Emilio Colindres exc231

Andrew Wood aew61

Jonathan Fische jxf259

Harry Nelken hrn10

**Crave Final Report**

**Application Background:**

While there are many existing applications that provide access to vast databases containing information specifically regarding different restaurants and their corresponding type of cuisine, there are no widely known applications that allow you search by specific food items. For example, say you as a user are specifically in the mood for Shrimp Fried Rice. Instead of being able to search for Shrimp Fried Rice directly, you must search for Chinese restaurants, Thai restaurants, and maybe even Japanese restaurants, and then search each of their menus for the dish. This is the issue that our group is attempting to solve. A lot of the time we make decisions about what restaurant we go to based on what dish we are in the mood for, rather than a type of restaurant. Our solution to this dilemma is *Crave*.

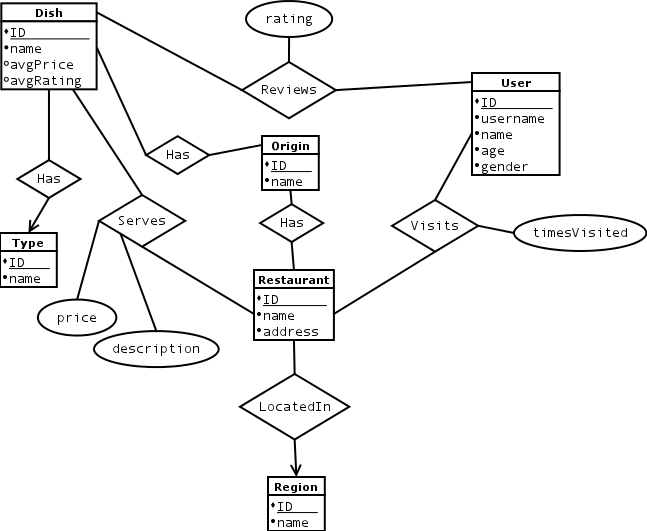
*Crave* is a desktop application written in Java, backed by a MySql database, that allows its users to search for food dishes in the University Circle area of Cleveland. The user is able to search for items by providing keywords, and then optionally specifying the type of dish, such as burgers or sushi, and the origin of the dish, such as Italian. Each user also has a profile page with a list of restaurants that they have visited, along with suggested dishes to try based on their history.

**Data Description:**

The database stores various types of program information. User data, such as usernames and passwords, are stored, as well as the rating that a user has given a dish. The dishes themselves, along with the restaurants that serve them, are also stored in the database with information connecting dishes to restaurants. The types, as well as origins, of food supported are stored in the database so that the user can select from a predetermined number of options. The different dishes are also linked to their origins and types in the database.

We decided to maintain a fixed set of dishes and restaurants, so that the only interaction the user has with the database is being able to register (store) a user, update their rating for a certain dish, or request to view certain information about dishes and restaurants.

**ER Diagram:**



**Functional Dependencies:**

Users ( ID, name, username )

{ ID->name,username, name,username->ID }

Restaurants ( ID, name, address )

{ ID->name, address, name,address->ID }

Dishes ( ID, name, avgPrice, avgRating )

{ ID->name,avgPrice,avgRating }

Types ( ID, name )

{ ID->name, name->ID }

Regions ( ID, name )

{ ID->name, name->ID }

Origins (ID, name )

{ ID->name, name->ID }

**Schema:**

ENTITIES

All ID attributes are auto-incrementing.

users( I​D,​ name, username ) ­ name not null, (username) unique not null

restaurants( ID,​ name,​ address ) ­ (name, address) unique not null

dishes( ID, n​ame, avgPrice, avgRating ) ­ name not null, avgPrice default 0.00, avgRating default 0

types( ID, name ) - (name) unique not null

regions( ID, n​ame ) - (name) unique not null

origins( ID, name )​ - (name) unique not null

RELATIONS​

All relations have foreign key constraints for the entities that they reference. Furthermore, all foreign key references cascade on both updates and deletions.

( Dish - Origin )

dishOrigin( dID, oID )

( Dish - Type )

dishType( dID, tID )

( Restaurant - Origin )

restaurantOrigin( rID, oID )

( Restaurant - Region )

locatedIn( resID, regID )

( Restaurant - Dish )

serves( rID, dID, price, description ) - price not null

( User - Restaurant - Dish )

reviews(uID, rID, dID, rating ) - rating not null

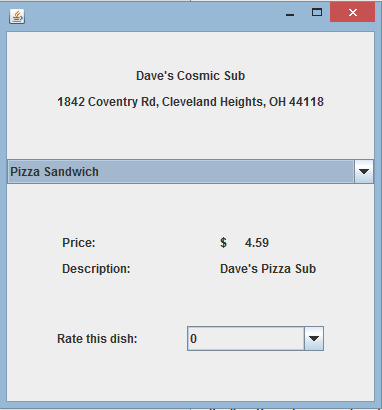
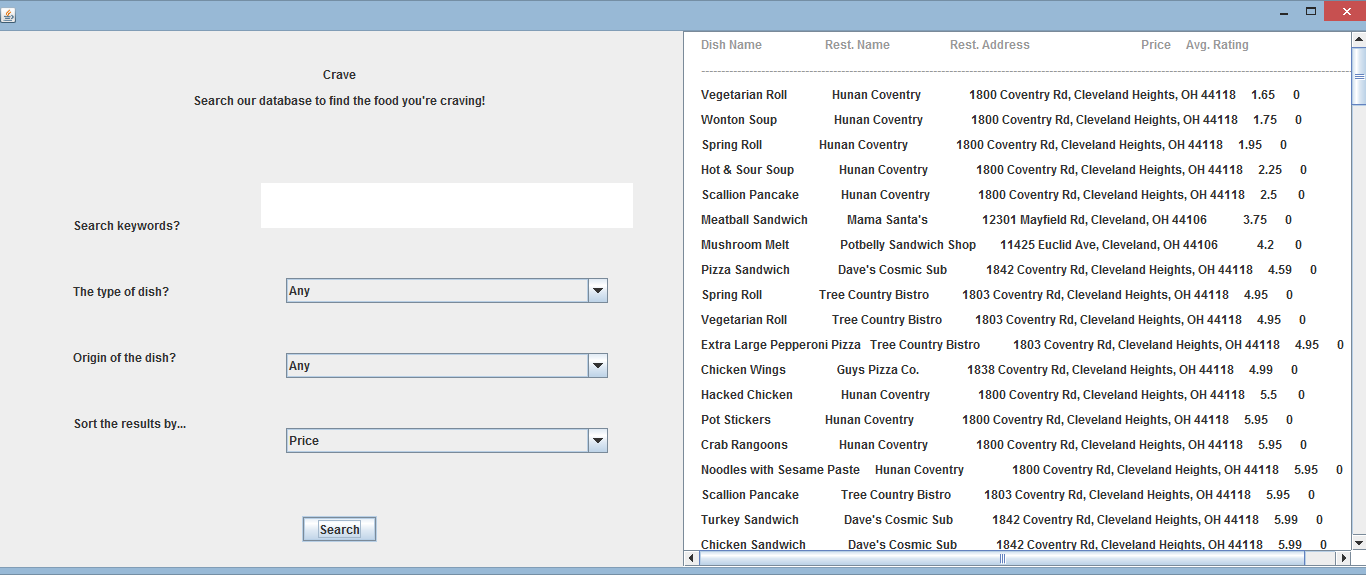
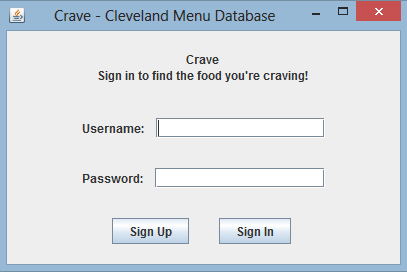
**Indexes:**

We do not explicitly specify indexes in our database, other than specifying Primary Keys and Uniqueness as stated above. Since each entity’s primary key is simply an ID, it is a very efficient index.

**Triggers and Assertions:**

The serves and reviews relations each have one trigger. The trigger for the serves relations updates the avgPrice attribute for the corresponding dish that the new tuple is referencing. The trigger for the reviews relation similarly updates the avgReview attribute for the corresponding dish that the new tuple represents a review for.

**User Interface:**



**Example Queries:**

SELECT d.name, d.ID, r.name, r.ID, r.address, serv.price, d.avgRating

FROM dishes as d, restaurants as r, serves as serv, origins as o, types as t, dishOrigin as do, dishType as dt

WHERE (serv.rID, serv.dID) = (r.ID, d.ID)

AND (d.ID, d.ID) = (do.dID, dt.dID)

AND (o.ID, t.ID) = (do.oID, dt.tID)

ORDER BY serv.price

**Implementation:**

*Crave* is written in Java and backed by a MySQL database. Our current implementation required the database to be stored locally, but the application can be easily modified to connect to an online database. Because of the nature of both Java and MySql, our application is completely cross-platform, whether used on a local database or not. We have tested and confirmed its reliability on OS X 10.10, Windows 7, and Windows 8.

**Team Member Roles:**

* Emilio Colindres was in charge of the design of the database schema and triggers, as well as the creation, deletion, and dummy data generation scripts. He also wrote the final report.
* Harry Nelken was the architect behind the user interface code. He devised the class hierarchy for the UI and helped integrate the dynamically generated queries into the application.
* Andrew Wood single-handedly wrote all of the query management code, on top of working on the user interface. He also put forth a substantial effort into integrating his query management system into the user interface.
* Jonathan Fische did all of the research on local restaurants and created Excel sheets to populate the database. He worked tirelessly to provide the rest of the group with a suitable amount of data to test the application with a realistic database.