**NEW RULE**

Due to our system now accessing multiple databases, all queries must use full database path.

**EXAMPLE**

select \* from inventory

**SHOULD NOW BE**

select \* from admin\_underverse.inventory

**EVERYTHING IN RED IS DEFUNCT**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **inventory table** | modifier\_id | item\_id | user\_id | base\_item\_id | health\_range\_percentage | added stats etc... |
| *example prefix* | 1 | 177 | 222 | 3 | 0.5 | ... |
| *example suffix* | 2 | 177 | 222 | 3 | 0.1 | ... |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **modifier table** | modifier\_id | mod\_name | HP (range) | added stats etc... |
| *example prefix* | 1 | Vital | 0|20 | ... |
| *example suffix* | 2 | of Health | 0|10 | ... |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **base item table** | base\_item\_id | base\_name | health | added stats etc... |
| *example base* | 3 | Guardian Pendent | 30 | ... |

**Extension pseudo code for GetInventory.java**:

*user (id 222) requests to see his inventory*

select distinct item\_id from inventory where user\_id = 222

// will return all distinct item\_ids

// will return (177) in this example

for all arrays returned:

**PROPOSAL**

Due to each table containing a high number of similar stat columns, it may be beneficial to make **one large table**. This will allow us to easily modify, add, and remove stats. It also increases efficiency by limiting database calls.

Base items, inventory modifiers, and base items will be distinguished by the **type** column.

note: **prefixes and suffixes** will now be referred to as **head mods and tail mods**

Definitions

**head** or **tail mod final** - the calculated head or tail mods for a specific item (large amount of these created dynamically)

**source head** or **tail** - source ranges and stats used to calculate final mods (as many as we define statically)

**base** - static base item stats (as many as we define)

**template** - row showing default values

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **items table** | type | mod\_id | item\_id | user\_id | base\_item\_id | hp | name | added stats etc... |
| *ex. head mod final* | head final | 1 | 177 | 222 | 3 | 0.5 | null | ... |
| *ex. tail mod final* | tail final | 2 | 177 | 222 | 3 | 0.1 | null | ... |
| *ex. source head* | head | 1 | null | null | null | 0|20 | Vital | ... |
| *ex. source tail* | tail | 2 | null | null | null | 0|10 | Health | ... |
| *ex. base* | base | null | null | null | 3 | 30 | Guardian Pendent | ... |
| *ex. base only item* | base final | null | 178 | 223 | 3 | null | null | ... |

**Extension pseudo code for GetInventory.java**:

*user (id 222) requests to see his inventory*

result = select distinct(item\_id),base\_item\_id from items where user\_id = 222

// will return all distinct item\_ids

// will return {(item\_id:177, base\_item\_id:3), ...} in this example

for all arrays returned:

keys = result.getKeys();

**itemID** = result[keys['item\_id']];

**baseItemID** = result[keys['base\_item\_id']];

// get base item

baseItem = select \* from items where type = 'base' AND base\_item\_id = **baseItemID**

// will return base item stats

// will return {... , hp:30, ...} in this example

loop through returned array

disregard all null values and IDs (leaving only stats)

dump remaining stats into an array with original keys //in this example baseStats = {name:"Guardian Pendent", hp:30};

// get modifiers

mods = select \* from items where type = 'head final' OR type = 'tail final' AND item\_id = **itemID**

filter out everything again leaving only stats EXCEPT also include the mod\_ids

//in this example **headFinalStats** = {mod\_id:1, hp:0.5};

//in this example **tailFinalStats** = {mod\_id:2, hp:0.1};

get modifiers and filter them again

select \* from items where type = 'head' AND mod\_id = **headFinalStats[mod\_id]**

//in this example sourceHeadStats = {name:"Vital", hp:0|20};

select \* from items where type = 'tail' AND mod\_id = **tailFinalStats[mod\_id]**

//in this example sourceTailStats = {name:"Health", hp:0|10};

to get final name: sourceHeadStats[name] + " " + baseStats[name] + " of " + sourceTailStats[name]

// in this example "Vital Guardian Pendent of Health"

to get final stats, loop through each key filtering out non-stats (in this example there is only one stat key: hp)

get min and max of source head and tail stat (e.g. Vital is 0 and 20)

multiply max - min by final ratio and add min (e.g. ((20 - 0) \* 0.5) + 0)) to get final stat = ((max - min) \* percentage) + min

**Note:** While this system is very flexible, the major disadvantage of this flexibility is that it must make several calls to the database to get the information on a single item. In my experience with major PHP systems, database calls in this low volume aren't very strenuous on the server (many forums make hundreds of database queries per user per request). However, it is unclear how SmartFox's Java environment and SQL extension will handle the stress.

If the load of this system does cause a problem, there is a simple solution. The system will stay how it is but we will have all the items pre-built in a separate table as well for efficient calling.

This will require the table to be re-built every time we make a modification to the source items but would make calling much more efficient.

Before doing something like that though a simple call to PHP through Java would probably also solve the problem.