The WITH RECURSIVE syntax

Using FrontAccounting, I needed a way to project recurring revenue for the next 12 months. I had a combination of a Perl script and a MySQL query to do this, but I wanted a single query that I could plug into my report tool.

Warnings

Just like it says, this is recursion, generally used to retrieve data from a hierarchical configuration.

It is very resource intensive if it has to recurse very deep. Be careful when implementing this in the real world, as a poorly written CTE can blow up your database server.

Original Task

After some looking around, the basic query turned out to be:

```
   SELECT
      id AS id,
      description AS description,
      salesorder.total AS total,
      debtor.name AS client,
      days AS days,
      monthly AS monthly,
      last_sent AS lastSent,
      BEGIN AS BEGIN,
      DATE_ADD(IF(last_sent='0000-00-00', BEGIN, last_sent), INTERVAL
invoices.monthly MONTH) AS nextDue,
      MONTH(DATE_ADD(IF(last_sent='0000-00-00', BEGIN, last_sent), INTERVAL
invoices.monthly MONTH)) AS nextMonthDue
   FROM
      0_recurrent_invoices invoices
      JOIN 0_sales_orders salesorder USING (order_no)
      JOIN 0_debtors_master debtor ON (invoices.debtor_no=debtor.debtor_no);
```

I then ran the output of this through a Perl script that calculated when invoices would come up and incremented a monthly amount by the invoice amount. Some invoices recur monthly, some quarterly, some semi-annually, and some annually (the integer value `monthly` in the query).

I was able to bypass the Perl script by learning the CTE (Common Table Expressions) syntax for MySQL (and MariaDB). This is similar to processing queries into temp tables, then running an additional query against them.

Solution

I'll say more about it later, but here is the resulting query.

fa_recurring.sql

```
/*
   This query calculates the total invoiced amount for each month based
on the 'recurrent' query.

   It generates a list of months starting from the earliest 'nextDue'
date and extends it by a
   specified number of months.

   For each month, it sums the 'total' from the 'recurrent' query where
the month matches the
   billing cycle defined by 'nextDue' and 'monthly'.

   This will run for 12 months starting from the earliest 'nextDue'
date, but you can adjust
   the number of months by changing the INTERVAL in the 'params' CTE.
*/
WITH RECURSIVE recurrent AS (
   SELECT
      id AS id,
      description AS description,
      salesorder.total AS total,
      debtor.name AS client,
      days AS days,
      monthly AS monthly,
      last_sent AS lastSent,
      BEGIN AS BEGIN,
      DATE_ADD(IF(last_sent='0000-00-00', BEGIN, last_sent), INTERVAL
invoices.monthly MONTH) AS nextDue,
      MONTH(DATE_ADD(IF(last_sent='0000-00-00', BEGIN, last_sent),
INTERVAL invoices.monthly MONTH)) AS nextMonthDue
   FROM
      0_recurrent_invoices invoices
      JOIN 0_sales_orders salesorder USING (order_no)
      JOIN 0_debtors_master debtor ON
(invoices.debtor_no=debtor.debtor_no)
),
params AS (
   SELECT
      DATE_FORMAT(MIN(nextDue), '%Y-%m-15') AS start_month,
      DATE_ADD(DATE_FORMAT(MIN(nextDue), '%Y-%m-15'), INTERVAL 12
MONTH) AS end_month
   FROM recurrent
),
months AS (
```

```
   SELECT start_month AS month_start, end_month
   FROM params
   UNION ALL
   SELECT DATE_ADD(month_start, INTERVAL 1 MONTH), end_month
   FROM months
   WHERE month_start < end_month
)
SELECT
   month_start,
   SUM(
      CASE
         WHEN month_start >= DATE_FORMAT(nextDue, '%Y-%m-15')
          AND MOD(
             PERIOD_DIFF(DATE_FORMAT(month_start, '%Y%m'),
DATE_FORMAT(nextDue, '%Y%m')),
             monthly
          ) = 0
         THEN total
         ELSE 0
      END
   ) AS invoiced_total
FROM months
JOIN recurrent
GROUP BY month_start
ORDER BY month_start;
```

- This projects 12 months into the future. Since our recurrent invoices have that as a maximum (annual), it works for us. If you want 6 or 24 months, change the constant 12 near the middle of the query: *INTERVAL 12 MONTH*.
- Our invoicing is done on the 15th of every month, so in three locations we hard-code the date format. Look for '*%Y-%m-15*'. Since we are working with monthly events, the day is arbitrary as long as the invoices are due on the 15th each time.

Explanation

Purpose

The query aggregates recurring invoice totals by month. Each invoice recurs every `monthly` months starting from its computed `nextDue` date.

Query Structure

The query uses three CTEs:

1. **`recurrent`**

- Builds a normalized recurring-invoice dataset from FrontAccounting tables.

- Computes:
  - `nextDue`: the next invoice date, based on `begin` or `last_sent` plus `monthly` months.
  - `nextMonthDue`: the month number of `nextDue` (kept for reference).

2. **`params`**

- Calculates the reporting window:
  - `start_month`: first month to report (earliest `nextDue`).
  - `end_month`: `start_month` plus a fixed interval (currently 12 months).

3. **`months`** (recursive)

- Generates a row per calendar month from `start_month` through `end_month`.

Final Aggregation

For each generated `month_start`, the query sums `total` for rows in `recurrent` that should be billed in that month:

* A row is billed when the difference in months between `month_start` and `nextDue` is a multiple of `monthly`. * This is evaluated with:

- `PERIOD_DIFF(DATE_FORMAT(month_start, '%Y%m'), DATE_FORMAT(nextDue, '%Y%m'))`
- `MOD(…, monthly) = 0`

Output

The result returns one row per month:

- `month_start`: 15th of the month (YYYY-MM-15)
- `invoiced_total`: total invoice amount expected for that month

Adjusting the Reporting Range

To change the number of months produced, edit the interval in the `params` CTE:

```
DATE_ADD(DATE_FORMAT(MIN(nextDue), '%Y-%m-15'), INTERVAL 12 MONTH)
```

For example, replace 12 with 24 to show two years.

Notes

- The query is compatible with MariaDB 10.11.
- If there are no `nextDue` values, the CTEs will return no rows.