Various MySQL Queries

——————————————————————————

DISTINCT

We can use the distinct tag if we only want particular values within a field to occur once- like if we want to know how many distinct titles were checked out during a month without seeing duplicates.

For example, this query will return 10347628 values, or in other words the item type of every single entry in inraw:

SELECT itemtype from inraw;

Whereas this query will only show us the 50 distinct item types that were found after scanning all these transactions:

SELECT DISTINCT itemtype from inraw;

——————————————————————————

GROUP

What if we want to know how many times the item was checked out along with its distinct title? Instead of using distinct, we need to use GROUP BY to group the transactions by their titles. We also add the count(*) field to show the count of each group (title):

SELECT title, count(*) from inraw where date(cout) = '2010-08-16' group by title order by count(*) DESC;

If we also want to see the itemtype of these titles, we can add the itemtype field like so:

SELECT title, itemtype, count(*) from inraw where date(cout) = '2010-08-16' group by title order by title DESC;

but there's a problem with this. The itemtype shown is only the itemtype of the last transaction added to this group. It doesn't necessarily mean that every item within that group has the same item type. In order to split these up correctly we need to add itemtype to the GROUP criteria:

SELECT title, itemtype, count(*) from inraw where date(cout) = '2010-08-16' group by title, itemtype order by title ASC;

We can make these groupings arbitrarily complicated, and since they're always broken down by each grouping the order we specify them doesn't matter (I think):

SELECT title, count(*) from inraw where date(cout) = '2010-08-16' group by title, itemtype order by title ASC;

———————————————————————————

HAVING

What if we only want to see titles that were checked out more than once? We need to use the term HAVING, which is like WHERE, except that it checks this value on the groups that we've created earlier in the query:

SELECT title, itemtype, count(*) from inraw where date(cout) = '2010-08-16' group by itemtype, title HAVING count(*) > 1;

HAVING is an example of a "group function" or a MySQL function that works on groups that we specify. The count(*) after HAVING, is the count of the members of the smaller groups, rather than the number of matches for the the whole query. Be careful not to use HAVING for things that should really be specified by WHERE or AND.

Bad:
SELECT title, itemtype, count(*) from inraw where date(cout) = '2010-08-16' group by itemtype, title HAVING title like "%girl%";

Good:
SELECT title, itemtype, count(*) from inraw where date(cout) = '2010-08-16' AND title like "%girl%" group by itemtype, title;

Although technically they both work with newer versions of MySQL.

———————————————————————————

SUBSTRING and SUBSTRING_INDEX

SUBSTRING is used to pull some characters out of a string, so for instance:

SELECT SUBSTRING('Quadratically',5);
    -> 'ratically'
SELECT SUBSTRING('foobarbar' FROM 4);
    -> 'barbar'
SELECT SUBSTRING('Quadratically',5,6);
    -> 'ratica'
SELECT SUBSTRING('Sakila', -3);
    -> 'ila'
SELECT SUBSTRING('Sakila', -5, 3);

-> 'aki'
SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
-> 'ki'

So if we want to only check dewey numbers that start with 398.2, we can do this:

select from inraw where SUBSTRING(deweyClass,1,5);* = '398.2';

but this is the same (and I think much easier to read):

select from inraw where deweyClass between 398.2 and 398.3;*

SELECT_INDEX allows us to select parts of a string, based on delimiters. These can be any character or group of characters, but a common delimiter is a space or comma:

select title from inraw where SUBSTRING_INDEX(title, ' ', -2) = 'and more';

gives us everything with a title that ends with "and more."

select title from inraw where SUBSTRING_INDEX(title, ' ', 1) like '%ing';

gives us every title that starts with a normal adjective: "cooking", "skiing", "microwaving".

——————————————————————————

AS

We use AS to give our columns logical titles- for instance if you're returning the sum of a number of books in a particular year, MySQL doesn't know how to label that, so it will just default to the function name:

*select floor(deweyClass/10)*10, SUM(CASE WHEN year(cout) = '2005' THEN 1 ELSE 0 END) from inraw where deweyClass < 1000 group by floor(deweyClass/10)*10 order by floor(deweyClass/10)*10;*

With AS, we can rename this column to something more useful:

*select floor(deweyClass/10)*10 as dewey, SUM(CASE WHEN year(cout) = '2005' THEN 1 ELSE 0 END) AS checkouts_in_2005 from inraw where deweyClass < 1000 group by floor(deweyClass/10)*10 order by dewey;*

This can help you keep track of ridiculously complicated queries and can help others to interpret them.   It also allows you to store variables within your script, so you don't constantly have to type out something like "floor(deweyClass/10)*10" to reference it.

——————————————————————————

AVG

Returns an average:

*select floor(deweyClass/10)*10 as dewey, count(*), AVG(TIMESTAMPDIFF(DAY,cout,cin)) as avgDaysOut from inraw where deweyClass != 'null' and month(cout) = '01' group by dewey order by avgDaysOut DESC;*

But wait, what's going on here? Why are some of the average transaction durations over 6000 days long? This seems impossible, so let's break the query into individual transactions and try to find any that are throwing off the average:

*select floor(deweyClass/10)*10 as dewey, TIMESTAMPDIFF(DAY,cout,cin) as daysOut, cout from inraw where deweyClass != 'null' and month(cout) = '01' and floor(deweyClass/10)*10 = '460' order by daysOut DESC;*

There's the problem. A bunch of entries that are marked as having been checked out at 1970-01-01 at 00:00:00. These must be incorrect, so how can we ignore them and get the average of everything else?

*select floor(deweyClass/10)*10 as dewey, count(*), AVG(TIMESTAMPDIFF(DAY,cout,cin)) as avgDaysOut from inraw where deweyClass != 'null' and month(cout) = '01' and date(cout) != '1970-01-01' group by dewey order by avgDaysOut DESC;*

These values make much more sense.

——————————————————————————

SOUNDEX

You give SOUNDEX a string and it tries to figure out what this string sounds like in order to match it to words that sound similar. For instance SOUNDEX('famine') will return things like "funny men", "five women", "funny ha ha", and "fuyu no ume":

select cin, title, deweyClass from inraw where Soundex(title) = Soundex ('famine');

This returns "Bill Russell":

select cin, title, deweyClass from inraw where Soundex(title) = Soundex ('blargle');

I'm not sure how useful this is for our purposes, but maybe it can be used to find some sort of subconscious "media buzz word" affect on behavior…

—————————————————————————

DATETIME functions

Alll the 19317 distinct titles checked out on September 16, 2010:

SELECT DISTINCT title from inraw where date(cout) = '2010-08-16' order by title;
We can use the DAYOFYEAR function on a date to find out what day of the year it was. (December 31 = 365th day of the year).

select title,cout,DAYOFYEAR(cout) as dayOfYear from inraw where deweyClass != 'null' and month(cout) = '02' and date(cout) != '1970-01-01' group by dayOfYear;

There's also DAYOFMONTH:

select title,cout,DAYOFMONTH(cout) as dayOfMonth from inraw where deweyClass != 'null' and month(cout) = '02' and date(cout) != '1970-01-01' group by dayOfMonth;

Returns time, title, callNumber, bibNumber, itemtype from 1pm on August 16, 2016 ordered by bibNumber

SELECT time(cout), title, callNumber, bibNumber, itemtype from outraw where date(cout) = '2016-08-16' and hour(cout) = '13:00' order by bibNumber;

How to find out what hour of the day is most popular for dewey checkouts:

select title,cout,HOUR(cout) as hourOfDay, count(*) from inraw where deweyClass != 'null' and month(cout) = '02' and date(cout) != '1970-01-01' group by hourOfDay;

MySQL datetime types are quite powerful and have many embedded functions to do basically whatever crazy time conversion you could need. For example:

WEEKOFYEAR() returns the week of the year.

UNIX_TIMESTAMP() returns the number of seconds since January 1st 1970.

CURDATE() returns the current date.

etc. For more examples, see this page: http://dev.mysql.com/doc/refman/5.1/en/date-and-time-functions.html

—————————————————————————

STDDEV and actual statistical analysis of the data

What if we want to know the standard deviation checkout times in minutes grouped by hours of the day? We can use the STDDEV function to compute this for us:

select title,cout,HOUR(cout) as hourOfDay, count(*), stddev(minute(cout)) from inraw where deweyClass != 'null' and month(cout) = '02' and date(cout) != '1970-01-01' group by hourOfDay;

Here's an advanced search with an average, standard variance, and standard deviation for the minute of checkout during each hour of the day:

select title,cout,HOUR(cout) as hourOfDay, count(*), AVG(minute(cout)), variance(minute(cout)), STDDEV(minute(cout)) from inraw where deweyClass != 'null' and month(cout) = '02' and date(cout) != '1970-01-01' group by hourOfDay;

Here's a site that explains what standard variance and deviation are using horrible cartoon dogs:

http://www.mathsisfun.com/data/standard-deviation.html

Here are more MySQL statistical tools:

http://dev.mysql.com/doc/refman/5.0/en/group-by-functions.html#function_stddev

————————————————————————————

Query within Query
from anastasiya: http://vislab.mat.ucsb.edu/2015/anastasiya/p1/index.html

Anastasiya is looking for co-occurences giving both overview and detail of what was checked out at the same time.

She used a query within a query. Also note the assigning of variables such as t1 and t2. The second select query is a correlated inner query where the inner query returns a list of item numbers and the outer query selects only the records that have items that are in the list.

```
SELECT
   t1.title,
   t2.title AS title2,
   COUNT(t2.title) AS freq,
   t1.deweyClass AS dewey1,
   t2.deweyClass AS dewey2
FROM
   inraw AS t1,
   inraw AS t2
WHERE
   t1.cout = t2.cout AND t1.cin = t2.cin
      AND t1.itemNumber IN (SELECT
          itemNumber
        FROM
          inraw
        WHERE
          title LIKE '%algorithms%')
        AND t2.title != ''
        AND t1.itemNumber != t2.itemNumber
GROUP BY title2
ORDER BY freq DESC
LIMIT 100;
```

————————————————————————————

From Kitty Currier: http://vislab.mat.ucsb.edu/2014/p1/Kitty/index.html

Kitty is looking for barcode anomalies.

```
SELECT
   a1.itemNumber,
   numBarcodes,
   COUNT(inraw.cout) AS numCout,
   itemtype
FROM
   (SELECT
      itemnumber, COUNT(DISTINCT barcode) AS numBarcodes
   FROM
      inraw
   GROUP BY itemNumber
   ORDER BY numBarcodes DESC
   LIMIT 26901) AS a1,
   inraw
WHERE
   (inraw.itemNumber = a1.itemNumber)
GROUP BY a1.itemNumber
ORDER BY numBarcodes DESC
```