Challenge 1: Inventory Management

Q1. Identify products that need reordering across all .

```
WITH stock AS (
  SELECT
    p.productid,
    p.productname,
    p.reorderlevel,
    IFNULL(SUM(i.quantity), 0) AS totalqty
  FROM products p
  LEFT JOIN inventory i ON i.productid = p.productid
  GROUP BY p.productid, p.productname, p.reorderlevel
)
SELECT
  productid,
  productname,
  totalqty,
  reorderlevel,
  GREATEST(reorderlevel - totalqty, 0) AS deficittoreorder
FROM stock
WHERE totalqty < reorderlevel
ORDER BY deficittoreorder DESC, productname;
```

Q2. Calculate the cost of restocking all products below reorder .

```
WITH stock AS (
  SELECT
    p.productid,
    p.productname,
    p.reorderlevel,
    p.costprice,
    IFNULL(SUM(i.quantity), 0) AS totalqty
  FROM products p
  LEFT JOIN inventory i ON i.productid = p.productid
  GROUP BY p.productid, p.productname, p.reorderlevel, p.costprice
),
needs AS (
```

```
  SELECT
    productid,
    productname,
    costprice,
    GREATEST(reorderlevel - totalqty, 0) AS deficit
  FROM stock
  WHERE totalqty < reorderlevel
)
SELECT
  productid,
  productname,
  deficit AS unitstoorder,
  costprice,
  (deficit * costprice) AS restockcost
FROM needs
ORDER BY restockcost DESC, productname;
-- Grand total: SELECT SUM(deficit * costprice) AS totalrestockcost FROM needs;
```

Q3. Recommend warehouse transfers to balance Query.

```
WITH bywh AS (
  SELECT productid, warehouseid, IFNULL(quantity, 0) AS qty
  FROM inventory
),
totals AS (
  SELECT
    productid,
    SUM(qty) AS totalqty,
    COUNT() AS whcount,
    CEIL(SUM(qty) / COUNT()) AS targetperwh
  FROM bywh
  GROUP BY productid
),
deltas AS (
  SELECT
    b.productid,
    b.warehouseid,
    b.qty,
```

```
    t.targetperwh,
    (b.qty - t.targetperwh) AS delta
  FROM bywh b
  JOIN totals t USING (productid)
),
surplus AS (
  SELECT productid, warehouseid AS fromwarehouseid, delta AS surplusqty
  FROM deltas
  WHERE delta > 0
),
deficit AS (
  SELECT productid, warehouseid AS towarehouseid, -delta AS deficitqty
  FROM deltas
  WHERE delta < 0
)
SELECT
  s.productid,
  s.fromwarehouseid,
  d.towarehouseid,
  LEAST(s.surplusqty, d.deficitqty) AS suggestedtransferqty
FROM surplus s
JOIN deficit d
  ON d.productid = s.productid
WHERE LEAST(s.surplusqty, d.deficitqty) > 0
ORDER BY s.productid, suggestedtransferqty DESC, s.fromwarehouseid, d.towarehouseid
LIMIT 200;
```

Challenge 2: Customer Analytics

Q4. Create a customer cohort analysis by registration

```
WITH cohort AS (
  SELECT
    c.customerid,
    DATEFORMAT(c.registrationdate, '%Y-%m-01') AS cohortmonth
  FROM customers c
),
activity AS (
```

```
  SELECT
    o.customerid,
    DATEFORMAT(o.orderdate, '%Y-%m-01') AS activitymonth
  FROM orders o
)
SELECT
  cohort.cohortmonth,
  activity.activitymonth,
  TIMESTAMPDIFF(MONTH, cohort.cohortmonth, activity.activitymonth) AS monthssincecohort,
  COUNT(DISTINCT cohort.customerid) AS activecustomers
FROM cohort
JOIN activity ON activity.customerid = cohort.customerid
GROUP BY cohort.cohortmonth, activity.activitymonth
ORDER BY cohort.cohortmonth, activity.activitymonth;
```

Q5. Calculate customer churn rate

```
WITH RECURSIVE months AS (
  SELECT DATEFORMAT(LEAST(
          (SELECT MIN(registrationdate) FROM customers),
          (SELECT MIN(orderdate) FROM orders)
        ), '%Y-%m-01') AS monthstart
  UNION ALL
  SELECT DATEFORMAT(DATEADD(monthstart, INTERVAL 1 MONTH), '%Y-%m-01')
  FROM months
  WHERE monthstart < DATEFORMAT(CURDATE(), '%Y-%m-01')
),
ordersbymonth AS (
  SELECT DATEFORMAT(o.orderdate, '%Y-%m-01') AS monthstart, o.customerid
  FROM orders o
  GROUP BY monthstart, o.customerid
),
activeprev AS (
  SELECT DATEFORMAT(DATEADD(m.monthstart, INTERVAL 1 MONTH), '%Y-%m-01') AS monthstart,
        obm.customerid
  FROM months m
  JOIN ordersbymonth obm ON obm.monthstart = m.monthstart
),
```

```
activecurr AS (
  SELECT m.monthstart, obm.customerid
  FROM months m
  LEFT JOIN ordersbymonth obm ON obm.monthstart = m.monthstart
),
churned AS (
  SELECT ap.monthstart, COUNT(DISTINCT ap.customerid) AS churnedcustomers
  FROM activeprev ap
  LEFT JOIN activecurr ac
    ON ac.monthstart = ap.monthstart AND ac.customerid = ap.customerid
  WHERE ac.customerid IS NULL
  GROUP BY ap.monthstart
),
base AS (
  SELECT DATEFORMAT(DATEADD(monthstart, INTERVAL -1 MONTH), '%Y-%m-01') AS monthstart,
       COUNT(DISTINCT customerid) AS activeprevmonth
  FROM ordersbymonth
  GROUP BY DATEFORMAT(DATEADD(monthstart, INTERVAL -1 MONTH), '%Y-%m-01')
)
SELECT
  b.monthstart,
  b.activeprevmonth,
  IFNULL(c.churnedcustomers, 0) AS churnedcustomers,
  CASE WHEN b.activeprevmonth > 0
      THEN ROUND(100 * IFNULL(c.churnedcustomers, 0) / b.activeprevmonth, 2)
      ELSE 0 END AS churnratepct
FROM base b
LEFT JOIN churned c USING (monthstart)
ORDER BY b.month_start;
```

Q6. Identify customers likely to upgrade loyalty tiers (last 90 days, rule-based).

```
WITH spend90d AS (
  SELECT o.customerid, SUM(o.totalamount) AS spend90d
  FROM orders o
  WHERE o.orderdate >= DATESUB(CURDATE(), INTERVAL 90 DAY)
  GROUP BY o.customerid
)
```

```
SELECT
  c.customerid,
  CONCAT(c.firstname, ' ', c.lastname) AS customername,
  c.loyaltytier AS currenttier,
  CASE c.loyaltytier
    WHEN 'Bronze' THEN 'Silver'
    WHEN 'Silver' THEN 'Gold'
    WHEN 'Gold'   THEN 'Platinum'
    ELSE NULL
  END AS nexttier,
  s.spend90d
FROM customers c
JOIN spend90d s ON s.customerid = c.customerid
WHERE (c.loyaltytier = 'Bronze'  AND s.spend90d >= 500)
   OR (c.loyaltytier = 'Silver'  AND s.spend90d >= 2000)
   OR (c.loyaltytier = 'Gold'    AND s.spend90d >= 5000)
ORDER BY FIELD(currenttier, 'Bronze','Silver','Gold'), s.spend_90d DESC;
```

Challenge 3: Revenue Optimization

Q7. Find the most profitable product combinations (pairs in the same order).

```
WITH pairprofit AS (
  SELECT
    LEAST(oi1.productid, oi2.productid) AS productida,
    GREATEST(oi1.productid, oi2.productid) AS productidb,
    oi1.orderid,
    SUM(
      ((oi1.unitprice oi1.quantity - IFNULL(oi1.discount,0)) - (p1.costprice oi1.quantity)) +
      ((oi2.unitprice oi2.quantity - IFNULL(oi2.discount,0)) - (p2.costprice oi2.quantity))
    ) AS pairprofitinorder
  FROM orderitems oi1
  JOIN orderitems oi2
    ON oi1.orderid = oi2.orderid
   AND oi1.productid < oi2.productid
  JOIN products p1 ON p1.productid = oi1.productid
  JOIN products p2 ON p2.productid = oi2.productid
  GROUP BY productida, productidb, oi1.orderid
```

```
)
SELECT
  productida,
  productidb,
  COUNT(DISTINCT orderid) AS orderswithpair,
  SUM(pairprofitinorder) AS totalpairprofit
FROM pairprofit
GROUP BY productida, productidb
HAVING COUNT(DISTINCT orderid) >= 3
ORDER BY totalpairprofit DESC, orderswithpair DESC
LIMIT 50;
```

Q8. Analyze discount effectiveness on revenue (bucketed)

```
WITH items AS (
  SELECT
    oi.orderid,
    oi.productid,
    oi.quantity,
    oi.unitprice,
    IFNULL(oi.discount, 0) AS discount,
    (oi.unitprice oi.quantity - IFNULL(oi.discount,0)) AS netrevenue,
    ((oi.unitprice oi.quantity - IFNULL(oi.discount,0))) - (p.costprice oi.quantity) AS grossmargin,
    CASE
      WHEN (oi.unitprice oi.quantity + IFNULL(oi.discount,0)) > 0
        THEN IFNULL(oi.discount,0) / (oi.unitprice oi.quantity + IFNULL(oi.discount,0))
      ELSE 0
    END AS discountpct
  FROM orderitems oi
  JOIN products p ON p.productid = oi.productid
),
bucketed AS (
  SELECT
    CASE
      WHEN discountpct < 0.05 THEN '0–5%'
      WHEN discountpct < 0.10 THEN '5–10%'
      WHEN discountpct < 0.15 THEN '10–15%'
      WHEN discountpct < 0.20 THEN '15–20%'
```

```
    WHEN discountpct < 0.25 THEN '20–25%'
    WHEN discountpct < 0.30 THEN '25–30%'
    WHEN discountpct < 0.40 THEN '30–40%'
    WHEN discountpct < 0.50 THEN '40–50%'
    ELSE '50%+'
  END AS discountbucket,
  netrevenue,
  grossmargin,
  quantity
 FROM items
)
SELECT
 discountbucket,
 COUNT() AS lines,
 SUM(quantity) AS units,
 ROUND(SUM(netrevenue), 2) AS revenue,
 ROUND(SUM(grossmargin), 2) AS grossmargin,
 CASE WHEN SUM(netrevenue) > 0
     THEN ROUND(100 * SUM(grossmargin) / SUM(netrevenue), 2)
     ELSE NULL END AS marginratepct
FROM bucketed
GROUP BY discountbucket
ORDER BY
 CASE discountbucket
  WHEN '0–5%' THEN 1 WHEN '5–10%' THEN 2 WHEN '10–15%' THEN 3
  WHEN '15–20%' THEN 4 WHEN '20–25%' THEN 5 WHEN '25–30%' THEN 6
  WHEN '30–40%' THEN 7 WHEN '40–50%' THEN 8 ELSE 9 END;
```

Q9. Calculate revenue per warehouse.

```
SELECT
 sh.warehouseid,
 w.warehousename,
 ROUND(SUM(oi.unitprice oi.quantity - IFNULL(oi.discount,0)), 2) AS revenue,
 ROUND(SUM( (oi.unitprice oi.quantity - IFNULL(oi.discount,0)) - (p.costprice * oi.quantity) ), 2) AS
grossmargin,
 COUNT(DISTINCT oi.orderid) AS ordersfulfilled
FROM shipments sh
```

```
JOIN orderitems oi ON oi.orderid = sh.orderid
JOIN products p ON p.productid = oi.productid
JOIN warehouses w ON w.warehouseid = sh.warehouseid
GROUP BY sh.warehouseid, w.warehouse_name
ORDER BY revenue DESC;
```

Challenge 4: Performance Tuning

Q10. Optimize the slowest queries using EXPLAIN / performance schema.

```
-- Find slowest normalized queries
SELECT
  DIGESTTEXT AS samplequery,
  COUNTSTAR AS calls,
  ROUND(SUMTIMERWAIT/1e12, 3) AS totaltimesec,
  ROUND(AVGTIMERWAIT/1e12, 3) AS avgtimesec,
  SUMROWSSENT AS rowssent
FROM performanceschema.eventsstatementssummarybydigest
ORDER BY avgtimesec DESC
LIMIT 10;

-- Example plan
EXPLAIN ANALYZE
SELECT
  p.productname,
  COUNT(oi.orderitemid) AS timesordered,
  SUM(oi.quantity) AS totalquantity
FROM products p
LEFT JOIN orderitems oi ON p.productid = oi.productid
GROUP BY p.productid, p.productname
ORDER BY timesordered DESC;
```

Q11. Create appropriate indexes.

```
CREATE INDEX IF NOT EXISTS idxorderscustomerdate ON orders(customerid, orderdate);
CREATE INDEX IF NOT EXISTS idxorderitemsorder ON orderitems(orderid);
CREATE INDEX IF NOT EXISTS idxorderitemsproduct ON orderitems(productid);

CREATE INDEX IF NOT EXISTS idxinventoryproduct ON inventory(productid);
```

CREATE INDEX IF NOT EXISTS idx*inventory*warehouse ON inventory(warehouse*id);*
*CREATE UNIQUE INDEX IF NOT EXISTS unique*product*warehouse ON inventory(product*id,
warehouse*id);*

*CREATE INDEX IF NOT EXISTS idx*shipment*sorder ON shipments(order*id);
CREATE INDEX IF NOT EXISTS idx*shipments*warehouse ON shipments(warehouse_id);

Q12. Rewrite a subquery using JOINs for better performance.

-- Products ordered in the last 90 days
SELECT DISTINCT
  p.product*id, p.product*name
FROM products p
JOIN order*items oi ON oi.product*id = p.product*id*
*JOIN orders o ON o.order*id = oi.order*id*
*WHERE o.order*date >= DATE_SUB(CURDATE(), INTERVAL 90 DAY);