1. **Easy**
   - Write a query to concatenate the `first_name` and `last_name` columns from the `customers` table, separating them with a space.

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM customers;
```

   - Find the starting position of the '@' character in the `email` column of the `customers` table.

```
SELECT POSITION('@' IN email) AS at_position
FROM customers;
```

   - Extract the first 3 characters from the `product_code` column in the `products` table.

```
SELECT SUBSTRING(product_code, 1, 3) AS first_3_chars
FROM products;
```

2. **Moderate**
   - Create a new column called `formatted_phone` that formats phone numbers from the `phone` column in the `customers` table as `(XXX) XXX-XXXX`.

```
SELECT CONCAT('(', SUBSTRING(phone, 1, 3), ') ',
              SUBSTRING(phone, 4, 3), '-',
              SUBSTRING(phone, 7)) AS formatted_phone
FROM customers;
```

   - Write a `CASE` statement that categorizes products in the `products` table into 'Electronics', 'Furniture', and 'Other' based on keywords in the `product_name` column.

```
SELECT product_name,
       CASE
           WHEN LOWER(product_name) LIKE '%electronics%' THEN
'Electronics'
           WHEN LOWER(product_name) LIKE '%furniture%' THEN
'Furniture'
           ELSE 'Other'
```

```
        END AS product_category
FROM products;
```

- Extract the initials (first initial and last initial) from the `first_name` and `last_name` columns in the `employees` table.

```
SELECT CONCAT(UPPER(SUBSTR(first_name, 1, 1)), '.',
              UPPER(SUBSTR(last_name, 1, 1)), '.') AS initials
FROM employees;
```

3. **Challenging**
   - Pad the `order_id` column in the `orders` table with leading zeros to ensure a minimum length of 6 characters.

```
SELECT LPAD(order_id, 6, '0') AS padded_order_id
FROM orders;
```

   - Replace all occurrences of the substring '@example.com' in the `email` column of the `customers` table with an empty string to extract the usernames.

```
SELECT REPLACE(email, '@example.com', '') AS username
FROM customers;
```

   - Write a nested `CASE` statement that categorizes orders in the `orders` table based on the `order_status` and `ship_method` columns (e.g., 'Open', 'In Progress - FedEx', 'In Progress - USPS', 'Closed').

```
SELECT order_id,
       CASE
           WHEN order_status = 'Pending' THEN 'Open'
           WHEN order_status = 'Shipping'
               CASE ship_method
                   WHEN 'FedEx' THEN 'In Progress - FedEx'
                   WHEN 'USPS' THEN 'In Progress - USPS'
                   ELSE 'In Progress - Unknown'
               END
           WHEN order_status = 'Delivered' THEN 'Closed'
           ELSE 'Unknown'
       END AS order_status_text
FROM orders;
```

4. **Advanced**
    - Split the `tags` column in the `products` table into separate rows using the `STRING_TO_TABLE()` function, and select the first 3 tags for each product.

```
SELECT product_name,
       SPLIT_PART(tags, ',', 1) AS tag_1,
       SPLIT_PART(tags, ',', 2) AS tag_2,
       SPLIT_PART(tags, ',', 3) AS tag_3
FROM products;
```

    - Create a function that takes a product description as input and returns the product category ('Camera', 'Lens', 'Tripod', 'Other') based on the presence of keywords in the description.

```
CREATE FUNCTION categorize_product(description TEXT)
RETURNS TEXT AS $$
BEGIN
    CASE
        WHEN LOWER(description) LIKE '%camera%' THEN 'Camera'
        WHEN LOWER(description) LIKE '%lens%' THEN 'Lens'
        WHEN LOWER(description) LIKE '%tripod%' THEN 'Tripod'
        ELSE 'Other'
    END;
END;
$$ LANGUAGE plpgsql;

SELECT product_name, categorize_product(description) AS
product_category
FROM products;
```