- Retrieve Records with NULL Values:

  SELECT * FROM employees WHERE column_name IS NULL;

- Retrieve Records with Non-NULL Values:

  SELECT * FROM employees WHERE column_name IS NOT NULL;

- Use LIKE Operator for Pattern Matching:

  SELECT * FROM employees WHERE column_name LIKE 'pattern%';

- Use BETWEEN Operator for Range:

  SELECT * FROM employees WHERE column_name BETWEEN value1 AND value2;

- Perform Aggregate Functions (SUM, AVG, MAX, MIN):

  SELECT SUM(column_name), AVG(column_name), MAX(column_name), MIN(column_name) FROM employees;

- Concatenate Columns:

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

- Use CASE Statements for Conditional Logic:

```
SELECT employee_id,

CASE

 WHEN age < 30 THEN 'Young'
 WHEN age >= 30 AND age < 50 THEN 'Middle-aged'
 ELSE 'Old'
END AS age_group
FROM employees;
```

- Use DISTINCT with Multiple Columns:

```
SELECT DISTINCT department, location FROM employees;
```

- Retrieve Random Records:

```
SELECT * FROM employees ORDER BY RANDOM() LIMIT 1;
```

- Perform Joins with Multiple Conditions:

```
SELECT * FROM employees
JOIN departments ON employees.department_id =
departments.department_id AND employees.location_id =
departments.location_id;
```

- Retrieve Records with a Date Range:

```
SELECT * FROM employees WHERE hire_date BETWEEN
'start_date' AND 'end_date';
```

- Retrieve Top N Records by Group:

```
SELECT * FROM (
SELECT *, ROW_NUMBER() OVER (PARTITION BY
department ORDER BY salary DESC) AS row_num
FROM employees
) AS ranked
WHERE row_num <= 3;
```

- Use EXISTS to Check for Existence:

```
SELECT * FROM employees WHERE EXISTS (SELECT 1 FROM
salaries WHERE employees.employee_id =
salaries.employee_id);
```

- Use NOT EXISTS to Check for Non-Existence:

```
SELECT * FROM employees WHERE NOT EXISTS (SELECT 1
FROM performance_reviews WHERE
employees.employee_id =
performance_reviews.employee_id);
```

- Retrieve Records from the Last N Days:

```
SELECT * FROM orders WHERE order_date >=
CURRENT_DATE() - INTERVAL 7 DAY;
```

- Use UNION to Combine Results from Multiple Queries:

```
(SELECT product_id, product_name FROM products)
UNION
(SELECT product_id, product_name FROM
discontinued_products);
```

- Use INTERSECT to Find Common Records:

```
(SELECT customer_id FROM new_customers)
INTERSECT
(SELECT customer_id FROM returning_customers);
```

- Use EXCEPT to Find Unique Records:

```
(SELECT product_id FROM current_stock)
EXCEPT
(SELECT product_id FROM pending_orders);
```

- Use TRY_CAST to Safely Cast Data Types:

```
SELECT TRY_CAST(price AS DECIMAL(10,2)) FROM products;
```

- Use STRING_AGG to Concatenate Strings:

```
SELECT order_id, STRING_AGG(product_name, ', ') AS
ordered_products FROM orders GROUP BY order_id;
```

- Use ARRAY_AGG to Aggregate Values into Arrays:

```
SELECT department_id, ARRAY_AGG(employee_name) AS
employees FROM employees GROUP BY department_id;
```

- Retrieve Records from the First N Rows:

```
SELECT * FROM orders FETCH FIRST 10 ROWS ONLY;
```

- Use ROLLUP for Generating Subtotals:

```
SELECT department_id, SUM(sales_amount) AS total_sales
FROM sales
GROUP BY ROLLUP(department_id);
```

- Use CUBE for Generating Cross-tabular Results:

```
SELECT department_id, product_id, SUM(quantity_sold) AS
total_quantity
FROM sales
GROUP BY CUBE(department_id, product_id);
```

- Use CROSS APPLY for Table-Valued Functions:

```
SELECT e.employee_id, e.full_name, s.sales_amount
FROM employees e
CROSS APPLY dbo.GetSalesAmount(e.employee_id) AS s;
```

- Use OUTER APPLY for Correlated Subqueries:

```
SELECT e.employee_id, e.full_name, oa.avg_sales_amount
FROM employees e
OUTER APPLY (
SELECT AVG(sales_amount) AS avg_sales_amount
FROM sales s
WHERE s.employee_id = e.employee_id
) AS oa;
```

- Use CROSS JOIN for Cartesian Products:

```
SELECT e.employee_id, d.department_id
FROM employees e
CROSS JOIN departments d;
```

- Use EXTRACT to Extract Parts of a Date:

```
SELECT order_id, EXTRACT(YEAR FROM order_date) AS
order_year
FROM orders;
```

- Use ROW_NUMBER() for Pagination:

```
SELECT *, ROW_NUMBER() OVER (ORDER BY order_date) AS
row_num

FROM orders
```

```
) AS ranked
 WHERE row_num BETWEEN 11 AND 20;
```