1. Scalar UDF: Final meal price after discount

```
DELIMITER $$

CREATE FUNCTION fn_discounted_meal_price(
  p_itemid CHAR(5),
  p_discount DECIMAL(4,2)
)
RETURNS DECIMAL(6,2)
DETERMINISTIC
BEGIN
  DECLARE base_price DECIMAL(6,2);

  SELECT price INTO base_price
  FROM items
  WHERE itemid = p_itemid;

  RETURN base_price * (1 - p_discount);
END$$

DELIMITER ;
```

---

2. Scalar UDF: Average number of ingredients per meal

```
DELIMITER $$

CREATE FUNCTION fn_avg_ingredients_per_meal()
RETURNS DECIMAL(5,2)
DETERMINISTIC
BEGIN
  DECLARE avg_count DECIMAL(5,2);

  SELECT AVG(ingredient_count)
  INTO avg_count
  FROM (
    SELECT COUNT(*) AS ingredient_count
    FROM madewith
    GROUP BY itemid
  ) t;

  RETURN avg_count;
END$$
```

```
DELIMITER ;
```

---

3. View: Meals with highest and lowest prices

```
CREATE VIEW vw_highest_lowest_priced_meals AS
SELECT *
FROM items
WHERE price = (SELECT MAX(price) FROM items)
  OR price = (SELECT MIN(price) FROM items);
```

---

4. Stored Procedure: Top N highest and lowest priced meals

```
DELIMITER $$

CREATE PROCEDURE sp_top_n_high_low_meals(IN p_n INT)
BEGIN
  (
    SELECT itemid, name, price, 'HIGHEST' AS category
    FROM items
    ORDER BY price DESC
    LIMIT p_n
  )
  UNION ALL
  (
    SELECT itemid, name, price, 'LOWEST' AS category
    FROM items
    ORDER BY price ASC
    LIMIT p_n
  );
END$$

DELIMITER ;
```

---

5. Trigger: Allow insert only if price within ±3 of average

```
DELIMITER $$

CREATE TRIGGER trg_validate_item_insert
```

```
BEFORE INSERT ON items
FOR EACH ROW
BEGIN
  DECLARE avg_price DECIMAL(6,2);

  SELECT AVG(price) INTO avg_price FROM items;

  IF NEW.price NOT BETWEEN avg_price - 3 AND avg_price + 3 THEN
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT = 'Meal price outside allowed range';
  END IF;
END$$

DELIMITER ;
```

---

6. Trigger: Recalculate meal price when quantity changes

```
DELIMITER $$

CREATE TRIGGER trg_update_price_on_quantity_change
AFTER UPDATE ON madewith
FOR EACH ROW
BEGIN
  UPDATE items it
  SET it.price = (
     SELECT 2 * SUM(m.quantity * i.unitprice)
     FROM madewith m
     JOIN ingredients i ON m.ingredientid = i.ingredientid
     WHERE m.itemid = it.itemid
  )
  WHERE it.itemid = NEW.itemid;
END$$

DELIMITER ;
```

---

7. Trigger: Prevent ingredient price > 0.90

```
DELIMITER $$

CREATE TRIGGER trg_check_ingredient_price
BEFORE INSERT ON ingredients
FOR EACH ROW
```

```
BEGIN
  IF NEW.unitprice > 0.90 THEN
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT = 'Ingredient price cannot exceed 0.90';
  END IF;
END$$

DELIMITER ;
```

---

8. Scalar UDF: Total ingredient cost of a meal

```
DELIMITER $$

CREATE FUNCTION fn_meal_ingredient_cost(p_itemid CHAR(5))
RETURNS DECIMAL(8,2)
DETERMINISTIC
BEGIN
  DECLARE total_cost DECIMAL(8,2);

  SELECT SUM(m.quantity * i.unitprice)
  INTO total_cost
  FROM madewith m
  JOIN ingredients i ON m.ingredientid = i.ingredientid
  WHERE m.itemid = p_itemid;

  RETURN IFNULL(total_cost, 0);
END$$

DELIMITER ;
```

---

9. Trigger: Update meal prices when ingredient price changes (100% markup)

```
DELIMITER $$

CREATE TRIGGER trg_update_price_on_ingredient_change
AFTER UPDATE ON ingredients
FOR EACH ROW
BEGIN
  UPDATE items it
  SET it.price = (
     SELECT 2 * SUM(m.quantity * i.unitprice)
```

```
    FROM madewith m
    JOIN ingredients i ON m.ingredientid = i.ingredientid
    WHERE m.itemid = it.itemid
  )
  WHERE it.itemid IN (
    SELECT itemid FROM madewith WHERE ingredientid = NEW.ingredientid
  );
END$$

DELIMITER ;
```

---

10. Scalar UDF: Vendor-wise total ingredient supply cost

```
DELIMITER $$

CREATE FUNCTION fn_vendor_total_cost(p_vendorid CHAR(5))
RETURNS DECIMAL(8,2)
DETERMINISTIC
BEGIN
  DECLARE total_cost DECIMAL(8,2);

  SELECT SUM(unitprice)
  INTO total_cost
  FROM ingredients
  WHERE vendorid = p_vendorid;

  RETURN IFNULL(total_cost, 0);
END$$

DELIMITER ;
```

---

11. Trigger: Store previous price & reject zero price

```
DELIMITER $$

CREATE TRIGGER trg_backup_price
BEFORE UPDATE ON items
FOR EACH ROW
BEGIN
  IF NEW.price = 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Price cannot be zero';
```

```
  END IF;

  SET NEW.prevprice = OLD.price;
END$$

DELIMITER ;
```

---

12. Stored Procedure: Discount meals above threshold

```
DELIMITER $$

CREATE PROCEDURE sp_discount_meals(
  IN p_threshold DECIMAL(6,2),
  IN p_discount DECIMAL(4,2)
)
BEGIN
  SELECT itemid,
       name,
       price AS original_price,
       CASE
          WHEN price > p_threshold THEN price * (1 - p_discount)
          ELSE price
       END AS discounted_price
  FROM items;
END$$

DELIMITER ;
```

---

13. Trigger: Prevent deletion of referenced meals

```
DELIMITER $$

CREATE TRIGGER trg_prevent_meal_delete
BEFORE DELETE ON items
FOR EACH ROW
BEGIN
  IF EXISTS (
     SELECT 1 FROM madewith WHERE itemid = OLD.itemid
  ) THEN
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT = 'Cannot delete meal referenced in madewith';
  END IF;
```

```
END$$

DELIMITER ;
```

---

14. Scalar UDF: Count vendors supplying a meal

```
DELIMITER $$

CREATE FUNCTION fn_vendor_count_for_meal(p_itemid CHAR(5))
RETURNS INT
DETERMINISTIC
BEGIN
  DECLARE vendor_count INT;

  SELECT COUNT(DISTINCT i.vendorid)
  INTO vendor_count
  FROM madewith m
  JOIN ingredients i ON m.ingredientid = i.ingredientid
  WHERE m.itemid = p_itemid;

  RETURN vendor_count;
END$$

DELIMITER ;
```

---

15. Trigger-based audit mechanism for price changes

```
DELIMITER $$

CREATE TRIGGER trg_audit_price_change
AFTER UPDATE ON items
FOR EACH ROW
BEGIN
  IF OLD.price <> NEW.price THEN
     INSERT INTO item_price_audit(itemid, old_price, new_price)
     VALUES (OLD.itemid, OLD.price, NEW.price);
  END IF;
END$$

DELIMITER ;
```