Got it, here's the handout with more detailed explanations:

SQL Text Manipulation Handout

1. Working with Text Columns

Selecting Text Columns

To work with text data in SQL, you first need to select the relevant text columns from your tables. This is done using the `SELECT` statement, specifying the column names containing text data.

```
SELECT first_name, last_name
FROM customers;
```

This query will return the `first_name` and `last_name` columns from the `customers` table.

Concatenating Text Columns

Often, you'll need to combine multiple text columns into a single value. This is known as concatenation, and in SQL, you can use the `CONCAT()` function or the `||` operator to achieve this.

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

SELECT first_name || ' ' || last_name AS full_name
FROM customers;
```

Both of these queries will create a new column called `full_name` that combines the `first_name` and `last_name` columns, separating them with a space.

Extracting Substrings

Sometimes, you may need to extract a portion of a text value, such as the first few characters or a specific part of the string. You can use the `SUBSTRING()` or `SUBSTR()`

functions for this purpose.

```
SELECT SUBSTRING(email, 1, 5) AS first_5_chars
FROM customers;

SELECT SUBSTR(phone, 1, 3) AS area_code
FROM customers;
```

The `SUBSTRING()` function takes three arguments: the column name, the starting position (1-based index), and the number of characters to extract. The `SUBSTR()` function works similarly, but the starting position is 1-based.

Converting Text Case

Changing the case of text can be useful for standardizing data or making it more readable. Kotlin provides the `UPPER()` and `LOWER()` functions for this purpose.

```
SELECT UPPER(first_name) AS uppercase_first_name
FROM customers;

SELECT LOWER(description) AS lowercase_description
FROM products;
```

These queries will convert the `first_name` column to all uppercase and the `description` column to all lowercase, respectively.

Trimming Whitespace

Text data can sometimes include leading or trailing whitespace, which can be undesirable. The `TRIM()` function can be used to remove this excess whitespace.

```
SELECT TRIM(address) AS trimmed_address
FROM customers;
```

This query will remove any leading or trailing spaces from the `address` column.

2. Transforming Text Data

Padding Text

You may need to pad text with leading or trailing characters, such as adding zeros to the beginning of a number or underscores to the end of a code. The `LPAD()` and `RPAD()` functions can be used for this purpose.

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

SELECT RPAD(product_code, 10, '_') AS padded_product_code
FROM products;
```

The `LPAD()` function adds leading characters (in this case, '0') to the left side of the `order_id` column, ensuring a minimum length of 6 characters. The `RPAD()` function adds trailing underscores to the right side of the `product_code` column, ensuring a minimum length of 10 characters.

Replacing Text

The `REPLACE()` function can be used to substitute one string with another within a text value.

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

This query will remove the '@example.com' part from the `email` column, effectively extracting the username.

Splitting Text into Rows

Sometimes, you may need to split a single text value into multiple rows, such as when working with comma-separated values. You can use functions like `SPLIT_PART()` or `STRING_TO_TABLE()` for this purpose.

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

```
SELECT *
FROM STRING_TO_TABLE(tags, ',') AS tag
FROM products;
```

The `SPLIT_PART()` function takes three arguments: the column name, the delimiter (in this case, a comma), and the index of the part you want to extract. The `STRING_TO_TABLE()` function splits the entire string into separate rows.

3. Using CASE Expressions

Basic CASE Statement

The `CASE` expression in SQL allows you to implement conditional logic and transform data based on specific conditions. The basic syntax is:

```
CASE WHEN condition THEN result
     WHEN condition THEN result
     ...
     ELSE result
END
```

Here's an example:

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

This will convert the numeric `order_status` values into more user-friendly text labels.

Searching for Patterns with CASE WHEN

You can also use the `CASE WHEN` syntax to check for specific patterns or conditions within the text data.

```
SELECT product_name,
       CASE
           WHEN product_name LIKE 'Camera%' THEN 'Camera'
           WHEN product_name LIKE '%Lens' THEN 'Lens'
           WHEN product_name LIKE '%Tripod' THEN 'Tripod'
           ELSE 'Other'
       END AS product_category
FROM products;
```

This query will categorize products based on the product name, using `LIKE` patterns to check for specific keywords.

Handling NULL Values

The `COALESCE()` and `NULLIF()` functions can be useful when working with nullable text columns.

```
SELECT customer_name,
       COALESCE(phone, 'No Phone Number') AS phone
FROM customers;

SELECT NULLIF(in_stock, 0) AS in_stock
FROM products;
```

`COALESCE()` returns the first non-NULL value from the provided arguments, which can be used to replace NULL values with a default value. `NULLIF()` returns NULL if the two provided values are equal, which can be useful for converting '0' to NULL.

Nested CASE Statements

You can also nest `CASE` statements within each other to handle more complex logic.

```
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;
```

This nested `CASE` statement first checks the `order_status`, and then, for the 'Shipping' status, it checks the `ship_method` to determine the appropriate status text.

4. Practical Examples

Formatting Phone Numbers

Here's an example of how you can format phone numbers by extracting the area code, exchange, and last four digits using `SUBSTRING()`:

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

This will transform a phone number like '5551234567' into '(555) 123-4567'.

Extracting Initials from Full Names

To extract the initials from full names, you can use a combination of `UPPER()` and `SUBSTR()`:

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

This will take the first character of the `first_name` and `last_name` columns, convert them to uppercase, and concatenate them with periods to create the initials.

Categorizing Products by Description

You can use `CASE WHEN` statements to categorize products based on keywords in the product description:

```
SELECT product_name,
       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 AS product_category
FROM products;
```

This will analyze the lowercased product descriptions and assign a category based on the presence of specific keywords.

Handling Mixed-Case and Inconsistent Data

To ensure consistent capitalization in text data, you can combine `UPPER()` and `LOWER()` with `SUBSTR()`:

```
SELECT CONCAT(UPPER(SUBSTR(first_name, 1, 1)), LOWER(SUBSTR(first_name,
2))) AS first_name,
       CONCAT(UPPER(SUBSTR(last_name, 1, 1)), LOWER(SUBSTR(last_name, 2)))
AS last_name
FROM customers;
```

This will capitalize the first letter of the `first_name` and `last_name` columns, while converting the rest of the letters to lowercase, even if the original data was inconsistent.