MySQL 8 Windows Functions - Complete Guide with Hotel Domain Examples

Table of Contents

Introduction to Window Functions {#introduction}

Window functions perform calculations across a set of table rows related to the current row. Unlike aggregate functions, they don't group rows into a single output row - they retain all individual rows while adding calculated values.

Syntax:

```sql
function_name([arguments]) OVER (
    [PARTITION BY column1, column2, ...]
    [ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...]
    [ROWS|RANGE BETWEEN ... AND ...]
)
```

Key Components:

- **PARTITION BY**: Divides result set into partitions (like GROUP BY but doesn't collapse rows)
- **ORDER BY**: Defines order within each partition
- **Frame Clause**: Defines subset of partition for calculation

---

Database Setup {#database-setup}

Create Database and Tables

```sql
```

```sql
-- Create Database
CREATE DATABASE hotel_management;
USE hotel_management;

-- Hotels Table
CREATE TABLE hotels (
    hotel_id INT PRIMARY KEY AUTO_INCREMENT,
    hotel_name VARCHAR(100) NOT NULL,
    city VARCHAR(50) NOT NULL,
    country VARCHAR(50) NOT NULL,
    rating DECIMAL(2,1) CHECK (rating >= 1 AND rating <= 5),
    established_year INT
);

-- Rooms Table
CREATE TABLE rooms (
    room_id INT PRIMARY KEY AUTO_INCREMENT,
    hotel_id INT,
    room_number VARCHAR(10) NOT NULL,
    room_type VARCHAR(30) NOT NULL,
    price_per_night DECIMAL(8,2) NOT NULL,
    max_occupancy INT NOT NULL,
    FOREIGN KEY (hotel_id) REFERENCES hotels(hotel_id)
);

-- Customers Table
CREATE TABLE customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15),
    country VARCHAR(50),
    loyalty_level VARCHAR(20) DEFAULT 'Bronze'
);

-- Bookings Table
CREATE TABLE bookings (
    booking_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    room_id INT,
    check_in_date DATE NOT NULL,
    check_out_date DATE NOT NULL,
    total_amount DECIMAL(10,2) NOT NULL,
    booking_status VARCHAR(20) DEFAULT 'Confirmed',
    booking_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (room_id) REFERENCES rooms(room_id)
);

-- Staff Table
CREATE TABLE staff (
    staff_id INT PRIMARY KEY AUTO_INCREMENT,
    hotel_id INT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department VARCHAR(30) NOT NULL,
    salary DECIMAL(8,2) NOT NULL,
    hire_date DATE NOT NULL,
    FOREIGN KEY (hotel_id) REFERENCES hotels(hotel_id)
);

-- Revenue Table (Monthly Revenue per Hotel)
CREATE TABLE monthly_revenue (
    revenue_id INT PRIMARY KEY AUTO_INCREMENT,
    hotel_id INT,
    year INT NOT NULL,
    month INT NOT NULL,
    revenue DECIMAL(12,2) NOT NULL,
    FOREIGN KEY (hotel_id) REFERENCES hotels(hotel_id)
);
```

Insert Sample Data

```
sql
```

```sql
-- Insert Hotels
INSERT INTO hotels (hotel_name, city, country, rating, established_year) VALUES
('Grand Palace Hotel', 'New York', 'USA', 4.8, 1995),
('Ocean View Resort', 'Miami', 'USA', 4.5, 2001),
('Mountain Peak Lodge', 'Denver', 'USA', 4.2, 1988),
('City Center Inn', 'New York', 'USA', 3.9, 2010),
('Luxury Suites', 'Los Angeles', 'USA', 4.7, 2005),
('Seaside Paradise', 'Miami', 'USA', 4.3, 1999),
('Business Hotel', 'Chicago', 'USA', 4.0, 2008),
('Boutique Retreat', 'San Francisco', 'USA', 4.6, 2012);

-- Insert Rooms
INSERT INTO rooms (hotel_id, room_number, room_type, price_per_night, max_occupancy) VALUES
-- Grand Palace Hotel (hotel_id = 1)
(1, '101', 'Standard', 250.00, 2),
(1, '102', 'Standard', 250.00, 2),
(1, '201', 'Deluxe', 350.00, 3),
(1, '301', 'Suite', 500.00, 4),
-- Ocean View Resort (hotel_id = 2)
(2, '101', 'Ocean View', 400.00, 2),
(2, '102', 'Ocean View', 400.00, 2),
(2, '201', 'Premium Suite', 600.00, 4),
-- Mountain Peak Lodge (hotel_id = 3)
(3, '101', 'Cabin', 180.00, 2),
(3, '102', 'Cabin', 180.00, 2),
(3, '201', 'Family Suite', 280.00, 6),
-- City Center Inn (hotel_id = 4)
(4, '101', 'Economy', 150.00, 2),
(4, '102', 'Economy', 150.00, 2),
(4, '201', 'Business', 220.00, 2),
-- Luxury Suites (hotel_id = 5)
(5, '101', 'Luxury', 450.00, 2),
(5, '201', 'Presidential', 800.00, 6);

-- Insert Customers
INSERT INTO customers (first_name, last_name, email, phone, country, loyalty_level) VALUES
('John', 'Smith', 'john.smith@email.com', '+1-555-0101', 'USA', 'Gold'),
('Emma', 'Johnson', 'emma.johnson@email.com', '+1-555-0102', 'USA', 'Silver'),
('Michael', 'Brown', 'michael.brown@email.com', '+1-555-0103', 'USA', 'Bronze'),
('Sarah', 'Davis', 'sarah.davis@email.com', '+1-555-0104', 'Canada', 'Platinum'),
('David', 'Wilson', 'david.wilson@email.com', '+1-555-0105', 'USA', 'Gold'),
('Lisa', 'Anderson', 'lisa.anderson@email.com', '+1-555-0106', 'UK', 'Silver'),
('Robert', 'Taylor', 'robert.taylor@email.com', '+1-555-0107', 'USA', 'Bronze'),
('Jennifer', 'Martinez', 'jennifer.martinez@email.com', '+1-555-0108', 'Mexico', 'Gold');

-- Insert Staff
```

```sql
INSERT INTO staff (hotel_id, first_name, last_name, department, salary, hire_date) VALUES
(1, 'Alice', 'Cooper', 'Front Desk', 45000.00, '2020-01-15'),
(1, 'Bob', 'Johnson', 'Housekeeping', 35000.00, '2019-03-20'),
(1, 'Carol', 'Smith', 'Management', 75000.00, '2018-01-10'),
(2, 'Dave', 'Wilson', 'Front Desk', 42000.00, '2021-06-01'),
(2, 'Eve', 'Brown', 'Maintenance', 40000.00, '2020-08-15'),
(3, 'Frank', 'Davis', 'Front Desk', 38000.00, '2022-01-01'),
(3, 'Grace', 'Miller', 'Housekeeping', 33000.00, '2021-11-10'),
(4, 'Henry', 'Garcia', 'Management', 68000.00, '2019-05-20'),
(5, 'Ivy', 'Martinez', 'Concierge', 55000.00, '2020-09-12');

-- Insert Bookings (2023-2024 data)
INSERT INTO bookings (customer_id, room_id, check_in_date, check_out_date, total_amount, booking_status, booking_d
(1, 1, '2024-01-15', '2024-01-18', 750.00, 'Completed', '2024-01-10 14:30:00'),
(2, 5, '2024-01-20', '2024-01-23', 1200.00, 'Completed', '2024-01-15 09:45:00'),
(3, 8, '2024-02-01', '2024-02-03', 360.00, 'Completed', '2024-01-25 16:20:00'),
(1, 3, '2024-02-10', '2024-02-14', 1400.00, 'Completed', '2024-02-05 11:15:00'),
(4, 13, '2024-03-05', '2024-03-08', 1350.00, 'Completed', '2024-03-01 10:30:00'),
(5, 2, '2024-03-15', '2024-03-17', 500.00, 'Confirmed', '2024-03-10 13:45:00'),
(6, 6, '2024-04-01', '2024-04-04', 1200.00, 'Completed', '2024-03-25 15:20:00'),
(7, 11, '2024-04-10', '2024-04-12', 300.00, 'Cancelled', '2024-04-05 12:10:00'),
(8, 14, '2024-05-01', '2024-05-05', 3200.00, 'Completed', '2024-04-20 14:00:00'),
(2, 7, '2024-05-15', '2024-05-18', 1800.00, 'Completed', '2024-05-10 09:30:00'),
(3, 4, '2024-06-01', '2024-06-03', 1000.00, 'Confirmed', '2024-05-25 16:45:00'),
(1, 5, '2024-07-15', '2024-07-20', 2000.00, 'Completed', '2024-07-10 11:20:00');

-- Insert Monthly Revenue Data
INSERT INTO monthly_revenue (hotel_id, year, month, revenue) VALUES
-- 2023 Data
(1, 2023, 1, 125000.00), (1, 2023, 2, 110000.00), (1, 2023, 3, 135000.00),
(1, 2023, 4, 145000.00), (1, 2023, 5, 160000.00), (1, 2023, 6, 175000.00),
(2, 2023, 1, 95000.00), (2, 2023, 2, 88000.00), (2, 2023, 3, 105000.00),
(2, 2023, 4, 120000.00), (2, 2023, 5, 140000.00), (2, 2023, 6, 155000.00),
(3, 2023, 1, 65000.00), (3, 2023, 2, 60000.00), (3, 2023, 3, 70000.00),
(3, 2023, 4, 80000.00), (3, 2023, 5, 90000.00), (3, 2023, 6, 100000.00),
-- 2024 Data
(1, 2024, 1, 130000.00), (1, 2024, 2, 115000.00), (1, 2024, 3, 140000.00),
(1, 2024, 4, 150000.00), (1, 2024, 5, 165000.00), (1, 2024, 6, 180000.00),
(2, 2024, 1, 100000.00), (2, 2024, 2, 92000.00), (2, 2024, 3, 110000.00),
(2, 2024, 4, 125000.00), (2, 2024, 5, 145000.00), (2, 2024, 6, 160000.00),
(3, 2024, 1, 68000.00), (3, 2024, 2, 63000.00), (3, 2024, 3, 73000.00),
(3, 2024, 4, 83000.00), (3, 2024, 5, 93000.00), (3, 2024, 6, 103000.00);
```

Window Function Types {#window-function-types}

1. ROW_NUMBER()

Assigns unique sequential integers to rows within a partition.

Example: Rank rooms by price within each hotel

```sql
SELECT
    h.hotel_name,
    r.room_number,
    r.room_type,
    r.price_per_night,
    ROW_NUMBER() OVER (PARTITION BY h.hotel_id ORDER BY r.price_per_night DESC) as price_rank
FROM rooms r
JOIN hotels h ON r.hotel_id = h.hotel_id
ORDER BY h.hotel_name, price_rank;
```

2. RANK() and DENSE_RANK()

- **RANK()**: Assigns same rank to equal values, skips next ranks
- **DENSE_RANK()**: Assigns same rank to equal values, no gaps

Example: Rank hotels by rating

```sql
SELECT
    hotel_name,
    city,
    rating,
    RANK() OVER (ORDER BY rating DESC) as hotel_rank,
    DENSE_RANK() OVER (ORDER BY rating DESC) as hotel_dense_rank
FROM hotels
ORDER BY rating DESC;
```

3. NTILE(n)

Divides rows into n approximately equal groups.

Example: Divide customers into 3 groups based on total spending

```sql
```

```
SELECT
    c.first_name,
    c.last_name,
    SUM(b.total_amount) as total_spent,
    NTILE(3) OVER (ORDER BY SUM(b.total_amount) DESC) as spending_tier
FROM customers c
JOIN bookings b ON c.customer_id = b.customer_id
WHERE b.booking_status = 'Completed'
GROUP BY c.customer_id, c.first_name, c.last_name
ORDER BY total_spent DESC;
```

4. LAG() and LEAD()

Access data from previous (LAG) or next (LEAD) rows.

Example: Compare monthly revenue with previous month

sql

```sql
SELECT
    h.hotel_name,
    mr.year,
    mr.month,
    mr.revenue,
    LAG(mr.revenue) OVER (PARTITION BY mr.hotel_id ORDER BY mr.year, mr.month) as prev_month_revenue,
    mr.revenue - LAG(mr.revenue) OVER (PARTITION BY mr.hotel_id ORDER BY mr.year, mr.month) as revenue_change
FROM monthly_revenue mr
JOIN hotels h ON mr.hotel_id = h.hotel_id
ORDER BY h.hotel_name, mr.year, mr.month;
```

5. FIRST_VALUE() and LAST_VALUE()

Return first or last value in the window frame.

Example: Show highest and lowest room prices per hotel

sql

```sql
SELECT
    h.hotel_name,
    r.room_number,
    r.room_type,
    r.price_per_night,
    FIRST_VALUE(r.price_per_night) OVER (
        PARTITION BY r.hotel_id
        ORDER BY r.price_per_night DESC
        ROWS UNBOUNDED PRECEDING
    ) as highest_price,
    LAST_VALUE(r.price_per_night) OVER (
        PARTITION BY r.hotel_id
        ORDER BY r.price_per_night DESC
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
    ) as lowest_price
FROM rooms r
JOIN hotels h ON r.hotel_id = h.hotel_id
ORDER BY h.hotel_name, r.price_per_night DESC;
```

6. Aggregate Functions as Window Functions

SUM(), COUNT(), AVG(), MIN(), MAX() can be used with OVER clause.

Example: Running total of bookings per customer

```sql
SELECT
    c.first_name,
    c.last_name,
    b.booking_date,
    b.total_amount,
    SUM(b.total_amount) OVER (
        PARTITION BY c.customer_id
        ORDER BY b.booking_date
        ROWS UNBOUNDED PRECEDING
    ) as running_total,
    COUNT(*) OVER (
        PARTITION BY c.customer_id
        ORDER BY b.booking_date
        ROWS UNBOUNDED PRECEDING
    ) as booking_count
FROM customers c
JOIN bookings b ON c.customer_id = b.customer_id
WHERE b.booking_status = 'Completed'
ORDER BY c.customer_id, b.booking_date;
```

7. PERCENT_RANK() and CUME_DIST()

- **PERCENT_RANK()**: Relative rank as percentage (0-1)
- **CUME_DIST()**: Cumulative distribution

Example: Staff salary percentiles

```sql
SELECT
    s.first_name,
    s.last_name,
    h.hotel_name,
    s.department,
    s.salary,
    PERCENT_RANK() OVER (ORDER BY s.salary) as salary_percentile,
    CUME_DIST() OVER (ORDER BY s.salary) as cumulative_distribution
FROM staff s
JOIN hotels h ON s.hotel_id = h.hotel_id
ORDER BY s.salary DESC;
```

Practical Examples {#practical-examples}

Example 1: Hotel Performance Dashboard

```sql
```

```sql
-- Monthly revenue comparison with year-over-year growth
SELECT
    h.hotel_name,
    mr.year,
    mr.month,
    mr.revenue,
    LAG(mr.revenue, 12) OVER (
        PARTITION BY mr.hotel_id
        ORDER BY mr.year, mr.month
    ) as same_month_last_year,
    ROUND(
        ((mr.revenue - LAG(mr.revenue, 12) OVER (
            PARTITION BY mr.hotel_id
            ORDER BY mr.year, mr.month
        )) / LAG(mr.revenue, 12) OVER (
            PARTITION BY mr.hotel_id
            ORDER BY mr.year, mr.month
        )) * 100, 2
    ) as yoy_growth_percent
FROM monthly_revenue mr
JOIN hotels h ON mr.hotel_id = h.hotel_id
WHERE mr.year IN (2023, 2024)
ORDER BY h.hotel_name, mr.year, mr.month;
```

Example 2: Customer Segmentation Analysis

```
sql
```

```
-- Customer value analysis with segmentation
SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    c.loyalty_level,
    COUNT(b.booking_id) as total_bookings,
    SUM(b.total_amount) as total_spent,
    AVG(b.total_amount) as avg_booking_value,
    NTILE(4) OVER (ORDER BY SUM(b.total_amount) DESC) as value_quartile,
    CASE
        WHEN NTILE(4) OVER (ORDER BY SUM(b.total_amount) DESC) = 1 THEN 'VIP Customer'
        WHEN NTILE(4) OVER (ORDER BY SUM(b.total_amount) DESC) = 2 THEN 'High Value'
        WHEN NTILE(4) OVER (ORDER BY SUM(b.total_amount) DESC) = 3 THEN 'Medium Value'
        ELSE 'Low Value'
    END as customer_segment
FROM customers c
LEFT JOIN bookings b ON c.customer_id = b.customer_id AND b.booking_status = 'Completed'
GROUP BY c.customer_id, c.first_name, c.last_name, c.loyalty_level
HAVING COUNT(b.booking_id) > 0
ORDER BY total_spent DESC;
```

Example 3: Room Utilization Analysis

```sql
-- Room occupancy analysis with rankings
SELECT
    h.hotel_name,
    r.room_number,
    r.room_type,
    COUNT(b.booking_id) as bookings_count,
    SUM(DATEDIFF(b.check_out_date, b.check_in_date)) as total_nights_booked,
    RANK() OVER (PARTITION BY r.hotel_id ORDER BY COUNT(b.booking_id) DESC) as popularity_rank,
    PERCENT_RANK() OVER (ORDER BY COUNT(b.booking_id) DESC) as popularity_percentile
FROM hotels h
JOIN rooms r ON h.hotel_id = r.hotel_id
LEFT JOIN bookings b ON r.room_id = b.room_id AND b.booking_status = 'Completed'
GROUP BY h.hotel_id, h.hotel_name, r.room_id, r.room_number, r.room_type
ORDER BY h.hotel_name, popularity_rank;
```

Practice Problems {#practice-problems}

Problem 1: Revenue Trends

Write a query to show each hotel's monthly revenue along with:

- 3-month moving average
- Percentage change from previous month
- Rank within each year

Problem 2: Staff Analysis

Create a query that shows:

- Each staff member's salary rank within their department
- Salary percentile across all staff
- Difference from department average salary

Problem 3: Customer Booking Patterns

Analyze customer booking behavior:

- Number each customer's bookings chronologically
- Calculate days between consecutive bookings
- Identify the customer's most expensive and cheapest bookings

Problem 4: Hotel Comparison

Compare hotels by:

- Average room price rank
- Total revenue rank
- Customer satisfaction rank (based on rating)

Problem 5: Seasonal Analysis

Analyze seasonal patterns:

- Identify peak and low seasons for each hotel
- Calculate quarter-over-quarter growth
- Rank months by total industry revenue

Solutions {#solutions}

Solution 1: Revenue Trends

```
sql
```

```sql
SELECT
    h.hotel_name,
    mr.year,
    mr.month,
    mr.revenue,
    -- 3-month moving average
    ROUND(AVG(mr.revenue) OVER (
        PARTITION BY mr.hotel_id
        ORDER BY mr.year, mr.month
        ROWS 2 PRECEDING
    ), 2) as moving_avg_3month,
    -- Percentage change from previous month
    ROUND(
        ((mr.revenue - LAG(mr.revenue) OVER (
            PARTITION BY mr.hotel_id
            ORDER BY mr.year, mr.month
        )) / LAG(mr.revenue) OVER (
            PARTITION BY mr.hotel_id
            ORDER BY mr.year, mr.month
        )) * 100, 2
    ) as month_change_percent,
    -- Rank within each year
    RANK() OVER (
        PARTITION BY mr.hotel_id, mr.year
        ORDER BY mr.revenue DESC
    ) as yearly_rank
FROM monthly_revenue mr
JOIN hotels h ON mr.hotel_id = h.hotel_id
ORDER BY h.hotel_name, mr.year, mr.month;
```

Solution 2: Staff Analysis

```sql
```

```
SELECT
    h.hotel_name,
    s.first_name,
    s.last_name,
    s.department,
    s.salary,
    -- Salary rank within department
    RANK() OVER (
        PARTITION BY s.department
        ORDER BY s.salary DESC
    ) as dept_salary_rank,
    -- Salary percentile across all staff
    ROUND(PERCENT_RANK() OVER (ORDER BY s.salary) * 100, 1) as salary_percentile,
    -- Department average salary
    ROUND(AVG(s.salary) OVER (PARTITION BY s.department), 2) as dept_avg_salary,
    -- Difference from department average
    ROUND(s.salary - AVG(s.salary) OVER (PARTITION BY s.department), 2) as diff_from_avg
FROM staff s
JOIN hotels h ON s.hotel_id = h.hotel_id
ORDER BY s.department, s.salary DESC;
```

Solution 3: Customer Booking Patterns

```
sql
```

```
SELECT
    c.first_name,
    c.last_name,
    b.booking_date,
    b.total_amount,
    -- Number bookings chronologically
    ROW_NUMBER() OVER (
        PARTITION BY c.customer_id
        ORDER BY b.booking_date
    ) as booking_sequence,
    -- Days between consecutive bookings
    DATEDIFF(
        b.booking_date,
        LAG(b.booking_date) OVER (
            PARTITION BY c.customer_id
            ORDER BY b.booking_date
        )
    ) as days_since_last_booking,
    -- Most expensive booking for this customer
    FIRST_VALUE(b.total_amount) OVER (
        PARTITION BY c.customer_id
        ORDER BY b.total_amount DESC
        ROWS UNBOUNDED PRECEDING
    ) as max_booking_amount,
    -- Cheapest booking for this customer
    FIRST_VALUE(b.total_amount) OVER (
        PARTITION BY c.customer_id
        ORDER BY b.total_amount ASC
        ROWS UNBOUNDED PRECEDING
    ) as min_booking_amount
FROM customers c
JOIN bookings b ON c.customer_id = b.customer_id
WHERE b.booking_status = 'Completed'
ORDER BY c.customer_id, b.booking_date;
```

Solution 4: Hotel Comparison

```
sql
```

```
SELECT
    h.hotel_id,
    h.hotel_name,
    h.city,
    h.rating,
    ROUND(AVG(r.price_per_night), 2) as avg_room_price,
    SUM(mr.revenue) as total_revenue,
    -- Average room price rank
    RANK() OVER (ORDER BY AVG(r.price_per_night) DESC) as price_rank,
    -- Total revenue rank
    RANK() OVER (ORDER BY SUM(mr.revenue) DESC) as revenue_rank,
    -- Customer satisfaction rank (rating)
    RANK() OVER (ORDER BY h.rating DESC) as satisfaction_rank,
    -- Overall performance score
    (
        RANK() OVER (ORDER BY AVG(r.price_per_night) DESC) +
        RANK() OVER (ORDER BY SUM(mr.revenue) DESC) +
        RANK() OVER (ORDER BY h.rating DESC)
    ) / 3 as avg_rank_score
FROM hotels h
LEFT JOIN rooms r ON h.hotel_id = r.hotel_id
LEFT JOIN monthly_revenue mr ON h.hotel_id = mr.hotel_id
GROUP BY h.hotel_id, h.hotel_name, h.city, h.rating
ORDER BY avg_rank_score;
```

Solution 5: Seasonal Analysis

```
sql
```

```
SELECT
    year,
    month,
    SUM(revenue) as total_industry_revenue,
    -- Rank months by total revenue
    RANK() OVER (PARTITION BY year ORDER BY SUM(revenue) DESC) as monthly_rank,
    -- Quarter calculation
    CASE
        WHEN month IN (1,2,3) THEN 'Q1'
        WHEN month IN (4,5,6) THEN 'Q2'
        WHEN month IN (7,8,9) THEN 'Q3'
        ELSE 'Q4'
    END as quarter,
    -- Quarter-over-quarter growth (simplified for available data)
    LAG(SUM(revenue), 3) OVER (ORDER BY year, month) as revenue_3months_ago,
    ROUND(
        ((SUM(revenue) - LAG(SUM(revenue), 3) OVER (ORDER BY year, month)) /
         LAG(SUM(revenue), 3) OVER (ORDER BY year, month)) * 100, 2
    ) as qoq_growth_percent
FROM monthly_revenue
GROUP BY year, month
ORDER BY year, month;
```

Key Takeaways

1. **Window functions don't reduce rows** - they add calculated columns to existing rows
2. **PARTITION BY** is like GROUP BY but doesn't collapse rows
3. **ORDER BY** within OVER clause determines calculation order
4. **Frame clauses** (ROWS/RANGE) define which rows to include in calculations
5. **Multiple window functions** can be used in the same query for different insights
6. **Performance**: Window functions are generally more efficient than correlated subqueries

Frame Clause Examples

```sql
```

```
-- Current row and all preceding rows
ROWS UNBOUNDED PRECEDING

-- Current row and 2 preceding rows
ROWS 2 PRECEDING

-- Between 1 preceding and 1 following row
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING

-- All rows in partition
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
```

Best Practices

1. Use appropriate indexes on columns in PARTITION BY and ORDER BY
2. Limit window functions in subqueries for better performance
3. Consider using CTEs for complex window function queries
4. Test with large datasets to ensure performance
5. Use EXPLAIN to analyze query execution plans