CS 61:
Database Systems

Aggregation

Adapted from Silberschatz, Korth, and Sundarshan unless otherwise noted
Augmented by Claude Opus

Review

Data is held in relations (tables)

SELECT <attributes>
FROM <table>
WHERE <condition>
ORDER BY <attributes>

Review

```
SELECT CAMIS, dba, `inspection date`,
    `violation code`, Score, Grade
    FROM restaurant_inspections
    WHERE boro='Manhattan'
    ORDER BY Score desc, `inspection date`;
```

CAMIS	dba	inspection d...	violati...	Score	Grade
50177707	JOYBITE TANGHULU	10/27/2025	05H	203	N
50177707	JOYBITE TANGHULU	10/27/2025	05D	203	N
50177707	JOYBITE TANGHULU	10/27/2025	10F	203	N
50177707	JOYBITE TANGHULU	10/27/2025	04J	203	N
50177707	JOYBITE TANGHULU	10/27/2025	05A	203	N
50177707	JOYBITE TANGHULU	10/27/2025	05C	203	N
50177707	JOYBITE TANGHULU	10/27/2025	28-06	203	N
50177707	JOYBITE TANGHULU	10/27/2025	03E	203	N
50177707	JOYBITE TANGHULU	10/27/2025	05F	203	N
50177707	JOYBITE TANGHULU	10/27/2025	05E	203	N
50127670	DELI & GRILL	08/28/2025	08C	175	
50127670	DELI & GRILL	08/28/2025	06D	175	
50127670	DELI & GRILL	08/28/2025	04N	175	
50127670	DELI & GRILL	08/28/2025	05A	175	
50127670	DELI & GRILL	08/28/2025	04A	175	
50127670	DELI & GRILL	08/28/2025	02G	175	
50127670	DELI & GRILL	08/28/2025	05F	175	
50127670	DELI & GRILL	08/28/2025	04H	175	
50127670	DELI & GRILL	08/28/2025	10B	175	
50127670	DELI & GRILL	08/28/2025	06B	175	
50127670	DELI & GRILL	08/28/2025	02B	175	

CAMIS = RestaurantID
DBA = Doing Business As (RestaurantName)
`inspection date` = when inspected occured
`violation code` = violations found
Score = numerical rating
Grade = inspection grade

Agenda

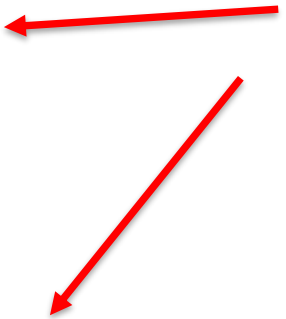
1. Aggregate functions and NULL
2. GROUP BY and HAVING
3. Nested queries
4. Conditional evaluation
5. Window functions

Aggregate function provide a scalar value for an attribute

Aggregate functions

Use in the **SELECT** clause

- **AVG:** average value
- **MIN:** minimum value
- **MAX:** maximum value
- **SUM:** sum of values
- **COUNT:** number of values



AVG and SUM must be numeric attributes, others need not be numeric

NULL not considered in aggregate functions

SELECT COUNT(*)

FROM table

WHERE p

NULL means the value is missing or unknown; can cause unexpected problems

Theoretically, these two queries should be the same!

```
SELECT AVG(score) AS AvgScore
   FROM restaurant_inspections; -- 25.59
```

```
SELECT SUM(score)/COUNT(*) AS AvgScore
   FROM restaurant_inspections; -- 23.62
```

Works if COUNT(*) changed to COUNT(Score)

Practice:

First query returns 25.59, the second 23.62 Why are they different?

How can we make them the same?

Remember, NULL not considered in aggregate functions

NULL in an arithmetic operation is NULL (e.g., 5 + NULL = NULL)

IN is a concise way to do multiple ORs

```
-- Check to see if restaurant in 1 of 5 boros
SELECT *
    FROM restaurant_inspections
    WHERE boro = 'Manhattan'
        OR boro = 'Bronx'
        OR boro = 'Brooklyn'
        OR boro = 'Queens'
        OR boro = 'Staten Island';
```

```
-- Equivalent boro check using IN
SELECT *
    FROM restaurant_inspections
    WHERE boro IN ('Manhattan', 'Bronx', 'Brooklyn', 'Queens', 'Staten Island');
```

Practice 1

use nyc_data;

- Where any restaurants ***not*** in one of the five boros?
- Find the ***min*** and ***max*** restaurant name
- What is the ***average*** score of all inspections scores?
- How many total restaurants inspection scores were recorded (consider where score != null)?
- Try to answer the last two questions with one **SELECT** query

Agenda

1. Aggregate functions and NULL
2. GROUP BY and HAVING
3. Nested queries
4. Conditional evaluation
5. Window functions

GROUP BY creates subgroups of tuples, you can perform aggregation over subgroups

```
SELECT ID, name, dept_name,
       FORMAT(salary,0) AS Salary
FROM instructor
ORDER BY dept_name;
```

Format Salary nicely

ID	name	dept_name	Salary
76766	Crick	Biology	72,000
10101	Srinivasan	Comp. Sci.	65,000
45565	Katz	Comp. Sci.	75,000
83821	Brandt	Comp. Sci.	92,000
98345	Kim	Elec. Eng.	80,000
12121	Wu	Finance	90,000
76543	Singh	Finance	80,000
32343	El Said	History	60,000
58583	Califieri	History	62,000
15151	Mozart	Music	40,000
22222	Einstein	Physics	95,000
33456	Gold	Physics	87,000

GROUP BY almost always used with aggregate function (sum, avg, count…)

Get avg salary by dept

```
SELECT dept_name,
       FORMAT(AVG(salary),0) AS AvgSalary
FROM instructor
GROUP BY dept_name
ORDER BY AvgSalary DESC;
```

dept_name	AvgSalary
Physics	91,000
Finance	85,000
Elec. Eng.	80,000
Comp. Sci.	77,333
Biology	72,000
History	61,000
Music	40,000

Selected attributes (e.g. dept_name and AvgSalary) should be in aggregate functions or in GROUP BY list

- Without grouping, AVG would return a single number for all departments
- Grouping allows aggregation of tuples with the same value for the GROUP BY attributes (e.g. dept_name)

HAVING works with GROUP BY to filter subgroups

```
SELECT dept_name,
       FORMAT(AVG(salary),0) AS AvgSalary
FROM instructor
GROUP BY dept_name
ORDER BY AvgSalary DESC;
```

dept_name	AvgSalary
Physics	91,000
Finance	85,000
Elec. Eng.	80,000
Comp. Sci.	77,333
Biology	72,000
History	61,000
Music	40,000

```
SELECT dept_name,
       FORMAT(AVG(salary),0) AS AvgSalary
FROM instructor
GROUP BY dept_name
HAVING AVG(salary) > 65000
ORDER BY AvgSalary DESC;
```

dept_name	AvgSalary
Physics	91,000
Finance	85,000
Elec. Eng.	80,000
Comp. Sci.	77,333
Biology	72,000

History and Music not included now

- **HAVING works with GROUP BY to filter results like how WHERE works with SELECT**
- **Note: predicates in the HAVING clause are applied *after* the formation of groups whereas predicates in the WHERE clause are applied before forming groups**

SQL evaluation proceeds start with FROM and proceeds to LIMIT

FROM
Create a new relation based on tables listed

GROUP BY
Aggregate tuples into subgroups

SELECT
Get attributes listed

LIMIT
Return top k items

WHERE
Remove tuples not matching criteria

HAVING
Remove non-matching subgroups

ORDER BY
Sort resulting tuples

https://www.mysqltutorial.org/mysql-group-by.aspx

Some advice about crafting SELECT commands

- Know your data
  - The importance of understanding the data model that you are working in cannot be overstated
  - Real-world databases are messy; many systems remain in service in an organization for decades

- Know the problem
  - Understand the question you are attempting to answer
  - Information reporting requests will come from a range of sources; may be one-time events or ongoing operations within an application

Some advice about crafting SELECT commands

Build query in this order

1. FROM
2. WHERE
3. SELECT
4. GROUP BY
5. HAVING
6. ORDER BY

Where does the data come from? Could be one or more tables, could be subquery Think of building one large relation

Filter out unwanted rows in relation from step 1

Which attributes do we want?

Create subgroups

Filter out unneeded subgroups

Finally, sort the results

Some advice about crafting SELECT commands

Build query in this order	Write SQL command in this order	
1. FROM	SELECT	column list
2. WHERE	FROM	table list
3. SELECT	[WHERE	condition list]
4. GROUP BY	[GROUP BY	column list]
5. HAVING	[HAVING	condition list]
6. ORDER BY	[ORDER BY	column list [ASC | DESC]];

Practice 2

use nyc_data

- Which is better? a low score or a high score? (Hint: consider the `critical flag`)
- Battle of boros! In one query, find the average health inspection score and number of inspections by boro. Which boro had the best avg scores? (Hint: sort by avg score)
- Do the same query as above, but calculate avg and count by boro and cuisine type. Which boro/cuisine were the best?
- For restaurants in Queens, find the average score and number of inspection scores for restaurants that have been inspected at least 5 times; sort by avg score, best first

Agenda

1. Aggregate functions and NULL
2. GROUP BY and HAVING
3. Nested queries
4. Conditional evaluation
5. Window functions

Nested queries have a subquery inside another query

Nested queries

Nesting can be done in the SELECT, FROM or WHERE clauses

SELECT $A_1, A_2, ..., A_n$
FROM $r_1, r_2, ..., r_m$
WHERE P

- **SELECT clause:**

  A_i can be replaced be a subquery that generates a single (scalar) value

- **FROM clause:** r_i can be replaced by any valid subquery because SELECT returns a relation

  More on this soon

- **WHERE clause:** P can be replaced with an expression of the form:

  A <operation> (subquery)

  A is an attribute and <operation> is <,>,IN, NOT IN, etc

Subqueries in the SELECT clause return a scalar value

Subquery in SELECT clause

SELECT
FROM
WHERE

- You can use a subquery in the SELECT clause in SQL
- Generally returns a scalar value (could be NULL)

```
-- compare restaurant score with this restaurant's max score
SELECT dba AS RestaurantName, Score,
    (SELECT MAX(Score)
        FROM restaurant_inspections r2
        WHERE r2.camis = r1.camis) AS MaxScore
FROM restaurant_inspections r1
WHERE dba LIKE 'Morris Park Bake%';
```

Select RestaurantID and Score for each row in table

- **Find max score for this restaurant; has same value for each returned row**
- **This is sometimes called a correlated subquery because camis from inner query using r2 is correlated with camis from outer query using r1**

Limit search to shorten query runtime
More on this when we get to query optimization

Subqueries in the FROM clause return a relation

Subquery in FROM clause

SELECT
FROM
WHERE

- You can use a subquery in the FROM clause in SQL

```
-- Create Manhattan table in the FROM clause
SELECT dba AS RestaurantName, Score
    FROM (SELECT *
            FROM restaurant_inspections
            WHERE boro = 'Manhattan') AS ManhattanRestaurants
  WHERE Score IS NOT NULL;
```

Create relation in the FROM clause
SELECT returns a relation

Subqueries can also be used in the WHERE clause

Subquery in WHERE clause

SELECT
FROM
WHERE

```
-- find scores for restaurant with min camis id
    SELECT camis AS RestaurantID, dba AS RestaurantName, Score
    FROM restaurant_inspections
    WHERE camis = (SELECT MIN(CAMIS)
                   FROM restaurant_inspections);

-- find inspections with max score from any inspection
    SELECT * FROM restaurant_inspections
    WHERE score = (SELECT MAX(Score)
                   FROM restaurant_inspections);
```

The WITH clause is also a subquery, but creates a queryable temporary relation

Subquery in WHERE clause

The **WITH** clause provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs

```
WITH TempRelationName (ColumnName1, columnName2...) AS
     (SELECT ...)
SELECT ...
```

Commonly called Common Table Extensions (CTEs)

```
-- First create table with max inspection score, query that table to find restaurants that have the max score
WITH MaxScoreTable AS
    (SELECT MAX(Score) AS MaxScore FROM restaurant_inspections)
SELECT * FROM restaurant_inspections
    WHERE score = (SELECT MaxScore FROM MaxScoreTable);
```

Practice 3

use nyc_data

1. For all each restaurant **not in Manhattan or Queens** return
   - RestaurantID, RestaurantName, Boro, and average score for that restaurant on one row
   - Sort the restaurants by average score descending
2. Use a WITH clause to calculate a temporary relation with a column for the average score of all inspections, then use that temporary table to return all rows with a score greater than average
3. Do the same as 2, but without using a WITH clause

Agenda

1. Aggregate functions and NULL
2. GROUP BY and HAVING
3. Nested queries
4. Conditional evaluation
5. Window functions

IF allows conditional evaluation, giving one of two values

IF command

If expr results in true or not null, return true value else return false value

Format:

SELECT IF (expr, true value, false value) **AS** name

Like a lambda expression in many programming languages

Practice: Some restaurant inspection grades are blank
If Grade ='', return 'N/A' else return Grade

Note: if attribute could come from more than relation, identify which one

```
USE nyc_data;
SELECT camis, dba, `inspection date`, Score,
      IF(Grade ='','N/A',Grade) AS Grade, `grade date`
FROM restaurant_inspections
WHERE camis = 30075445;
```

Blank grades now 'N/A'

Restau...	RestaurantName	inspection d...	Score	Grade	grade date
30075445	MORRIS PARK BAKE SHOP	01/31/2023	21	N/A	
30075445	MORRIS PARK BAKE SHOP	01/31/2023	21	N/A	
30075445	MORRIS PARK BAKE SHOP	02/03/2023	13	P	02/03/2023
30075445	MORRIS PARK BAKE SHOP	01/31/2023	21	N/A	
30075445	MORRIS PARK BAKE SHOP	01/31/2023	21	N/A	
30075445	MORRIS PARK BAKE SHOP	01/31/2023	21	N/A	
30075445	MORRIS PARK BAKE SHOP	08/22/2023	12	A	08/22/2023

COALESCE returns the first non-null value in a list of arguments

COALESCE

Format:

SELECT COALESCE(value$_1$, value$_2$, … value$_n$) **AS** name

value$_x$ can be an attribute or literal

If value$_1$ is NULL, SQL tries value$_2$, if value$_2$ NULL try value$_3$, …

Returns first non-null value

If all values are NULL, returns NULL

Given a table of customer orders

- **Try using State as Location**
- **If State is NULL, try Country**
- **If Country is NULL, use 'N/A'**

Example:

SELECT OrderID, **COALESCE**(State, Country, 'N/A') **AS** Location

FROM Orders

CASE provides if-then-else logic in one of two formats

CASE

Value of attribute

```
SELECT CASE attribute
    WHEN value1 THEN result1
    WHEN value2 THEN result2
    [ELSE else result]
END AS AttributeName
```

OR

No attribute here, it is in expression

```
SELECT CASE
    WHEN expr1 THEN result1
    WHEN expr2 THEN result2
    [ELSE else result]
END AS AttributeName
```

Don't forget END!

Practice: Categorize restaurants as having infrequent (<10), moderate (10-15), or frequent (>10) inspections

Can use BETWEEN x AND y; or
Could have used count(*) >10 AND count(*) > 15

```
SELECT camis, dba, count(*) AS `Total Inspections`,
CASE
      WHEN count(*) < 10 THEN 'Infrequent inspections '
      WHEN count(*) BETWEEN 10 AND 15 THEN 'Moderate inspections'
      ELSE 'Frequent inspections'
END AS Frequency
FROM restaurant_inspections
GROUP BY CAMIS
HAVING COUNT(*) > 1;
```

camis	dba	Total Inspectio...	Frequency
40876618	BIJAN'S	10	Moderate inspections
50010248	CAFE GRUMPY	19	Frequent inspections
41503108	TULCINGO RESTAURANT	28	Frequent inspections
41403890	TOWERS CAFE	16	Frequent inspections
41580023	SAGE	22	Frequent inspections
50093744	MORI MORI	3	Infrequent inspections

Practice

CASE practice

We can combine CASE with aggregate operations. For each restaurant provide the number of times an inspection resulted in each possible letter Grade

- First determine the grades possible (or at least those than have been given)
- Then on one line provide the number of times each restaurant received a distinct grade (e.g. CAMIS, RestaurantName, A, B, C…, NULL, Total)
- Output should look like this:

CAMIS	RestaurantName	A	B	C	G	N	P	Z	Total
50010248	CAFE GRUMPY	3	0	6	0	0	0	0	9
41403890	TOWERS CAFE	4	0	0	0	0	1	0	5
50093744	MORI MORI	3	0	0	0	0	0	0	3
50122709	CARIBBEAN KRAVE	7	0	0	0	0	0	0	7
50088988	CITY ONE	4	5	0	0	0	0	0	9
50095943	65 MARKET PLACE	5	6	0	0	0	0	0	11
41248866	BARAKA BUFFET	3	0	0	0	0	0	0	3
50034742	BLUJEEN	2	0	6	0	0	0	0	8
50098716	EATALY NY LLC (KIOSK)	4	0	0	0	0	1	0	5
41032752	DONS BOGAM	6	0	0	0	0	0	0	6
41086967	IL SOLE	6	0	0	0	0	0	0	6
41713504	ISIS RESTAURANT	5	6	0	0	0	0	0	11
41546006	JUNIOR'S TACOS	3	0	0	0	0	0	0	3
41163019	ABILENE	6	0	0	0	0	0	0	6

Practice

CASE practice

We can combine CASE with aggregate operations. For each restaurant provide the number of times an inspection resulted in each possible letter Grade

- First determine the grades possible (or at least those than have been given)
- Then on one line provide the number of times each restaurant received a distinct grade (e.g. CAMIS, RestaurantName, A, B, C…, NULL, Total)
- Output should look like this:

```
SELECT DISTINCT grade FROM inspections ORDER BY grade; -- A, B, C, G, N, P, Z, ‘’
SELECT CAMIS, dba AS RestaurantName,
   SUM(CASE Grade WHEN 'A' THEN 1 ELSE 0 END) AS 'A',
   SUM(CASE Grade WHEN 'B' THEN 1 ELSE 0 END) AS 'B',
   SUM(CASE Grade WHEN 'C' THEN 1 ELSE 0 END) AS 'C',
   SUM(CASE Grade WHEN 'G' THEN 1 ELSE 0 END) AS 'G',
   SUM(CASE Grade WHEN 'N' THEN 1 ELSE 0 END) AS 'N',
   SUM(CASE Grade WHEN 'P' THEN 1 ELSE 0 END) AS 'P',
   SUM(CASE Grade WHEN 'Z' THEN 1 ELSE 0 END) AS 'Z',
   SUM(CASE WHEN Grade = '' THEN 1 ELSE 0 END) AS 'Empty',
   COUNT(*) AS Total
FROM restaurant_inspections
GROUP BY CAMIS, dba;
```

Practice

CASE practice

We can combine CASE with aggregate operations. For each restaurant provide the number of times an inspection resulted in each possible letter Grade

- First determine the grades possible (or at least those than have been given)
- Then on one line provide the number of times each restaurant received a distinct grade (e.g. CAMIS, RestaurantName, A, B, C…, NULL, Total)
- Output should look like this:

```
SELECT DISTINCT grade FROM inspections ORDER BY grade; -- A, B, C, G, N, P, Z, ‘’
SELECT CAMIS, dba AS RestaurantName,
        COUNT(IF (Grade = 'A', Grade, NULL)) AS 'A',
        COUNT(IF (Grade = 'B', Grade, NULL)) AS 'B',
        COUNT(IF (Grade = 'C', Grade, NULL)) AS 'C',
        COUNT(IF (Grade = 'G', Grade, NULL)) AS 'G',
        COUNT(IF (Grade = 'N', Grade, NULL)) AS 'N',
        COUNT(IF (Grade = 'P', Grade, NULL)) AS 'P',
        COUNT(IF (Grade = 'Z', Grade, NULL)) AS 'Z',
        COUNT(IF (Grade = '' , 1, NULL)) AS 'NULL',
        COUNT(*) as Total
FROM restaurant_inspections
GROUP BY CAMIS, dba;
```

Another solution using COUNT IF

Notice that COUNT does not count the row if it is NULL

Agenda

1. Aggregate functions and NULL
2. GROUP BY and HAVING
3. Nested queries
4. Conditional evaluation
5. Window functions

We have previously seen how to use GROUP BY to aggregate data

Given sales table

productLine	orderYear	orderValue
Vintage Cars	2003	619161.48
Classic Cars	2003	1374832.22
Trucks and Buses	2003	376657.12
Trains	2003	65822.05
Ships	2003	222182.08
Planes	2003	309784.20
Motorcycles	2003	348909.24
Classic Cars	2004	1763136.73
Vintage Cars	2004	854551.85
Trains	2004	96285.53
Ships	2004	337326.10
Planes	2004	471971.46
Motorcycles	2004	527243.84
Trucks and Buses	2004	465390.00
Motorcycles	2005	245273.04
Classic Cars	2005	715953.54
Vintage Cars	2005	323846.30
Trucks and Buses	2005	182066.45
Trains	2005	26425.34
Ships	2005	104490.16
Planes	2005	172881.88

Can use group by to get sales per product line

```
SELECT productline,
  FORMAT(SUM(orderValue),2) AS totalOrders
    FROM  sales
    GROUP BY productline;
```

productLine	totalOrders
Vintage Cars	1,797,559.63
Classic Cars	3,853,922.49
Trucks and Buses	1,024,113.57
Trains	188,532.92
Ships	663,998.34
Planes	954,637.54
Motorcycles	1,121,426.12

No total line of all sales, just sales by product line

Adapted from: https://www.mysqltutorial.org/mysql-rollup/

You can add a summary row by using UNION

Given sales table

productLine	orderYear	orderValue
Vintage Cars	2003	619161.48
Classic Cars	2003	1374832.22
Trucks and Buses	2003	376657.12
Trains	2003	65822.05
Ships	2003	222182.08
Planes	2003	309784.20
Motorcycles	2003	348909.24
Classic Cars	2004	1763136.73
Vintage Cars	2004	854551.85
Trains	2004	96285.53
Ships	2004	337326.10
Planes	2004	471971.46
Motorcycles	2004	527243.84
Trucks and Buses	2004	465390.00
Motorcycles	2005	245273.04
Classic Cars	2005	715953.54
Vintage Cars	2005	323846.30
Trucks and Buses	2005	182066.45
Trains	2005	26425.34
Ships	2005	104490.16
Planes	2005	172881.88

UNION adds new *rows* to result (JOINs adds new columns)

```
SELECT productline,
   FORMAT(SUM(orderValue),2) AS totalOrders
      FROM sales
      GROUP BY productline;
UNION ALL
SELECT
   NULL,
   FORMAT(SUM(orderValue),2) AS totalOrders
FROM sales;
```

UNION ALL returns duplicate rows (UNION DISTINCT does not)

Must have same number of columns with compatible data types

SQL has an easier way to add the summary row using ROLLUP

productLine	totalOrders
Vintage Cars	1,797,559.63
Classic Cars	3,853,922.49
Trucks and Buses	1,024,113.57
Trains	188,532.92
Ships	663,998.34
Planes	954,637.54
Motorcycles	1,121,426.12
NULL	9604190.61

- **UNION returns new row with total**
- **Note: NULL for productLine in second SELECT**

Adapted from: https://www.mysqltutorial.org/mysql-rollup/

ROLLUP can be used similarly to create subtotals based on grouping

Given sales table

productLine	orderYear	orderValue
Vintage Cars	2003	619161.48
Classic Cars	2003	1374832.22
Trucks and Buses	2003	376657.12
Trains	2003	65822.05
Ships	2003	222182.08
Planes	2003	309784.20
Motorcycles	2003	348909.24
Classic Cars	2004	1763136.73
Vintage Cars	2004	854551.85
Trains	2004	96285.53
Ships	2004	337326.10
Planes	2004	471971.46
Motorcycles	2004	527243.84
Trucks and Buses	2004	465390.00
Motorcycles	2005	245273.04
Classic Cars	2005	715953.54
Vintage Cars	2005	323846.30
Trucks and Buses	2005	182066.45
Trains	2005	26425.34
Ships	2005	104490.16
Planes	2005	172881.88

ROLLUP creates total

```
SELECT productline,
  FORMAT(SUM(orderValue),2) AS totalOrders
    FROM  sales
    GROUP BY productline  WITH ROLLUP;
```

WITH ROLLUP adds extra row with totals for grouped by attributes like UNION did

productLine	totalOrders
Classic Cars	3,853,922.49
Motorcycles	1,121,426.12
Planes	954,637.54
Ships	663,998.34
Trains	188,532.92
Trucks and Buses	1,024,113.57
Vintage Cars	1,797,559.63
NULL	9,604,190.61

Now have total for all sales in a row called a super-aggregate

Adapted from: https://www.mysqltutorial.org/mysql-rollup/

ROLLUP can operate over multiple columns

Given sales table

productLine	orderYear	orderValue
Vintage Cars	2003	619161.48
Classic Cars	2003	1374832.22
Trucks and Buses	2003	376657.12
Trains	2003	65822.05
Ships	2003	222182.08
Planes	2003	309784.20
Motorcycles	2003	348909.24
Classic Cars	2004	1763136.73
Vintage Cars	2004	854551.85
Trains	2004	96285.53
Ships	2004	337326.10
Planes	2004	471971.46
Motorcycles	2004	527243.84
Trucks and Buses	2004	465390.00
Motorcycles	2005	245273.04
Classic Cars	2005	715953.54
Vintage Cars	2005	323846.30
Trucks and Buses	2005	182066.45
Trains	2005	26425.34
Ships	2005	104490.16
Planes	2005	172881.88

Can roll up multiple attributes

```
SELECT
  productLine,
  orderYear,
  FORMAT(SUM(orderValue),2) AS totalOrders
FROM  sales
GROUP BY productline, orderYear WITH ROLLUP;
```

productLine	orderYear	totalOrders
Trains	2003	65,822.05
Trains	2004	96,285.53
Trains	2005	26,425.34
Trains	NULL	188,532.92
Trucks and…	2003	376,657.12
Trucks and…	2004	465,390.00
Trucks and…	2005	182,066.45
Trucks and…	NULL	1,024,113.57
Vintage Cars	2003	619,161.48
Vintage Cars	2004	854,551.85
Vintage Cars	2005	323,846.30
Vintage Cars	NULL	1,797,559.63
NULL	NULL	9,604,190.61

Now have super-aggregate row by product line

Hierarchy determined by GROUP BY order (product line first)

Also have grand total over all super-aggregate rows

Adapted from: https://www.mysqltutorial.org/mysql-rollup/

ROLLUP can operate over multiple columns

Given sales table

productLine	orderYear	orderValue
Vintage Cars	2003	619161.48
Classic Cars	2003	1374832.22
Trucks and Buses	2003	376657.12
Trains	2003	65822.05
Ships	2003	222182.08
Planes	2003	309784.20
Motorcycles	2003	348909.24
Classic Cars	2004	1763136.73
Vintage Cars	2004	854551.85
Trains	2004	96285.53
Ships	2004	337326.10
Planes	2004	471971.46
Motorcycles	2004	527243.84
Trucks and Buses	2004	465390.00
Motorcycles	2005	245273.04
Classic Cars	2005	715953.54
Vintage Cars	2005	323846.30
Trucks and Buses	2005	182066.45
Trains	2005	26425.34
Ships	2005	104490.16
Planes	2005	172881.88

Can roll up multiple attributes

```
SELECT
  orderYear,
   productLine,
  FORMAT(SUM(orderValue),2) AS totalOrders
FROM  sales
GROUP BY orderYear, productLine WITH ROLLUP;
```

GROUP BY reversed

orderYear	productLine	totalOrders
2003	Classic Cars	1,374,832.22
2003	Motorcycles	348,909.24
2003	Planes	309,784.20
2003	Ships	222,182.08
2003	Trains	65,822.05
2003	Trucks and Buses	376,657.12
2003	Vintage Cars	619,161.48
2003	NULL	3,317,348.39
2004	Classic Cars	1,763,136.73
2004	Motorcycles	527,243.84
2004	Planes	471,971.46
2004	Ships	337,326.10
2004	Trains	96,285.53
2004	Trucks and Buses	465,390.00
2004	Vintage Cars	854,551.85
2004	NULL	4,515,905.51
2005	Classic Cars	715,953.54
2005	Motorcycles	245,273.04
2005	Planes	172,881.88
2005	Ships	104,490.16
2005	Trains	26,425.34
2005	Trucks and Buses	182,066.45
2005	Vintage Cars	323,846.30
2005	NULL	1,770,936.71
NULL	NULL	9,604,190.61

Grouping order reversed (orderYear first)

Adapted from: https://www.mysqltutorial.org/mysql-rollup/

ROLLUP with IFNULL can give super-aggregate rows a meaningful label

Given sales table

productLine	orderYear	orderValue
Vintage Cars	2003	619161.48
Classic Cars	2003	1374832.22
Trucks and Buses	2003	376657.12
Trains	2003	65822.05
Ships	2003	222182.08
Planes	2003	309784.20
Motorcycles	2003	348909.24
Classic Cars	2004	1763136.73
Vintage Cars	2004	854551.85
Trains	2004	96285.53
Ships	2004	337326.10
Planes	2004	471971.46
Motorcycles	2004	527243.84
Trucks and Buses	2004	465390.00
Motorcycles	2005	245273.04
Classic Cars	2005	715953.54
Vintage Cars	2005	323846.30
Trucks and Buses	2005	182066.45
Trains	2005	26425.34
Ships	2005	104490.16
Planes	2005	172881.88

Can roll up multiple attributes

```
SELECT IFNULL(orderYear,'GRAND TOTAL') AS orderYear,
       IFNULL(productLine,'Line Total') AS productLine,
       FORMAT(SUM(orderValue),2) AS totalOrders
       FROM sales
       GROUP BY orderYear, productline WITH ROLLUP;
```

orderYear	productLine	totalOrders
2003	Classic Cars	1,374,832.22
2003	Motorcycles	348,909.24
2003	Planes	309,784.20
2003	Ships	222,182.08
2003	Trains	65,822.05
2003	Trucks and Buses	376,657.12
2003	Vintage Cars	619,161.48
2003	Line Total	3,317,348.39
2004	Classic Cars	1,763,136.73
2004	Motorcycles	527,243.84
2004	Planes	471,971.46
2004	Ships	337,326.10
2004	Trains	96,285.53
2004	Trucks and Buses	465,390.00
2004	Vintage Cars	854,551.85
2004	Line Total	4,515,905.51
2005	Classic Cars	715,953.54
2005	Motorcycles	245,273.04
2005	Planes	172,881.88
2005	Ships	104,490.16
2005	Trains	26,425.34
2005	Trucks and Buses	182,066.45
2005	Vintage Cars	323,846.30
2005	Line Total	1,770,936.71
GRAND...	Line Total	9,604,190.61

IFNULL chooses first item if not NULL, second item if NULL

Super aggregate rows will be NULL

NULL replaced with Line Total or GRAND TOTAL

Adapted from: https://www.mysqltutorial.org/mysql-rollup/

GROUPING can give super-aggregate rows a meaningful label

Given sales table

productLine	orderYear	orderValue
Vintage Cars	2003	619161.48
Classic Cars	2003	1374832.22
Trucks and Buses	2003	376657.12
Trains	2003	65822.05
Ships	2003	222182.08
Planes	2003	309784.20
Motorcycles	2003	348909.24
Classic Cars	2004	1763136.73
Vintage Cars	2004	854551.85
Trains	2004	96285.53
Ships	2004	337326.10
Planes	2004	471971.46
Motorcycles	2004	527243.84
Trucks and Buses	2004	465390.00
Motorcycles	2005	245273.04
Classic Cars	2005	715953.54
Vintage Cars	2005	323846.30
Trucks and Buses	2005	182066.45
Trains	2005	26425.34
Ships	2005	104490.16
Planes	2005	172881.88

GROUPING returns 1 if super-aggregate row, 0 otherwise

```
SELECT IF(GROUPING(orderYear),'All Years',orderYear)AS orderYear,
IF(GROUPING(productLine),'All Products',productLine)AS productLine
SUM(orderValue)AS totalOrderValue
FROM sales
GROUP BY orderYear , productline WITH ROLLUP;
```

orderYear	productLine	totalOrderVal...
2003	Classic Cars	1374832.22
2003	Motorcycles	348909.24
2003	Planes	309784.20
2003	Ships	222182.08
2003	Trains	65822.05
2003	Trucks and Buses	376657.12
2003	Vintage Cars	619161.48
2003	All Products	3317348.39
2004	Classic Cars	1763136.73
2004	Motorcycles	527243.84
2004	Planes	471971.46
2004	Ships	337326.10
2004	Trains	96285.53
2004	Trucks and Buses	465390.00
2004	Vintage Cars	854551.85
2004	All Products	4515905.51
2005	Classic Cars	715953.54
2005	Motorcycles	245273.04
2005	Planes	172881.88
2005	Ships	104490.16
2005	Trains	26425.34
2005	Trucks and Buses	182066.45
2005	Vintage Cars	323846.30
2005	All Products	1770936.71
All Years	All Products	9604190.61

Remember how IF works: first value if true, otherwise second value

Super-aggregate rows now have reasonable names (not just NULL)

Adapted from: https://www.mysqltutorial.org/mysql-rollup/

Example WITH ROLLUP and GROUPING

```
use nyc_inspections;

-- Find average score by zipcode for each boro Manhattan or Bronx,
report avg for each boro and total average
SELECT IF(GROUPING(boro),'TOTAL',boro) AS Boro,
    IF(GROUPING(zipcode),'Boro Avg',zipcode) AS ZipCode,
    AVG(Score) AS AvgScore
    FROM restaurant_inspections
    WHERE boro IN ('Manhattan', 'Bronx')
        AND zipcode <> ''
        AND score IS NOT NULL
  GROUP BY boro, zipcode WITH ROLLUP;
```

Boro	ZipCode	AvgScore
Manhattan	10178	17.3077
Manhattan	10179	30.7222
Manhattan	10271	10.0455
Manhattan	10279	11.8000
Manhattan	10280	22.8657
Manhattan	10281	17.0075
Manhattan	10282	21.1139
Manhattan	Boro Avg	24.8849
TOTAL	Boro Avg	24.7606

Each boro has its own average
TOTAL is the average for all boro averages

RANK assigns an increasing number to each row returned

RANK

Sometimes you want to assign a numerical value to rows to indicate their rank (e.g., first row has rank 1, second row has rank 2, …)

- RANK() assigns a rank to each row within the partition of a result set
- The rank of a row is specified by one plus the number of ranks that come before it

PARTITION BY works like GROUP BY – splits results into groups based on attribute listed

Can have more than one partition (partition by cuisine type, then boro for example)

Format:

SELECT RANK() **OVER** (
  PARTITION BY A_1 [,A_2,…A_n]
  ORDER BY <expression> [ASC|DESC], [{,<expression>...}]) **AS** RankName

Sort each partition by ORDER BY

- **Rank numbering starts at 1 for each partition**
- **If tie on partition, all tying rows get same rank (e.g., if three row tie for first, all three get rank of 1, next row gets rank of 4)**
- **Use ROW_NUMBER() instead of RANK() to ensure no gaps between rank values assigned (e.g., first three ties get rank 1 though three, next row still gets rank of 4)**

Adapted from: https://www.mysqltutorial.org/mysql-window-functions/mysql-rank-function/

RANK assigns an increasing number to each row returned

RANK

PARTITION BY (works like GROUP BY) is optional

Example:

```
-- find rank of each boro based on average score of American cuisine
SELECT RANK() OVER (ORDER BY AVG(score)) AS `Rank`, boro,  AVG(score) AS Avg
    FROM restaurant_inspections
    WHERE `cuisine description` LIKE 'American%’
    GROUP BY boro;
```

Rank	boro	Avg
1	Bronx	21.4972
2	Staten Island	22.2342
3	Manhattan	22.4976
4	Queens	22.9800
5	Brooklyn	23.9712

Use WITH and LIMIT to get top *k* results

RANK

Will limit to top 2 restaurants

Example:

```
SET @k = 2; -- return top k=2
WITH RestaurantRanks AS (
    SELECT RANK() OVER (ORDER BY AVG(Score)) AS `Rank`, boro, AVG(score) AS Avg
    FROM restaurant_inspections
    WHERE `cuisine description` LIKE 'American%'
    GROUP BY boro)
SELECT * FROM RestaurantRanks WHERE `Rank`<= @k; -- top 2
```

Note: PARTITION BY is optional, if omitted, use all rows (here all boros)

WITH created temporary table, SELECT from that on RANK to get top k results

Rank	boro	Avg
1	Bronx	21.4972
2	Staten Island	22.2342

If rows tie, they get the same rank

So, this query could return more than 2 rows if multiple boros tie