```
create database eyewear;
use eyewear;

-- Disable foreign key checks temporarily to allow creation/recreation in any order
SET FOREIGN_KEY_CHECKS = 0;

-- ----------------------------------------------------
-- Phase 1: Create Core Dimension Tables
-- These tables define the core "who, what, when, where" and have no external FK dependencies.
-- ----------------------------------------------------

-- 1. Table `dim_date`
-- Central date dimension.
CREATE TABLE IF NOT EXISTS `dim_date` (
  `date_id` DATE NOT NULL,
  `day` INT NULL,
  `month` INT NULL,
  `quarter` INT NULL,
  `year` INT NULL,
  PRIMARY KEY (`date_id`)
) ENGINE = InnoDB;

-- 2. Table `dim_campaign`
-- Details about marketing campaigns.
CREATE TABLE IF NOT EXISTS `dim_campaign` (
  `campaign_id` VARCHAR(50) NOT NULL,
  `campaign_name` VARCHAR(255) NULL,
  `channel` VARCHAR(100) NULL,
  `campaign_cost` DECIMAL(10,2) NULL,
  PRIMARY KEY (`campaign_id`)
) ENGINE = InnoDB;

-- 3. Table `dim_customer`
-- Demographic and loyalty information about customers.
CREATE TABLE IF NOT EXISTS `dim_customer` (
  `customer_id` VARCHAR(50) NOT NULL,
  `age` INT NULL,
  `gender` VARCHAR(10) NULL,
  `location` VARCHAR(100) NULL,
  `loyalty_tier` VARCHAR(50) NULL,
  PRIMARY KEY (`customer_id`)
) ENGINE = InnoDB;

-- 4. Table `dim_product`
-- Details about products sold.
CREATE TABLE IF NOT EXISTS `dim_product` (
  `product_id` VARCHAR(50) NOT NULL,
  `sku` VARCHAR(100) UNIQUE NULL,
  `name` VARCHAR(255) NULL,
  `brand` VARCHAR(100) NULL,
  `type` VARCHAR(100) NULL,
  `price` DECIMAL(10,2) NULL,
  PRIMARY KEY (`product_id`)
) ENGINE = InnoDB;

-- 5. Table `dim_store`
-- Information about retail store locations.
CREATE TABLE IF NOT EXISTS `dim_store` (
  `store_id` VARCHAR(50) NOT NULL,
  `city` VARCHAR(100) NULL,
```

```
  `region` VARCHAR(100) NULL,
  `type` VARCHAR(50) NULL,
  PRIMARY KEY (`store_id`)
) ENGINE = InnoDB;


-- -----------------------------------------------------
-- Phase 2: Create Dependent Dimension/Auxiliary Tables
-- These tables have foreign keys that reference tables created in Phase 1.
-- -----------------------------------------------------

-- 6. Table `dim_product_cost` (SCD Type 2 Dimension)
-- Tracks historical cost information for products.
CREATE TABLE IF NOT EXISTS `dim_product_cost` (
  `product_id` VARCHAR(50) NOT NULL,
  `cost_per_unit` DECIMAL(10,2) NULL,
  `sale_price_per_unit` DECIMAL(10,2) NULL,
  `gross_profit_per_unit` DECIMAL(10,2) NULL,
  `supplier` VARCHAR(255) NULL,
  `last_updated` DATE NULL,
  `valid_from_date` DATE NOT NULL, -- Crucial for SCD Type 2
  `valid_to_date` DATE NULL,
  PRIMARY KEY (`product_id`, `valid_from_date`), -- Composite PK for SCD Type 2
  INDEX `fk_dim_product_cost_product_id_idx` (`product_id` ASC),
  CONSTRAINT `fk_dim_product_cost_product_id`
    FOREIGN KEY (`product_id`)
    REFERENCES `dim_product` (`product_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;

-- 7. Table `weather_data`
-- External data, linked to dim_date for date context.
CREATE TABLE IF NOT EXISTS `weather_data` (
  `date` DATE NOT NULL,
  `city` VARCHAR(100) NOT NULL,
  `avg_temperature_celsius` DECIMAL(5,2) NULL,
  `precipitation_mm` DECIMAL(5,2) NULL,
  `humidity_percent` DECIMAL(5,2) NULL,
  `wind_speed_kph` DECIMAL(5,2) NULL,
  `weather_condition` VARCHAR(100) NULL,
  PRIMARY KEY (`date`, `city`),
  INDEX `fk_weather_data_date_idx` (`date` ASC),
  CONSTRAINT `fk_weather_data_date`
    FOREIGN KEY (`date`)
    REFERENCES `dim_date` (`date_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;

-- 8. Table `external_cost_data`
-- Stores external operational costs per store per month.
CREATE TABLE IF NOT EXISTS `external_cost_data` (
  `store_id` VARCHAR(50) NOT NULL,
  `month` DATE NOT NULL, -- Represents the first day of the month for linking to dim_date
  `operational_cost` DECIMAL(10,2) NULL,
  PRIMARY KEY (`store_id`, `month`),
  INDEX `fk_external_cost_data_store_id_idx` (`store_id` ASC),
  INDEX `fk_external_cost_data_month_idx` (`month` ASC),
  CONSTRAINT `fk_external_cost_data_store_id`
```

```
    FOREIGN KEY (`store_id`)
    REFERENCES `dim_store` (`store_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_external_cost_data_month`
    FOREIGN KEY (`month`)
    REFERENCES `dim_date` (`date_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;


-- -----------------------------------------------------
-- Phase 3: Create Fact Tables
-- These tables record business events and link to the dimensions.
-- -----------------------------------------------------

-- 9. Table `sales_fact`
-- Core fact table for sales transactions.
CREATE TABLE IF NOT EXISTS `sales_fact` (
  `sale_id` VARCHAR(50) NOT NULL,
  `store_id` VARCHAR(50) NOT NULL,
  `product_id` VARCHAR(50) NOT NULL,
  `customer_id` VARCHAR(50) NOT NULL,
  `date_id` DATE NOT NULL,
  `quantity` INT NULL,
  `discount_applied` DECIMAL(10,2) NULL,
  `revenue` DECIMAL(10,2) NULL,
  PRIMARY KEY (`sale_id`, `product_id`), -- Assuming sale_id + product_id is unique for an item in a sale
  INDEX `fk_sales_fact_store_id_idx` (`store_id` ASC),
  INDEX `fk_sales_fact_product_id_idx` (`product_id` ASC),
  INDEX `fk_sales_fact_customer_id_idx` (`customer_id` ASC),
  INDEX `fk_sales_fact_date_id_idx` (`date_id` ASC),
  CONSTRAINT `fk_sales_fact_store_id`
    FOREIGN KEY (`store_id`)
    REFERENCES `dim_store` (`store_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_sales_fact_product_id`
    FOREIGN KEY (`product_id`)
    REFERENCES `dim_product` (`product_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_sales_fact_customer_id`
    FOREIGN KEY (`customer_id`)
    REFERENCES `dim_customer` (`customer_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_sales_fact_date_id`
    FOREIGN KEY (`date_id`)
    REFERENCES `dim_date` (`date_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;

-- 10. Table `return_fact`
-- Records product returns, linked to sales if applicable.
CREATE TABLE IF NOT EXISTS `return_fact` (
  `return_id` VARCHAR(50) NOT NULL,
  `sale_id` VARCHAR(50) NOT NULL, -- Links to original sale transaction
```

```
  `product_id` VARCHAR(50) NOT NULL,
  `customer_id` VARCHAR(50) NOT NULL,
  `store_id` VARCHAR(50) NOT NULL,
  `return_date` DATE NOT NULL,
  `reason` VARCHAR(255) NULL,
  `refund_amount` DECIMAL(10,2) NULL,
  PRIMARY KEY (`return_id`),
  INDEX `fk_return_fact_sale_id_idx` (`sale_id` ASC),
  INDEX `fk_return_fact_product_id_idx` (`product_id` ASC),
  INDEX `fk_return_fact_customer_id_idx` (`customer_id` ASC),
  INDEX `fk_return_fact_store_id_idx` (`store_id` ASC),
  INDEX `fk_return_fact_return_date_idx` (`return_date` ASC),
  CONSTRAINT `fk_return_fact_sale_id`
    FOREIGN KEY (`sale_id`)
    REFERENCES `sales_fact` (`sale_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_return_fact_product_id`
    FOREIGN KEY (`product_id`)
    REFERENCES `dim_product` (`product_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_return_fact_customer_id`
    FOREIGN KEY (`customer_id`)
    REFERENCES `dim_customer` (`customer_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_return_fact_store_id`
    FOREIGN KEY (`store_id`)
    REFERENCES `dim_store` (`store_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_return_fact_return_date`
    FOREIGN KEY (`return_date`)
    REFERENCES `dim_date` (`date_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;

-- 11. Table `inventory_fact`
-- Records inventory levels.
CREATE TABLE IF NOT EXISTS `inventory_fact` (
  `inventory_id` VARCHAR(50) NOT NULL,
  `store_id` VARCHAR(50) NOT NULL,
  `product_id` VARCHAR(50) NOT NULL,
  `stock_on_hand` INT NULL,
  `reorder_level` INT NULL,
  `last_restocked` DATE NOT NULL,
  PRIMARY KEY (`inventory_id`),
  INDEX `fk_inventory_fact_store_id_idx` (`store_id` ASC),
  INDEX `fk_inventory_fact_product_id_idx` (`product_id` ASC),
  INDEX `fk_inventory_fact_last_restocked_idx` (`last_restocked` ASC),
  CONSTRAINT `fk_inventory_fact_store_id`
    FOREIGN KEY (`store_id`)
    REFERENCES `dim_store` (`store_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_inventory_fact_product_id`
    FOREIGN KEY (`product_id`)
    REFERENCES `dim_product` (`product_id`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_inventory_fact_last_restocked`
    FOREIGN KEY (`last_restocked`)
    REFERENCES `dim_date` (`date_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;

-- 12. Table `campaign_fact`
-- Records campaign conversions, linked to sales if applicable.
CREATE TABLE IF NOT EXISTS `campaign_fact` (
  `order_id` VARCHAR(50) NOT NULL,
  `sale_id` VARCHAR(50) NULL, -- Optional link to sales_fact
  `campaign_id` VARCHAR(50) NOT NULL,
  `customer_id` VARCHAR(50) NOT NULL,
  `conversion_date` DATE NOT NULL,
  PRIMARY KEY (`order_id`, `campaign_id`, `customer_id`),
  INDEX `fk_campaign_fact_campaign_id_idx` (`campaign_id` ASC),
  INDEX `fk_campaign_fact_customer_id_idx` (`customer_id` ASC),
  INDEX `fk_campaign_fact_conversion_date_idx` (`conversion_date` ASC),
  INDEX `fk_campaign_fact_sale_id_idx` (`sale_id` ASC),
  CONSTRAINT `fk_campaign_fact_campaign_id`
    FOREIGN KEY (`campaign_id`)
    REFERENCES `dim_campaign` (`campaign_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_campaign_fact_customer_id`
    FOREIGN KEY (`customer_id`)
    REFERENCES `dim_customer` (`customer_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_campaign_fact_conversion_date`
    FOREIGN KEY (`conversion_date`)
    REFERENCES `dim_date` (`date_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_campaign_fact_sale_id`
    FOREIGN KEY (`sale_id`)
    REFERENCES `sales_fact` (`sale_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;

-- 13. Table `customer_session_data`
-- Records customer website/app session details.
CREATE TABLE IF NOT EXISTS `customer_session_data` (
  `session_id` VARCHAR(50) NOT NULL,
  `customer_id` VARCHAR(50) NOT NULL,
  `session_date` DATE NOT NULL,
  `session_duration_min` INT NULL,
  `device_type` VARCHAR(50) NULL,
  `products_viewed` INT NULL,
  `pages_visited` INT NULL,
  `time_of_day` VARCHAR(50) NULL, -- UPDATED: To VARCHAR for time ranges
  `purchase_made` BOOLEAN NULL,
  `session_type` VARCHAR(50) NULL,
  PRIMARY KEY (`session_id`),
  INDEX `fk_customer_session_data_customer_id_idx` (`customer_id` ASC),
  INDEX `fk_customer_session_data_session_date_idx` (`session_date` ASC),
```

```
  CONSTRAINT `fk_customer_session_data_customer_id`
    FOREIGN KEY (`customer_id`)
    REFERENCES `dim_customer` (`customer_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_customer_session_data_session_date`
    FOREIGN KEY (`session_date`)
    REFERENCES `dim_date` (`date_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;

-- 14. Table `footfall_data`
-- Records store footfall metrics.
CREATE TABLE IF NOT EXISTS `footfall_data` (
  `date` DATE NOT NULL,
  `store_id` VARCHAR(50) NOT NULL,
  `total_footfall` INT NULL,
  `conversion_rate_percent` DECIMAL(5,2) NULL,
  `avg_stay_minutes` DECIMAL(5,2) NULL,
  `peak_hour` VARCHAR(50) NULL, -- UPDATED: To VARCHAR for time ranges
  PRIMARY KEY (`date`, `store_id`),
  INDEX `fk_footfall_data_date_idx` (`date` ASC),
  INDEX `fk_footfall_data_store_id_idx` (`store_id` ASC),
  CONSTRAINT `fk_footfall_data_date`
    FOREIGN KEY (`date`)
    REFERENCES `dim_date` (`date_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_footfall_data_store_id`
    FOREIGN KEY (`store_id`)
    REFERENCES `dim_store` (`store_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;

-- 15. Table `restock_orders`
-- Tracks inventory restock orders.
CREATE TABLE IF NOT EXISTS `restock_orders` (
  `order_id` VARCHAR(50) NOT NULL,
  `store_id` VARCHAR(50) NOT NULL,
  `product_id` VARCHAR(50) NOT NULL,
  `order_date` DATE NOT NULL,
  `expected_delivery_date` DATE NULL,
  `actual_delivery_date` DATE NULL,
  PRIMARY KEY (`order_id`),
  INDEX `fk_restock_orders_store_id_idx` (`store_id` ASC),
  INDEX `fk_restock_orders_product_id_idx` (`product_id` ASC),
  INDEX `fk_restock_orders_order_date_idx` (`order_date` ASC),
  INDEX `fk_restock_orders_expected_delivery_date_idx` (`expected_delivery_date` ASC),
  INDEX `fk_restock_orders_actual_delivery_date_idx` (`actual_delivery_date` ASC),
  CONSTRAINT `fk_restock_orders_store_id`
    FOREIGN KEY (`store_id`)
    REFERENCES `dim_store` (`store_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_restock_orders_product_id`
    FOREIGN KEY (`product_id`)
    REFERENCES `dim_product` (`product_id`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_restock_orders_order_date`
    FOREIGN KEY (`order_date`)
    REFERENCES `dim_date` (`date_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_restock_orders_expected_delivery_date`
    FOREIGN KEY (`expected_delivery_date`)
    REFERENCES `dim_date` (`date_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_restock_orders_actual_delivery_date`
    FOREIGN KEY (`actual_delivery_date`)
    REFERENCES `dim_date` (`date_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;


-- Re-enable foreign key checks after all tables are created
SET FOREIGN_KEY_CHECKS = 1;


-- # DATA QUALITY CHECKS - TABLE ROW COUNTS

SELECT 'sales_fact' AS table_name, COUNT(*) AS row_count FROM sales_fact
UNION ALL
SELECT 'return_fact', COUNT(*) FROM return_fact
UNION ALL
SELECT 'inventory_fact', COUNT(*) FROM inventory_fact
UNION ALL
SELECT 'campaign_fact', COUNT(*) FROM campaign_fact
UNION ALL
SELECT 'restock_orders', COUNT(*) FROM restock_orders
UNION ALL
SELECT 'customer_session_data', COUNT(*) FROM customer_session_data
UNION ALL
SELECT 'footfall_data', COUNT(*) FROM footfall_data
UNION ALL
SELECT 'dim_product_cost', COUNT(*) FROM dim_product_cost
UNION ALL
SELECT 'weather_data', COUNT(*) FROM weather_data
UNION ALL
SELECT 'dim_store', COUNT(*) FROM dim_store
UNION ALL
SELECT 'dim_product', COUNT(*) FROM dim_product
UNION ALL
SELECT 'dim_date', COUNT(*) FROM dim_date
UNION ALL
SELECT 'dim_customer', COUNT(*) FROM dim_customer
UNION ALL
SELECT 'dim_campaign', COUNT(*) FROM dim_campaign
UNION ALL
SELECT 'external_cost_data', COUNT(*) FROM external_cost_data;


-- # DATA QUALITY CHECKS - NULL VALUES FOR ALL TABLES

-- 1. sales_fact
SELECT
```

```
    'sales_fact' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN sale_id IS NULL OR sale_id = '' THEN 1 ELSE 0 END) AS null_sale_id,
    SUM(CASE WHEN store_id IS NULL OR store_id = '' THEN 1 ELSE 0 END) AS null_store_id,
    SUM(CASE WHEN product_id IS NULL OR product_id = '' THEN 1 ELSE 0 END) AS null_product_id,
    SUM(CASE WHEN customer_id IS NULL OR customer_id = '' THEN 1 ELSE 0 END) AS null_customer_id,
    SUM(CASE WHEN date_id IS NULL THEN 1 ELSE 0 END) AS null_date_id,
    SUM(CASE WHEN quantity IS NULL THEN 1 ELSE 0 END) AS null_quantity,
    SUM(CASE WHEN discount_applied IS NULL THEN 1 ELSE 0 END) AS null_discount_applied,
    SUM(CASE WHEN revenue IS NULL THEN 1 ELSE 0 END) AS null_revenue
FROM sales_fact;

-- 2. return_fact
SELECT
    'return_fact' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN return_id IS NULL OR return_id = '' THEN 1 ELSE 0 END) AS null_return_id,
    SUM(CASE WHEN sale_id IS NULL OR sale_id = '' THEN 1 ELSE 0 END) AS null_sale_id,
    SUM(CASE WHEN product_id IS NULL OR product_id = '' THEN 1 ELSE 0 END) AS null_product_id,
    SUM(CASE WHEN customer_id IS NULL OR customer_id = '' THEN 1 ELSE 0 END) AS null_customer_id,
    SUM(CASE WHEN store_id IS NULL OR store_id = '' THEN 1 ELSE 0 END) AS null_store_id,
    SUM(CASE WHEN return_date IS NULL THEN 1 ELSE 0 END) AS null_return_date,
    SUM(CASE WHEN reason IS NULL OR reason = '' THEN 1 ELSE 0 END) AS null_reason,
    SUM(CASE WHEN refund_amount IS NULL THEN 1 ELSE 0 END) AS null_refund_amount
FROM return_fact;

-- 3. inventory_fact
SELECT
    'inventory_fact' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN inventory_id IS NULL OR inventory_id = '' THEN 1 ELSE 0 END) AS null_inventory_id,
    SUM(CASE WHEN store_id IS NULL OR store_id = '' THEN 1 ELSE 0 END) AS null_store_id,
    SUM(CASE WHEN product_id IS NULL OR product_id = '' THEN 1 ELSE 0 END) AS null_product_id,
    SUM(CASE WHEN stock_on_hand IS NULL THEN 1 ELSE 0 END) AS null_stock_on_hand,
    SUM(CASE WHEN reorder_level IS NULL THEN 1 ELSE 0 END) AS null_reorder_level,
    SUM(CASE WHEN last_restocked IS NULL THEN 1 ELSE 0 END) AS null_last_restocked
FROM inventory_fact;

-- 4. campaign_fact
SELECT
    'campaign_fact' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN order_id IS NULL OR order_id = '' THEN 1 ELSE 0 END) AS null_order_id,
    SUM(CASE WHEN sale_id IS NULL OR sale_id = '' THEN 1 ELSE 0 END) AS null_sale_id,
    SUM(CASE WHEN campaign_id IS NULL OR campaign_id = '' THEN 1 ELSE 0 END) AS null_campaign_id,
    SUM(CASE WHEN customer_id IS NULL OR customer_id = '' THEN 1 ELSE 0 END) AS null_customer_id,
    SUM(CASE WHEN conversion_date IS NULL THEN 1 ELSE 0 END) AS null_conversion_date
FROM campaign_fact;

-- 5. customer_session_data
SELECT
    'customer_session_data' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN session_id IS NULL OR session_id = '' THEN 1 ELSE 0 END) AS null_session_id,
    SUM(CASE WHEN customer_id IS NULL OR customer_id = '' THEN 1 ELSE 0 END) AS null_customer_id,
    SUM(CASE WHEN session_date IS NULL THEN 1 ELSE 0 END) AS null_session_date,
    SUM(CASE WHEN session_duration_min IS NULL THEN 1 ELSE 0 END) AS null_session_duration_min,
    SUM(CASE WHEN device_type IS NULL OR device_type = '' THEN 1 ELSE 0 END) AS null_device_type,
    SUM(CASE WHEN products_viewed IS NULL THEN 1 ELSE 0 END) AS null_products_viewed,
    SUM(CASE WHEN pages_visited IS NULL THEN 1 ELSE 0 END) AS null_pages_visited,
```

```
    SUM(CASE WHEN time_of_day IS NULL OR time_of_day = '' THEN 1 ELSE 0 END) AS null_time_of_day,
    SUM(CASE WHEN purchase_made IS NULL THEN 1 ELSE 0 END) AS null_purchase_made,
    SUM(CASE WHEN session_type IS NULL OR session_type = '' THEN 1 ELSE 0 END) AS null_session_type
FROM customer_session_data;

-- 6. footfall_data
SELECT
    'footfall_data' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN date IS NULL THEN 1 ELSE 0 END) AS null_date,
    SUM(CASE WHEN store_id IS NULL OR store_id = '' THEN 1 ELSE 0 END) AS null_store_id,
    SUM(CASE WHEN total_footfall IS NULL THEN 1 ELSE 0 END) AS null_total_footfall,
    SUM(CASE WHEN conversion_rate_percent IS NULL THEN 1 ELSE 0 END) AS null_conversion_rate_percent,
    SUM(CASE WHEN avg_stay_minutes IS NULL THEN 1 ELSE 0 END) AS null_avg_stay_minutes,
    SUM(CASE WHEN peak_hour IS NULL OR peak_hour = '' THEN 1 ELSE 0 END) AS null_peak_hour
FROM footfall_data;

-- 7. restock_orders
SELECT
    'restock_orders' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN order_id IS NULL OR order_id = '' THEN 1 ELSE 0 END) AS null_order_id,
    SUM(CASE WHEN store_id IS NULL OR store_id = '' THEN 1 ELSE 0 END) AS null_store_id,
    SUM(CASE WHEN product_id IS NULL OR product_id = '' THEN 1 ELSE 0 END) AS null_product_id,
    SUM(CASE WHEN order_date IS NULL THEN 1 ELSE 0 END) AS null_order_date,
    SUM(CASE WHEN expected_delivery_date IS NULL THEN 1 ELSE 0 END) AS null_expected_delivery_date,
    SUM(CASE WHEN actual_delivery_date IS NULL THEN 1 ELSE 0 END) AS null_actual_delivery_date
FROM restock_orders;

-- 8. weather_data
SELECT
    'weather_data' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN date IS NULL THEN 1 ELSE 0 END) AS null_date,
    SUM(CASE WHEN city IS NULL OR city = '' THEN 1 ELSE 0 END) AS null_city,
    SUM(CASE WHEN avg_temperature_celsius IS NULL THEN 1 ELSE 0 END) AS null_avg_temperature_celsius,
    SUM(CASE WHEN precipitation_mm IS NULL THEN 1 ELSE 0 END) AS null_precipitation_mm,
    SUM(CASE WHEN humidity_percent IS NULL THEN 1 ELSE 0 END) AS null_humidity_percent,
    SUM(CASE WHEN wind_speed_kph IS NULL THEN 1 ELSE 0 END) AS null_wind_speed_kph,
    SUM(CASE WHEN weather_condition IS NULL OR weather_condition = '' THEN 1 ELSE 0 END) AS null_weather_cond
FROM weather_data;

-- 9. dim_campaign
SELECT
    'dim_campaign' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN campaign_id IS NULL OR campaign_id = '' THEN 1 ELSE 0 END) AS null_campaign_id,
    SUM(CASE WHEN campaign_name IS NULL OR campaign_name = '' THEN 1 ELSE 0 END) AS null_campaign_name,
    SUM(CASE WHEN channel IS NULL OR channel = '' THEN 1 ELSE 0 END) AS null_channel,
    SUM(CASE WHEN campaign_cost IS NULL THEN 1 ELSE 0 END) AS null_campaign_cost
FROM dim_campaign;

-- 10. dim_customer
SELECT
    'dim_customer' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN customer_id IS NULL OR customer_id = '' THEN 1 ELSE 0 END) AS null_customer_id,
    SUM(CASE WHEN age IS NULL THEN 1 ELSE 0 END) AS null_age,
    SUM(CASE WHEN gender IS NULL OR gender = '' THEN 1 ELSE 0 END) AS null_gender,
    SUM(CASE WHEN location IS NULL OR location = '' THEN 1 ELSE 0 END) AS null_location,
```

```
    SUM(CASE WHEN loyalty_tier IS NULL OR loyalty_tier = '' THEN 1 ELSE 0 END) AS null_loyalty_tier
FROM dim_customer;

-- 11. dim_date
SELECT
    'dim_date' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN date_id IS NULL THEN 1 ELSE 0 END) AS null_date_id,
    SUM(CASE WHEN day IS NULL THEN 1 ELSE 0 END) AS null_day,
    SUM(CASE WHEN month IS NULL THEN 1 ELSE 0 END) AS null_month,
    SUM(CASE WHEN quarter IS NULL THEN 1 ELSE 0 END) AS null_quarter,
    SUM(CASE WHEN year IS NULL THEN 1 ELSE 0 END) AS null_year
FROM dim_date;

-- 12. dim_product
SELECT
    'dim_product' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN product_id IS NULL OR product_id = '' THEN 1 ELSE 0 END) AS null_product_id,
    SUM(CASE WHEN sku IS NULL OR sku = '' THEN 1 ELSE 0 END) AS null_sku,
    SUM(CASE WHEN name IS NULL OR name = '' THEN 1 ELSE 0 END) AS null_name,
    SUM(CASE WHEN brand IS NULL OR brand = '' THEN 1 ELSE 0 END) AS null_brand,
    SUM(CASE WHEN type IS NULL OR type = '' THEN 1 ELSE 0 END) AS null_type,
    SUM(CASE WHEN price IS NULL THEN 1 ELSE 0 END) AS null_price
FROM dim_product;

-- 13. dim_product_cost
SELECT
    'dim_product_cost' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN product_id IS NULL OR product_id = '' THEN 1 ELSE 0 END) AS null_product_id,
    SUM(CASE WHEN cost_per_unit IS NULL THEN 1 ELSE 0 END) AS null_cost_per_unit,
    SUM(CASE WHEN sale_price_per_unit IS NULL THEN 1 ELSE 0 END) AS null_sale_price_per_unit,
    SUM(CASE WHEN gross_profit_per_unit IS NULL THEN 1 ELSE 0 END) AS null_gross_profit_per_unit,
    SUM(CASE WHEN supplier IS NULL OR supplier = '' THEN 1 ELSE 0 END) AS null_supplier,
    SUM(CASE WHEN last_updated IS NULL THEN 1 ELSE 0 END) AS null_last_updated,
    SUM(CASE WHEN valid_from_date IS NULL THEN 1 ELSE 0 END) AS null_valid_from_date,
    SUM(CASE WHEN valid_to_date IS NULL THEN 0 ELSE 1 END) AS count_not_null_valid_to_date -- Checks for *popula
FROM dim_product_cost;

-- 14. dim_store
SELECT
    'dim_store' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN store_id IS NULL OR store_id = '' THEN 1 ELSE 0 END) AS null_store_id,
    SUM(CASE WHEN city IS NULL OR city = '' THEN 1 ELSE 0 END) AS null_city,
    SUM(CASE WHEN region IS NULL OR region = '' THEN 1 ELSE 0 END) AS null_region,
    SUM(CASE WHEN type IS NULL OR type = '' THEN 1 ELSE 0 END) AS null_type
FROM dim_store;

-- 15. external_cost_data
SELECT
    'external_cost_data' AS table_name,
    COUNT(*) AS total_rows,
    SUM(CASE WHEN store_id IS NULL OR store_id = '' THEN 1 ELSE 0 END) AS null_store_id,
    SUM(CASE WHEN month IS NULL THEN 1 ELSE 0 END) AS null_month,
    SUM(CASE WHEN operational_cost IS NULL THEN 1 ELSE 0 END) AS null_operational_cost
FROM external_cost_data;
```

```
-- # DATA QUALITY CHECKS - DUPLICATE KEYS
-- ####################################

-- 1. sales_fact: Check for duplicates based on its COMPOSITE Primary Key (sale_id, product_id)
SELECT
    'sales_fact' AS table_name,
    sale_id,
    product_id,
    COUNT(*) AS occurrences
FROM sales_fact
GROUP BY sale_id, product_id
HAVING COUNT(*) > 1;

-- 2. return_fact: Check for duplicates based on its Primary Key (return_id)
SELECT
    'return_fact' AS table_name,
    return_id,
    COUNT(*) AS occurrences
FROM return_fact
GROUP BY return_id
HAVING COUNT(*) > 1;

-- 3. dim_customer: Check for duplicates based on its Primary Key (customer_id)
SELECT
    'dim_customer' AS table_name,
    customer_id,
    COUNT(*) AS occurrences
FROM dim_customer
GROUP BY customer_id
HAVING COUNT(*) > 1;

-- 4. dim_date: Check for duplicates based on its Primary Key (date_id)
SELECT
    'dim_date' AS table_name,
    date_id,
    COUNT(*) AS occurrences
FROM dim_date
GROUP BY date_id
HAVING COUNT(*) > 1;

-- 5. dim_campaign: Check for duplicates based on its Primary Key (campaign_id)
SELECT
    'dim_campaign' AS table_name,
    campaign_id,
    COUNT(*) AS occurrences
FROM dim_campaign
GROUP BY campaign_id
HAVING COUNT(*) > 1;

-- 6. dim_product: Check for duplicates based on its Primary Key (product_id)
SELECT
    'dim_product (PK)' AS table_name,
    product_id,
    COUNT(*) AS occurrences
FROM dim_product
GROUP BY product_id
HAVING COUNT(*) > 1;

-- 6a. dim_product: Check for duplicates based on its UNIQUE Key (sku)
SELECT
```

```
    'dim_product (Unique SKU)' AS table_name,
    sku,
    COUNT(*) AS occurrences
FROM dim_product
GROUP BY sku
HAVING COUNT(*) > 1;

-- 7. dim_store: Check for duplicates based on its Primary Key (store_id)
SELECT
    'dim_store' AS table_name,
    store_id,
    COUNT(*) AS occurrences
FROM dim_store
GROUP BY store_id
HAVING COUNT(*) > 1;

-- 8. dim_product_cost: Check for duplicates based on its COMPOSITE Primary Key (product_id, valid_from_date)
SELECT
    'dim_product_cost' AS table_name,
    product_id,
    valid_from_date,
    COUNT(*) AS occurrences
FROM dim_product_cost
GROUP BY product_id, valid_from_date
HAVING COUNT(*) > 1;

-- 9. weather_data: Check for duplicates based on its COMPOSITE Primary Key (date, city)
SELECT
    'weather_data' AS table_name,
    date,
    city,
    COUNT(*) AS occurrences
FROM weather_data
GROUP BY date, city
HAVING COUNT(*) > 1;

-- 10. external_cost_data: Check for duplicates based on its COMPOSITE Primary Key (store_id, month)
SELECT
    'external_cost_data' AS table_name,
    store_id,
    month,
    COUNT(*) AS occurrences
FROM external_cost_data
GROUP BY store_id, month
HAVING COUNT(*) > 1;

-- 11. inventory_fact: Check for duplicates based on its Primary Key (inventory_id)
SELECT
    'inventory_fact' AS table_name,
    inventory_id,
    COUNT(*) AS occurrences
FROM inventory_fact
GROUP BY inventory_id
HAVING COUNT(*) > 1;

-- 12. campaign_fact: Check for duplicates based on its COMPOSITE Primary Key (order_id, campaign_id, customer_id
SELECT
    'campaign_fact' AS table_name,
    order_id,
    campaign_id,
```

```
    customer_id,
    COUNT(*) AS occurrences
FROM campaign_fact
GROUP BY order_id, campaign_id, customer_id
HAVING COUNT(*) > 1;

-- 13. customer_session_data: Check for duplicates based on its Primary Key (session_id)
SELECT
    'customer_session_data' AS table_name,
    session_id,
    COUNT(*) AS occurrences
FROM customer_session_data
GROUP BY session_id
HAVING COUNT(*) > 1;

-- 14. footfall_data: Check for duplicates based on its COMPOSITE Primary Key (date, store_id)
SELECT
    'footfall_data' AS table_name,
    date,
    store_id,
    COUNT(*) AS occurrences
FROM footfall_data
GROUP BY date, store_id
HAVING COUNT(*) > 1;

-- 15. restock_orders: Check for duplicates based on its Primary Key (order_id)
SELECT
    'restock_orders' AS table_name,
    order_id,
    COUNT(*) AS occurrences
FROM restock_orders
GROUP BY order_id
HAVING COUNT(*) > 1;




-- □ SALES & PERFORMANCE INSIGHTS

-- 1. Which products are generating the highest revenue overall and by region?

-------- 1. Top Product by Revenue in Each Region
WITH RegionProductRevenue AS (
    -- Get product revenue by region
    SELECT
        p.name AS product_name,
        s.region,
        SUM(sf.revenue) AS total_revenue_in_region,
        ROW_NUMBER() OVER (PARTITION BY s.region ORDER BY SUM(sf.revenue) DESC) AS region_rank
    FROM sales_fact sf
    JOIN dim_product p ON sf.product_id = p.product_id
    JOIN dim_store s ON sf.store_id = s.store_id
    GROUP BY p.name, s.region
)
-- Select only top product per region
SELECT
    region,
    product_name,
    total_revenue_in_region
```

```
FROM RegionProductRevenue
WHERE region_rank = 1
ORDER BY total_revenue_in_region DESC;


-------- 2. Overall Top Products Across All Regions
SELECT
   p.name AS product_name,
   SUM(sf.revenue) AS global_total_revenue
FROM sales_fact sf
JOIN dim_product p ON sf.product_id = p.product_id
GROUP BY p.name
ORDER BY global_total_revenue DESC
LIMIT 5;

-- 2. What are the top-selling eyewear types (frames, lenses, sunglasses)?
SELECT
   dp.type AS product_type,
   SUM(sf.quantity) AS total_units_sold,
   ROUND(SUM(sf.revenue), 2) AS total_revenue_INR,
   ROUND(SUM(sf.revenue) / SUM(sf.quantity), 2) AS avg_price_per_unit
FROM sales_fact sf
JOIN dim_product dp ON sf.product_id = dp.product_id
GROUP BY dp.type
ORDER BY total_units_sold DESC;

-- 3. Which stores consistently outperform others in terms of monthly sales?

-------- 1. Top Store Each Month
WITH MonthlyRevenue AS (
   SELECT
      sf.store_id,
      -- UPDATED: Use dim_date for month
      dd.year,
      dd.month,
      SUM(sf.revenue) AS monthly_revenue,
      ROW_NUMBER() OVER (PARTITION BY dd.year, dd.month ORDER BY SUM(sf.revenue) DESC) AS rn
   FROM sales_fact sf
   JOIN dim_date dd ON sf.date_id = dd.date_id -- Join with dim_date
   GROUP BY sf.store_id, dd.year, dd.month
)
-- Select only top store per month
SELECT
   year,
   month,
   store_id,
   monthly_revenue
FROM MonthlyRevenue
WHERE rn = 1
ORDER BY year, month;

-------- 2. Stores That Were Top Performers Consistently
WITH MonthlyRevenue AS (
   SELECT
      sf.store_id,
      -- UPDATED: Use dim_date for month
      dd.year,
      dd.month,
      SUM(sf.revenue) AS monthly_revenue,
      ROW_NUMBER() OVER (PARTITION BY dd.year, dd.month ORDER BY SUM(sf.revenue) DESC) AS rn
```

```
    FROM sales_fact sf
    JOIN dim_date dd ON sf.date_id = dd.date_id -- Join with dim_date
    GROUP BY sf.store_id, dd.year, dd.month
),
TopStoresByMonth AS (
    -- Get top store per month
    SELECT
        store_id,
        year,
        month
    FROM MonthlyRevenue
    WHERE rn = 1
),
ConsistentPerformers AS (
    -- Count how many times each store was top performer
    SELECT
        store_id,
        COUNT(*) AS months_as_top_store
    FROM TopStoresByMonth
    GROUP BY store_id
    HAVING COUNT(*) > 1
)
-- Final output: List consistent top stores
SELECT
    store_id,
    months_as_top_store
FROM ConsistentPerformers
ORDER BY months_as_top_store DESC;

-- 4. What is the sales growth trend over time (daily, monthly, quarterly)?

-- Daily: (No change needed, date_id is already a proper date column)
SELECT date_id, SUM(revenue) AS daily_revenue
FROM sales_fact
GROUP BY date_id
ORDER BY date_id;

-- Monthly:
SELECT
    dd.year, -- UPDATED: Use dim_date for year and month
    dd.month,
    SUM(sf.revenue) AS monthly_revenue
FROM sales_fact sf
JOIN dim_date dd ON sf.date_id = dd.date_id -- Join with dim_date
GROUP BY dd.year, dd.month
ORDER BY dd.year, dd.month;

-- Quarterly:
SELECT
    dd.year, -- UPDATED: Use dim_date for year and quarter
    dd.quarter,
    SUM(sf.revenue) AS quarterly_revenue
FROM sales_fact sf
JOIN dim_date dd ON sf.date_id = dd.date_id -- Join with dim_date
GROUP BY dd.year, dd.quarter
ORDER BY dd.year, dd.quarter;

-- 5. What percentage of revenue comes from repeat customers vs new customers?

WITH FirstPurchase AS (
```

```
    -- Get first purchase date per customer
    SELECT
        customer_id,
        MIN(date_id) AS first_purchase_date
    FROM sales_fact
    GROUP BY customer_id
),
CustomerType AS (
    -- Classify each sale as New or Repeat
    SELECT
        sf.customer_id,
        CASE
            WHEN sf.date_id > fp.first_purchase_date THEN 'Repeat'
            ELSE 'New'
        END AS purchase_type,
        sf.revenue
    FROM sales_fact sf
    JOIN FirstPurchase fp ON sf.customer_id = fp.customer_id
)
-- Final output: Revenue split between new and repeat customers
SELECT
    purchase_type,
    SUM(revenue) AS total_revenue,
    ROUND(SUM(revenue) * 100.0 / (SELECT SUM(revenue) FROM CustomerType), 2) AS percentage_of_total_revenue
FROM CustomerType
GROUP BY purchase_type
ORDER BY total_revenue DESC;


-- 6. What are the peak sales days of the week or month?

-- Peak Sales Days of the Week
-- No major change needed here. While dim_date has 'day', it doesn't have 'day_of_week_num' or 'day_name' directly
-- Using functions like DAYOFWEEK() and DAYNAME() directly on date_id (which is a DATE type and usually indexed) is
SELECT
    DAYOFWEEK(sf.date_id) AS day_of_week_num, -- MySQL's DAYOFWEEK returns 1=Sunday, 2=Monday, etc.
    DAYNAME(sf.date_id) AS day_of_week,
    SUM(sf.revenue) AS total_revenue
FROM sales_fact sf
GROUP BY day_of_week_num, day_of_week
ORDER BY total_revenue DESC;

-- Peak Sales Days of the Month
-- UPDATED: Joining dim_date to use its 'day' column for consistency with dimensional model
SELECT
    dd.day AS day_of_month,
    SUM(sf.revenue) AS total_revenue
FROM sales_fact sf
JOIN dim_date dd ON sf.date_id = dd.date_id -- Join with dim_date
GROUP BY dd.day
ORDER BY total_revenue DESC;

-- 7. What is the average transaction value per customer across stores?

SELECT
    sf.store_id,
    ds.region,
    ds.city,
    COUNT(DISTINCT sf.customer_id) AS unique_customers,
    COUNT(sf.sale_id) AS total_transactions,
```

```
    SUM(sf.revenue) AS total_store_revenue,
    ROUND(AVG(sf.revenue), 2) AS avg_transaction_value,
    -- UPDATED: Changed column name for clarity. This is Average Revenue Per Customer for the period queried.
    -- Added NULLIF to prevent division by zero if unique_customers is 0.
    ROUND(SUM(sf.revenue) / NULLIF(COUNT(DISTINCT sf.customer_id), 0), 2) AS avg_revenue_per_customer
FROM sales_fact sf
JOIN dim_store ds ON sf.store_id = ds.store_id
GROUP BY sf.store_id, ds.region, ds.city
ORDER BY avg_transaction_value DESC;


-- 8. Which customer segments (age, loyalty tier) spend the most?

SELECT
    FLOOR(dc.age / 10) * 10 AS age_group, -- Creates age groups (e.g., 20-29, 30-39)
    dc.loyalty_tier,
    COUNT(DISTINCT dc.customer_id) AS customer_count,
    COUNT(sf.sale_id) AS total_transactions,
    SUM(sf.revenue) AS total_revenue,
    ROUND(AVG(sf.revenue), 2) AS avg_transaction_value
FROM sales_fact sf
JOIN dim_customer dc ON sf.customer_id = dc.customer_id
WHERE dc.loyalty_tier IN ('Platinum', 'Gold', 'Silver') -- Filter for specific loyalty tiers
GROUP BY age_group, dc.loyalty_tier
HAVING age_group BETWEEN 10 AND 60 -- Filter for relevant age groups
ORDER BY total_revenue DESC;

-- 9. How many units per product type are sold per region/store?

-- Units Sold Per Product Type by Region
WITH RegionProductSales AS (
    SELECT
        s.region,
        p.type AS product_type,
        SUM(sf.quantity) AS units_sold
    FROM sales_fact sf
    JOIN dim_product p ON sf.product_id = p.product_id
    JOIN dim_store s ON sf.store_id = s.store_id
    GROUP BY s.region, p.type
),
RegionTotalSales AS (
    SELECT
        region,
        SUM(units_sold) AS total_units_sold
    FROM RegionProductSales
    GROUP BY region
)
SELECT
    rps.region,
    rps.product_type,
    rps.units_sold,
    -- Calculate percentage share of units sold for each product type within its region
    ROUND(rps.units_sold * 100.0 / NULLIF(rts.total_units_sold, 0), 2) AS percent_share
FROM RegionProductSales rps
JOIN RegionTotalSales rts ON rps.region = rts.region
ORDER BY rps.region, rps.units_sold DESC;

-- Units Sold Per Product Type by Store
WITH StoreProductSales AS (
    SELECT
```

```
        s.store_id,
        s.city,
        s.region,
        p.type AS product_type,
        SUM(sf.quantity) AS units_sold
    FROM sales_fact sf
    JOIN dim_product p ON sf.product_id = p.product_id
    JOIN dim_store s ON sf.store_id = s.store_id
    GROUP BY s.store_id, s.city, s.region, p.type
),
StoreTotalSales AS (
    SELECT
        store_id,
        SUM(units_sold) AS total_units_sold
    FROM StoreProductSales
    GROUP BY store_id
)
SELECT
    sps.store_id,
    sps.city,
    sps.region,
    sps.product_type,
    sps.units_sold,
    -- Calculate percentage share of units sold for each product type within its store
    ROUND(sps.units_sold * 100.0 / NULLIF(sts.total_units_sold, 0), 2) AS percent_share
FROM StoreProductSales sps
JOIN StoreTotalSales sts ON sps.store_id = sts.store_id
ORDER BY sps.region, sps.store_id, sps.units_sold DESC;


-- 10. Which store-customer combinations generate the most business?

WITH StoreCustomerRevenue AS (
    -- Calculate total revenue per unique store-customer combination
    SELECT
        sf.store_id,
        sf.customer_id,
        SUM(sf.revenue) AS total_revenue,
        -- Rank customers within each store by their total revenue
        ROW_NUMBER() OVER (PARTITION BY sf.store_id ORDER BY SUM(sf.revenue) DESC) AS customer_rank
    FROM sales_fact sf
    GROUP BY sf.store_id, sf.customer_id
)
-- Select only the top customer for each store (ranked 1st)
SELECT
    sc.store_id,
    sc.customer_id,
    sc.total_revenue
FROM StoreCustomerRevenue sc
WHERE customer_rank = 1
ORDER BY sc.total_revenue DESC;



-- INVENTORY & SUPPLY CHAIN

-- 11. Which products are frequently low on stock and need restocking?

WITH LatestInventory AS (
    -- Get the most recent stock level for each product per store
```

```
    SELECT
        i.product_id,
        i.store_id,
        i.stock_on_hand,
        i.reorder_level,
        ROW_NUMBER() OVER (PARTITION BY i.product_id, i.store_id ORDER BY i.last_restocked DESC) AS rn
    FROM inventory_fact i
),
UnderstockedProducts AS (
    -- Count how many times each product went understocked historically
    -- (This assumes inventory_fact stores historical snapshots)
    SELECT
        product_id,
        COUNT(*) AS times_understocked
    FROM inventory_fact
    WHERE stock_on_hand < reorder_level
    GROUP BY product_id
),
CurrentUnderstocked AS (
    -- Current understocked products per store, based on the LATEST inventory data
    SELECT
        li.product_id,
        li.store_id,
        li.stock_on_hand,
        li.reorder_level,
        (li.reorder_level - li.stock_on_hand) AS units_needed_to_restock
    FROM LatestInventory li
    WHERE li.rn = 1 -- Ensure we're only taking the latest record
    AND li.stock_on_hand < li.reorder_level
),
ProductNames AS (
    -- Get product names and types from dim_product
    SELECT
        product_id,
        name AS product_name,
        type AS product_type
    FROM dim_product
)
-- Final output: Combine historical + current understock data
SELECT
    p.product_id,
    p.product_name,
    p.product_type,
    COALESCE(usp.times_understocked, 0) AS historical_restock_needed_count,
    SUM(COALESCE(cu.units_needed_to_restock, 0)) AS total_units_needed_now -- SUM and COALESCE for aggregated
FROM ProductNames p
LEFT JOIN UnderstockedProducts usp ON p.product_id = usp.product_id
LEFT JOIN CurrentUnderstocked cu ON p.product_id = cu.product_id
GROUP BY p.product_id, p.product_name, p.product_type, usp.times_understocked -- Grouping by usp.times_unders
ORDER BY historical_restock_needed_count DESC, total_units_needed_now DESC;

-- 12. What’s the average inventory turnover rate by store/product?

-- Note: These calculations assume 'inventory_fact' contains historical snapshots of stock levels
-- to derive a meaningful 'average stock level' over a period. If 'inventory_fact' only holds
-- the current snapshot, these "average stock levels" would represent the average of all *current*
-- product stock levels within a store/overall, which is not the standard definition for turnover.

-- 1. Inventory Turnover Rate by Store
```

```
-- Store-level inventory turnover
WITH StoreSales AS (
    SELECT
        store_id,
        SUM(quantity) AS total_units_sold
    FROM sales_fact
    GROUP BY store_id
),
StoreAvgStock AS (
    SELECT
        store_id,
        ROUND(AVG(stock_on_hand), 2) AS avg_stock_level
    FROM inventory_fact
    GROUP BY store_id
)
SELECT
    ss.store_id,
    ss.total_units_sold,
    sas.avg_stock_level,
    ROUND(ss.total_units_sold / NULLIF(sas.avg_stock_level, 0), 2) AS inventory_turnover_rate -- Added NULLIF
FROM StoreSales ss
JOIN StoreAvgStock sas ON ss.store_id = sas.store_id
WHERE sas.avg_stock_level > 0
ORDER BY inventory_turnover_rate DESC;

-- 2. Inventory Turnover Rate by Product

-- Product-level inventory turnover
WITH ProductSales AS (
    SELECT
        product_id,
        SUM(quantity) AS total_units_sold
    FROM sales_fact
    GROUP BY product_id
),
ProductAvgStock AS (
    SELECT
        product_id,
        ROUND(AVG(stock_on_hand), 2) AS avg_stock_level
    FROM inventory_fact
    GROUP BY product_id
)
SELECT
    ps.product_id,
    dp.name AS product_name,
    dp.type AS product_type,
    ps.total_units_sold,
    pas.avg_stock_level,
    ROUND(ps.total_units_sold / NULLIF(pas.avg_stock_level, 0), 2) AS inventory_turnover_rate -- Added NULLIF
FROM ProductSales ps
JOIN ProductAvgStock pas ON ps.product_id = pas.product_id
JOIN dim_product dp ON ps.product_id = dp.product_id
WHERE pas.avg_stock_level > 0
ORDER BY inventory_turnover_rate DESC;

-- 13. Which stores are overstocked or understocked across key SKUs?

WITH ProductRevenues AS (
    SELECT
        sf.product_id,
```

```
        SUM(sf.revenue) AS total_revenue,
        NTILE(20) OVER (ORDER BY SUM(sf.revenue) DESC) AS revenue_percentile -- Top 5% (100/20 = 5)
    FROM sales_fact sf
    GROUP BY sf.product_id
),
KeySKUs AS (
    -- Select only top revenue-generating products (top 5%)
    SELECT DISTINCT product_id
    FROM ProductRevenues
    WHERE revenue_percentile = 1
),
LatestInventory AS (
    -- Get the most recent stock level per store/product for key SKUs
    SELECT
        i.store_id,
        i.product_id,
        i.stock_on_hand,
        i.reorder_level,
        ROW_NUMBER() OVER (
            PARTITION BY i.store_id, i.product_id
            ORDER BY i.last_restocked DESC
        ) AS rn
    FROM inventory_fact i
    WHERE i.product_id IN (SELECT product_id FROM KeySKUs)
),
FilteredInventory AS (
    -- Keep only latest records and classify stock status
    SELECT
        store_id,
        product_id,
        stock_on_hand,
        reorder_level,
        CASE
            WHEN stock_on_hand > 2 * reorder_level THEN 'Overstocked' -- Custom rule for overstocking
            WHEN stock_on_hand < reorder_level THEN 'Understocked'
            ELSE 'Normal'
        END AS stock_status
    FROM LatestInventory
    WHERE rn = 1
),
ProductDetails AS (
    -- Add product names/types
    SELECT
        product_id,
        name AS product_name,
        type AS product_type
    FROM dim_product
)
-- Final output: Only stores with overstocked/understocked key SKUs
SELECT
    fi.store_id,
    fi.product_id,
    pd.product_name,
    pd.product_type,
    fi.stock_on_hand,
    fi.reorder_level,
    fi.stock_status
FROM FilteredInventory fi
JOIN ProductDetails pd ON fi.product_id = pd.product_id
WHERE fi.stock_status IN ('Overstocked', 'Understocked')
```

```
ORDER BY fi.stock_status DESC, fi.store_id;


-- 14. How often are restocks happening, and are they timely?

WITH RestockHistory AS (
    -- Get previous restock date and compute interval between restocks for each product-store
    SELECT
        store_id,
        product_id,
        last_restocked,
        LAG(last_restocked) OVER (PARTITION BY store_id, product_id ORDER BY last_restocked) AS prev_restocked,
        DATEDIFF(last_restocked, LAG(last_restocked) OVER (PARTITION BY store_id, product_id ORDER BY last_restocke
    FROM inventory_fact
),
RestockStats AS (
    -- Compute average restock frequency and consistency (std dev)
    SELECT
        store_id,
        product_id,
        COUNT(*) AS restock_count,
        AVG(days_between_restocks) AS avg_days_between_restocks,
        STD(days_between_restocks) AS std_dev_restock_interval,
        CASE
            WHEN STD(days_between_restocks) > 0.5 * AVG(days_between_restocks) THEN 'Inconsistent'
            ELSE 'Consistent'
        END AS restock_consistency_status
    FROM RestockHistory
    WHERE days_between_restocks IS NOT NULL
    GROUP BY store_id, product_id
),
CurrentStockStatus AS (
    -- UPDATED: Get the LATEST stock status from inventory_fact for each product-store
    SELECT
        store_id,
        product_id,
        stock_on_hand,
        reorder_level,
        last_restocked,
        ROW_NUMBER() OVER (PARTITION BY store_id, product_id ORDER BY last_restocked DESC) AS rn
    FROM inventory_fact
)
-- Final output with timeliness check
SELECT
    css.store_id,
    dp.name AS product_name,
    dp.type AS product_type,
    ROUND(rs.avg_days_between_restocks, 2) AS avg_restock_interval_days,
    -- UPDATED: Use CURDATE() for dynamic timeliness check
    CASE
        WHEN rs.avg_days_between_restocks IS NULL THEN 'First Restock Needed/No History' -- Handle products with n
        WHEN DATEDIFF(CURDATE(), css.last_restocked) > rs.avg_days_between_restocks THEN 'Delayed'
        ELSE 'On Time'
    END AS restock_timeliness_status,
    CASE
        WHEN css.stock_on_hand < css.reorder_level THEN 'Low Stock - Urgent'
        ELSE 'Stock OK'
    END AS current_stock_level_status
FROM CurrentStockStatus css
LEFT JOIN RestockStats rs
```

```
    ON css.store_id = rs.store_id
    AND css.product_id = rs.product_id
JOIN dim_product dp ON css.product_id = dp.product_id
WHERE css.rn = 1 -- Ensure we are considering only the latest stock status
ORDER BY restock_timeliness_status, current_stock_level_status;


-- 15. What is the lead time between restock and product availability?

SELECT
    ro.order_id,
    ro.store_id,
    ro.product_id,
    dp.name AS product_name,
    ro.order_date,
    ro.expected_delivery_date,
    ro.actual_delivery_date,
    DATEDIFF(ro.expected_delivery_date, ro.order_date) AS expected_lead_days,
    DATEDIFF(ro.actual_delivery_date, ro.order_date) AS actual_lead_days,
    DATEDIFF(ro.actual_delivery_date, ro.expected_delivery_date) AS delivery_variance_days -- Positive means late, neg
FROM restock_orders ro
JOIN dim_product dp ON ro.product_id = dp.product_id
WHERE ro.actual_delivery_date IS NOT NULL -- Only consider orders that have been actually delivered
ORDER BY actual_lead_days DESC;


-- 16. How effective is the reorder level setting per product category?

-- Step 1: Compute average stock vs reorder level per product
WITH ProductStockStats AS (
    SELECT
        product_id,
        AVG(stock_on_hand) AS avg_stock,
        AVG(reorder_level) AS avg_reorder_level,
        AVG(stock_on_hand - reorder_level) AS diff_avg_stock_vs_reorder
    FROM inventory_fact
    GROUP BY product_id
)
-- Step 2: Aggregate at product type level
SELECT
    dp.type AS product_type,
    COUNT(pss.product_id) AS product_count,
    ROUND(AVG(pss.avg_stock), 2) AS avg_stock_level,
    ROUND(AVG(pss.avg_reorder_level), 2) AS avg_reorder_level,
    ROUND(AVG(pss.diff_avg_stock_vs_reorder), 2) AS avg_diff_stock_reorder,
    -- Interpretation
    CASE
        WHEN AVG(pss.diff_avg_stock_vs_reorder) < 0 THEN 'Understocked - Increase Reorder Level'
        ELSE 'Overstocked - Decrease Reorder Level'
    END AS reorder_recommendation
FROM ProductStockStats pss
JOIN dim_product dp ON pss.product_id = dp.product_id
GROUP BY dp.type
ORDER BY avg_diff_stock_reorder ASC;

-- □ RETURNS & REFUNDS

-- 17. What is the return rate by product type, brand, or store?

-- □ 1. Return Rate by Product Type
```

```
SELECT
    dp.type AS product_type,
    COUNT(DISTINCT sf.sale_id) AS total_sales,
    COUNT(DISTINCT rf.sale_id) AS total_returns,
    ROUND(COUNT(DISTINCT rf.sale_id) * 100.0 / NULLIF(COUNT(DISTINCT sf.sale_id), 0), 2) AS return_rate_percent
FROM sales_fact sf
JOIN dim_product dp ON sf.product_id = dp.product_id
LEFT JOIN return_fact rf ON sf.sale_id = rf.sale_id
GROUP BY dp.type
HAVING total_sales > 0
ORDER BY return_rate_percent DESC;

-- □ 2. Return Rate by Brand

SELECT
    dp.brand,
    COUNT(DISTINCT sf.sale_id) AS total_sales,
    COUNT(DISTINCT rf.sale_id) AS total_returns,
    ROUND(COUNT(DISTINCT rf.sale_id) * 100.0 / NULLIF(COUNT(DISTINCT sf.sale_id), 0), 2) AS return_rate_percent
FROM sales_fact sf
JOIN dim_product dp ON sf.product_id = dp.product_id
LEFT JOIN return_fact rf ON sf.sale_id = rf.sale_id
WHERE dp.brand IS NOT NULL
GROUP BY dp.brand
HAVING total_sales > 0
ORDER BY return_rate_percent DESC;

-- □ 3. Return Rate by Store

SELECT
    ds.region,
    ds.city,
    sf.store_id,
    COUNT(DISTINCT sf.sale_id) AS total_sales,
    COUNT(DISTINCT rf.sale_id) AS total_returns,
    ROUND(COUNT(DISTINCT rf.sale_id) * 100.0 / NULLIF(COUNT(DISTINCT sf.sale_id), 0), 2) AS return_rate_percent
FROM sales_fact sf
LEFT JOIN return_fact rf ON sf.sale_id = rf.sale_id
JOIN dim_store ds ON sf.store_id = ds.store_id
GROUP BY sf.store_id, ds.region, ds.city
HAVING total_sales > 0
ORDER BY return_rate_percent DESC;


-- 18. What are the top reasons for returns, and how can we reduce them?

-- Step 1: Count total distinct sales
WITH TotalSales AS (
    SELECT
        COUNT(DISTINCT sale_id) AS total_sales
    FROM sales_fact
),
-- Step 2: Count how many sales were returned for each reason
ReturnCountByReason AS (
    SELECT
        rf.reason,
        COUNT(DISTINCT rf.sale_id) AS sales_with_returns
    FROM return_fact rf
    GROUP BY rf.reason
```

```
)
-- Final output: Return rate by reason + actionable recommendations
SELECT
    rcr.reason,
    rcr.sales_with_returns,
    ts.total_sales,
    ROUND((rcr.sales_with_returns * 100.0 / NULLIF(ts.total_sales, 0)), 2) AS return_rate_of_total_sales,
    CASE
        WHEN rcr.reason = 'Wrong Prescription' THEN 'Improve prescription validation process'
        WHEN rcr.reason = 'Size Issue' THEN 'Add virtual try-on or better size guide'
        WHEN rcr.reason = 'Not Liked' THEN 'Enhance product descriptions/photos'
        WHEN rcr.reason = 'Damaged' THEN 'Review packaging/shipping practices'
        ELSE 'Investigate further'
    END AS recommendation
FROM ReturnCountByReason rcr
CROSS JOIN TotalSales ts
ORDER BY return_rate_of_total_sales DESC;

-- 19. What is the financial impact (in INR) of returns by category and location?

SELECT
    dp.type AS product_type,
    ds.region,
    ds.city,
    COUNT(DISTINCT sf.sale_id) AS total_sales_count,
    SUM(sf.quantity) AS total_units_sold,
    COUNT(rf.return_id) AS total_return_records,
    SUM(sf.revenue) AS total_sales_INR,
    COALESCE(SUM(rf.refund_amount), 0) AS total_refunds_INR,
    ROUND(
        COALESCE(SUM(rf.refund_amount) * 100.0 / NULLIF(SUM(sf.revenue), 0), 0), 2
    ) AS refund_percentage_of_sales,
    ROUND(
        COALESCE(SUM(rf.refund_amount) / NULLIF(COUNT(rf.return_id), 0), 0), 2
    ) AS avg_refund_per_return
FROM sales_fact sf
JOIN dim_product dp ON sf.product_id = dp.product_id
JOIN dim_store ds ON sf.store_id = ds.store_id
LEFT JOIN return_fact rf ON sf.sale_id = rf.sale_id
GROUP BY dp.type, ds.region, ds.city
HAVING COUNT(DISTINCT sf.sale_id) > 0
ORDER BY refund_percentage_of_sales DESC;

-- 20. Are certain stores experiencing higher return rates than others?

WITH StoreReturnStats AS (
    SELECT
        sf.store_id,
        ds.region,
        ds.city,
        COUNT(DISTINCT sf.sale_id) AS total_sales,
        COUNT(DISTINCT rf.sale_id) AS sales_with_returns,
        COUNT(rf.return_id) AS total_returns,
        ROUND(
            COUNT(DISTINCT rf.sale_id) * 100.0 / NULLIF(COUNT(DISTINCT sf.sale_id), 0), 2
        ) AS return_rate_percent
    FROM sales_fact sf
    JOIN dim_store ds ON sf.store_id = ds.store_id
    LEFT JOIN return_fact rf ON sf.sale_id = rf.sale_id
    GROUP BY sf.store_id, ds.region, ds.city
```

```
)
-- Final output: Stores with highest return rate
SELECT
    store_id,
    region,
    city,
    total_sales,
    sales_with_returns,
    total_returns,
    return_rate_percent
FROM StoreReturnStats
WHERE total_sales > 0
ORDER BY return_rate_percent DESC;


-- 21. What is the average time between purchase and return?

SELECT
    ROUND(AVG(DATEDIFF(rf.return_date, sf.date_id)), 2) AS avg_days_to_return
FROM sales_fact sf
JOIN return_fact rf ON sf.sale_id = rf.sale_id
WHERE rf.return_date >= sf.date_id;

-- 22. Which customer segments are more likely to return products?

SELECT
    dc.loyalty_tier,
    FLOOR(dc.age / 10) * 10 AS age_group,
    dc.gender,
    COUNT(DISTINCT sf.sale_id) AS total_sales,
    COUNT(DISTINCT rf.sale_id) AS total_sales_with_returns,
    SUM(sf.quantity) AS total_sold_items,
    COUNT(rf.return_id) AS total_returned_items,
    ROUND(
        COUNT(DISTINCT rf.sale_id) * 100.0 / NULLIF(COUNT(DISTINCT sf.sale_id), 0), 2
    ) AS return_rate_by_transaction,
    ROUND(
        COUNT(rf.return_id) * 100.0 / NULLIF(SUM(sf.quantity), 0), 2
    ) AS return_rate_by_item
FROM sales_fact sf
LEFT JOIN return_fact rf ON sf.sale_id = rf.sale_id
JOIN dim_customer dc ON sf.customer_id = dc.customer_id
GROUP BY dc.loyalty_tier, age_group, dc.gender
HAVING total_sales > 0
ORDER BY return_rate_by_item DESC;


-- CUSTOMER BEHAVIOR & MARKETING

-- 23. What is the average customer lifetime value (CLV)?

-- Calculate overall CLV statistics
SELECT
    COUNT(*) AS total_customers,
    ROUND(MIN(clv), 2) AS min_clv,
    ROUND(AVG(clv), 2) AS avg_clv,
    ROUND(MAX(clv), 2) AS max_clv
FROM (
    -- Compute CLV per customer
    SELECT
```

```
        customer_id,
        SUM(revenue) AS clv
    FROM sales_fact
    GROUP BY customer_id
) AS clv_data;


-- 24. Which loyalty tier has the highest engagement and revenue contribution?

-- Loyalty Tier Analysis: Engagement and Revenue Contribution
SELECT
    dc.loyalty_tier,
    COUNT(DISTINCT dc.customer_id) AS unique_customers,
    COUNT(sf.sale_id) AS total_transactions,
    SUM(sf.revenue) AS total_revenue,
    ROUND(SUM(sf.revenue) / COUNT(DISTINCT dc.customer_id), 2) AS avg_clv,
    ROUND(AVG(sf.revenue), 2) AS avg_transaction_value,
    ROUND((SUM(sf.revenue) * 100.0) / (SELECT SUM(revenue) FROM sales_fact), 2) AS revenue_share_percent
FROM sales_fact sf
JOIN dim_customer dc ON sf.customer_id = dc.customer_id
GROUP BY dc.loyalty_tier
ORDER BY total_revenue DESC;


-- 25. What are the most popular products among different age groups?

WITH AgeGroupProductStats AS (
    SELECT
        FLOOR(dc.age / 10) * 10 AS age_group,
        dp.name AS product_name,
        dp.type AS product_type,
        COUNT(sf.sale_id) AS transaction_count,
        SUM(sf.quantity) AS units_sold,
        ROUND(SUM(sf.revenue), 2) AS total_revenue,
        ROUND(AVG(sf.revenue), 2) AS avg_transaction_value
    FROM sales_fact sf
    JOIN dim_product dp ON sf.product_id = dp.product_id
    JOIN dim_customer dc ON sf.customer_id = dc.customer_id
    GROUP BY age_group, dp.name, dp.type
),
RankedProducts AS (
    SELECT *,
           RANK() OVER (PARTITION BY age_group ORDER BY units_sold DESC) AS rank_by_units
    FROM AgeGroupProductStats
)
-- Select top 5 products per age group
SELECT
    age_group,
    product_name,
    product_type,
    units_sold,
    total_revenue,
    transaction_count,
    avg_transaction_value,
    rank_by_units
FROM RankedProducts
WHERE rank_by_units <= 5
ORDER BY age_group, rank_by_units;
```

```sql
-- 26. How do customer preferences vary by geography?

WITH LocationProductStats AS (
    SELECT
        dc.location,
        dp.type AS product_type,
        SUM(sf.quantity) AS units_sold,
        SUM(sf.revenue) AS total_revenue
    FROM sales_fact sf
    JOIN dim_customer dc ON sf.customer_id = dc.customer_id
    JOIN dim_product dp ON sf.product_id = dp.product_id
    GROUP BY dc.location, dp.type
),
LocationTotals AS (
    SELECT
        location,
        SUM(units_sold) AS total_units_sold
    FROM LocationProductStats
    GROUP BY location
)
SELECT
    lps.location,
    lps.product_type,
    lps.units_sold,
    lps.total_revenue,
    ROUND(
        lps.units_sold * 100.0 / NULLIF(lt.total_units_sold, 0), 2
    ) AS percent_share_of_sales
FROM LocationProductStats lps
JOIN LocationTotals lt ON lps.location = lt.location
ORDER BY lps.location, percent_share_of_sales DESC;


-- 27. Which campaigns yield the best ROI?

WITH CampaignRevenue AS (
    SELECT
        cf.campaign_id,
        dc.channel,
        dc.campaign_name,
        SUM(sf.revenue) AS total_revenue,
        COUNT(DISTINCT cf.customer_id) AS customers_converted
    FROM campaign_fact cf
    JOIN sales_fact sf ON cf.sale_id = sf.sale_id
    JOIN dim_campaign dc ON cf.campaign_id = dc.campaign_id
    GROUP BY cf.campaign_id, dc.channel, dc.campaign_name
),
CampaignCost AS (
    SELECT
        campaign_id,
        channel,
        campaign_name,
        campaign_cost
    FROM dim_campaign
)
-- Final output: Campaign ROI by Channel
SELECT
    cr.campaign_id,
    cr.campaign_name,
    cr.channel,
```

```
    cc.campaign_cost,
    cr.total_revenue,
    ROUND(cr.total_revenue - cc.campaign_cost, 2) AS net_profit,
    ROUND((cr.total_revenue - cc.campaign_cost) * 100.0 / cc.campaign_cost, 2) AS roi_percentage,
    cr.customers_converted
FROM CampaignRevenue cr
JOIN CampaignCost cc ON cr.campaign_id = cc.campaign_id
WHERE cc.campaign_cost > 0
ORDER BY roi_percentage DESC;


-- 28. What are the upsell and cross-sell opportunities based on past purchases?

WITH CustomerProducts AS (
    -- Get distinct product purchases per customer
    SELECT DISTINCT customer_id, product_id
    FROM sales_fact
),
CoPurchasedProductPairs AS (
    -- Find all product pairs bought by same customer
    SELECT
        a.product_id AS product_a,
        b.product_id AS product_b,
        p1.name AS name_a,
        p2.name AS name_b,
        p1.type AS type_a,
        p2.type AS type_b,
        COUNT(*) AS co_purchase_count
    FROM CustomerProducts a
    JOIN CustomerProducts b
        ON a.customer_id = b.customer_id AND a.product_id < b.product_id
    JOIN dim_product p1 ON a.product_id = p1.product_id
    JOIN dim_product p2 ON b.product_id = p2.product_id
    WHERE p1.type <> p2.type  -- Exclude same-type combinations
    GROUP BY a.product_id, b.product_id, p1.name, p2.name, p1.type, p2.type
)
-- Classify as upsell or cross-sell
SELECT
    name_a AS product_a,
    name_b AS product_b,
    type_a,
    type_b,
    co_purchase_count,
    CASE
        WHEN type_a = 'Frame' AND type_b = 'Lens' THEN 'Upsell (Frame  Lens)'
        WHEN type_a = 'Lens' AND type_b = 'Sunglasses' THEN 'Cross-sell (Lens  Sunglasses)'
        WHEN type_a = 'Frame' AND type_b = 'Sunglasses' THEN 'Cross-sell (Frame  Sunglasses)'
        WHEN type_a = 'Lens' AND type_b = 'Lens Cleaner' THEN 'Cross-sell (Lens  Accessory)'
        WHEN type_a = 'Frame' AND type_b = 'Case' THEN 'Cross-sell'
        ELSE 'Cross-sell'
    END AS opportunity_type
FROM CoPurchasedProductPairs
ORDER BY co_purchase_count DESC
LIMIT 50;


-- 29. How many first-time customers convert into repeat buyers within 90 days?

WITH FirstPurchases AS (
    -- Get each customer's first purchase date and cohort month
```

```
    SELECT
        customer_id,
        MIN(date_id) AS first_purchase_date,
        DATE_FORMAT(MIN(date_id), '%Y-%m') AS cohort_month
    FROM sales_fact
    GROUP BY customer_id
),
RepeatPurchases AS (
    -- Find customers who made at least one more purchase within 90 days
    SELECT
        fp.customer_id,
        fp.cohort_month
    FROM FirstPurchases fp
    JOIN sales_fact sf
        ON fp.customer_id = sf.customer_id
        AND sf.date_id > fp.first_purchase_date
        AND sf.date_id <= DATE_ADD(fp.first_purchase_date, INTERVAL 90 DAY)
    GROUP BY fp.customer_id, fp.cohort_month
    HAVING COUNT(sf.sale_id) >= 1
)
-- Aggregate results by cohort (month)
SELECT
    fp.cohort_month,
    COUNT(DISTINCT fp.customer_id) AS total_new_customers,
    COUNT(DISTINCT rp.customer_id) AS repeat_customers_within_90_days,
    ROUND(
        COUNT(DISTINCT rp.customer_id) * 100.0 / NULLIF(COUNT(DISTINCT fp.customer_id), 0),
        2
    ) AS conversion_rate_percent
FROM FirstPurchases fp
LEFT JOIN RepeatPurchases rp
    ON fp.customer_id = rp.customer_id
GROUP BY fp.cohort_month
ORDER BY fp.cohort_month;

-- □ BUSINESS STRATEGY & EXPANSION

-- 30. Which city or region should MyGlasses expand into next based on current demand patterns?

WITH CustomerDemand AS (
    SELECT
        dc.location AS region,
        COUNT(DISTINCT dc.customer_id) AS total_customers,
        COUNT(sf.sale_id) AS total_transactions,
        SUM(sf.revenue) AS total_revenue
    FROM dim_customer dc
    JOIN sales_fact sf ON dc.customer_id = sf.customer_id
    GROUP BY dc.location
),
StorePresence AS (
    SELECT
        region,
        COUNT(*) AS store_count
    FROM dim_store
    GROUP BY region
)
-- Top 5 regions with no stores and high demand
SELECT
    cd.region,
    cd.total_customers,
```

```
    cd.total_transactions,
    cd.total_revenue,
    COALESCE(sp.store_count, 0) AS store_count
FROM CustomerDemand cd
LEFT JOIN StorePresence sp ON cd.region = sp.region
WHERE COALESCE(sp.store_count, 0) = 0
ORDER BY cd.total_revenue DESC
LIMIT 5;
```

Top 50 Business Questions

```
-- □ Q1: Products with Declining Sales Despite Campaign Exposure
WITH CampaignSales AS (
    SELECT
        sf.product_id,
        dp.name AS product_name,
        sf.date_id,
        SUM(sf.revenue) AS total_revenue
    FROM campaign_fact cf
    JOIN sales_fact sf ON cf.sale_id = sf.sale_id
    JOIN dim_product dp ON sf.product_id = dp.product_id
    GROUP BY sf.product_id, dp.name, sf.date_id
),
MonthlyProductSales AS (
    SELECT
        product_id,
        product_name,
        YEAR(date_id) AS year,
        MONTH(date_id) AS month,
        SUM(total_revenue) AS monthly_revenue,
        LAG(SUM(total_revenue), 1) OVER (PARTITION BY product_id ORDER BY YEAR(date_id), MONTH(date_id)) AS prev
    FROM CampaignSales
    GROUP BY product_id, product_name, year, month
)
SELECT
    product_id,
    product_name,
    year,
    month,
    monthly_revenue,
    prev_month_revenue,
    ROUND(
        (monthly_revenue - prev_month_revenue) * 100.0 / NULLIF(prev_month_revenue, 0), 2
    ) AS revenue_growth_percent
FROM MonthlyProductSales
WHERE prev_month_revenue IS NOT NULL
  AND ((monthly_revenue - prev_month_revenue) * 100.0 / prev_month_revenue) < 0
ORDER BY revenue_growth_percent ASC;


-- □ Q2: What causes stockouts on popular products?

-- □ Popular products that frequently fall below reorder level
```

```
WITH ProductPopularity AS (
    SELECT
        sf.product_id,
        dp.name AS product_name,
        COUNT(sf.sale_id) AS total_sales,
        SUM(sf.quantity) AS units_sold
    FROM sales_fact sf
    JOIN dim_product dp ON sf.product_id = dp.product_id
    GROUP BY sf.product_id, dp.name
    ORDER BY units_sold DESC
),
StockAlerts AS (
    SELECT
        product_id,
        COUNT(*) AS stockout_count
    FROM inventory_fact
    WHERE stock_on_hand < reorder_level
    GROUP BY product_id
)
SELECT
    pp.product_id,
    pp.product_name,
    pp.units_sold,
    sa.stockout_count,
    ROUND(sa.stockout_count * 100.0 / pp.units_sold, 2) AS stockout_rate
FROM ProductPopularity pp
JOIN StockAlerts sa ON pp.product_id = sa.product_id
ORDER BY stockout_rate DESC;


-- Q3: Which stores have the highest return rates?

-- □ Return rate per store (returns vs total sales)

SELECT
    ds.store_id,
    ds.region,
    ds.city,
    COUNT(rf.return_id) * 100.0 / COUNT(sf.sale_id) AS return_rate_percent
FROM dim_store ds
LEFT JOIN sales_fact sf ON ds.store_id = sf.store_id
LEFT JOIN return_fact rf ON sf.sale_id = rf.sale_id
GROUP BY ds.store_id, ds.region, ds.city
ORDER BY return_rate_percent DESC;


-- Q4: Why are certain products returned more frequently?

-- □ Top reasons for returns by product type

SELECT
    dp.type AS product_type,
    rf.reason,
    COUNT(*) AS return_count
FROM return_fact rf
JOIN sales_fact sf ON rf.sale_id = sf.sale_id
JOIN dim_product dp ON sf.product_id = dp.product_id
GROUP BY dp.type, rf.reason
ORDER BY product_type, return_count DESC;
```

```
-- Q5: What is the average time to restock items?

-- □ Avg days between order and actual delivery

SELECT
    ro.product_id,
    dp.name AS product_name,
    dp.type AS product_type,
    COUNT(*) AS orders_count,
    ROUND(AVG(DATEDIFF(ro.actual_delivery_date, ro.order_date)), 2) AS avg_lead_time_days
FROM restock_orders ro
JOIN dim_product dp ON ro.product_id = dp.product_id
WHERE ro.actual_delivery_date IS NOT NULL
GROUP BY ro.product_id, dp.name, dp.type
ORDER BY avg_lead_time_days DESC;


-- Q6: How does customer buying behavior vary by region?

-- □ Sales patterns by region

SELECT
    ds.region,
    dc.gender,
    dc.loyalty_tier,
    COUNT(sf.sale_id) AS transaction_count,
    SUM(sf.revenue) AS total_revenue,
    ROUND(AVG(sf.revenue), 2) AS avg_transaction_value
FROM sales_fact sf
JOIN dim_store ds ON sf.store_id = ds.store_id
JOIN dim_customer dc ON sf.customer_id = dc.customer_id
GROUP BY ds.region, dc.gender, dc.loyalty_tier
ORDER BY region, total_revenue DESC;


-- Q7: Which customer segments contribute most to revenue?

-- □ Revenue contribution by age group + loyalty tier

SELECT
    FLOOR(dc.age / 10) * 10 AS age_group,
    dc.loyalty_tier,
    COUNT(sf.sale_id) AS transactions,
    SUM(sf.revenue) AS total_revenue,
    ROUND(SUM(sf.revenue) / COUNT(sf.sale_id), 2) AS avg_order_value
FROM sales_fact sf
JOIN dim_customer dc ON sf.customer_id = dc.customer_id
GROUP BY age_group, dc.loyalty_tier
ORDER BY total_revenue DESC;


-- Q8: How often do new customers return to purchase again?

-- □ Repeat rate of new customers (within 90 days)

WITH FirstPurchases AS (
    SELECT
        customer_id,
        MIN(date_id) AS first_purchase_date
```

```
    FROM sales_fact
    GROUP BY customer_id
),
RepeatPurchases AS (
    SELECT
        fp.customer_id
    FROM FirstPurchases fp
    JOIN sales_fact sf
        ON fp.customer_id = sf.customer_id
        AND sf.date_id > fp.first_purchase_date
        AND sf.date_id <= DATE_ADD(fp.first_purchase_date, INTERVAL 90 DAY)
    GROUP BY fp.customer_id
)
SELECT
    COUNT(DISTINCT rp.customer_id) * 100.0 / COUNT(DISTINCT fp.customer_id) AS repeat_conversion_rate
FROM FirstPurchases fp
LEFT JOIN RepeatPurchases rp ON fp.customer_id = rp.customer_id;


-- Q9: Are there sales trends linked to seasons or festivals?

-- □ Monthly sales trend with seasonality

SELECT
    d.year,
    d.month,
    SUM(sf.revenue) AS total_revenue,
    ROUND(AVG(sf.revenue), 2) AS avg_transaction_value
FROM sales_fact sf
JOIN dim_date d ON sf.date_id = d.date_id
GROUP BY d.year, d.month
ORDER BY d.year, d.month;


-- Q10: What is the financial impact of returns on overall profitability?

-- □ Total refunds as % of revenue

SELECT
    ROUND(SUM(sf.revenue), 2) AS total_revenue,
    ROUND(SUM(rf.refund_amount), 2) AS total_refunds,
    ROUND(SUM(rf.refund_amount) * 100.0 / SUM(sf.revenue), 2) AS refund_percent_of_total
FROM sales_fact sf
LEFT JOIN return_fact rf ON sf.sale_id = rf.sale_id;









-- ########################################################
-- # QUESTIONS 11 - 50: BUSINESS ANALYTICS QUERIES #
-- ########################################################

-- □ Q11: Which stores perform best in customer satisfaction proxies?

SELECT
```

```
    ds.store_id,
    ds.region,
    ds.city,
    COUNT(rf.return_id) AS total_returns,
    ROUND(AVG(rf.refund_amount), 2) AS avg_refund,
    COUNT(rf.return_id) * 100.0 / COUNT(sf.sale_id) AS return_rate_percent
FROM sales_fact sf
JOIN dim_store ds ON sf.store_id = ds.store_id
LEFT JOIN return_fact rf ON sf.sale_id = rf.sale_id
GROUP BY ds.store_id, ds.region, ds.city
ORDER BY return_rate_percent ASC;

-- □ Q12: How well do loyalty tiers correlate with repeat customers?

WITH FirstPurchases AS (
    SELECT
        customer_id,
        MIN(date_id) AS first_purchase_date
    FROM sales_fact
    GROUP BY customer_id
),
RepeatCustomers AS (
    SELECT
        fp.customer_id
    FROM FirstPurchases fp
    JOIN sales_fact sf
        ON fp.customer_id = sf.customer_id
        AND sf.date_id > DATE_ADD(fp.first_purchase_date, INTERVAL 30 DAY)
    GROUP BY fp.customer_id
)
SELECT
    dc.loyalty_tier,
    COUNT(DISTINCT dc.customer_id) AS total_customers,
    COUNT(DISTINCT rc.customer_id) AS repeat_customers,
    ROUND(COUNT(DISTINCT rc.customer_id) * 100.0 / NULLIF(COUNT(DISTINCT dc.customer_id), 0), 2) AS repeat_rate
FROM dim_customer dc
LEFT JOIN RepeatCustomers rc ON dc.customer_id = rc.customer_id
GROUP BY dc.loyalty_tier
ORDER BY repeat_rate DESC;


-- □ Q13: What is the SKU-level inventory turnover?

SELECT
    inv.store_id,
    inv.product_id,
    dp.name AS product_name,
    SUM(sf.quantity) AS units_sold,
    AVG(inv.stock_on_hand) AS avg_stock_level,
    ROUND(SUM(sf.quantity) / NULLIF(AVG(inv.stock_on_hand), 0), 2) AS turnover_rate
FROM inventory_fact inv
JOIN sales_fact sf ON inv.store_id = sf.store_id AND inv.product_id = sf.product_id
JOIN dim_product dp ON inv.product_id = dp.product_id
GROUP BY inv.store_id, inv.product_id, dp.name
ORDER BY turnover_rate DESC;


-- □ Q14: Which products have high margin but low sales?

SELECT
```

```
    dp.product_id,
    dp.name,
    dp.type,
    dp.price,
    dpc.cost_per_unit,
    ROUND(dp.price - dpc.cost_per_unit, 2) AS gross_profit,
    ROUND((dp.price - dpc.cost_per_unit) * 100.0 / dp.price, 2) AS profit_margin_percent,
    SUM(sf.quantity) AS total_units_sold
FROM dim_product dp
JOIN dim_product_cost dpc ON dp.product_id = dpc.product_id
LEFT JOIN sales_fact sf ON dp.product_id = sf.product_id
GROUP BY dp.product_id, dp.name, dp.type, dp.price, dpc.cost_per_unit
HAVING total_units_sold < 100 -- Adjust threshold as needed
ORDER BY profit_margin_percent DESC;

-- □ Q15: What is the effect of pricing changes on sales volume?

WITH ProductPriceHistory AS (
    SELECT
        dp.product_id,
        sf.date_id,
        dp.price,
        SUM(sf.quantity) AS units_sold
    FROM sales_fact sf
    JOIN dim_product dp ON sf.product_id = dp.product_id
    GROUP BY dp.product_id, sf.date_id, dp.price
)
SELECT
    product_id,
    date_id,
    price,
    units_sold,
    LEAD(units_sold, 1) OVER (PARTITION BY product_id ORDER BY date_id) AS next_sale_units,
    ROUND(
        ((LEAD(units_sold, 1) OVER (PARTITION BY product_id ORDER BY date_id) - units_sold) * 100.0 / NULLIF(units_sold
    ) AS unit_change_after_price_change
FROM ProductPriceHistory
ORDER BY product_id, date_id;

-- □ Q16: Are there regional product preferences?
-- □ Top-selling product types by region
SELECT
    ds.region,
    dp.type AS product_type,
    SUM(sf.quantity) AS units_sold,
    ROUND(SUM(sf.revenue), 2) AS total_revenue
FROM sales_fact sf
JOIN dim_store ds ON sf.store_id = ds.store_id
JOIN dim_product dp ON sf.product_id = dp.product_id
GROUP BY ds.region, dp.type
ORDER BY ds.region, total_revenue DESC;

-- □ Q17: What products generate the highest repeat purchase rate?
-- □ Products bought multiple times by same customer
WITH CustomerProductPurchases AS (
    SELECT
        sf.customer_id,
        sf.product_id,
        COUNT(sf.sale_id) AS purchase_count
    FROM sales_fact sf
```

```
    GROUP BY sf.customer_id, sf.product_id
)
SELECT
    dp.product_id,
    dp.name AS product_name,
    dp.type AS product_type,
    COUNT(cpp.customer_id) AS customers_who_repurchased,
    SUM(cpp.purchase_count) AS total_purchases,
    ROUND(COUNT(cpp.customer_id) * 100.0 / NULLIF(COUNT(DISTINCT sf.customer_id), 0), 2) AS repurchase_rate
FROM CustomerProductPurchases cpp
JOIN dim_product dp ON cpp.product_id = dp.product_id
JOIN sales_fact sf ON cpp.product_id = sf.product_id
GROUP BY dp.product_id, dp.name, dp.type
ORDER BY repurchase_rate DESC;

-- □ Q18: How efficient is the supply chain in meeting demand?
-- □ Stockouts vs sales performance
SELECT
    inv.product_id,
    dp.name AS product_name,
    inv.store_id,
    ds.region,
    inv.stock_on_hand,
    inv.reorder_level,
    COUNT(sf.sale_id) AS total_sales,
    CASE
        WHEN inv.stock_on_hand <= 0 THEN 'Out of Stock'
        WHEN inv.stock_on_hand < inv.reorder_level THEN 'Low Stock'
        ELSE 'Stocked Well'
    END AS stock_status
FROM inventory_fact inv
JOIN sales_fact sf ON inv.store_id = sf.store_id AND inv.product_id = sf.product_id
JOIN dim_store ds ON inv.store_id = ds.store_id
JOIN dim_product dp ON inv.product_id = dp.product_id
GROUP BY inv.product_id, inv.store_id, inv.stock_on_hand, inv.reorder_level, ds.region, dp.name
ORDER BY stock_status DESC, total_sales DESC;

-- □ Q19: What is the percentage of online vs in-store sales?
-- □ Understand channel split
SELECT
    ds.type AS store_type,
    COUNT(sf.sale_id) AS transaction_count,
    SUM(sf.revenue) AS total_revenue,
    ROUND(COUNT(sf.sale_id) * 100.0 / (SELECT COUNT(*) FROM sales_fact), 2) AS percent_of_total_sales
FROM sales_fact sf
JOIN dim_store ds ON sf.store_id = ds.store_id
GROUP BY ds.type
ORDER BY total_revenue DESC;

-- □ Q20: Which marketing campaigns have the best ROI?
-- □ Campaign cost vs revenue generated
WITH CampaignRevenue AS (
    SELECT
        cf.campaign_id,
        SUM(sf.revenue) AS total_revenue
    FROM campaign_fact cf
    JOIN sales_fact sf ON cf.sale_id = sf.sale_id
    GROUP BY cf.campaign_id
),
CampaignCosts AS (
```

```
    SELECT campaign_id, campaign_cost
    FROM dim_campaign
)
SELECT
    cc.campaign_id,
    cr.total_revenue,
    cc.campaign_cost,
    ROUND(cr.total_revenue - cc.campaign_cost, 2) AS net_profit,
    ROUND((cr.total_revenue - cc.campaign_cost) * 100.0 / cc.campaign_cost, 2) AS roi_percentage
FROM CampaignRevenue cr
JOIN CampaignCosts cc ON cr.campaign_id = cc.campaign_id
WHERE cc.campaign_cost > 0
ORDER BY roi_percentage DESC;

-- □ Q21: What customer demographics are underserved?
-- □ Sales distribution by age group and location
SELECT
    FLOOR(dc.age / 10) * 10 AS age_group,
    dc.location,
    dc.gender,
    COUNT(DISTINCT dc.customer_id) AS customer_count,
    SUM(sf.revenue) AS total_revenue,
    ROUND(SUM(sf.revenue) / COUNT(DISTINCT dc.customer_id), 2) AS avg_revenue_per_customer
FROM dim_customer dc
JOIN sales_fact sf ON dc.customer_id = sf.customer_id
GROUP BY age_group, dc.location, dc.gender
ORDER BY avg_revenue_per_customer DESC;

-- □ Q22: Are customers satisfied with product delivery times?
-- □ Average days between order and sale (for online orders)
-- □ Q22: Customer satisfaction via purchase timing (online orders only)

SELECT
    csd.device_type,
    csd.time_of_day,
    csd.session_duration_min,
    ROUND(AVG(DATEDIFF(sf.date_id, csd.session_date)), 2) AS avg_days_to_purchase
FROM customer_session_data csd
JOIN sales_fact sf ON csd.customer_id = sf.customer_id
WHERE csd.purchase_made = 1
GROUP BY csd.device_type, csd.time_of_day, csd.session_duration_min
ORDER BY avg_days_to_purchase;


-- □ Q23 (Diagnosis): Inventory Discrepancies or Shrinkage Detection

SELECT
    inv.store_id,
    inv.product_id,
    dp.name AS product_name,
    MAX(inv.stock_on_hand) AS current_stock,
    SUM(sf.quantity) AS total_sold_last_90_days,
    MAX(inv.stock_on_hand) - SUM(sf.quantity) AS expected_stock_left
FROM inventory_fact inv
JOIN sales_fact sf
    ON inv.store_id = sf.store_id
    AND inv.product_id = sf.product_id
    AND DATE(sf.date_id) >= DATE_SUB(CURDATE(), INTERVAL 90 DAY)
JOIN dim_product dp ON inv.product_id = dp.product_id
GROUP BY inv.store_id, inv.product_id, dp.name
```

```
HAVING expected_stock_left < 0
ORDER BY expected_stock_left ASC;


-- □ Q24: Impact of product bundling on sales

WITH CustomerBundles AS (
    SELECT
        sf.customer_id,
        sf.date_id,
        GROUP_CONCAT(DISTINCT dp.type ORDER BY dp.type SEPARATOR '+') AS bundle_type,
        COUNT(DISTINCT dp.type) AS unique_types_in_bundle,
        SUM(sf.quantity) AS total_items,
        SUM(sf.revenue) AS bundle_revenue
    FROM sales_fact sf
    JOIN dim_product dp ON sf.product_id = dp.product_id
    GROUP BY sf.customer_id, sf.date_id
    HAVING unique_types_in_bundle > 1
),
BundleAgg AS (
    SELECT
        bundle_type,
        COUNT(*) AS transaction_count,
        ROUND(AVG(bundle_revenue), 2) AS avg_revenue_per_bundle,
        ROUND(SUM(bundle_revenue), 2) AS total_revenue_from_bundles
    FROM CustomerBundles
    GROUP BY bundle_type
)
SELECT *
FROM BundleAgg
ORDER BY avg_revenue_per_bundle DESC;

-- □ Q25: How Does Weather Affect Eyewear Sales?

SELECT
    wd.weather_condition,
    wd.avg_temperature_celsius,
    wd.precipitation_mm,
    SUM(sf.revenue) AS total_revenue,
    COUNT(DISTINCT sf.sale_id) AS transaction_count,
    ROUND(SUM(sf.revenue) / NULLIF(COUNT(DISTINCT sf.sale_id), 0), 2) AS avg_revenue_per_day
FROM sales_fact sf
JOIN dim_store ds ON sf.store_id = ds.store_id
JOIN weather_data wd ON sf.date_id = wd.date AND ds.city = wd.city
GROUP BY wd.weather_condition, wd.avg_temperature_celsius, wd.precipitation_mm
ORDER BY transaction_count DESC;

-- □ Q26: Are there any patterns in returns related to product batch?
-- □ Returns grouped by product type and reason
SELECT
    dp.type AS product_type,
    rf.reason,
    COUNT(rf.return_id) AS return_count
FROM return_fact rf
JOIN sales_fact sf ON rf.sale_id = sf.sale_id
JOIN dim_product dp ON sf.product_id = dp.product_id
GROUP BY dp.type, rf.reason
ORDER BY return_count DESC;

-- □ Q27: Which stores are ready for expansion based on sales growth?
```

```sql
-- □ Growth rate over last 6 months
WITH MonthlySales AS (
  SELECT
    store_id,
    DATE_FORMAT(date_id, '%Y-%m') AS month,
    SUM(revenue) AS monthly_revenue
  FROM sales_fact
  WHERE date_id >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
  GROUP BY store_id, DATE_FORMAT(date_id, '%Y-%m')
),
StoreGrowth AS (
  SELECT
    store_id,
    LAG(monthly_revenue, 1) OVER (PARTITION BY store_id ORDER BY month) AS prev_revenue,
    monthly_revenue
  FROM MonthlySales
)
SELECT
  store_id,
  ROUND(
    (monthly_revenue - prev_revenue) * 100.0 / NULLIF(prev_revenue, 0), 2
  ) AS growth_rate
FROM StoreGrowth
WHERE prev_revenue IS NOT NULL
ORDER BY growth_rate DESC;

-- □ Q28: What is the average purchase frequency per customer segment?
-- □ Days between first and second purchases
WITH PurchaseFrequency AS (
  SELECT
    customer_id,
    COUNT(sale_id) AS total_purchases,
    DATEDIFF(MAX(date_id), MIN(date_id)) AS days_between_first_last
  FROM sales_fact
  GROUP BY customer_id
  HAVING total_purchases > 1
)
SELECT
  dc.loyalty_tier,
  FLOOR(dc.age / 10) * 10 AS age_group,
  ROUND(AVG(days_between_first_last / (total_purchases - 1)), 2) AS avg_days_between_purchases
FROM PurchaseFrequency pf
JOIN dim_customer dc ON pf.customer_id = dc.customer_id
GROUP BY dc.loyalty_tier, age_group
ORDER BY avg_days_between_purchases;

-- □ Q29: Which products drive the highest customer acquisition?
-- □ Count how many unique customers bought each product
SELECT
  dp.product_id,
  dp.name AS product_name,
  COUNT(DISTINCT sf.customer_id) AS unique_customers,
  SUM(sf.revenue) AS total_revenue
FROM sales_fact sf
JOIN dim_product dp ON sf.product_id = dp.product_id
GROUP BY dp.product_id, dp.name
ORDER BY unique_customers DESC;

-- □ Q30: Are customers dropping off after the first purchase?
-- □ New customers who never returned
```

```
WITH FirstLastPurchase AS (
    SELECT
        customer_id,
        MIN(date_id) AS first_purchase,
        MAX(date_id) AS last_purchase
    FROM sales_fact
    GROUP BY customer_id
)
SELECT
    dc.loyalty_tier,
    FLOOR(dc.age / 10) * 10 AS age_group,
    COUNT(*) AS new_customers,
    COUNT(fl.last_purchase) AS returning_customers,
    ROUND(
        (COUNT(*) - COUNT(fl.last_purchase)) * 100.0 / COUNT(*), 2
    ) AS drop_off_rate
FROM dim_customer dc
LEFT JOIN FirstLastPurchase fl ON dc.customer_id = fl.customer_id
GROUP BY dc.loyalty_tier, age_group
ORDER BY drop_off_rate DESC;

-- □ Q31: Rate of product defects causing returns
SELECT
    dp.type AS product_type,
    rf.reason,
    COUNT(rf.return_id) AS return_count,
    SUM(sf.quantity) AS total_units_sold,
    ROUND(COUNT(rf.return_id) * 100.0 / NULLIF(SUM(sf.quantity), 0), 2) AS return_rate
FROM return_fact rf
JOIN sales_fact sf ON rf.sale_id = sf.sale_id
JOIN dim_product dp ON sf.product_id = dp.product_id
WHERE rf.reason IN ('Damaged', 'Defective', 'Faulty', 'Manufacturing Defect')
GROUP BY dp.type, rf.reason
ORDER BY return_rate DESC;

-- □ Q32: How effective is the reorder level in preventing stockouts?
-- □ Compare reorder level vs actual stock used
SELECT
    inv.product_id,
    dp.name,
    inv.store_id,
    AVG(inv.reorder_level) AS avg_reorder_level,
    AVG(inv.stock_on_hand) AS avg_stock_level,
    SUM(CASE WHEN inv.stock_on_hand < inv.reorder_level THEN 1 ELSE 0 END) AS understocked_count
FROM inventory_fact inv
JOIN dim_product dp ON inv.product_id = dp.product_id
GROUP BY inv.product_id, inv.store_id
HAVING understocked_count > 0
ORDER BY avg_stock_level DESC;

-- □ Q33: Which stores have the highest average transaction size?
-- □ Revenue per sale by store
SELECT
    ds.store_id,
    ds.region,
    ds.city,
    COUNT(sf.sale_id) AS transactions,
    SUM(sf.revenue) AS total_revenue,
    ROUND(AVG(sf.revenue), 2) AS avg_transaction_value
FROM sales_fact sf
```

```
JOIN dim_store ds ON sf.store_id = ds.store_id
GROUP BY ds.store_id, ds.region, ds.city
ORDER BY avg_transaction_value DESC;

-- □ Q34: Impact of new product launches on overall sales
SELECT
    dp.product_id,
    dp.name AS product_name,
    dp.type,
    dp.brand,
    SUM(CASE WHEN sf.date_id BETWEEN '2023-01-01' AND '2023-01-31' THEN sf.revenue ELSE 0 END) AS pre_launch_
    SUM(CASE WHEN sf.date_id BETWEEN '2023-02-01' AND '2023-02-28' THEN sf.revenue ELSE 0 END) AS post_launch
    ROUND(
        (SUM(CASE WHEN sf.date_id BETWEEN '2023-02-01' AND '2023-02-28' THEN sf.revenue ELSE 0 END) -
         SUM(CASE WHEN sf.date_id BETWEEN '2023-01-01' AND '2023-01-31' THEN sf.revenue ELSE 0 END)
        ) * 100.0 / NULLIF(SUM(CASE WHEN sf.date_id BETWEEN '2023-01-01' AND '2023-01-31' THEN sf.revenue ELSE 0
        2
    ) AS growth_percent
FROM dim_product dp
LEFT JOIN sales_fact sf ON dp.product_id = sf.product_id
GROUP BY dp.product_id, dp.name
HAVING post_launch_sales > pre_launch_sales
ORDER BY growth_percent DESC;

-- □ Q35: Which customer segments are price sensitive?
-- □ Analyze discount usage across demographics
SELECT
    dc.loyalty_tier,
    FLOOR(dc.age / 10) * 10 AS age_group,
    dc.gender,
    ROUND(AVG(sf.discount_applied), 2) AS avg_discount_used,
    COUNT(*) AS total_purchases,
    COUNT(CASE WHEN sf.discount_applied > 0 THEN 1 END) AS discounted_purchases,
    ROUND(
        COUNT(CASE WHEN sf.discount_applied > 0 THEN 1 END) * 100.0 / COUNT(*), 2
    ) AS discount_usage_percent
FROM sales_fact sf
JOIN dim_customer dc ON sf.customer_id = dc.customer_id
GROUP BY dc.loyalty_tier, age_group, dc.gender
ORDER BY discount_usage_percent DESC;

-- □ Q36: Improve cross-selling based on purchase patterns
-- □ Most co-purchased product pairs
WITH CoPurchases AS (
    SELECT
        a.product_id AS product_a,
        b.product_id AS product_b,
        COUNT(*) AS co_purchase_count
    FROM sales_fact a
    JOIN sales_fact b ON a.sale_id = b.sale_id AND a.product_id < b.product_id
    GROUP BY product_a, product_b
)
SELECT
    p1.name AS product_a,
    p2.name AS product_b,
    cp.co_purchase_count,
    p1.type AS type_a,
    p2.type AS type_b
FROM CoPurchases cp
JOIN dim_product p1 ON cp.product_a = p1.product_id
```

```
JOIN dim_product p2 ON cp.product_b = p2.product_id
WHERE p1.type <> p2.type
ORDER BY co_purchase_count DESC
LIMIT 50;

-- □ Q37: Ratio of in-store to online returns
-- □ Helps optimize processes
SELECT
    ds.type AS store_type,
    COUNT(rf.return_id) AS return_count,
    COUNT(sf.sale_id) AS total_sales,
    ROUND(COUNT(rf.return_id) * 100.0 / NULLIF(COUNT(sf.sale_id), 0), 2) AS return_rate
FROM return_fact rf
JOIN sales_fact sf ON rf.sale_id = sf.sale_id
JOIN dim_store ds ON sf.store_id = ds.store_id
GROUP BY ds.type
ORDER BY return_rate DESC;

-- □ Q38: Stores with highest footfall
-- □ Helps plan staffing and promotions
SELECT
    fd.store_id,
    ds.region,
    ds.city,
    fd.total_footfall,
    fd.conversion_rate_percent,
    fd.avg_stay_minutes,
    fd.peak_hour
FROM footfall_data fd
JOIN dim_store ds ON fd.store_id = ds.store_id
ORDER BY fd.total_footfall DESC;

-- □ Q39: Inventory restock around holiday periods
SELECT
    i.product_id,
    p.name AS product_name,
    i.store_id,
    i.last_restocked,
    i.stock_on_hand,
    i.reorder_level,
    CASE
        WHEN DAYOFYEAR(i.last_restocked) >= 355 OR DAYOFYEAR(i.last_restocked) <= 10 THEN 'Holiday Period'
        ELSE 'Normal'
    END AS season_type
FROM inventory_fact i
JOIN dim_product p ON i.product_id = p.product_id
ORDER BY i.last_restocked DESC;

-- □ Q40: Are customer complaints related to specific products or stores?
-- □ Return reasons by store and product
SELECT
    ds.region,
    ds.city,
    dp.type AS product_type,
    rf.reason,
    COUNT(*) AS return_count
FROM return_fact rf
JOIN sales_fact sf ON rf.sale_id = sf.sale_id
JOIN dim_store ds ON sf.store_id = ds.store_id
JOIN dim_product dp ON sf.product_id = dp.product_id
```

```
GROUP BY ds.region, ds.city, dp.type, rf.reason
ORDER BY return_count DESC;

-- □ Q41: Customer Satisfaction Impact on Repeat Purchases

WITH FirstPurchase AS (
    SELECT
        customer_id,
        MIN(date_id) AS first_purchase_date
    FROM sales_fact
    GROUP BY customer_id
),
CustomerSatisfaction AS (
    SELECT
        dc.customer_id,
        dc.loyalty_tier,
        FLOOR(dc.age / 10) * 10 AS age_group,
        dc.gender,
        COUNT(rf.return_id) AS returns,
        COUNT(sf.sale_id) AS total_purchases,
        ROUND(COUNT(rf.return_id) * 100.0 / NULLIF(COUNT(sf.sale_id), 0), 2) AS return_rate
    FROM dim_customer dc
    JOIN sales_fact sf ON dc.customer_id = sf.customer_id
    LEFT JOIN return_fact rf ON sf.sale_id = rf.sale_id
    GROUP BY dc.customer_id, dc.loyalty_tier, age_group, dc.gender
)
SELECT
    loyalty_tier,
    age_group,
    gender,
    ROUND(AVG(return_rate), 2) AS avg_return_rate,
    COUNT(DISTINCT customer_id) AS total_customers,
    COUNT(DISTINCT CASE WHEN return_rate < 10 THEN customer_id END) AS satisfied_customers,
    ROUND(
        COUNT(DISTINCT CASE WHEN return_rate < 10 THEN customer_id END) * 100.0 / COUNT(DISTINCT customer_id)
    ) AS satisfaction_percent
FROM CustomerSatisfaction
GROUP BY loyalty_tier, age_group, gender
ORDER BY satisfaction_percent DESC;

-- □ Q42: Ratio of New vs Returning Customers Over Time
-- □ Helps understand acquisition vs retention

WITH FirstPurchase AS (
    SELECT
        sf.customer_id,
        sf.date_id,
        sf.revenue,
        MIN(sf.date_id) OVER (PARTITION BY sf.customer_id) AS first_purchase_date
    FROM sales_fact sf
),
CustomerType AS (
    SELECT
        fp.customer_id,
        fp.date_id,
        fp.revenue,
        fp.first_purchase_date,
        CASE
            WHEN fp.date_id = fp.first_purchase_date THEN 'New'
            ELSE 'Returning'
```

```
        END AS customer_type
    FROM FirstPurchase fp
)
SELECT
    DATE_FORMAT(date_id, '%Y-%m') AS month,
    customer_type,
    COUNT(customer_id) AS customer_count,
    SUM(revenue) AS total_revenue
FROM CustomerType
GROUP BY month, customer_type
ORDER BY month, customer_type;

-- □ Q43: Store Revenue vs Operational Costs

SELECT
    ds.store_id,
    ds.region,
    ds.city,
    SUM(sf.revenue) AS total_revenue,
    COALESCE(oc.operational_cost, 0) AS operational_costs,
    ROUND(SUM(sf.revenue) - oc.operational_cost, 2) AS net_profit,
    ROUND((SUM(sf.revenue) - oc.operational_cost) * 100.0 / NULLIF(oc.operational_cost, 0), 2) AS profit_vs_cost_ratio
FROM sales_fact sf
JOIN dim_store ds ON sf.store_id = ds.store_id
LEFT JOIN (
    SELECT store_id, SUM(operational_cost) AS operational_cost
    FROM external_cost_data
    WHERE month = '2023-01-01'
    GROUP BY store_id
) oc ON sf.store_id = oc.store_id
GROUP BY ds.store_id, ds.region, ds.city, oc.operational_cost
ORDER BY profit_vs_cost_ratio DESC;

-- □ Q44: Geographic areas underserved by current store coverage
-- □ Based on customer demand vs actual store presence

WITH LocationProductStats AS (
    SELECT
        dc.location,
        dp.type AS product_type,
        SUM(sf.quantity) AS units_sold,
        SUM(sf.revenue) AS total_revenue
    FROM sales_fact sf
    JOIN dim_customer dc ON sf.customer_id = dc.customer_id
    JOIN dim_product dp ON sf.product_id = dp.product_id
    GROUP BY dc.location, dp.type
),
LocationTotals AS (
    SELECT
        location,
        SUM(units_sold) AS total_units_sold,
        SUM(total_revenue) AS total_revenue
    FROM LocationProductStats
    GROUP BY location
),
StorePresence AS (
    SELECT DISTINCT city AS location
    FROM dim_store
)
SELECT
```

```
    lt.location,
    lt.total_units_sold,
    lt.total_revenue,
    COALESCE(sp.location, 'Not Covered') AS store_coverage_status
FROM LocationTotals lt
LEFT JOIN StorePresence sp ON lt.location = sp.location
WHERE sp.location IS NULL
ORDER BY lt.total_revenue DESC;

-- □ Q45: Impact of discounts on margins
-- □ Profitability analysis with and without discount
SELECT
    dp.type AS product_type,
    ROUND(AVG(sf.discount_applied), 2) AS avg_discount,
    ROUND(AVG(sf.revenue), 2) AS avg_revenue,
    ROUND(AVG(sf.revenue - sf.discount_applied), 2) AS net_avg_revenue
FROM sales_fact sf
JOIN dim_product dp ON sf.product_id = dp.product_id
GROUP BY dp.type
ORDER BY net_avg_revenue DESC;

-- □ Q46: Seasonal demand variability by product type
-- □ Sales trends by quarter
SELECT
    dp.type AS product_type,
    YEAR(sf.date_id) AS year,
    QUARTER(sf.date_id) AS quarter,
    SUM(sf.quantity) AS units_sold,
    SUM(sf.revenue) AS total_revenue
FROM sales_fact sf
JOIN dim_product dp ON sf.product_id = dp.product_id
GROUP BY product_type, year, quarter
ORDER BY product_type, year, quarter;

-- □ Q47: Common customer journey paths (online & offline)
-- □ Combine session + sales data
SELECT
    cs.session_type,
    cs.device_type,
    cs.time_of_day,
    COUNT(*) AS sessions,
    SUM(cs.purchase_made) AS conversions,
    ROUND(SUM(cs.purchase_made) * 100.0 / COUNT(*), 2) AS conversion_rate
FROM customer_session_data cs
GROUP BY session_type, device_type, time_of_day
ORDER BY conversion_rate DESC;

-- □ Q48: Customers Buying Multiple Categories

WITH MultiCategoryBuyers AS (
    SELECT
        customer_id,
        COUNT(DISTINCT dp.type) AS unique_types_bought
    FROM sales_fact sf
    JOIN dim_product dp ON sf.product_id = dp.product_id
    GROUP BY customer_id
    HAVING unique_types_bought > 1
)
SELECT
    dp.type AS product_type,
```

```
    COUNT(*) AS buyers_count,
    ROUND(COUNT(*) * 100.0 / (SELECT COUNT(DISTINCT customer_id) FROM sales_fact), 2) AS percent_buyers
FROM MultiCategoryBuyers mcb
JOIN sales_fact sf ON mcb.customer_id = sf.customer_id
JOIN dim_product dp ON sf.product_id = dp.product_id
GROUP BY dp.type
ORDER BY buyers_count DESC;

-- □ Q49: How does store location affect basket size?
-- □ Avg revenue per sale by city/region
SELECT
    ds.region,
    ds.city,
    COUNT(sf.sale_id) AS total_transactions,
    ROUND(SUM(sf.revenue), 2) AS total_revenue,
    ROUND(AVG(sf.revenue), 2) AS avg_basket_size
FROM sales_fact sf
JOIN dim_store ds ON sf.store_id = ds.store_id
GROUP BY ds.region, ds.city
ORDER BY avg_basket_size DESC;

-- □ Q50: Key drivers for customer churn
-- □ No recent purchase = churned
WITH LastPurchase AS (
    SELECT
        customer_id,
        MAX(date_id) AS last_purchase_date
    FROM sales_fact
    GROUP BY customer_id
)
SELECT
    dc.loyalty_tier,
    FLOOR(dc.age / 10) * 10 AS age_group,
    dc.gender,
    dc.location,
    COUNT(*) AS churned_customers
FROM dim_customer dc
JOIN LastPurchase lp ON dc.customer_id = lp.customer_id
WHERE lp.last_purchase_date < DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY dc.loyalty_tier, age_group, dc.gender, dc.location
ORDER BY churned_customers DESC;
```