SQL Clauses

1. WHERE Clause

The WHERE clause is used to filter records based on specified conditions.

Syntax:

```
SELECT column1, column2, ...FROM table_name WHERE condition;
```

Examples:

a. Simple comparison:

```
SELECT * FROM employees WHERE salary > 50000;
```

This retrieves all employees with a salary greater than 50,000.

b. Multiple conditions using AND:

```
SELECT * FROM products  WHERE category = 'Electronics' AND price < 1000;
```

This selects all electronic products priced under 1,000.

c. Using OR:

```
SELECT * FROM customers WHERE country = 'USA' OR country = 'Canada';
```

This retrieves customers from either the USA or Canada.

d. Using IN:

```
SELECT * FROM orders WHERE status IN ('Shipped', 'Delivered');
```

This selects orders with a status of either 'Shipped' or 'Delivered'.
e. Using LIKE for pattern matching:

```
SELECT * FROM employees WHERE last_name LIKE 'S%';
```

This retrieves employees whose last name starts with 'S'.

f. Using BETWEEN:

```
SELECT * FROM products WHERE price BETWEEN 100 AND 500;
```

This selects products with prices between 100 and 500, inclusive.

2. DISTINCT Clause

The DISTINCT clause is used to return only distinct (unique) values in the result set.

Syntax:

```
SELECT DISTINCT column1, column2, … FROM table_name;
```

Examples:

a. Single column:

```
SELECT DISTINCT category FROM products;
```

This retrieves a list of unique product categories.

b. Multiple columns:

```
SELECT DISTINCT city, state FROM addresses;
```

This returns unique combinations of city and state from the addresses table.

c. With WHERE clause:

SELECT DISTINCT department FROM employees WHERE salary > 60000;

This retrieves unique departments that have employees earning over 60,000.

d. With COUNT:

SELECT COUNT(DISTINCT customer_id) AS unique_customers FROM orders;

This counts the number of unique customers who have placed orders.

3. AS Clause (Alias)

The AS clause is used to give a table or a column a temporary name (alias).

Syntax:

For columns: SELECT column_name AS alias_name
For tables: SELECT column FROM table_name AS alias_name

Examples:
a. Column alias:

SELECT first_name AS name, last_name AS surname FROM employees;

This renames the columns in the output to 'name' and 'surname'.

b. Table alias:
SELECT e.first_name, d.department_name FROM employees AS e
JOIN departments AS d ON e.department_id = d.department_id;

This uses aliases 'e' for employees and 'd' for departments to make the query more readable.

c. Alias for calculations:

```
SELECT product_name, price, price * 1.1 AS price_with_tax FROM products;
```

This calculates the price with tax and gives it an alias.

d. Alias in WHERE clause:

```
SELECT product_name, price, price * quantity AS total_value FROM order_items
WHERE price * quantity > 1000;
```

Note: You can't use the alias 'total_value' in the WHERE clause here, as it's processed before SELECT.

e. Alias with functions:

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name,
   YEAR(CURDATE()) - YEAR(birth_date) AS age FROM employees;
```

This creates aliases for concatenated names and calculated ages. These clauses can be combined to create more complex and specific queries.

```
SELECT DISTINCT p.category,
      AVG(p.price) AS avg_price
FROM products AS p
JOIN order_items AS oi ON p.product_id = oi.product_id
WHERE oi.order_date >= '2023-01-01'
GROUP BY p.category
HAVING AVG(p.price) > 100
ORDER BY avg_price DESC;
```

This query retrieves distinct product categories with their average prices for items ordered since January 1, 2023, where the average price is over 100, sorted by average price in descending order.