Sarah’s Cupboard

CSCI 332 Final Project

Application Structure

-Cooks : a collection of users with a password, id, username

-Food: a collection of Food available to all users

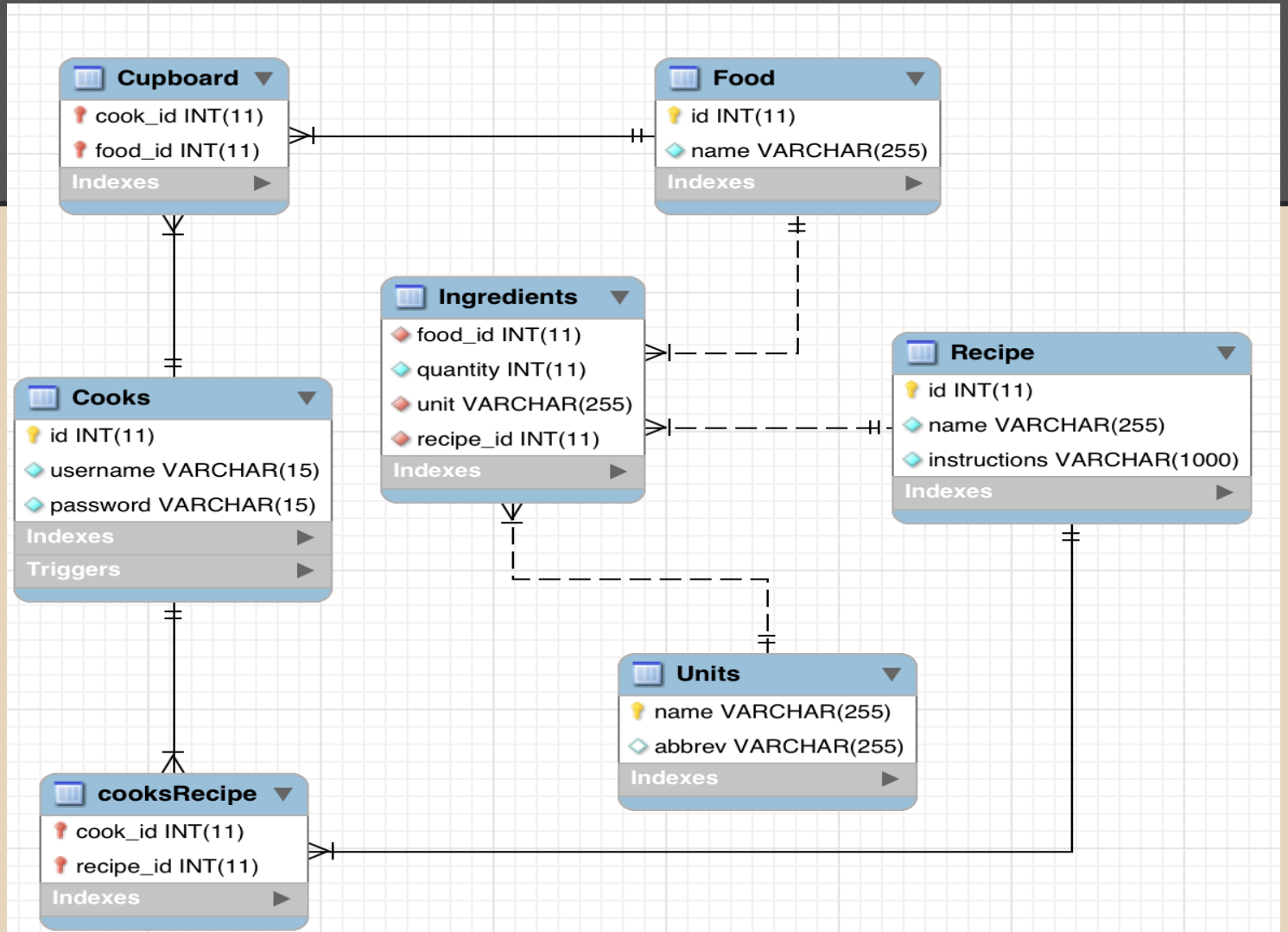
-Recipe: a collection of Recipes available to all users

-cooksRecipe: a collection mapping users to recipes which they have saved (many to many)

-Cupboard: a collection mapping users to food items which they have saved (many to many)

-Units: a static collection of units which can be used with Ingredients, used chiefly for integrity enforcement

-Ingredients: a collection of recipes mapped to food items which are their ingredients (many to many)


Cupboard
cook_id INT(11)
food_id INT(11)
Indexes
Food
id INT(11)
name VARCHAR(255)
Indexes
Ingredients
food_id INT(11)
quantity INT(11)
unit VARCHAR(255)
recipe_id INT(11)
Indexes
Recipe
id INT(11)
name VARCHAR(255)
instructions VARCHAR(1000)
Indexes
Cooks
id INT(11)
username VARCHAR(15)
password VARCHAR(15)
Indexes
Triggers
Units
name VARCHAR(255)
abbrev VARCHAR(255)
Indexes
cooksRecipe
cook_id INT(11)
recipe_id INT(11)
Indexes

Normalization

4NF because there are no multivalued dependencies and every primary key is the left side of every functional dependency is a primary key.

Examples:

Ingredients: recipe_id, food_id -> quantity, unit

Food: id -> name

Isolation Level

Due to the nature of the application and its lack of sensitive data storage and the very low possibility of concurrent updates, the isolation level remains at the default, repeatable read.

Integrity Enforcement

-All ids are auto incrementing and used alone or in combination with each other as primary keys

-Foreign keys used to ensure user data as valid

- cascading updates and deletes used with recipes and ingredients but not with food because if a food item is deleted, the ingredients of a recipe could be messed up

Stored Procedures

-Stored procedures used for almost all CRUD operations

-Example: when a user inserts a food item into their cupboard which is not already in the Food table, it inserts the item into the Food table and then into the user's cupboard entry

```
create procedure cupboardInsert(IN cook int, IN foodName varchar(255))
begin
select @x:=id from Food where foodName = Food.name;
if @x is null then
select @y:=id from Food order by id DESC limit 1;
insert into Food(Food.name) values(foodName);
insert into Cupboard (Cupboard.cook_id,Cupboard.food_id)values(cook,@y+1);
else
insert into Cupboard (Cupboard.cook_id,Cupboard.food_id)values(cook,@x);
end if;
end
```

Triggers

Stops users from being able to create a username or password that already exists

```
CREATE DEFINER=`sarandford`@`localhost` trigger validateUser before insert on Cooks for each row
begin
IF exists(select username from Cooks where username=NEW.username) THEN
                SIGNAL SQLSTATE VALUE '45000'
                        SET MESSAGE_TEXT = '[table:Cooks] - `username` column is not valid';
elseif exists(select password from Cooks where password=NEW.password) THEN
        SIGNAL SQLSTATE VALUE '45000'
                SET MESSAGE_TEXT = '[table:Cooks] - `password` column is not valid';
        END IF;
    end
```

Functions

Only one function is used and it is to return a total of the number of items in a user's cupboard

```
delimiter //
create function totalItems(id int)
returns int
begin
declare x int;
select count(*) into x from Cupboard where cook_id=id group by cook_id;
return x;
end//
```

Views/Reports

-Multiple views are used to aid in selecting data from the tables with many to many relationships.

-As an example the report is a view:

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `sarah`@`153.9.0.105`
    SQL SECURITY DEFINER
VIEW `popularFood` AS
    SELECT
        `Food`.`name` AS `name`, COUNT(0) AS `count(*)`
    FROM
        (`Food`
        JOIN `Cupboard` ON ((`Food`.`id` = `Cupboard`.`food_id`)))
    GROUP BY `Cupboard`.`food_id`
    ORDER BY COUNT(0) DESC
```

View the site

http://sarahscupboard.com/