SQL – Aggregations

Aggregation: Takes values in multiple rows of data and returns one value

```
SUM(FIELD_NAME)
AVG(FIELD_NAME)
MIN(FIELD_NAME)
MAX(FIELD_NAME)
COUNT(FIELD_NAME) or COUNT(*)
```

- `SUM` and `AVG` require that the field be numeric
- `MIN` and `MAX` will work with a variety of data types

NULL: Represents the absence of data

- Not the same as a zero (numeric value) a space (text value)
- All aggregate functions except `COUNT` ignore NULL values
- `COUNT(FIELD_NAME)` and `COUNT(*)` give different values if NULLs present

SQL – Whole Table Aggregations

TRANSACTIONS

Transaction_ID	Customer_Id	Channel	Product	Price	Discount
1000123	60067	Web	Book	9.95	
1000124	12345	Store	Book	11.95	
1000125	23451	Store	DVD	14.95	
1000126	70436	Reseller	DVD	19.95	5
1000127	66772	Store	Magazine	3.25	
1000128	60067	Web	Book	29.95	
1000129	72045	Web	DVD	9.95	
1000130	82371	Reseller	Magazine	2.5	0.25
1000131	12345	Store	Book	7.95	

```
SELECT COUNT(*)
FROM TRANSACTIONS
```

9

OR

COUNT(*)
9

```
SELECT COUNT(*) AS NUM_ROWS
FROM TRANSACTIONS
```

NUM_ROWS
9

- NUM_ROWS is the 'Alias' for COUNT(*) and is designated using 'AS'

SQL – The GROUP BY Clause

GROUP BY	Defines the level of aggregation I want if I'm summarizing data

```
SELECT GROUP_FIELD_1, ..., GROUP_FIELD_N,
       <AGGREGATE_1>, ..., <AGGREGATE_2>
FROM TABLE_NAME
GROUP BY GROUP_FIELD_1, ..., GROUP_FIELD_N
```

- `GROUP_FIELD(s)` must match in `SELECT` and `GROUP BY` clauses
- Everything else in the `SELECT` statement must be an aggregate function, e.g.:

```
SUM(FIELD_NAME)
AVG(FIELD_NAME)
MIN(FIELD_NAME)
MAX(FIELD_NAME)
COUNT(FIELD_NAME) or COUNT(*)
```

SQL – The GROUP BY Clause

Let's say I want to know the following by Product:

- Total number of purchases by product
- Total dollar value of purchases by product
- Average dollar value of purchases by product

Transaction_ID	Customer_Id	Channel	Product	Price	Discount
1000123	60067	Web	Book	9.95	
1000124	12345	Store	Book	11.95	
1000125	23451	Store	DVD	14.95	
1000126	70436	Reseller	DVD	19.95	5
1000127	66772	Store	Magazine	3.25	
1000128	60067	Web	Book	29.95	
1000129	72045	Web	DVD	9.95	
1000130	82371	Reseller	Magazine	2.5	0.25
1000131	12345	Store	Book	7.95	

Purchases = # Rows = `COUNT(*)`

`SUM(Price)`
`AVG(Price)`

Be Boulder.
University of Colorado Boulder

SQL – The GROUP BY Clause

```
SELECT PRODUCT, COUNT(*) AS PURCHASES,
       SUM(PRICE) AS TOTAL_SALES,
       AVG(PRICE) AS AVG_SALES
FROM TRANSACTIONS
GROUP BY PRODUCT
```

PRODUCT	PURCHASES	TOTAL_SALES	AVG_SALES
Book	4	59.8	14.95
DVD	3	44.85	14.95
Magazine	2	5.75	2.875

SQL – The HAVING Clause

HAVING	Adds filters that restrict what aggregated rows / records are retrieved

```
SELECT GROUP_FIELD_1, ..., GROUP_FIELD_N,
       <AGGREGATE_1>, ..., <AGGREGATE_2>
FROM TABLE_NAME
GROUP BY GROUP_FIELD_1, ..., GROUP_FIELD_N
HAVING <logical conditions(s)>
```

In this case conditions must be applied to one or more of the aggregate functions defined in the `SELECT` statement.

SQL – The HAVING Clause

Let's start with the results from the GROUP BY example:

```
SELECT PRODUCT, COUNT(*) AS PURCHASES,
       SUM(PRICE) AS TOTAL_SALES,
       AVG(PRICE) AS AVG_SALES
FROM TRANSACTIONS
GROUP BY PRODUCT
```

PRODUCT	PURCHASES	TOTAL_SALES	AVG_SALES
Book	4	59.8	14.95
DVD	3	44.85	14.95
Magazine	2	5.75	2.875

Now suppose that I only wanted to see products with average sales over $10

SQL – The HAVING Clause

```
SELECT PRODUCT, COUNT(*) AS PURCHASES,
       SUM(PRICE) AS TOTAL_SALES,
       AVG(PRICE) AD AVG_SALES
FROM TRANSACTIONS
GROUP BY PRODUCT
HAVING AVG_SALES > 10
```

PRODUCT	PURCHASES	TOTAL_SALES	AVG_SALES
Book	4	59.8	14.95
DVD	3	44.85	14.95

SQL – The HAVING Clause

```
SELECT PRODUCT, COUNT(*) AS PURCHASES,
       SUM(PRICE) AS TOTAL_SALES,
       AVG(PRICE) AD AVG_SALES
FROM TRANSACTIONS
WHERE CHANNEL <> 'RESELLER'
GROUP BY PRODUCT
HAVING AVG_SALES > 10
```

PRODUCT	PURCHASES	TOTAL_SALES	AVG_SALES
Book	4	59.8	14.95
DVD	3	44.85	12.45

SQL – The ORDER BY Clause

ORDER BY	Defines the sort order of the results

```
SELECT FIELD_1, FIELD_2, ..., FIELD_N
FROM TABLE_NAME
ORDER BY FIELD_i, ...., FIELD_n
```

- By default, sorts in ascending order, but descending can be specified as follows:

```
ORDER BY FIELD_i DESC
```

- Sorting field(s) can be anything in the source table, whether or not it's in the `SELECT` statement

SQL – The ORDER BY Clause

```
SELECT *
FROM TRANSACTIONS
ORDER BY PRICE DESC
```

Transaction_ID	Customer_Id	Channel	Product	Price	Discount
1000128	60067	Web	Book	29.95	
1000126	70436	Reseller	DVD	19.95	5
1000125	23451	Store	DVD	14.95	
1000124	12345	Store	Book	11.95	
1000123	60067	Web	Book	9.95	
1000129	72045	Web	DVD	9.95	
1000131	12345	Store	Book	7.95	
1000127	66772	Store	Magazine	3.25	
1000130	82371	Reseller	Magazine	2.5	0.25