Inserting, updating, and deleting tutorial

In this tutorial, you'll write SQL queries that demonstrate:

- How to add data to a database using `INSERT`
- How to change data in a database using `UPDATE`
- How to remove data from a database using `DELETE`

Open the `inserting-updating-deleting-tutorial.sql` file in pgAdmin using the `PizzaShop` database. Add the SQL statements for this tutorial under the corresponding sections.

Part one: Query for orders

You've been asked to help some friends with the site for their new pizzeria, "Archimedes Pie." Fortunately, you have some experience with pizza shop databases and are willing to help.

The basic queries to select, add, update, and remove data from the database have been sketched out, and you're in the process of finalizing them. In particular, you're posing various scenarios to yourself and seeing how well the queries work within these scenarios.

One of these hypothetical situations involves a customer calling to add one more pizza to their existing order. The first step is to confirm the existing order with the customer. The customer is "Elenore Mamwell" who lives at "561 Claremont Alley."

Write, or copy and paste the following `SELECT` statement:

```sql
SELECT c.last_name, c.first_name, c.street_address, s.sale_id, p.pizza_id, p.size_id,
p.crust, p.sauce, pt.topping_name
FROM customer AS c
JOIN sale AS s ON c.customer_id = s.customer_id
JOIN pizza AS p ON s.sale_id = p.sale_id
LEFT JOIN pizza_topping AS pt ON p.pizza_id = pt.pizza_id
WHERE c.last_name = 'Mamwell' AND c.first_name = 'Elenore'
ORDER BY c.last_name, c.first_name, s.sale_id, p.pizza_id;
```

Run the query and you'll get these results:

last_name	first_name	street_address	sale_id	pizza_id	size_id	crust	sauce	topping_name
Mamwell	Elenore	561 Claremont Alley	4	4	L	Thin	Normal	Mushrooms
Mamwell	Elenore	561 Claremont Alley	37	49	L	Regular	Normal	Extra Cheese
Mamwell	Elenore	561 Claremont Alley	37	50	L	Regular	Normal	Bacon
Mamwell	Elenore	561 Claremont Alley	37	51	L	Regular	Normal	Tomatoes

last_name	first_name	street_address	sale_id	pizza_id	size_id	crust	sauce	topping_name
Mamwell	Elenore	561 Claremont Alley	**50**	**67**	M	Pan	Normal	Ham

Looking at the `sale_id` column, you see that Elenore previously placed three orders with ids of `4`, `37`, and `50`. Since `sale_id` is a `serial` column, and the rows in the result set are in part ordered by `sale_id` in ascending order, you can assume the higher the value, the more recent the order. With this in mind, you confirm the last order for a "medium pan pizza with ham" with Elenore.

Note that the `sale_id` for Elenore's order is `50`, and the `pizza_id` is `67`. You'll need these values shortly.

> Note: *Serial column values may differ from those shown*.
>
> Depending upon any changes you may have performed in your instance of the `PizzaShop` database, when you run the query, the actual values of `serial` columns, such as `sale_id` and `pizza_id`, may differ from the ones shown in this tutorial.
>
> *This isn't an error.* The database is responsible for generating `serial` values, and as long as primary keys are unique within individual tables, the actual value is largely unimportant. However, it does mean that you may have to reconcile the differing values between the result set shown in the tutorial and the ones displayed in your query tool.

Part two: Add additional pizza

Continuing with the scenario, having confirmed the existing order, Elenore wants to add a second pizza. The additional pizza is a "large thin crust with sausage, onions, and mushrooms."

You need a `sale_id` to insert a new pizza into the `pizza` table, which has a foreign key constraint on `sale.sale_id`. Use the value of the `sale_id` you noted at the end of part one for the value of the `pizza.sale_id` in the `INSERT` to add a new large thin crust pizza to the Elenore's order:

```
INSERT INTO pizza
(sale_id, size_id, crust, sauce, price)
VALUES
(50, (SELECT size_id FROM size WHERE size_description = 'Large'), 'Thin', 'Normal',
17.24)
RETURNING pizza_id;
```

Run the `INSERT` statement. The `RETURNING` clause on the end of the `INSERT` returns the `pizza_id` of the newly added pizza:

pizza_id
96

Use this value to add toppings to the new pizza:

```
INSERT INTO pizza_topping
(pizza_id, topping_name)
VALUES
```

```
  (96, 'Sausage'),
  (96, 'Onions'),
  (96, 'Mushrooms');
```

Then, rerun the `SELECT` query to confirm the new pizza with the correct toppings has been added to Elenore's order:

last_name	first_name	street_address	sale_id	pizza_id	size_id	crust	sauce	topping_name
Mamwell	Elenore	561 Claremont Alley	4	4	L	Thin	Normal	Mushrooms
Mamwell	Elenore	561 Claremont Alley	37	49	L	Regular	Normal	Extra Cheese
Mamwell	Elenore	561 Claremont Alley	37	50	L	Regular	Normal	Bacon
Mamwell	Elenore	561 Claremont Alley	37	51	L	Regular	Normal	Tomatoes
Mamwell	Elenore	561 Claremont Alley	50	67	M	Pan	Normal	Ham
Mamwell	Elenore	561 Claremont Alley	**50**	**96**	**L**	**Thin**	**Normal**	**Sausage**
Mamwell	Elenore	561 Claremont Alley	**50**	**96**	**L**	**Thin**	**Normal**	**Onions**
Mamwell	Elenore	561 Claremont Alley	**50**	**96**	**L**	**Thin**	**Normal**	**Mushrooms**

Read the result set carefully. The last three rows represent *one* pizza (`pizza_id: 96`) with three toppings: `Sausage`, `Onions`, and `Mushrooms`.

Part three: Change existing pizza

Next, you imagine a request from Elenore to change the other pizza from a "medium" to a "large" pan pizza with ham.

Since there are no changes to the pizza's toppings, the change from "medium" to "large" can be accomplished with a single `UPDATE` to the `pizza` table. The update is limited in the `WHERE` clause to the value of `pizza_id` you noted earlier at the end of part one when you confirmed the order with Elenore:

```
UPDATE pizza SET
size_id = (SELECT size_id FROM size WHERE size_description = 'Large'),
price = price + 2
WHERE pizza_id = 67;
```

Run the `UPDATE` statement, and then rerun the `SELECT` to confirm your change:

last_name	first_name	street_address	sale_id	pizza_id	size_id	crust	sauce	topping_name

last_name	first_name	street_address	sale_id	pizza_id	size_id	crust	sauce	topping_name
Mamwell	Elenore	561 Claremont Alley	4	4	L	Thin	Normal	Mushrooms
Mamwell	Elenore	561 Claremont Alley	37	49	L	Regular	Normal	Extra Cheese
Mamwell	Elenore	561 Claremont Alley	37	50	L	Regular	Normal	Bacon
Mamwell	Elenore	561 Claremont Alley	37	51	L	Regular	Normal	Tomatoes
Mamwell	Elenore	561 Claremont Alley	50	67	**L**	Pan	Normal	Ham
Mamwell	Elenore	561 Claremont Alley	50	96	L	Thin	Normal	Sausage
Mamwell	Elenore	561 Claremont Alley	50	96	L	Thin	Normal	Onions
Mamwell	Elenore	561 Claremont Alley	50	96	L	Thin	Normal	Mushrooms

Part four: Remove a pizza

Finally, you imagine a scenario where Elenore decides the additional pizza isn't needed after all.

To remove the large thin crust pizza with sausage, onions, and mushrooms, you need to delete the toppings from the `pizza_topping` table first because it has a foreign key constraint on `pizza.pizza_id`. You use the same `pizza_id` value used when adding the toppings in part three in the `WHERE` of the `DELETE`:

```
DELETE FROM pizza_topping
WHERE pizza_id = 96;
```

Then run the `DELETE` statement.

After the toppings have been removed, it's safe to delete the pizza. You'll use the same `pizza_id` value to delete the pizza from the `pizza` table:

```
DELETE FROM pizza
WHERE pizza_id = 96;
```

Run this `DELETE` statement, and rerun the `SELECT` query for the last time to confirm the additional pizza has been removed:

last_name	first_name	street_address	sale_id	pizza_id	size_id	crust	sauce	topping_name
Mamwell	Elenore	561 Claremont Alley	4	4	L	Thin	Normal	Mushrooms

last_name	first_name	street_address	sale_id	pizza_id	size_id	crust	sauce	topping_name
Mamwell	Elenore	561 Claremont Alley	37	49	L	Regular	Normal	Extra Cheese
Mamwell	Elenore	561 Claremont Alley	37	50	L	Regular	Normal	Bacon
Mamwell	Elenore	561 Claremont Alley	37	51	L	Regular	Normal	Tomatoes
Mamwell	Elenore	561 Claremont Alley	50	67	L	Pan	Normal	Ham

The additional pizza is gone, while the size of the pan crust pizza with ham remains large.

Next steps

If you'd like more practice with inserting, updating, and deleting, here are a few ideas:

- Build a complete order, starting with creating a new customer all the way to creating a pizza with at least one topping.
- Change a topping for an existing pizza. For example, swap "Green Peppers" for "Onions", or "Bacon" for "Ham."
- Cancel an order by removing it from the database.
- Notice that the pizza price was hardcoded in the `INSERT` statement, and the price difference was hardcoded in the `UPDATE` statement. Can you write a statement that would `UPDATE pizza.price` based on `size.base_price` and `topping.additional_price`?
- The changes in parts two and three also didn't update `sale.total`. Can you write a statement that adds up the price of all pizzas for an order, and `UPDATE sale.total`?
- Remove the word `LEFT` from the last `JOIN` in the `SELECT` statement used to retrieve Elenore's orders, add a new pizza without toppings to the last order, and then re-query Elenore's orders with the modified `SELECT`. Did the new pizza show up in the result set? If not, why not?