Overview of Advanced Queries

Increasingly people are expecting answers to more complex questions based on the data in their databases. This includes tasks such as ranking, calculating moving averages and rolling up data. The queries at first might seem difficult but these are commonly needed business tasks.

Sometimes these problems were solved by using a programming approach or very complex SQL. The various dbms have been adding more features and functions to SQL to help solve these problems. If these techniques are built into the dbms, they can be optimized by the optimizer and using these functions is generally more efficient than other approaches

Many of these tasks involve the use of analytical functions which can be as simple as the Avg. Sum and Count functions we have been using. We can use these functions to calculate aggregate values over a group of rows.

MySQL has not directly implemented the major functions for these tasks and we will use techniques we have already covered to accomplish these.

One thing that we can do with these techniques is display both detail and aggregated data in the same query. You have done some of this already with subqueries. Suppose we want to display the following for each order in the first 6 months of 2013: cust_id, ord_id, ord_date, order_total; that is a simple group by with aggregate query. You should be able to do that with no problems. These are some rows from the result set.

```
+--------+-------+------------+----------+
| CustID | OrdID | OrdDate    | OrdTotal |
+--------+-------+------------+----------+
. . .
| 403050 |   527 | 2013-05-01 |   440.47 |
| 403050 |   507 | 2013-02-01 |   145.99 |
| 404000 |   302 | 2013-06-04 |   469.95 |
| 404100 |   504 | 2013-01-11 |   149.99 |
| 404100 |   503 | 2013-01-04 |   149.99 |
| 404100 |   303 | 2013-06-10 |   125.00 |
```

Now we want to add some higher level data; we want to know what percent of the customer total each order is; how much each order is above or below the average order total for that customer and the percent that order is of all of the customer orders in that time range. These are some rows from the result set.

Customer 403050 has two orders; the customer's total is 586.46.The first of this customers orders is about 75% of the customer's total and is about 2% of all of the orders. Customer 404000 has only one order so that order is 100% of the customer's total. Customer 404100 has three orders; the average of these three orders is 141.66; the first and second of this customer's orders are 8.33 above average for that customer and the third is 16.66 below the average for that customer. We can do this with correlated subqueries.

When you write these queries, pay more attention to using the proper groupings and the proper correlations. MySQL will let you write group by clauses which are logically wrong but which will still execute.

```
+--------+-------+------------+----------+-----------+------------+------------------+-----------|
| CustID | ordId | OrdDate    | OrdTotal | CustTotal | %CustTotal | OverUnderAvgCust | %AllTotal |
+--------+-------+------------+----------+-----------+------------+------------------+-----------|
. . .
| 403050 |   527 | 2013-05-01 |   440.47 |    586.46 |       75.1 |           147.24 |       2.3 |
| 403050 |   507 | 2013-02-01 |   145.99 |    586.46 |       24.9 |          -147.24 |       0.8 |
| 404000 |   302 | 2013-06-04 |   469.95 |    469.95 |      100.0 |             0.00 |       2.4 |
| 404100 |   504 | 2013-01-11 |   149.99 |    424.98 |       35.3 |             8.33 |       0.8 |
| 404100 |   503 | 2013-01-04 |   149.99 |    424.98 |       35.3 |             8.33 |       0.8 |
| 404100 |   303 | 2013-06-10 |   125.00 |    424.98 |       29.4 |           -16.66 |       0.6 |
```

These results of these functions are more meaningful with large amounts of data, but to keep the examples simple, we will use small tables. Although these queries might seem complex at first, they are important for analyzing large amounts of data. This week's material may seem overwhelming but I think you can handle working out the tasks for the assignment.

We can use the a_emp.employees table. Because I don't need all of the data, I will create a view on that table. create view adv_emp as

```
create or replace view a_emp.adv_emp as
(select
  emp_id
, name_last
, emp_mng as mng
, dept_id
, year(hire_date) as year_hired
, cast(coalesce(salary,0) as signed integer) as salary
from a_emp.employees);
```

The sales table which will have one row for each day's sales for a range of dates. Create this in the a_oe database.

```
Create table a_oe.adv_sales (
  sales_day     date primary key
, sales integer not null check (sales >= 0));
```

Inserts are in the demo for this document. Some sample inserts

```
Insert into a_oe.adv_sales values('2011-05-06', 000);
Insert into a_oe.adv_sales values('2011-05-07', 000);
Insert into a_oe.adv_sales values('2011-05-08', 200);
```

We are creating this as a separate table but you can see how you could use a query to get this type of set of data.

```
select  ord_date
, cast(coalesce(sum(quantity_ordered * quoted_price),0) as signed integer)  as
sales
from a_oe.order_headers OH
join a_oe.order_details OD on OH.ord_id = OD.ord_id
group by ord_date;
```

You could use a view or store the data in a temp table for doing the analytical analysis.

Topics discussed include:

14_02:

- using aggregates to build more interesting queries

14_03

- Using rollup

14_04

- building running totals and moving aggregates

14_05

- ranking queries

14_06

- number tables and calendars