Advanced SQL Guidebook — MySQL (Outputs Included)

DDL/DML • Joins • CASE • Windows • CTEs • Sets • Dates/Strings

Basics: SELECT/WHERE/ORDER BY/LIMIT

SQL:

```
SELECT customer_id, first_name || ' ' || last_name AS full_name, email, created_at
       FROM customers
       WHERE created_at BETWEEN '2025-09-01' AND '2025-09-30'
       ORDER BY created_at DESC, customer_id DESC
       LIMIT 10;
```

Output:

```
 customer_id   full_name                   email created_at
           4  Dina Singh  dina.singh@example.com 2025-09-20
           3 Carlos Diaz carlos.diaz@example.com 2025-09-15
           2     Ben Kim     ben.kim@example.com 2025-09-10
           1    Asha Rao    asha.rao@example.com 2025-09-01
```

Aggregates + GROUP BY + HAVING

SQL:

```
SELECT p.category,
              SUM(oi.quantity)                    AS total_units,
              ROUND(SUM(oi.quantity * oi.unit_price), 2) AS gross_revenue
       FROM order_items oi
       JOIN products p ON p.product_id = oi.product_id
       GROUP BY p.category
       HAVING total_units > 0
       ORDER BY gross_revenue DESC;
```

Output:

```
 category  total_units  gross_revenue
   Snacks           15          36.35
   Pantry            3          16.27
Beverages            3          14.87
  Produce           15          11.45
    Dairy            6           6.94
```

INNER JOIN: paid orders

SQL:

```
SELECT o.order_id, o.status, o.order_date, p.amount, p.method
       FROM orders o
       JOIN payments p ON p.order_id = o.order_id
       ORDER BY o.order_date, o.order_id;
```

Output:

```
 order_id    status order_date  amount method
```

Advanced SQL Guidebook — MySQL (Outputs Included)

DDL/DML • Joins • CASE • Windows • CTEs • Sets • Dates/Strings

```
      101      PAID 2025-09-21   17.45   CARD
      102 DELIVERED 2025-09-22    3.95   CASH
      103      PAID 2025-09-22   10.25   CARD
      105   SHIPPED 2025-10-01   15.67   BANK
      107      PAID 2025-10-03    7.16 WALLET
```

LEFT JOIN anti-join: orders without payments

SQL:

```
SELECT o.order_id, o.status, o.order_date
       FROM orders o
       LEFT JOIN payments p ON p.order_id = o.order_id
       WHERE p.payment_id IS NULL
       ORDER BY o.order_id;
```

Output:

```
 order_id    status order_date
      104 CANCELLED 2025-09-25
      106    PLACED 2025-10-02
```

RIGHT JOIN (MySQL) — output via SQLite equivalence

SQL:

```
-- RIGHT JOIN in MySQL:
       -- SELECT o.order_id, o.status, p.amount
       -- FROM payments p RIGHT JOIN orders o ON o.order_id = p.order_id
       -- WHERE p.payment_id IS NULL
       -- ORDER BY o.order_id;
       -- SQLite equivalent (used for output):
       SELECT o.order_id, o.status, p.amount
       FROM orders o LEFT JOIN payments p ON p.order_id = o.order_id
       WHERE p.payment_id IS NULL
       ORDER BY o.order_id;
```

Output:

```
                          error
'NoneType' object is not iterable
```

CASE WHEN buckets

SQL:

```
SELECT o.order_id, o.status,
              CASE
                WHEN o.status IN ('PLACED','PAID') THEN 'OPEN'
                WHEN o.status IN ('SHIPPED','DELIVERED') THEN 'CLOSED'
                WHEN o.status = 'CANCELLED' THEN 'VOID'
                ELSE 'UNKNOWN'
              END AS lifecycle_bucket
       FROM orders o
       ORDER BY o.order_id;
```

Output:

```
 order_id    status lifecycle_bucket
      101      PAID             OPEN
      102 DELIVERED           CLOSED
      103      PAID             OPEN
      104 CANCELLED             VOID
      105   SHIPPED           CLOSED
      106    PLACED             OPEN
      107      PAID             OPEN
```

Window: per-customer recency (ROW_NUMBER)

SQL:

```
WITH order_totals AS (
         SELECT o.order_id, o.customer_id, o.order_date,
                SUM(oi.quantity * oi.unit_price) AS order_revenue
         FROM orders o JOIN order_items oi ON oi.order_id = o.order_id
```

Advanced SQL Guidebook — MySQL (Outputs Included)

DDL/DML • Joins • CASE • Windows • CTEs • Sets • Dates/Strings

```
        GROUP BY o.order_id, o.customer_id, o.order_date
      )
      SELECT c.customer_id,
             c.first_name || ' ' || c.last_name AS customer,
             o.order_id, o.order_date, o.order_revenue,
             ROW_NUMBER() OVER (PARTITION BY c.customer_id ORDER BY o.order_date DESC, o.order_id DESC) AS rn
      FROM order_totals o JOIN customers c ON c.customer_id = o.customer_id
      ORDER BY c.customer_id, rn;
```

Output:

customer_id	customer	order_id	order_date	order_revenue	rn
1	Asha Rao	102	2025-09-22	3.95	1
1	Asha Rao	101	2025-09-21	17.45	2
2	Ben Kim	104	2025-09-25	23.90	1
2	Ben Kim	103	2025-09-22	10.25	2
3	Carlos Diaz	106	2025-10-02	7.50	1
3	Carlos Diaz	105	2025-10-01	15.67	2
4	Dina Singh	107	2025-10-03	7.16	1

Window: store revenue rank (DENSE_RANK)

SQL:

```
SELECT s.store_id, s.store_name,
             ROUND(SUM(oi.quantity * oi.unit_price),2) AS store_revenue,
             DENSE_RANK() OVER (ORDER BY SUM(oi.quantity * oi.unit_price) DESC) AS revenue_rank
      FROM stores s
      JOIN orders o  ON o.store_id = s.store_id
      JOIN order_items oi ON oi.order_id = o.order_id
      GROUP BY s.store_id, s.store_name
      ORDER BY revenue_rank, s.store_id;
```

Output:

store_id	store_name	store_revenue	revenue_rank
1	Downtown Durham	36.06	1
2	Chapel Hill Central	34.15	2
3	RTP Express	15.67	3

Window: LAG to detect SKU price changes

SQL:

```
WITH sku_prices AS (
        SELECT oi.order_id, p.sku, oi.unit_price, o.order_date
        FROM order_items oi
        JOIN products p ON p.product_id = oi.product_id
        JOIN orders o   ON o.order_id = oi.order_id
        WHERE p.sku = 'SKU-TBR-002'
      )
      SELECT order_id, sku, unit_price, order_date,
             LAG(unit_price) OVER (ORDER BY order_date, order_id) AS prior_price
      FROM sku_prices
      ORDER BY order_date, order_id;
```

Output:

order_id	sku	unit_price	order_date	prior_price
101	SKU-TBR-002	2.49	2025-09-21	NaN
104	SKU-TBR-002	2.39	2025-09-25	2.49
107	SKU-TBR-002	2.49	2025-10-03	2.39

Recursive CTE calendar (Q11 fixed) + COALESCE

SQL:

```
WITH RECURSIVE calendar(d) AS (
        VALUES(date('2025-09-20'))
        UNION ALL
        SELECT date(d, '+1 day') FROM calendar WHERE d < date('2025-10-05')
      ),
      order_totals AS (
        SELECT date(o.order_date) AS d,
```

Advanced SQL Guidebook — MySQL (Outputs Included)

DDL/DML • Joins • CASE • Windows • CTEs • Sets • Dates/Strings

```
               ROUND(SUM(oi.quantity * oi.unit_price),2) AS revenue
        FROM orders o JOIN order_items oi ON oi.order_id = o.order_id
        GROUP BY date(o.order_date)
      )
      SELECT c.d AS date, COALESCE(ot.revenue, 0) AS revenue
      FROM calendar c LEFT JOIN order_totals ot ON ot.d = c.d
      ORDER BY c.d;
```

Output:

```
      date  revenue
2025-09-20     0.00
2025-09-21    17.45
2025-09-22    14.20
2025-09-23     0.00
2025-09-24     0.00
2025-09-25    23.90
2025-09-26     0.00
2025-09-27     0.00
2025-09-28     0.00
2025-09-29     0.00
2025-09-30     0.00
2025-10-01    15.67
2025-10-02     7.50
2025-10-03     7.16
2025-10-04     0.00
2025-10-05     0.00
```

UNION and EXCEPT/anti-join

SQL:

```
-- UNION: shoppers at store 1 or 2
      WITH s1 AS (SELECT DISTINCT customer_id FROM orders WHERE store_id = 1),
           s2 AS (SELECT DISTINCT customer_id FROM orders WHERE store_id = 2)
      SELECT 'UNION' AS op, customer_id FROM s1
      UNION
      SELECT 'UNION', customer_id FROM s2
      ORDER BY customer_id;

      -- EXCEPT (SQLite uses EXCEPT; MySQL emulates with NOT EXISTS, here we show EXCEPT for brevity)
      SELECT 'EXCEPT' AS op, customer_id FROM s1
      EXCEPT
      SELECT 'EXCEPT', customer_id FROM s2
      ORDER BY customer_id;
```

Output:

```
   op  customer_id
UNION            1
UNION            2
UNION            3
UNION            4
```