Let's go through each of the advanced joins with two code examples for each using SQL.

1. CROSS JOIN:

A CROSS JOIN returns the Cartesian product of two tables, meaning it combines each row from the first table with every row from the second table.

Example 1:

```sql
-- Creating two sample tables
CREATE TABLE employees (
    employee_id INT,
    employee_name VARCHAR(50)
);

CREATE TABLE departments (
    department_id INT,
    department_name VARCHAR(50)
);

-- Inserting sample data
INSERT INTO employees VALUES (1, 'Alice'), (2, 'Bob');
INSERT INTO departments VALUES (101, 'HR'), (102, 'IT');

-- Performing a CROSS JOIN
SELECT * FROM employees CROSS JOIN departments;
```

In this example, the result will have all combinations of employee and department rows.

Example 2:

```sql
-- Creating two sample tables
CREATE TABLE table1 (
    column1 INT
);

CREATE TABLE table2 (
    column2 INT
);

-- Inserting sample data
```

```
INSERT INTO table1 VALUES (1), (2);
INSERT INTO table2 VALUES (10), (20);

-- Performing a CROSS JOIN
SELECT * FROM table1 CROSS JOIN table2;
```

This example demonstrates a simple CROSS JOIN between two tables with one column each.

2. SELF JOIN:

A SELF JOIN is a regular join, but the table is joined with itself. It is useful when dealing with hierarchical data or when you need to compare rows within the same table.

Example 1:

```sql
-- Creating a sample table with hierarchical data
CREATE TABLE employees_hierarchy (
    employee_id INT,
    employee_name VARCHAR(50),
    manager_id INT
);

-- Inserting sample data
INSERT INTO employees_hierarchy VALUES (1, 'Alice', NULL), (2, 'Bob', 1), (3, 'Charlie', 2);

-- Performing a SELF JOIN to get manager and employee information
SELECT e.employee_name AS employee, m.employee_name AS manager
FROM employees_hierarchy e
JOIN employees_hierarchy m ON e.manager_id = m.employee_id;
```

In this example, the query retrieves the employee and their respective manager.

Example 2:

```sql
-- Creating a sample table for self-join
CREATE TABLE friends (
    person_id INT,
    friend_id INT
);
```

```
-- Inserting sample data
INSERT INTO friends VALUES (1, 2), (2, 3), (3, 1);

-- Performing a SELF JOIN to find mutual friends
SELECT f1.person_id AS person, f1.friend_id AS friend, f2.friend_id AS mutual_friend
FROM friends f1
JOIN friends f2 ON f1.friend_id = f2.person_id AND f1.person_id = f2.friend_id;
```

This example finds mutual friends between people in the 'friends' table.

3. Non-equijoin:

A Non-equijoin is a type of join that involves comparing columns using operators other than equality (e.g., `>`, `<`, `!=`).

Example 1:

```sql
-- Creating two sample tables
CREATE TABLE products (
    product_id INT,
    product_name VARCHAR(50),
    price INT
);

CREATE TABLE discounts (
    discount_id INT,
    discount_percentage INT
);

-- Inserting sample data
INSERT INTO products VALUES (1, 'Laptop', 1000), (2, 'Phone', 500);
INSERT INTO discounts VALUES (1, 10), (2, 20);

-- Performing a Non-equijoin to find discounted products
SELECT * FROM products, discounts
WHERE products.price > discounts.discount_percentage;
```

In this example, it retrieves products where the price is greater than the discount percentage.

Example 2:

```sql
-- Creating two sample tables
CREATE TABLE employees_salary (
    employee_id INT,
    employee_name VARCHAR(50),
    salary INT
);

CREATE TABLE salary_ranges (
    range_id INT,
    min_salary INT,
    max_salary INT
);

-- Inserting sample data
INSERT INTO employees_salary VALUES (1, 'Alice', 60000), (2, 'Bob', 80000);
INSERT INTO salary_ranges VALUES (1, 50000, 70000), (2, 70001, 90000);

-- Performing a Non-equijoin to find employees within salary ranges
SELECT * FROM employees_salary, salary_ranges
WHERE employees_salary.salary BETWEEN salary_ranges.min_salary AND
salary_ranges.max_salary;
```

In this example, it retrieves employees within specific salary ranges using a Non-equijoin.

Certainly! Let's continue with more examples for advanced joins:

4. NATURAL JOIN:

A NATURAL JOIN is a join that automatically matches the columns with the same name in both tables.

Example 1:

```sql
-- Creating two sample tables
CREATE TABLE employees_info (
    employee_id INT,
    employee_name VARCHAR(50),
    department_id INT
);

CREATE TABLE departments_info (
```

```
    department_id INT,
    department_name VARCHAR(50)
);

-- Inserting sample data
INSERT INTO employees_info VALUES (1, 'Alice', 101), (2, 'Bob', 102);
INSERT INTO departments_info VALUES (101, 'HR'), (102, 'IT');

-- Performing a NATURAL JOIN
SELECT * FROM employees_info NATURAL JOIN departments_info;
```

In this example, the NATURAL JOIN is based on the common column `department_id`.

Example 2:

```sql
-- Creating two sample tables
CREATE TABLE students (
    student_id INT,
    student_name VARCHAR(50),
    grade INT
);

CREATE TABLE courses (
    course_id INT,
    course_name VARCHAR(50),
    grade INT
);

-- Inserting sample data
INSERT INTO students VALUES (1, 'Alice', 85), (2, 'Bob', 92);
INSERT INTO courses VALUES (101, 'Math', 85), (102, 'English', 92);

-- Performing a NATURAL JOIN
SELECT * FROM students NATURAL JOIN courses;
```

This example demonstrates a NATURAL JOIN based on the common column `grade`.

5. ANTI JOIN:

An ANTI JOIN returns rows where there is no match in the second table.

Example 1:

```sql
-- Creating two sample tables
CREATE TABLE customers (
    customer_id INT,
    customer_name VARCHAR(50)
);

CREATE TABLE orders (
    order_id INT,
    customer_id INT,
    order_date DATE
);

-- Inserting sample data
INSERT INTO customers VALUES (1, 'Alice'), (2, 'Bob');
INSERT INTO orders VALUES (101, 1, '2023-01-01'), (102, 3, '2023-01-05');

-- Performing an ANTI JOIN to find customers without orders
SELECT * FROM customers
WHERE customer_id NOT IN (SELECT customer_id FROM orders);
```

In this example, it retrieves customers who haven't placed any orders.

Example 2:

```sql
-- Creating two sample tables
CREATE TABLE employees_departmentA (
    employee_id INT,
    employee_name VARCHAR(50)
);

CREATE TABLE employees_departmentB (
    employee_id INT,
    employee_name VARCHAR(50)
);

-- Inserting sample data
INSERT INTO employees_departmentA VALUES (1, 'Alice'), (2, 'Bob');
INSERT INTO employees_departmentB VALUES (3, 'Charlie'), (4, 'David');
```

```
-- Performing an ANTI JOIN to find employees not in department A
SELECT * FROM employees_departmentB
WHERE employee_id NOT IN (SELECT employee_id FROM employees_departmentA);
```

In this example, it retrieves employees who are not in department A.

These examples showcase different scenarios and use cases for advanced joins in SQL.