Pizza Project (doc)

A college, to attract more students, has decided to offer free pizza in the dormitory.

You have been selected to implement the needed automated ordering system, a webapp of course…

Pizza Dynamics

The pizzas will be available in 2 sizes, Small and Large.

Toppings beyond the basic tomato sauce and cheese can be selected from an expandable set of options:

- Pepperoni
- Onions
- Mushrooms
- …

A pizza just ordered has status PREPARING
A pizza later becomes BAKED
A pizza with acknowledged delivery is FINISHED

Web app user actions

- A student can order any subset of these toppings, and choose the size.
- The student id (user id) and current day is remembered as well.
- The students should be able to ask if their pizza(s) are done, and their size and toppings
- When a student acknowledges receipt of the pizza(s), those pizza orders are marked completed.

How does a pizza become BAKED?

- We could make the server keep track of time… but that’s unusual.
- Every active website needs an admin
- The pizza shop admin tells the system when the next pizza is done (they come out of the oven in order).
- The admin also says when a day is done. When a day is done, all the orders are complete for that day.
- The admin also can add a topping, list orders, reinitialize, etc.

Designing the UI

- *When designing modern user interfaces, think objects, then actions*.
- Looking at the user and admin actions, we see they can be grouped as involving objects that are toppings, sizes, orders, and days. Also the users themselves.
- For simplicity, the sizes are just Small and Large, not changeable by the UI
- Thus we propose the top-level topics:
  - Toppings
  - Orders
  - Days
  - Users

Designing the UI

- When we manage a collection like toppings, we don’t make the user enter/choose commands like “list”, “add”, …
- We just show the current collection to the user, with a button/link to add something to the collection, and a button on each item for its delete (and another for its update, if needed)
- This UI pattern is first shown in the book in Chap. 4, in the Product Manager.
  - Here we are managing a collection of Products (guitars, basses, etc.).
  - A user (an admin) can add a Product, or delete one.
  - This approach only involves two pages, one for listing the collection and one for adding a new element to it.

Designing the Database

- We want to be able to add a new topping to the system
- So we need a table for orders, another for toppings
- A single order can have many toppings
- A single topping can be used in many orders
- Thus we could model this as a N-N relationship between orders and toppings
- But then it's hard to delete a topping since it is still in use with older orders
- In reality, there's a difference between the idea of a certain topping being available (on the menu), and its use in a particular pizza
- So let's go back to basics and look at one pizza…

A Pizza Order

- A pizza order has a set of toppings and a single size
- For example, order 10 has size “Small” and toppings “pepperoni” and “onions”
- So the pizza_order table has “size” as a column, so the row for order 10 can have “size=small”.
- We need to attach toppings “pepperoni” and “onions” onto this order.
  - This is like employees and hobbies, a standard example of a *multi-valued attribute*. Each employee may have multiple hobbies.
  - The relational solution is to have a employee_hobby table with (empid, hobby) rows and FK on empid. The PK is (empid, hobby).
- So here we need an order_topping table with (orderid, topping) rows.

Pizza Database after a topping and size are added (No orders yet)

pizza_orders table: id is PK, empty to start

Id	user_id	size	day	status

shop_users: id is PK, users “joe” and “sue”

id	username	room
1	joe	6
2	sue	3

order_topping: (orderid, topping) is PK: empty

order_id	topping

status_values

status_value
Preparing
Baked
Finished

menu_toppings: id is PK, topping is unique

id	topping
1	pepperoni

menu_sizes: id is PK, size is unique

id	size	diameter
1	Small	12
2	Large	16

pizza_sys_tab (one row table)

current_day
1

Pizza Database after an order by sue is recorded

pizza_orders table: now has one order

Id	user_id	size	day	status
1	2	Small	1	Preparing

shop_users: id is PK, users “joe” and “sue”

id	username	room
1	joe	6
2	sue	3

order_topping: (orderid, topping) is PK

order_id	topping
1	pepperoni

status_values

status_value
Preparing
Baked
Finished

menu_toppings: id is PK, topping is unique

id	topping
1	pepperoni

menu_sizes: id is PK, size is unique

id	size	diameter
1	Small	12
2	Large	16

pizza_sys_tab (one row table)

current_day
1

Pizza Shop actions and database contents

Pizza status values;
Preparing → Baked → Finished

Suppose have one pizza size « small », one topping « pepperoni », then two orders:

First pizza order by sue:

1. Ordered (status = Preparing)
2. Admin said pizza ready (status=Baked)
3. Student received it (status=Finished)

Second pizza order by joe:

1. Ordered (status=Preparing)
2. Day ended, status=Finished

pizza_orders table: id is PK

<u>Id</u>	user_id	size	day	status
1	2	small	1	Finished
2	1	small	1	Finished

order_topping: (orderid, topping) is PK

<u>Order_id</u>	<u>topping</u>
1	pepperoni
2	pepperoni

Final state of database

pizza_orders

id	user_id	size	day	status
1	2	small	1	Finished
2	1	small	1	Finished

order_topping

order_id	topping
1	pepperoni
2	pepperoni

pizza_sys_tab (one row table)

current_day
2

menu_toppings: id is PK, topping is unique

id	topping
1	pepperoni

status_values

status_value
Preparing
Baked
Finished

menu_sizes: id is PK, size is unique

id	size	diameter
1	Small	12
2	Large	16

Foreign Keys

- We need a FK from order_id in order_topping to orders to make sure that order exists.
- Note we are not planning to delete orders in this app.
- It’s tempting to put a FK from topping in order_topping to topping in menu_toppings
- But then a topping can’t be deleted when it’s in use in old orders
- Similarly the size in pizza_orders can’t have a FK to size in menu_sizes.
- We could consider “on delete set null” for the FK on size, an advanced option. But we want to keep things simple.
- Thus we’ll stick with one FK on order_id, and one to make sure the status is valid.