for the client:

- Window.onload(). This occurs when the page has been loaded into the browser of a user's machine. This starts a timer on the client machine that keeps track of how long the page has been open. This function could later be extended to track the amount of time that has been spent on the page with blur events included.
- Window.onunload(). This occurs when the page is closing. This event causes the client browser to send a request to the analytics server informing it of the amount of time that the user spent on the page, the date and time at which the load and unload events occurred, and the origin public IP address (for usage graphs in the future). The server receives this request and then is able to create the appropriate structures and update the appropriate tables. NB: in the current version of the software, only the URL and the visit duration are sent. This is a synchronous send event because we want the browser to hang, making sure that the actual request gets published.

For the server, the events are:
- Summarize analytics information. This occurs on the server itself when a client requests the summary information for a Site or a Page. The analytics data are calculated lazily (only when the client requests them, not constantly updated in the database).
- Receive hit notification. The server recieves a ping from a client, parses out the unique site identifier, path to the actual request (page), and the page visit duration from the request. The server then updates its database tables as necessary, creating listings for pages if they were not in existence already.

Note that the JS on each client browser will include the page URL as a parameter to be processed by the SitesController, as well as the duration of the visit.