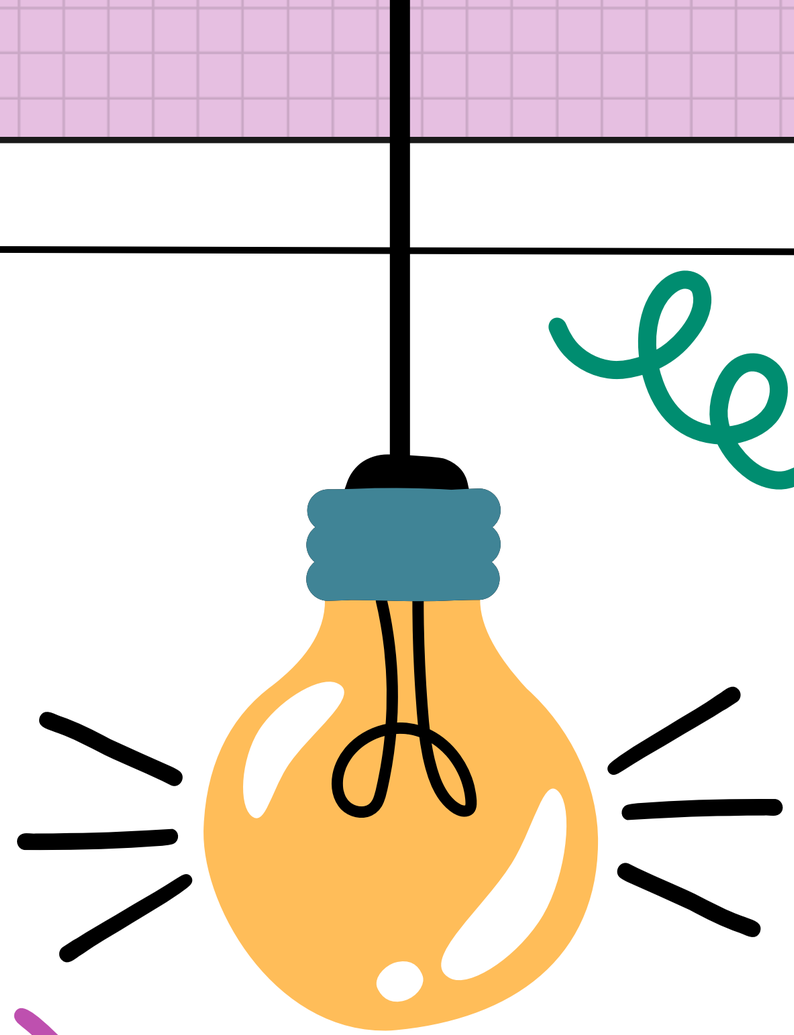
# DELIVR EATERY SQL ANALYSIS



By Andraine Wallace

## A comprehensive executive report presenting the quarterly rankings of eateries based on the number of unique users who have placed orders with them.

```
-- Import tablefunc
CREATE EXTENSION IF NOT EXISTS tablefunc;

-- Pivot the previous query by quarter
SELECT * FROM CROSSTAB($$
  WITH eatery_users AS  (
    SELECT
      eatery,
      -- Format the order date so "2018-06-01" becomes "Q2 2018"
      TO_CHAR(order_date, '"Q"Q YYYY') AS delivr_quarter,
      -- Count unique users
      COUNT(DISTINCT user_id) AS users
    FROM meals
    JOIN orders ON meals.meal_id = orders.meal_id
    GROUP BY eatery, delivr_quarter
    ORDER BY delivr_quarter, users)

  SELECT
    -- Select eatery and quarter
    eatery,
    delivr_quarter,
    -- Rank rows, partition by quarter and order by users
    RANK() OVER
      (PARTITION BY delivr_quarter
       ORDER BY users DESC) :: INT AS users_rank
  FROM eatery_users
  ORDER BY eatery, delivr_quarter;
$$)
-- Select the columns of the pivoted table
AS  ct (eatery TEXT,
        "Q2 2018" INT,
        "Q3 2018" INT,
        "Q4 2018" INT)
ORDER BY "Q4 2018";
```

## Results:

| eatery | Q2 2018 | Q3 2018 | Q4 2018 |
|---|---|---|---|
| 'The Moon Wok' | 1 | 1 | 1 |
| 'Burgatorio' | 2 | 2 | 2 |
| 'Beam Me Up Scotty' | 3 | 3 | 3 |
| 'Leaning Tower of Pizza' | 4 | 4 | 4 |
| 'Life of Pie' | 5 | 5 | 5 |

## Average monthly cost before September 2018

```sql
-- CTE named monthly_cost
WITH monthly_cost AS (
  SELECT
    DATE_TRUNC('month', stocking_date)::DATE AS delivr_month,
    SUM(meal_cost * stocked_quantity) AS cost
  FROM meals
  JOIN stock ON meals.meal_id = stock.meal_id
  GROUP BY delivr_month)

SELECT
  -- Average monthly cost before September
  AVG(cost)
FROM monthly_cost
WHERE delivr_month < '2018-09-01';
```

## Results:

| meal_id | cost |
|---|---|
| 5 | 12248 |
| 4 | 10211.5 |
| 6 | 8219.75 |
| 13 | 6648.75 |

Showing 5 out of 5 rows

## Determined the profitability of each eatery to enhance their negotiation capabilities of the Business Development team.

```sql
-- CTE named monthly_cost
WITH monthly_cost AS (
  SELECT
    DATE_TRUNC('month', stocking_date)::DATE AS delivr_month,
    SUM(meal_cost * stocked_quantity) AS cost
  FROM meals
  JOIN stock ON meals.meal_id = stock.meal_id
  GROUP BY delivr_month)

SELECT
  -- Average monthly cost before September
  AVG(cost)
FROM monthly_cost
WHERE delivr_month < '2018-09-01';
```

## Results:

| avg |
| --- |
| 3727.5833333333335 |

## Tracked Delivr profits per month

```sql
-- Revenue CTE
WITH revenue AS (
    SELECT
        DATE_TRUNC('month', order_date) :: DATE AS delivr_month,
        SUM(meal_price * order_quantity) AS revenue
    FROM meals
    JOIN orders ON meals.meal_id = orders.meal_id
    GROUP BY delivr_month),
-- Cost CTE
  cost AS (
    SELECT
        DATE_TRUNC('month', stocking_date) :: DATE AS delivr_month,
        SUM(meal_cost * stocked_quantity) AS cost
    FROM meals
    JOIN stock ON meals.meal_id = stock.meal_id
    GROUP BY delivr_month)
-- Calculate profit by joining CTEs
SELECT
    revenue - cost AS profit,
    revenue.delivr_month
FROM revenue
JOIN cost ON revenue.delivr_month = cost.delivr_month
ORDER BY revenue.delivr_month ASC;
```

## Results:

| profit | delivr_month |
|---|---|
| 4073.5 | 2018-06-01 |
| 6575.6 | 2018-07-01 |
| 9974.25 | 2018-08-01 |
| 15339.5 | 2018-09-01 |
| 23087.5 | 2018-10-01 |
| 38743 | 2018-11-01 |
| 70300.5 | 2018-12-01 |

**The comparison between the monthly active users (MAUs) of the previous and current month serves as a signal to the Product team. If the number of active users in the current month is lower than that of the previous month, it raises a concern that prompts the Product team's attention.**

```sql
WITH mau AS (
  SELECT
    DATE_TRUNC('month', order_date) :: DATE AS delivr_month,
    COUNT(DISTINCT user_id) AS mau
  FROM orders
  GROUP BY delivr_month)

SELECT
  -- Select the month and the MAU
  delivr_month,
  mau,
  COALESCE(
    LAG(mau) OVER(ORDER BY delivr_month ASC),
  0) AS last_mau
FROM mau
-- Order by month in ascending order
ORDER BY delivr_month ASC;
```

## Results:

| delivr_month | mau | last_mau |
|---|---|---|
| 2018-06-01 | 123 | 0 |
| 2018-07-01 | 226 | 123 |
| 2018-08-01 | 337 | 226 |
| 2018-09-01 | 489 | 337 |
| 2018-10-01 | 689 | 489 |
| 2018-11-01 | 944 | 689 |
| 2018-12-01 | 1267 | 944 |

## The overall average revenue per user (ARPU)

```sql
SELECT
  -- Select the user ID and calculate revenue
  user_id,
  SUM(m.meal_price * o.order_quantity) AS revenue
FROM meals AS m
JOIN orders AS o ON m.meal_id = o.meal_id
GROUP BY user_id
ORDER BY revenue DESC;
```

## Results:

| user_id | revenue |
|---|---|
| 18 | 626 |
| 76 | 553.25 |
| 73 | 537 |
| 45 | 521.25 |
| 30 | 497.75 |
| 101 | 494.25 |
| 70 | 480.25 |

Showing 100 out of 1304 rows