Data Quality Committee Monthly Meeting

Friday, November 8, 2019

12 – 1 pm ET

Chair(s): MisChele Vickers, Eunice Santos

Ambassador: Krystal Collier

CSTE contact: Hayleigh McCall

Agenda

DQC
Data Quality Committee

- Next meeting: Friday, December 13, 2019 12 – 1 pm ET
- **CSTE Updates, other updates**
- 2019 Year End Review and Future Endeavors
- Preparing data with SQL in R Studio
- Open Mic

CSTE Announcements

DQC
Data Quality Committee

- Call agenda, recordings, and brief notes now on Basecamp
- 2020 CSTE Annual Conference | Seattle, WA | June 28-July 2
  - Call for Abstracts now open!
- CSTE supported EHR for SyS project – Thought Bridge, LLC

Agenda

DQC
Data Quality Committee

- Next meeting: Friday, December 13, 2019 12 – 1 pm ET
- CSTE Updates, other updates
- **2019 Year End Review and Future Endeavors**
- Preparing data with SQL in R Studio
- Open Mic

Year End Review | 2019

DQC
Data Quality Committee

Connecting ideas with people for impact

Year End Review | 2019

DQC
Data Quality Committee

ideas

- NSSP products and improvements for DQ
- Interactive DQ dashboards using R Shiny
- Monitoring Data Drop offs with SaTScan

- Sending additional Epic diagnosis code fields
- Re-mapping data fields to improve DQ
- Data visualization using dashboards

Year End Review | 2019

DQC
Data Quality Committee

people

- NSSP
- Alex Senetcky and Kristen Soto

- Sharon Greene from NYC Department of Health and Mental Hygiene
- HealthMonitoring

Year End Review | 2019

DQC
Data Quality Committee

impact

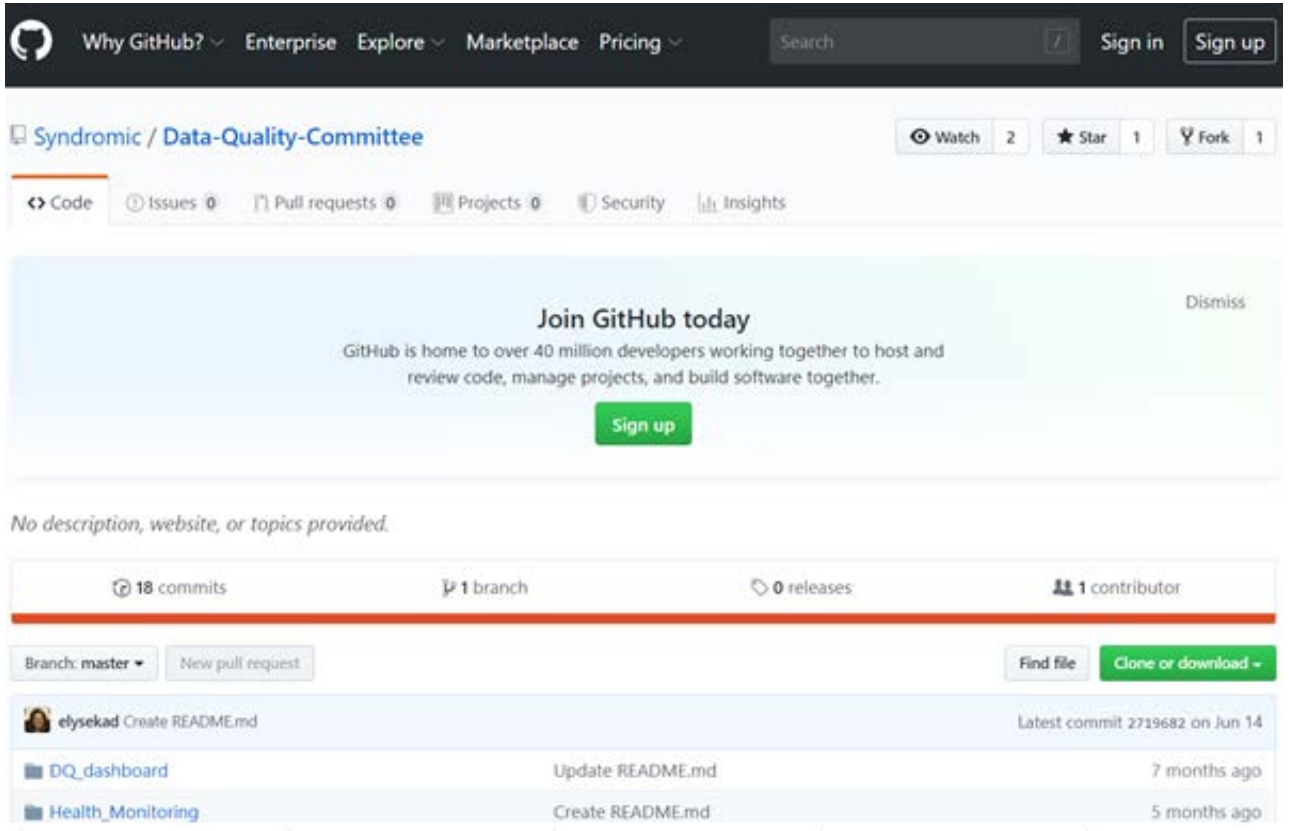
- Github resource
- NSSP Data Quality Dashboard
- ESSENCE Data Quality FIlters
- ESSENCE API
- NSSP Daily Site Summary Emails

- AMC Adminer
- AMC SAS Studio Data Quality on Demand Reports

Year End Review | 2019

DQC
Data Quality Committee

impact



National Syndromic Surveillance Program • Community of Practice

KNOWLEDGE REPOSITORY

Data Quality Committee (DQC) Recordings

Description:

October 11, 2019

September 13, 2019

August 9, 2019

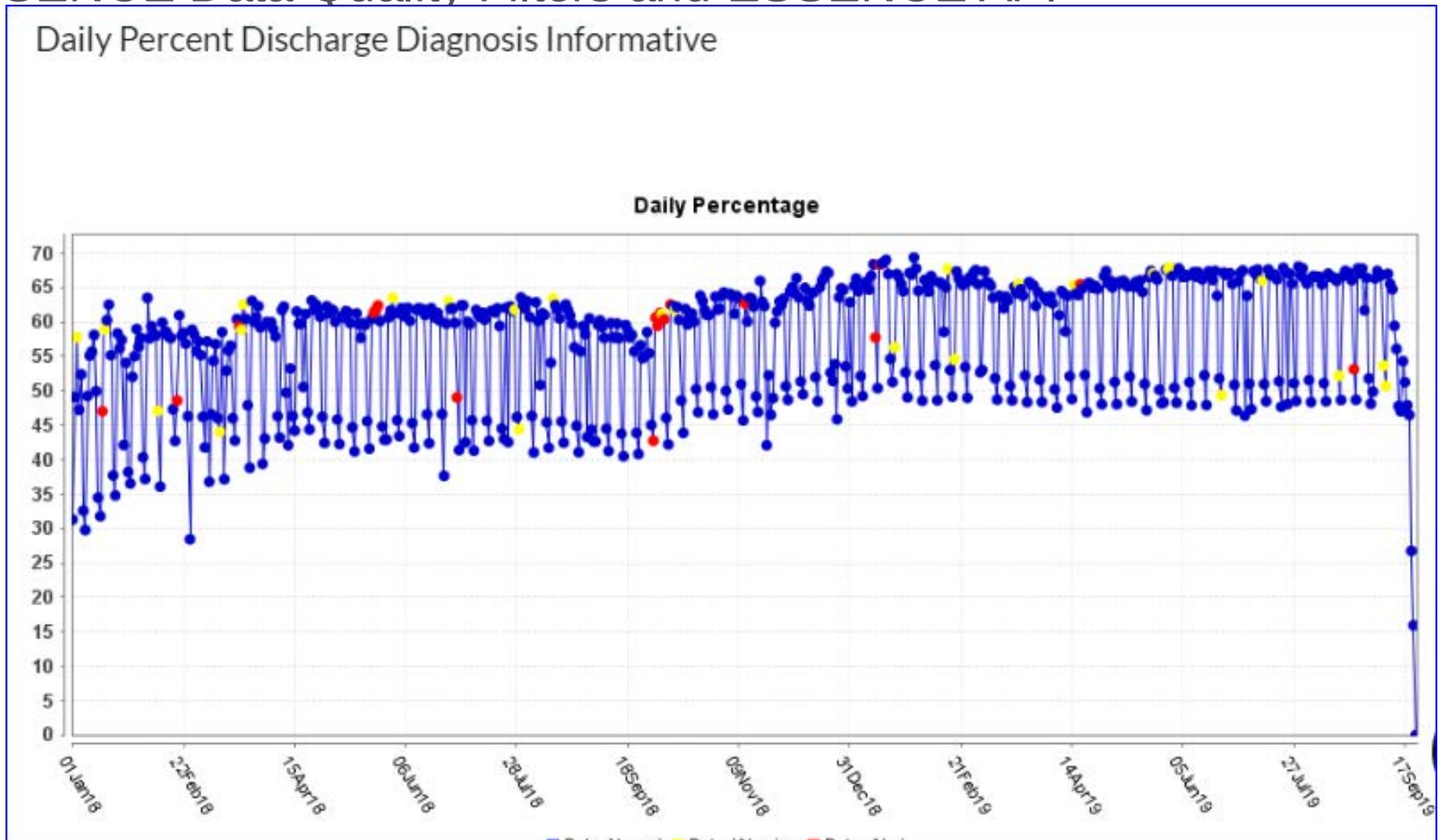
July 12, 2019

Author:

Data Quality Committee (DQC)

Year End Review | 2019

2019 ESSENCE Data Quality Filters and ESSENCE API



Year End Review | 2019

2019 NSSP SAS Studio Data Quality on Demand Reports

```
/* Directory Where DQOD Programs are stored and create subfolders to store output */
/*%let    sdir=/opt/sas/shared/homes/&sysuserid.,1,8)/public/NSSP/pgms/sas/dqod,*/
%let    sdir=/opt/sas/shared/repository/programs/DQOD_secondary_programs;
%put &sdir;
%include "&sdir/dqod_first.sas";    /* Creates DQOD and LOG, REPORT, and DATA subfolders in the users HOME directory */

/*********************************
    USERS INPUT BELOW
*********************************/
/* Site Information */
%let site_id= ;                     /* EX: %let site_id=960;          See your site_profile      */
%let site_short_name= ;             /* EX: %let site_short_name=AK;   See your site profile      */
/*Date Range over which data will be pulled to Create the DQOD reports */
%let start_date=20150101;           /* EX:  20180901  Selected Data Range Start Date in YYYYMMDD format  */
%let end_date=20150107;             /* EX:  20181110  Selected Data Range End Date   in YYYYMMDD format  */
%let dologeh;                       /* Detailed logs: Y=Write to log, N=Do Not Write To Log              */
/* Choose whether the program should output Completeness, Timeliness, Validity, or ALL  */
%let report_type=C;                 /* EX: %let report_type=( Completeness = C, Timeliness = T, Validity = V, All=A */

/*********************************
    Optional Custom Output
*********************************/

/* Validity Min and Max Columns for Provi
%let vlminc1;           /* First Column
%let vlmaxc81;          /* Last Column

%include "&sdir/dqodmain.sas";
ods results off;
%dqodmain;
```

Insert site ID and name

Select date range

Select DQ reports of interest

NSSP Exceptions Report-> Site: 930-Utah

Centers for Disease Control and Prevention

Dqod Report: 20150101 to 20150107

Feed: All - dqod

Updated at 23SEP2019 at 5:04 AM)

Data Flow Data Dictionary

Identification						Sending and Treating Facilities		Total Records Found in Processed, Exceptions and Filtered			
Feed Name	Parent Organization	Vendor Name	Facility Type	C Diseases Facility ID	Facility Name	Sending Facility ID	Treating Facility ID	Total Records	Filtered Records	% Total Records Filtered	Total Read
SAll Feed	SAll Parents	SAll Vendors	SAll Types	ALL	Facilities	SAll Sending	SAll Treating	1	0	0.000	1

Year End Review | 2019

2019 NSSP Data Quality Dashboard

Year End Review | 2019

DQC
Data Quality Committee

Connecting ideas with people for impact

Thank you to all the participants, contributors, and NSSP!

Year End Review | 2019 -> 2020

DQC
Data Quality Committee

Connecting ideas with people for impact

- DQ challenges in 2019?
- Ideas implemented at your site?
- Focus areas for 2020?
- CSTE Abstract ?

Agenda

DQC
Data Quality Committee

- Next meeting: Friday, December 13, 2019 12 – 1 pm ET
- CSTE Updates, other updates
- 2019 Year End Review and Future Endeavors
- **Preparing data with SQL in R Studio**
- Open Mic

Preparing data with SQL in R Studio

Eunice Santos
11/08/2019

Resources

- NSSP User Guides
- PostgreSQL Documentation | Chapter 3 | 3.5 Window Functions https://www.postgresql.org/docs/current/tutorial-window.html
- Lynda.com course Title: Advanced SQL for data science: Time Series
- SQL Server MVP Deep Dives
- Murach's SQL Server 2016 for developers
- Forecasting: Principles and Practice (free online)

Use case 1

LEN to Calculate string length of text from chief

```
#SQL Step
library(RODBC)
con<-odbcConnect("BioSense_Platform")

CCT_length_E <-paste ("SELECT LEN (pp.Chief_Complaint_Text) cct_length
, pp.Chief_Complaint_Text
FROM [dbo].[XX_PR_Processed] pp
JOIN [dbo].[XX_MFT] m on pp.C_Biosense_Facility_ID = m.C_Biosense_Facility_ID
WHERE pp.arrived_date between '2019-09-01' and '2019-09-30'
and m.facility_status= 'Active'
and m.facility_type = 'Emergency Care'
group by pp.Chief_Complaint_Text
order by cct_length;")

CCT_length_E <- sqlQuery (con, CCT_length_E)
close (con)
```

cct_length	Chief_Complaint_Text
26	abcdefghijklmnop….xyz
1	a

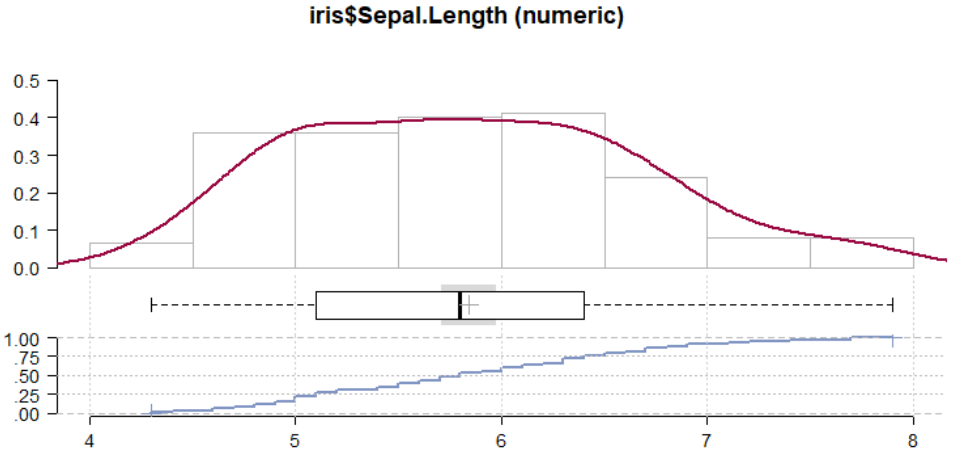
Use case 1

DescTools package for descriptive stats

```
#R Step
library (DescTools)
Desc(iris$Sepal.Length)
```

```
> Desc(iris$Sepal.Length)
-------------------------------------------------------------
-------------------------------------------------------------
iris$Sepal.Length (numeric)

  length       n     NAs  unique      0s    mean  meanCI
     150     150       0      35       0   5.843   5.710
          100.0%    0.0%            0.0%           5.977

     .05     .10     .25  median     .75     .90     .95
   4.600   4.800   5.100   5.800   6.400   6.900   7.255

   range      sd   vcoef     mad     IQR    skew    kurt
   3.600   0.828   0.142   1.038   1.300   0.309  -0.606

lowest : 4.3, 4.4 (3), 4.5, 4.6 (4), 4.7 (2)
highest: 7.3, 7.4, 7.6, 7.7 (4), 7.9
```

max



Use Case 2

IS NULL for Percent of all ED visits missing CC data

Data source: DataMart
Numerator: IS NULL
Denominator: IS NULL + IS NOT NULL
Percent = ((IS NULL)/ (IS NULL + IS NOT NULL))*100

IS NULL

```
select count (*), chief_complaint_text
FROM [dbo].[UT_PR_Processed] pp
   JOIN [dbo].[UT_MFT] m on pp.C_Biosense_Facility_ID =
m.C_Biosense_Facility_ID
 WHERE pp.arrived_date between '2019-09-01' and '2019-09-30'
   and m.facility_status= 'Active'
   and m.facility_type = 'Emergency Care'
   and pp.Chief_Complaint_Text IS NULL
         group by pp.Chief_Complaint_Text
```

IS NOT NULL

```
select count (*)
FROM [dbo].[UT_PR_Processed] pp
   JOIN [dbo].[UT_MFT] m on pp.C_Biosense_Facility_ID =
m.C_Biosense_Facility_ID
 WHERE pp.arrived_date between '2019-09-01' and '2019-09-30'
   and m.facility_status= 'Active'
   and m.facility_type = 'Emergency Care'
         and pp.Chief_Complaint_Text IS NOT NULL
```

Use Case 3

Percent of all ED visit missing CC data

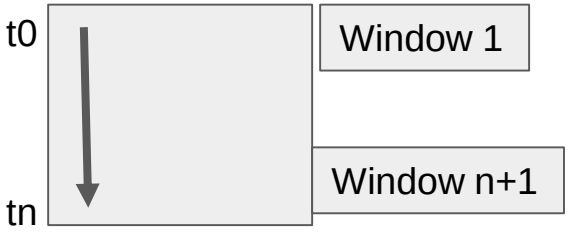
Data source: ESSENCE

Start date: 20190901
End date: 20190930
site= utah
Patient class = Emergency
Chief complaint Available = No

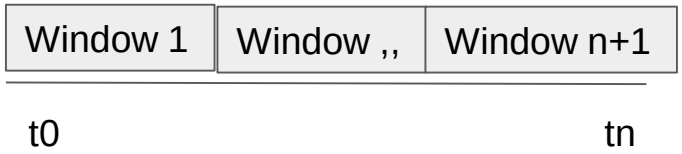
Data Table	
Chief Complaint Available	Site
	SiteName
No	10 (10 / 100)

Use case 4

Data Table



Time axis

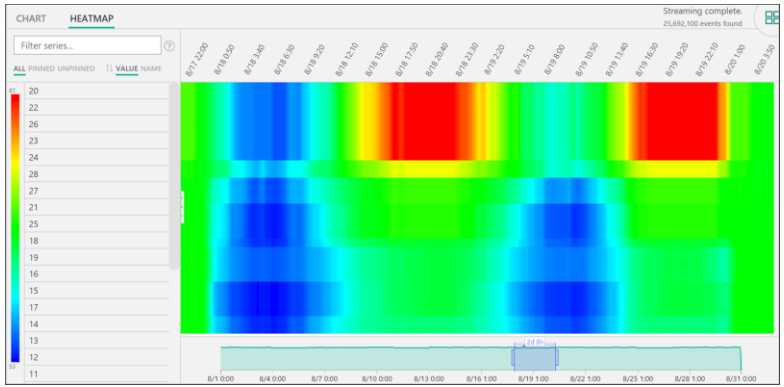


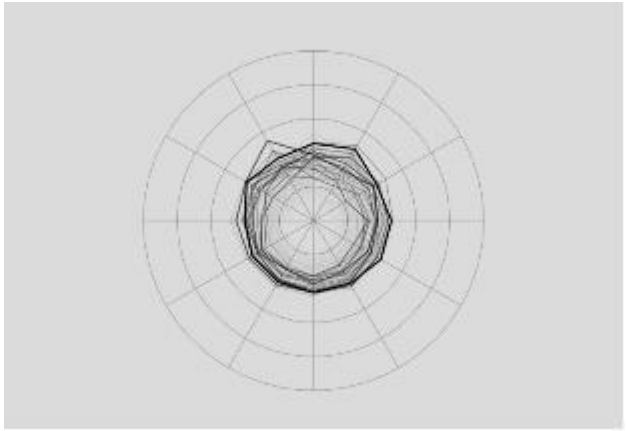
Analytics: Moving Average

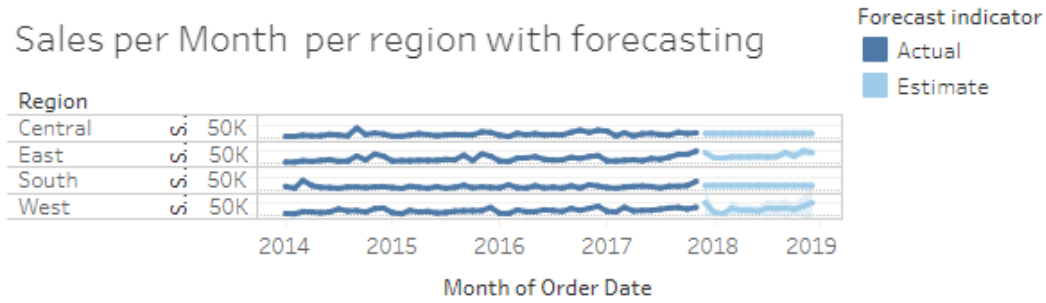
- One SQL solution
  - Windowing operation
- Factors that impact performance
  - Time unit for index (i.e. timestamp when query is run on calendar date)

Use case 5

Time is the main axis







Use case 6

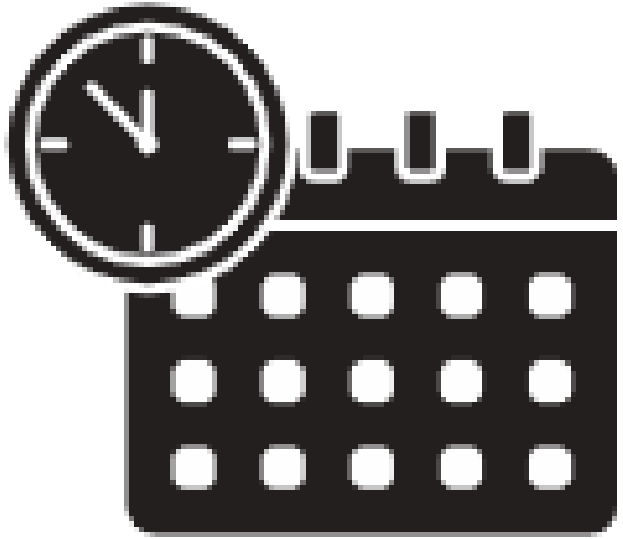
Common Table Expression to calculate previous day

```
#SQL Step
library(RODBC)
con<-odbcConnect("BioSense_Platform")
PreviousDay<- paste ("WITH daily_count AS
(SELECT date_trunc ('day', message_date_time) message_date
  , count(*) dcount
  FROM [dbo].[UT_PR_Processed]
  GROUP BY date_trunc ('day', message_date_time))
SELECT message_date, dcount, (SELECT dcount FROM daily_count dc2
   WHERE dc2.message_date = dc1.message_date - interval '1' day) previous_count
FROM daily_count dc1
WHERE message_date between '2019-10-09' and '2019-10-11'

order by message_date
;
")
```

Message_date	dcount	previous_count
11/09/2019	100	50
11/10/2019	100	100

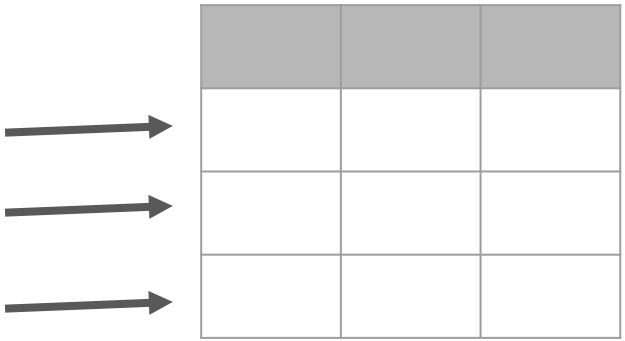
Overview of time series data

- Data values organized by time
- Sensor data in microseconds, daily closing stock price, health conditions in years



Record ingestion

- Captured in order of time
- Usually insert rather than an update to your database
- Data is added incrementally.
  - Usually the most recent data is queried more often



Aggregated tables/views

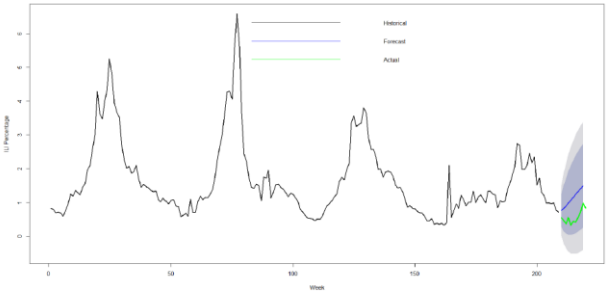
	Wide or Columnar	Long or Interleaved
Pros	Interpretability	New series does not affect the data structure
Cons	New columns affect the data structure	Magnitude must be the same for each series

Time	Series 1	Series 2	Series nth
201901	0.10	0.20	0.30
201902	0.15	0.25	0.35

Time	Series	Magnitude
201901	Value 1	0.10
201901	Value 2	0.20
201903	Value 3	0.30
201902	Value 1	0.15
201902	Value 2	0.25
201902	Value 3	0.35

Analytics and reporting

- Examples of analysis: historical trends, real-time alerts, predictive modeling, or forecasting
- Change is measured over time, enabling you to look backward and to predict future change.
- Visualized for trends, seasonality, cyclic patterns
- Autocorrelation between lagged values of a time series.

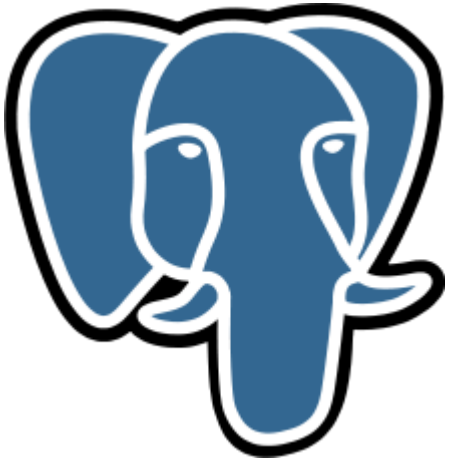


Why SQL?

Time Series	SQL feature
Sequential data	Window operations
Lots of data	Index column, partitions
Analyze for changes over time or forecasting	Analytic functions
Autocorrelation	LAG to access data in previous row(s)

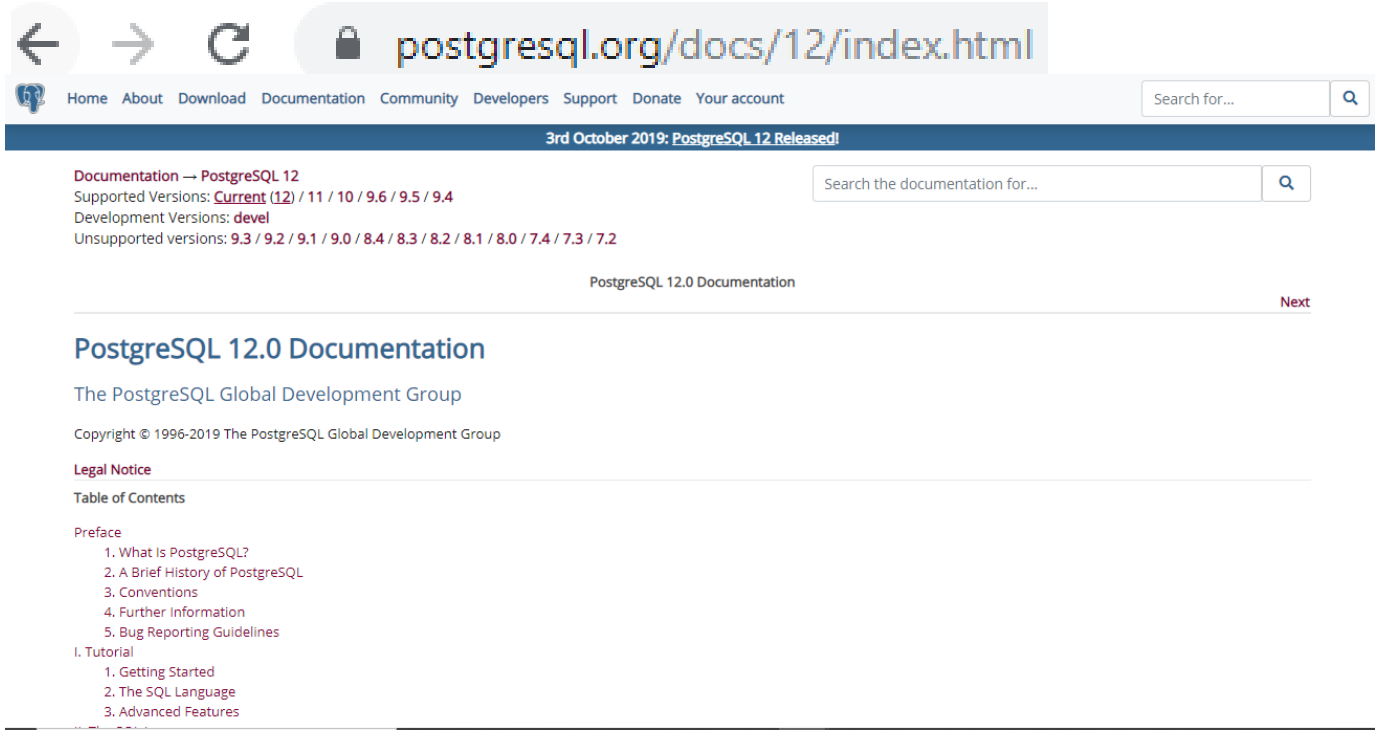
SQL features

- Common Table Expressions
- Window Functions
- Aggregate Function



Documentation for PostgreSQL

Table of Contents and Tutorial



Documentation for PostgreSQL

Common Table Expressions (CTE)

The syntax of a CTE

```
WITH cte_name1 AS (query_definition1)
[, cte_name2 AS (query_definition2)]
[...]
sql_statement
```

Two CTEs and a query that uses them

```
WITH Summary AS
(
    SELECT VendorState, VendorName, SUM(InvoiceTotal)
        AS SumOfInvoices
    FROM Invoices JOIN Vendors
      ON Invoices.VendorID = Vendors.VendorID
    GROUP BY VendorState, VendorName
),
TopInState AS
(
    SELECT VendorState, MAX(SumOfInvoices) AS SumOfInvoices
    FROM Summary
    GROUP BY VendorState
)
SELECT Summary.VendorState, Summary.VendorName,
    TopInState.SumOfInvoices
FROM Summary JOIN TopInState
    ON Summary.VendorState = TopInState.VendorState AND
       Summary.SumOfInvoices = TopInState.SumOfInvoices
ORDER BY Summary.VendorState;
```

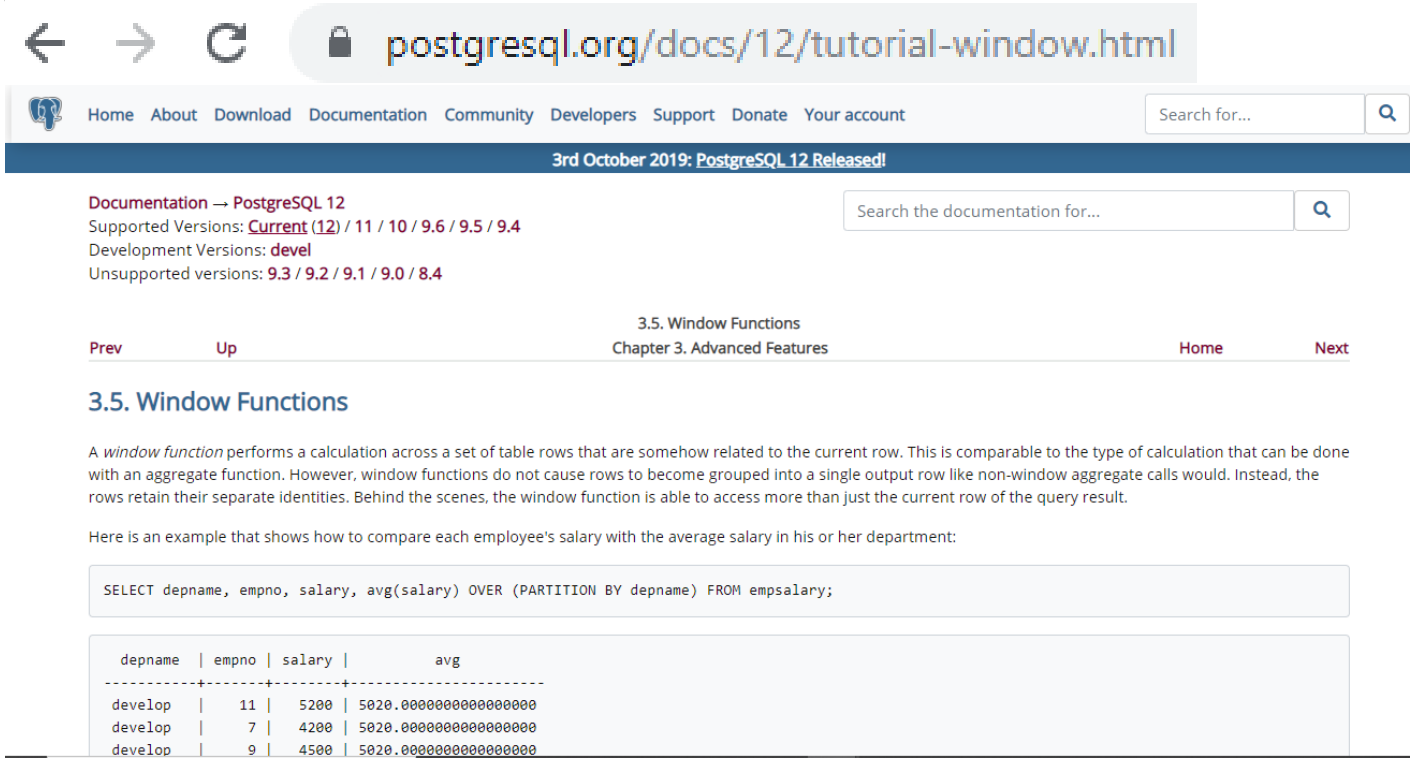
The result set

	VendorState	VendorName	SumOfInvoices
1	AZ	Wells Fargo Bank	662.00
2	CA	Digital Dreamworks	7125.34
3	DC	Reiter's Scientific & Pro Books	600.00
4	MA	Dean Witter Reynolds	1367.50
5	MI	Malloy Lithographing Inc	119892.41
6	NV	United Parcel Service	23177.96
7	OH	Edward Data Services	207.78
8	PA	Cardinal Business Media, Inc.	265.36

(10 rows)

Documentation for PostgreSQL

Window Functions



A query that calculates a cumulative total and moving average

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,
    SUM(InvoiceTotal) OVER (ORDER BY InvoiceDate) AS CumTotal,
    COUNT(InvoiceTotal) OVER (ORDER BY InvoiceDate) AS Count,
    AVG(InvoiceTotal) OVER (ORDER BY InvoiceDate) AS MovingAvg
FROM Invoices;
```

The result set

	InvoiceNumber	InvoiceDate	InvoiceTotal	CumTotal	Count	MovingAvg
1	989319-457	2015-12-08 00:00:00	3813.33	3813.33	1	3813.33
2	263253241	2015-12-10 00:00:00	40.20	3853.53	2	1926.765
3	963253234	2015-12-13 00:00:00	138.75	3992.28	3	1330.76
4	2-000-2993	2015-12-16 00:00:00	144.70	4195.23	6	699.205
5	963253251	2015-12-16 00:00:00	15.50	4195.23	6	699.205
6	963253261	2015-12-16 00:00:00	42.75	4195.23	6	699.205

The syntax of the OVER clause

```
aggregate_function OVER ([partition_by_clause]
     [order_by_clause])
```

A query that groups the summary data by date

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,
  SUM(InvoiceTotal) OVER (PARTITION BY InvoiceDate) AS DateTotal,
  COUNT(InvoiceTotal) OVER (PARTITION BY InvoiceDate) AS DateCount,
  AVG(InvoiceTotal) OVER (PARTITION BY InvoiceDate) AS DateAvg
FROM Invoices;
```

The result set

	InvoiceNumber	InvoiceDate	InvoiceTotal	DateTotal	DateCount	DateAvg
1	989319-457	2015-12-08 00:00:00	3813.33	3813.33	1	3813.33
2	263253241	2015-12-10 00:00:00	40.20	40.20	1	40.20
3	963253234	2015-12-13 00:00:00	138.75	138.75	1	138.75
4	2-000-2993	2015-12-16 00:00:00	144.70	202.95	3	67.65
5	963253251	2015-12-16 00:00:00	15.50	202.95	3	67.65
6	963253261	2015-12-16 00:00:00	42.75	202.95	3	67.65

The same query grouped by TermsID

```
SELECT InvoiceNumber, TermsID, InvoiceDate, InvoiceTotal,
    SUM(InvoiceTotal) OVER
        (PARTITION BY TermsID ORDER BY InvoiceDate) AS CumTotal,
    COUNT(InvoiceTotal) OVER
        (PARTITION BY TermsID ORDER BY InvoiceDate) AS Count,
    AVG(InvoiceTotal) OVER
        (PARTITION BY TermsID ORDER BY InvoiceDate) AS MovingAvg
FROM Invoices;
```

The result set

	InvoiceNumber	TermsID	InvoiceDate	InvoiceTotal	CumTotal	Count	MovingAvg
22	97-1024A	2	2016-03-20 00:00:00	356.48	9415.08	16	588.4425
23	31361833	2	2016-03-21 00:00:00	579.42	9994.50	17	587.9117
24	134116	2	2016-03-28 00:00:00	90.36	10084.86	18	560.27
25	989319-457	3	2015-12-08 00:00:00	3813.33	3813.33	1	3813.33
26	263253241	3	2015-12-10 00:00:00	40.20	3853.53	2	1926.765
27	963253234	3	2015-12-13 00:00:00	138.75	3992.28	3	1330.76

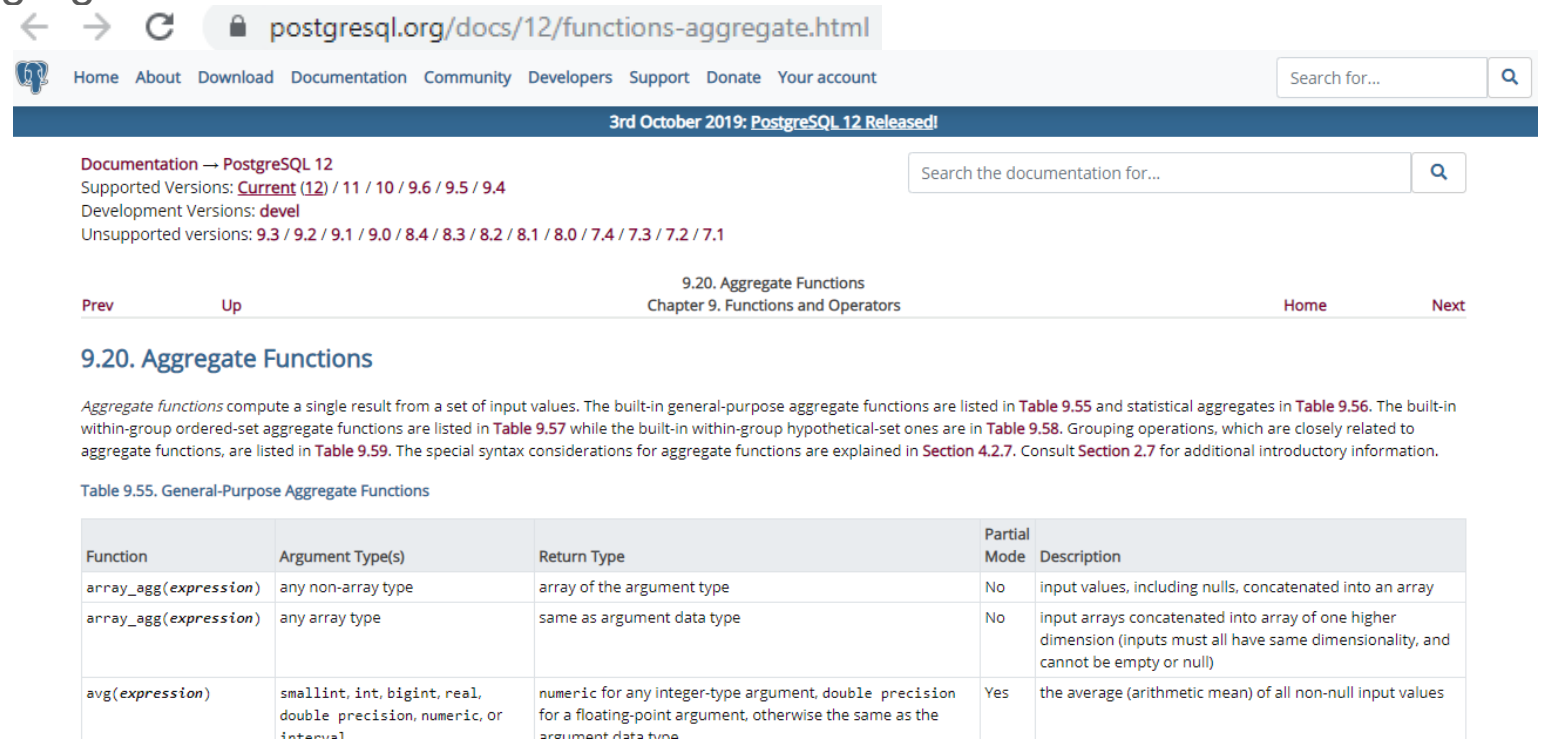
A query that uses the LAG function

```
SELECT RepID, SalesYear, SalesTotal AS CurrentSales,
    LAG(SalesTotal, 1, 0)
        OVER (PARTITION BY RepID ORDER BY SalesYear)
            AS LastSales,
    SalesTotal - LAG(SalesTotal, 1, 0)
        OVER (PARTITION BY REPID ORDER BY SalesYear)
            AS Change
FROM SalesTotals;
```

	RepID	SalesYear	CurrentSales	LastSales	Change
1	1	2014	1274856.38	0.00	1274856.38
2	1	2015	923746.85	1274856.38	-351109.53
3	1	2016	998337.46	923746.85	74590.61
4	2	2014	978465.99	0.00	978465.99
5	2	2015	974853.81	978465.99	-3612.18
6	2	2016	887695.75	974853.81	-87158.06

Documentation for PostgreSQL

Aggregate Functions



postgresql.org/docs/12/functions-aggregate.html

Home About Download Documentation Community Developers Support Donate Your account

Search for...

3rd October 2019: PostgreSQL 12 Released!

Documentation → PostgreSQL 12
Supported Versions: Current (12) / 11 / 10 / 9.6 / 9.5 / 9.4
Development Versions: devel
Unsupported versions: 9.3 / 9.2 / 9.1 / 9.0 / 8.4 / 8.3 / 8.2 / 8.1 / 8.0 / 7.4 / 7.3 / 7.2 / 7.1

Search the documentation for...

Prev Up 9.20. Aggregate Functions Chapter 9. Functions and Operators Home Next

9.20. Aggregate Functions

Aggregate functions compute a single result from a set of input values. The built-in general-purpose aggregate functions are listed in Table 9.55 and statistical aggregates in Table 9.56. The built-in within-group ordered-set aggregate functions are listed in Table 9.57 while the built-in within-group hypothetical-set ones are in Table 9.58. Grouping operations, which are closely related to aggregate functions, are listed in Table 9.59. The special syntax considerations for aggregate functions are explained in Section 4.2.7. Consult Section 2.7 for additional introductory information.

Table 9.55. General-Purpose Aggregate Functions

Function	Argument Type(s)	Return Type	Partial Mode	Description
`array_agg(`*`expression`*`)`	any non-array type	array of the argument type	No	input values, including nulls, concatenated into an array
`array_agg(`*`expression`*`)`	any array type	same as argument data type	No	input arrays concatenated into array of one higher dimension (inputs must all have same dimensionality, and cannot be empty or null)
`avg(`*`expression`*`)`	`smallint`, `int`, `bigint`, `real`, `double precision`, `numeric`, or `interval`	`numeric` for any integer-type argument, `double precision` for a floating-point argument, otherwise the same as the argument data type	Yes	the average (arithmetic mean) of all non-null input values

The columns in the SalesReps table

Column name	Data type
RepID	int
RepFirstName	varchar(50)
RepLastName	varchar(50)

The columns in the SalesTotals table

Column name	Data type
RepID	int
SalesYear	char(4)
SalesTotal	money

A query that uses the FIRST_VALUE and LAST_VALUE functions

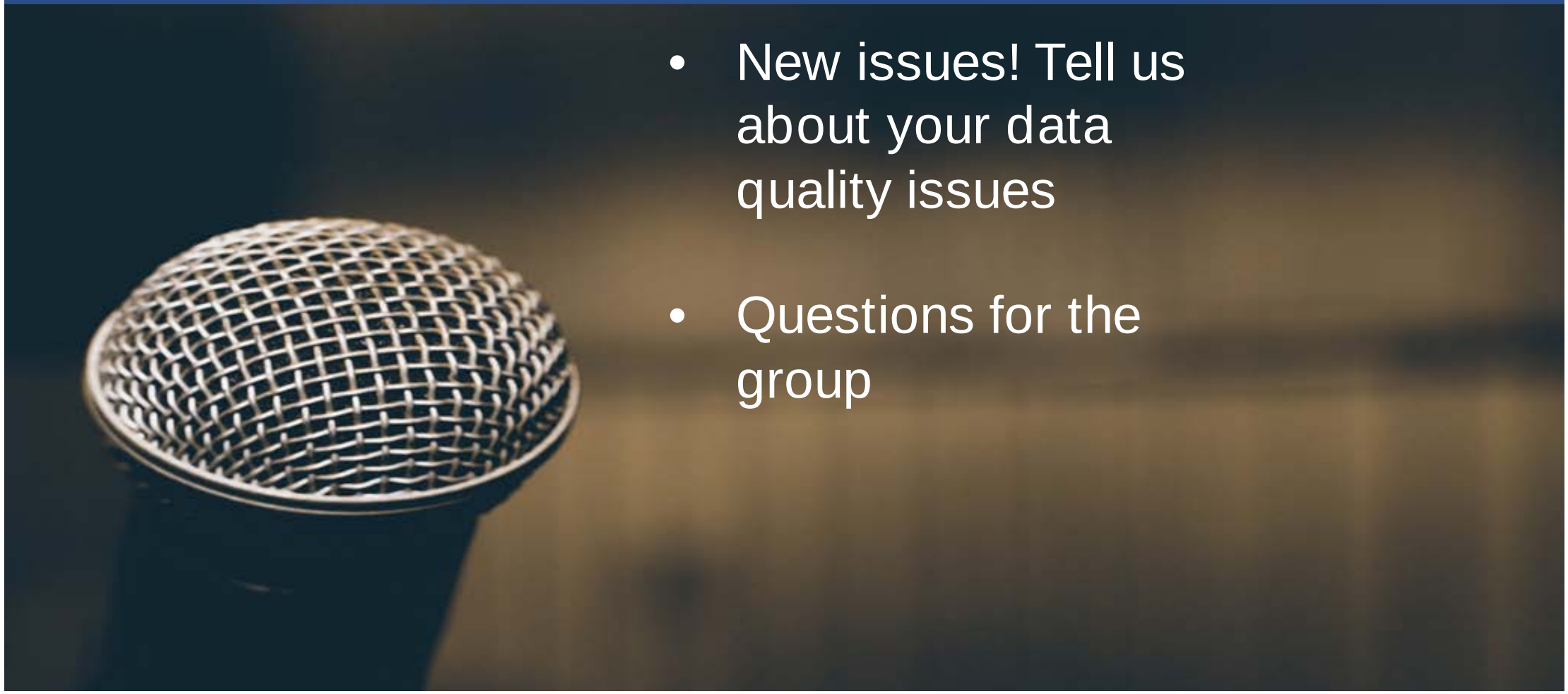
```
SELECT SalesYear, RepFirstName + ' ' +
       RepLastName AS RepName, SalesTotal,
       FIRST_VALUE(RepFirstName + ' ' + RepLastName)
           OVER (PARTITION BY SalesYear
               ORDER BY SalesTotal DESC)
           AS HighestSales,
       LAST_VALUE(RepFirstName + ' ' + RepLastName)
           OVER (PARTITION BY SalesYear
               ORDER BY SalesTotal DESC
              RANGE BETWEEN UNBOUNDED PRECEDING AND
                            UNBOUNDED FOLLOWING)
           AS LowestSales
FROM SalesTotals JOIN SalesReps
  ON SalesTotals.RepID = SalesReps.RepID;
```

The result set

	SalesYear	RepName	SalesTotal	HighestSales	LowestSales
1	2014	Jonathon Thomas	1274856.38	Jonathon Thomas	Sonja Martinez
2	2014	Andrew Markasian	1032875.48	Jonathon Thomas	Sonja Martinez
3	2014	Sonja Martinez	978465.99	Jonathon Thomas	Sonja Martinez
4	2015	Andrew Markasian	1132744.56	Andrew Markasian	Lydia Kramer
5	2015	Sonja Martinez	974853.81	Andrew Markasian	Lydia Kramer
6	2015	Jonathon Thomas	923746.85	Andrew Markasian	Lydia Kramer
7	2015	Phillip Winters	655786.92	Andrew Markasian	Lydia Kramer
8	2015	Lydia Kramer	422847.86	Andrew Markasian	Lydia Kramer
9	2016	Jonathon Thomas	998337.46	Jonathon Thomas	Lydia Kramer
10	2016	Sonja Martinez	887695.75	Jonathon Thomas	Lydia Kramer
11	2016	Phillip Winters	72443.37	Jonathon Thomas	Lydia Kramer
12	2016	Lydia Kramer	45182.44	Jonathon Thomas	Lydia Kramer

Open Mic

DQC
Data Quality Committee



- New issues! Tell us about your data quality issues
- Questions for the group