Cross-Grain Brewing — Distribution Analysis

Subqueries, CASE expressions and multi-table analysis across brewing, distribution and taproom operations.

Instructions

Answer each of the 8 questions below using a single SQL query per question. Every table reference must be schema-qualified using the craftbrewery prefix (e.g., craftbrewery.beers), and every column reference must be table-qualified (e.g., beers.beerName) — do not use table aliases or bare column names. Write your queries in a single .sql file with each query preceded by a comment indicating its question number (e.g., -- Question 1). Submit your .sql file to the course portal before the deadline; late submissions will not be accepted. Point values are listed beside each question and total 100 points.

Scenario

Craft Brewery Co. is scaling up its operations and needs data-driven insights to manage its beers, batches, taproom sales, and wholesale accounts more effectively. The analytics team has asked you to write SQL queries against the craftbrewery database to answer key business questions spanning production, sales, and distribution. Your work will directly inform decisions about which beers to promote, which territories are performing best, and how taproom revenue is distributed.

Database Structure

Ironroot Craft Brewery is an independent craft brewery that brews a rotating lineup of ales, lagers, and seasonal specialties. It sells kegs and cases wholesale to bars and bottle shops across multiple regional territories, and also operates an on-site taproom where customers can purchase pints and growlers directly. The business tracks its brewing production, product catalog, wholesale customers, sales orders, and taproom transactions.

Questions

1. Retrieve the beer name, ABV, and IBU for all beers that are currently active (isActive = 1). Order the results by ABV in descending order. *(8 pts)*

2. List each beer's name along with its style name and description. Only include beers that have a matching style. Order the results alphabetically by style name, then by beer name. *(11 pts)*

---

3. For each account, show the account name, account type, and the total number of orders placed. Include accounts that have never placed an order (show 0 for those). Order by total orders descending. *(11 pts)*

> *Hint: A LEFT JOIN ensures accounts with no orders are still included in the result.*

---

4. Find all beer styles that have more than 5 active beers associated with them. Display the style name and the count of active beers, ordered by the count descending. *(12 pts)*

> *Hint: Filter on active beers before grouping, then use HAVING to restrict the grouped results.*

---

5. For each territory, show the territory name, the sales representative, and the total revenue (sum of lineTotal) generated from all orders placed by accounts in that territory. Only include orders with a status of 'Delivered'. Order by total revenue descending. *(12 pts)*

> *Hint: You will need to chain multiple joins from territories down to orderLines.*

6. List the batch code, brew date, volume in litres, and status for all batches that belong to beers tagged as 'Hoppy'. Use a subquery with IN to find the relevant beer IDs from the beerTags table. Order results by brew date ascending. *(15 pts)*

> *Hint: Use a subquery inside an IN clause to first identify which beers have the 'Hoppy' tag, then filter batches accordingly.*

---

7. For each beer that has taproom sales, show the beer name and a breakdown of total sale amount by serving category: 'Small' (servingType IN ('Flight', 'Half Pint')), 'Standard' (servingType = 'Pint'), and 'Large' (servingType IN ('Growler Fill', 'Crowler')). Display columns: beerName, smallTotal, standardTotal, largeTotal. Order by beerName ascending. *(16 pts)*

> *Hint: Wrap each CASE WHEN expression inside a SUM() to pivot the serving types into separate columns.*

---

8. Identify accounts whose average order line unit price is greater than the overall average unit price across all order lines. Display the account name, city, account type, and their average unit price (rounded to 2 decimal places), ordered by average unit price descending. *(15 pts)*

> *Hint: Use a scalar subquery inside the HAVING clause to compute the global average unit price to compare against each account's average.*
