Matthew A Beck

IST659 Project Deliverable

[**Part 1: Design**](#_heading=h.gjdgxs) **2**

[Project Narrative](#_heading=h.3znysh7) 2

[Data Dictionary](#_heading=h.2et92p0) 2

[Data Questions](#_heading=h.2s8eyo1) 3

[Entity Relationship Diagram](#_heading=h.3dy6vkm) 3

[Logical Model Diagram](#_heading=h.1t3h5sf) 4

[Normalized Model](#_heading=h.4d34og8) 5

[**Part 2: Implementation**](https://docs.google.com/document/d/1eYOD0k4RUN7ok-umpmdUqCal6Bc4sKuECRO9tG1gm4c/edit#heading=h.spfj7uy2zkwz) **6**

[SQL DDL](https://docs.google.com/document/d/1eYOD0k4RUN7ok-umpmdUqCal6Bc4sKuECRO9tG1gm4c/edit#heading=h.2kprgnbxmo9w) 6

[SQL DML](https://docs.google.com/document/d/1eYOD0k4RUN7ok-umpmdUqCal6Bc4sKuECRO9tG1gm4c/edit#heading=h.wsdi3utu5iag) 16

[INSERT Statements](https://docs.google.com/document/d/1eYOD0k4RUN7ok-umpmdUqCal6Bc4sKuECRO9tG1gm4c/edit#heading=h.4trtr3797uke) 16

[Answers to Data Questions (SQL SELECT Statements)](https://docs.google.com/document/d/1eYOD0k4RUN7ok-umpmdUqCal6Bc4sKuECRO9tG1gm4c/edit#heading=h.2vclphkmrhup) 20

[GUI Prototype](https://docs.google.com/document/d/1eYOD0k4RUN7ok-umpmdUqCal6Bc4sKuECRO9tG1gm4c/edit#heading=h.wpwklx1y3r05) 22

[Reflection](https://docs.google.com/document/d/1eYOD0k4RUN7ok-umpmdUqCal6Bc4sKuECRO9tG1gm4c/edit#heading=h.mx5id0nsesu4) 23

# 

# Part 1: Design

## Project Narrative

My company is made up of a complex hierarchical structure with many business leaders overseeing one or many departments. Historically, success metrics and their associated visual representation have varied significantly from one area of the business to the other. The goal of my project is to link these metrics to the vital questions of the business in an extensible framework that can be easily accessed and updated by question askers and those responsible for answering. By simplifying both the collection and display of these metrics and their associated metadata, I can help leaders in the business focus on driving action instead of sifting through dense analytic reports.

## Data Dictionary

After conducting interviews with several business leaders and studying available frameworks for metric collection, I chose to document the following entities

|  |  |  |
| --- | --- | --- |
| **Entity** | **Attribute** | **Properties** |
| User | User | Required. Name of Employee, user of the metrics engine |
| Role | Role | Required and Unique. |
| Business Question | Question | Required and Unique. |
| Metric | Metric | Required. |
|  | Description | Explanation of what metric represents |
|  | Direction | Which direction (positive or negative) does the business want the metric to go? |
|  | Population | What groups are represented by this data |
|  | Data Steward | Who is responsible for maintaining this metric |
|  | Reference | Link to where more information on the metric can be found. |
| Metric Value | Value | Required. Recorded value of a metric |
|  | Date Entered | Date value entered the system |
|  | Period | Period of time from which a value was captured |
| Benchmark | Benchmark | Required. Business-determined value for an appropriate benchmark for a given metric |
|  | Entry Date | Date benchmark was entered the system |
|  | Valid From | Should benchmarks later be changed, a timestamp of when a given benchmark was first used |
|  | Valid To | Should benchmarks later be changed, a timestamp of when a given benchmark ceased being used. |
| Forecast | Forecast | Forecasted value for a linked metric |
|  | Period | Period of time for which a forecast was made |
|  | Entry Date | Date forecast was entered the system |
|  | Valid From | Should forecasts later be changed, a timestamp of when a given forecast was first used |
|  | Valid To | Should forecasts later be changed, a timestamp of when a given forecast ceased being used. |

|  |  |
| --- | --- |
| Business Rules | |
| A user has exactly one role  A role may be held by many users  A role belongs to only one supervisory organization at a time  A supervisory organization contains many roles  A role may ask one or many business questions  A business question may be asked by one or many roles  A business question may be answered by one or many metrics  A metric may answer one or many business questions  A metric may have one or many benchmarks (over time)  A benchmark may be used for exactly one metric  A metric may represent one or many metric values  A metric value may be used for exactly one metric  A metric may produce many forecasts  A forecast is produced for exactly one metric | |

The relevant stakeholders for this project are, for a given department:

**Executive** - Primary audience for business KPIs, reviews them with managers to gauge department performance

**Manager** - Relies on KPIs to measure their department’s performance, reviews with executives, analysts, and staff.

**Analyst** - Helps identify trends and produces reporting for managers to use in conversations with executives.

**Staff** - Individuals who generate the actual business output measured by KPIs.

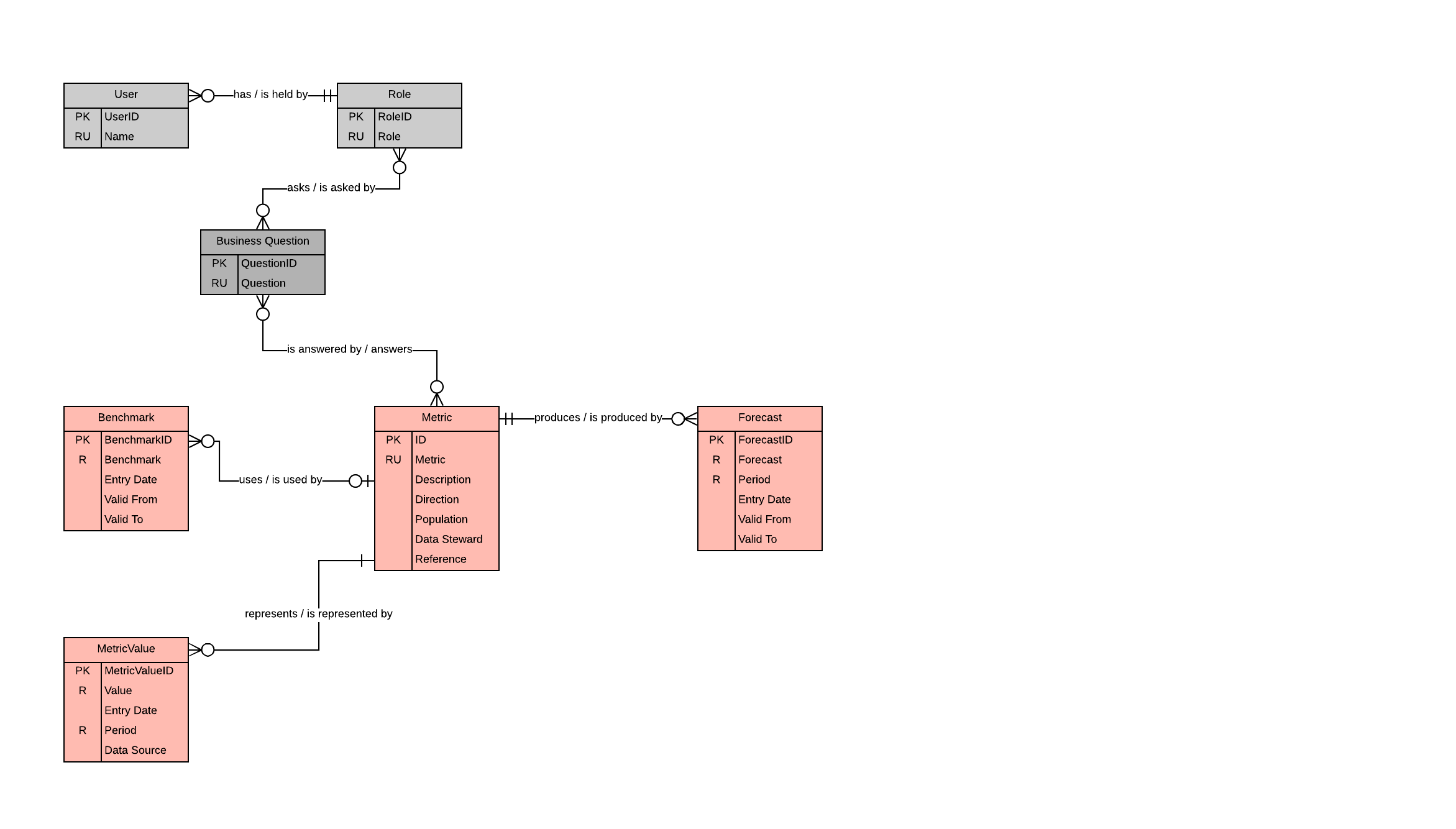
## Data Questions

Some questions this database must be able to answer:

1. How many roles and users are linked to a given business question?
2. Which business questions does a particular role need to ask?
3. What is our current company performance, based on the questions we care about, and relative to our benchmarks?
4. For each of our metrics, how did our actuals compare with our forecasts?
5. Which employees (users) can answer a given business question?

## Entity Relationship Diagram

When connected via relationships, the ERD is found on the following page. To manage the scope of this project, I have chosen to limit the database with the assumption that the database as presented will only contain data for one department. After the approach is piloted within the business, this framework can be expanded to accept a “department” entity.

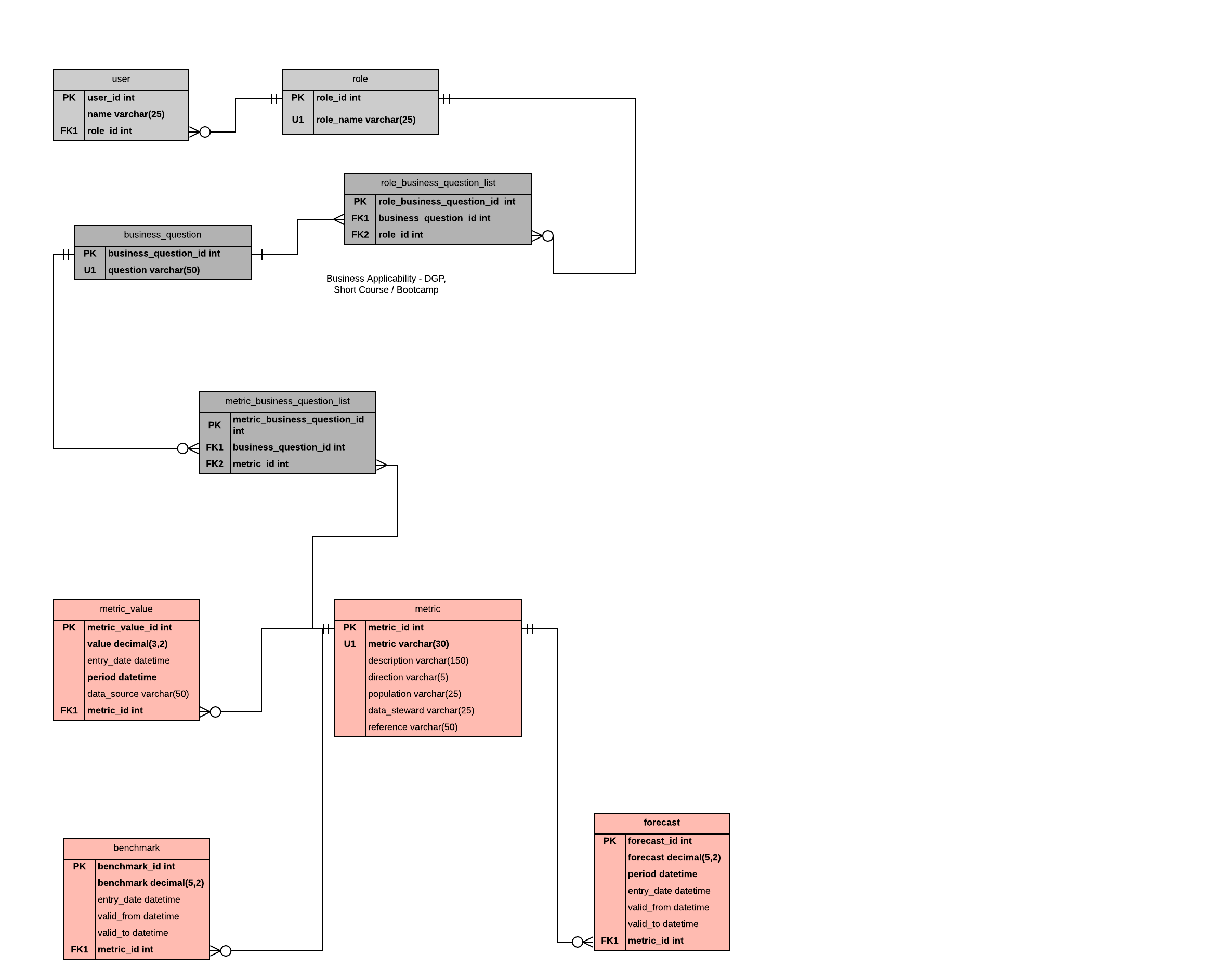


## Logical Model Diagram

In order to convert this conceptual ERD to a logical model, I did the following:

1. Reframed the existing entities as tables, renamed fields to be more system-friendly and added surrogate keys where necessary.
2. Assigned data types to fields listed in the data dictionary
3. Mapped relationships
   1. Established primary and foreign keys, documented the relevant constraints
   2. Created associative tables for the relationship between roles and business questions, and between business questions and their respective metrics.

Following these steps produced the following diagram:



## Normalized Model

The effort to create the logical model resulted in relations in third-normal form, so no further normalization was required.

# Part 2: Implementation

## SQL DDL

/\*

Author : Matt Beck

Course : IST659 M400

Term : October, 2019

\*/

/\*

Drop the tables first, if they exist

Dropped in reverse order of creation to avoid any problems with

foreign key references

Using DROP method

\*/

DROP TABLE IF EXISTS forecast

DROP TABLE IF EXISTS metric\_value

DROP TABLE IF EXISTS benchmark

DROP TABLE IF EXISTS metric\_business\_question\_list

DROP TABLE IF EXISTS metric

DROP TABLE IF EXISTS role\_business\_question\_list

DROP TABLE IF EXISTS business\_question

DROP TABLE IF EXISTS [user]

DROP TABLE IF EXISTS [role]

DROP FUNCTION IF EXISTS bq\_count\_roles

DROP FUNCTION IF EXISTS bq\_count\_users

DROP FUNCTION IF EXISTS max\_benchmark

DROP FUNCTION IF EXISTS derive\_latest\_value

DROP FUNCTION IF EXISTS next\_forecast

DROP PROCEDURE IF EXISTS dbo.add\_new\_benchmark

DROP PROCEDURE IF EXISTS dbo.add\_new\_forecast

DROP PROCEDURE IF EXISTS dbo.add\_new\_mbq\_mapping

DROP PROCEDURE IF EXISTS dbo.add\_new\_metric

DROP PROCEDURE IF EXISTS dbo.add\_new\_question

DROP PROCEDURE IF EXISTS dbo.add\_new\_rbq\_mapping

DROP PROCEDURE IF EXISTS dbo.add\_new\_role

DROP PROCEDURE IF EXISTS dbo.add\_new\_user

DROP PROCEDURE IF EXISTS dbo.delete\_mbq\_mapping

DROP PROCEDURE IF EXISTS dbo.delete\_rbq\_mapping

DROP PROCEDURE IF EXISTS dbo.delete\_user

DROP PROCEDURE IF EXISTS dbo.update\_existing\_benchmark\_value

DROP PROCEDURE IF EXISTS dbo.update\_existing\_forecast\_value

DROP PROCEDURE IF EXISTS dbo.update\_metric\_value

DROP PROCEDURE IF EXISTS dbo.update\_question

DROP PROCEDURE IF EXISTS dbo.update\_role\_name

DROP PROCEDURE IF EXISTS dbo.update\_user\_name

DROP VIEW IF EXISTS bq\_role\_user\_counts

DROP VIEW IF EXISTS business\_questions\_by\_role

DROP VIEW IF EXISTS company\_performance

DROP VIEW IF EXISTS forecast\_actual\_comparison

DROP VIEW IF EXISTS bq\_employee\_lookup

/\*

Creating Tables for Database

\*/

-- Creating the Role table

CREATE TABLE [role] (

--Columns for the Role table

role\_id int identity,

role\_name varchar(25) not null,

--Constraints on the role Table

CONSTRAINT PK\_role PRIMARY KEY (role\_id),

CONSTRAINT U1\_role UNIQUE(role\_name)

)

--End Creating the Role Table

GO

-- Creating the User table

CREATE TABLE [user] (

--Columns for the User table

user\_id int identity,

name varchar(25) not null,

role\_id int not null,

--Constraints on the User Table

CONSTRAINT PK\_user PRIMARY KEY (user\_id),

CONSTRAINT FK1\_user FOREIGN KEY (role\_id) REFERENCES role(role\_id)

)

--End Creating the User Table

GO

-- Creating the business\_question table

CREATE TABLE business\_question (

--Columns for the Role table

business\_question\_id int identity,

question varchar(100) not null,

--Constraints on the role Table

CONSTRAINT PK\_business\_question PRIMARY KEY (business\_question\_id),

CONSTRAINT U1\_question UNIQUE(question)

)

--End Creating the business question Table

GO

-- Creating the role\_business\_question\_list table

CREATE TABLE role\_business\_question\_list (

--Columns for the role\_business\_question\_list table

role\_business\_question\_id int identity,

business\_question\_id int not null,

role\_id int not null,

--Constraints on the role\_business\_question\_list Table

CONSTRAINT PK\_role\_business\_question\_list PRIMARY KEY (role\_business\_question\_id),

CONSTRAINT FK1\_role\_business\_question\_list FOREIGN KEY (business\_question\_id) REFERENCES business\_question(business\_question\_id),

CONSTRAINT FK2\_role\_business\_question\_list FOREIGN KEY (role\_id) REFERENCES role(role\_id)

)

--End Creating the role\_business\_question\_list table

GO

--Creating the metric table

CREATE TABLE metric (

metric\_id int identity,

metric varchar(30) not null,

description varchar(150),

direction varchar(5),

population varchar(25),

data\_steward varchar(25),

reference varchar(50),

CONSTRAINT PK\_metric PRIMARY KEY (metric\_id),

CONSTRAINT U1\_metric UNIQUE(metric)

);

--End Creating the metric Table

GO

-- Creating the metric\_business\_question\_list table

CREATE TABLE metric\_business\_question\_list (

--Columns for the metric\_business\_question\_list table

metric\_business\_question\_id int identity,

business\_question\_id int not null,

metric\_id int not null,

--Constraints on the metric\_business\_question\_list Table

CONSTRAINT PK\_metric\_business\_question\_list PRIMARY KEY (metric\_business\_question\_id),

CONSTRAINT FK1\_metric\_business\_question\_list FOREIGN KEY (business\_question\_id) REFERENCES business\_question(business\_question\_id),

CONSTRAINT FK2\_metric\_business\_question\_list FOREIGN KEY (metric\_id) REFERENCES metric(metric\_id)

)

--End Creating the role\_business\_question\_list table

GO

--Creating the metric\_value table

CREATE TABLE metric\_value (

--Columns for metric value table

metric\_value\_id int identity,

value decimal(3,2) not null,

entry\_date datetime default GetDate(),

period datetime not null,

metric\_id int not null,

--Constraints on the metric\_value table

CONSTRAINT PK\_metric\_value PRIMARY KEY (metric\_value\_id),

CONSTRAINT FK1\_metric\_value FOREIGN KEY (metric\_id) REFERENCES metric(metric\_id)

)

--End Creating the metric\_value table

GO

--Creating the benchmark table

CREATE TABLE benchmark (

--Columns for benchmark table

benchmark\_id int identity,

benchmark decimal(5,2) not null,

entry\_date datetime default GetDate(),

valid\_from datetime,

valid\_to datetime,

metric\_id int not null,

--Constraints on the benchmark table

CONSTRAINT PK\_benchmark PRIMARY KEY (benchmark\_id),

CONSTRAINT FK1\_benchmark FOREIGN KEY (metric\_id) REFERENCES metric(metric\_id)

)

--End Creating the benchmark table

GO

--Creating the forecast table

CREATE TABLE forecast (

--Columns for forecast table

forecast\_id int identity,

forecast decimal(5,2) not null,

period datetime,

entry\_date datetime default GetDate(),

valid\_from datetime,

valid\_to datetime,

metric\_id int not null,

--Constraints on the forecast table

CONSTRAINT PK\_forecast PRIMARY KEY (forecast\_id),

CONSTRAINT FK1\_forecast FOREIGN KEY (metric\_id) REFERENCES metric(metric\_id)

)

--End Creating the forecast table

GO

--Functions

--Create function for counting the number of roles based on a business question (or abstract to allow the user to include table name)

CREATE FUNCTION bq\_count\_roles(@bqID int)

RETURNS int AS

BEGIN

DECLARE @returnValue int --matches the function's return type

/\*

Get the count of roles for the provided question and

assign that value to @returnValue.

\*/

SELECT @returnValue = COUNT(DISTINCT role\_id) FROM role\_business\_question\_list rbq

WHERE rbq.business\_question\_id = @bqID

--Return @returnValue to the calling code.

RETURN @returnValue

END

GO

--Create function for counting # of Users based on business question provided

CREATE FUNCTION bq\_count\_users(@bqID int)

RETURNS int AS

BEGIN

DECLARE @returnValue int --matches the function's return type

/\*

Get the count of users for the provided question and

assign that value to @returnValue.

\*/

SELECT @returnValue = COUNT(DISTINCT u.user\_id) FROM role\_business\_question\_list rbq

JOIN [role] r ON rbq.role\_id=r.role\_id

JOIN [user] u ON u.role\_id=r.role\_id

WHERE rbq.business\_question\_id = @bqID

--Return @returnValue to the calling code.

RETURN @returnValue

END

GO

--Create function for pulling the most recent value for a given metric

CREATE FUNCTION derive\_latest\_value(@metric\_id int)

RETURNS decimal(5,2) AS --COUNT is an integer, so shall it be returned

BEGIN

DECLARE @returnValue decimal(5,2) --match function return type

DECLARE @max\_period datetime

select @max\_period = MAX(period) from metric\_value WHERE metric\_id = @metric\_id;

/\*

Pull the latest value from the metrics table, with the ID provided.

\*/

WITH CTE\_latest\_value

AS

(

SELECT

mv.\*,

ROW\_NUMBER() OVER(PARTITION BY mv.metric\_id ORDER BY IIF(mv.period >=@max\_period,1,2) ASC) rnk

FROM

metric\_value mv

)

SELECT @returnValue = value FROM CTE\_latest\_value WHERE rnk = 1 and metric\_id = @metric\_id

--Return @returnValue to the calling code.

RETURN @returnValue

END

GO

--Create function for pulling the most recent benchmark for a given metric

CREATE FUNCTION max\_benchmark(@metric\_id int)

RETURNS decimal(5,2) AS --COUNT is an integer, so shall it be returned

BEGIN

DECLARE @returnValue decimal(5,2) --match function return type

/\*

Collect the maximum benchmark value, given the provided metric ID

and assign that to the return value.

\*/

SELECT @returnValue = MAX(benchmark) FROM benchmark

WHERE benchmark.metric\_id = @metric\_id and valid\_to IS NULL

--Return @returnValue to the calling code.

RETURN @returnValue

END

GO

--Create function for pulling the predicted forecast for a given metric

CREATE FUNCTION next\_forecast(@metric\_id int)

RETURNS decimal(5,2) AS --COUNT is an integer, so shall it be returned

BEGIN

DECLARE @returnValue decimal(5,2) --match function return type

DECLARE @next\_forecast\_period datetime

select @next\_forecast\_period = MIN(period) from forecast f

WHERE metric\_id = @metric\_id and period > GETDATE();

/\*

Collect the next upcoming forecast, given the provided metric ID

and assign that to the return value.

\*/

WITH temp\_latest\_value

AS

(

SELECT

mv.\*,

ROW\_NUMBER() OVER(PARTITION BY mv.metric\_id ORDER BY IIF(mv.period >=@next\_forecast\_period,1,2) ASC) rnk

FROM

metric\_value mv

)

SELECT @returnValue = value FROM temp\_latest\_value WHERE rnk = 1 and metric\_id = @metric\_id

--Return @returnValue to the calling code.

RETURN @returnValue

END

GO

--Stored Procedures

/\*

Stored Procedure for updating a metric value

Stored Procedure for updating a benchmark value

Stored Procedure for updating a forecast

Stored Procedure for updating a role name

Stored Procedure for updating a user name

Stored Procedure for updating a business question

Stored Procedure for adding a new user

Stored Procedure for adding a new role

Stored Procedure for adding a new metric

Stored Procedure for adding a new metric value

Stored Procedure for adding a new benchmark

Stored Procedure for adding a new forecast

Stored Procedure for adding a role to question mapping

Stored Procedure for adding a metric to question mapping

Stored Procedure for deleting a user

Stored Procedure for deleting a role to question mapping

Stored Procedure for deleting a metric to question mapping

\*/

--Create Stored Procedure for updating a metric value

CREATE PROCEDURE update\_metric\_value(@metric\_id int, @period datetime, @value decimal(5,2))

AS

BEGIN

UPDATE metric\_value SET [value] = @value

WHERE metric\_id = @metric\_id AND period = @period

END

GO

--Create Stored Procedure for updating an existing benchmark value, in the case of entry error.

CREATE PROCEDURE update\_existing\_benchmark\_value(@benchmark\_id int,@metric\_id int, @period datetime, @benchmark decimal(5,2))

AS

BEGIN

UPDATE benchmark SET benchmark = @benchmark

WHERE metric\_id = @metric\_id AND benchmark\_id = @benchmark\_id

END

GO

--Create Stored Procedure for updating a forecast, in case of entry error.

CREATE PROCEDURE update\_existing\_forecast\_value(@metric\_id int, @period datetime, @forecast decimal(5,2))

AS

BEGIN

UPDATE forecast SET forecast = @forecast

WHERE metric\_id = @metric\_id AND [period] = @period

END

GO

--Create Stored Procedure for updating an existing role name

CREATE PROCEDURE update\_role\_name(@old\_role varchar(25), @new\_role varchar(25))

AS

BEGIN

UPDATE [role] SET role\_name = @new\_role

WHERE role\_name = @old\_role

END

GO

--Create Stored Procedure for updating an existing user name

CREATE PROCEDURE update\_user\_name(@old\_name varchar(25), @new\_name varchar(25))

AS

BEGIN

UPDATE [user] SET name = @new\_name

WHERE name = @old\_name

END

GO

--Create Stored Procedure for updating an existing business question

CREATE PROCEDURE update\_question(@bq\_id int, @new\_question\_lang varchar(25))

AS

BEGIN

UPDATE business\_question SET question = @new\_question\_lang

WHERE business\_question\_id = @bq\_id

END

GO

--Create Stored Procedure for adding a new forecast

CREATE PROCEDURE add\_new\_forecast(@metric\_id int, @forecast decimal(5,2), @period datetime, @valid\_from datetime = NULL ,@valid\_to datetime = NULL) AS

BEGIN

--Update valid\_to timestamp to ensure that the most recent forecast is used

--in the table

UPDATE forecast SET valid\_to = GetDate()

WHERE metric\_id = @metric\_id and valid\_to IS NULL

--Ensure valid\_from timestamp is from today's date.

SELECT @valid\_from = GetDate()

-- Now we can add the row using an INSERT Statement

INSERT INTO forecast (metric\_id,forecast,period,valid\_from,valid\_to)

VALUES(@metric\_id,@forecast,@period,@valid\_from,@valid\_to)

--Now return the @@identity so the calling code knows where

--the data ended up

RETURN @@identity

END

GO

--Create Stored Procedure for adding a new benchmark

CREATE PROCEDURE add\_new\_benchmark(@metric\_id int, @benchmark decimal(5,2), @valid\_from datetime = NULL ,@valid\_to datetime = NULL) AS

BEGIN

--Update valid\_to timestamp to ensure that the most recent forecast is used

--in the table

UPDATE benchmark SET valid\_to = GetDate()

WHERE metric\_id = @metric\_id and valid\_to IS NULL

--Ensure valid\_from timestamp is from today's date.

SELECT @valid\_from = GetDate()

-- Now we can add the row using an INSERT Statement

INSERT INTO benchmark (metric\_id,benchmark,valid\_from,valid\_to)

VALUES(@metric\_id,@benchmark,@valid\_from,@valid\_to)

--Now return the @@identity so the calling code knows where

--the data ended up

RETURN @@identity

END

GO

--Create Stored Procedure for adding a new user

CREATE PROCEDURE add\_new\_user(@name varchar(25), @role\_name varchar(20)) AS

BEGIN

DECLARE @role\_id INT

SELECT @role\_id = role\_id FROM [role] WHERE role\_name=@role\_name

-- Now we can add the row using an INSERT Statement

INSERT INTO [user] (name, role\_id)

VALUES(@name, @role\_id)

--Now return the @@identity so the calling code knows where

--the data ended up

RETURN @@identity

END

GO

--Create Stored Procedure for adding a new role

CREATE PROCEDURE add\_new\_role(@role\_name varchar(25)) AS

BEGIN

--Adding the row using an INSERT Statement

INSERT INTO [role] (role\_name)

VALUES(@role\_name)

--Now return the @@identity so the calling code knows where

--the data ended up

RETURN @@identity

END

GO

--Create Stored Procedure for adding a new business question

CREATE PROCEDURE add\_new\_question(@question varchar(100)) AS

BEGIN

--Adding the row using an INSERT Statement

INSERT INTO business\_question (question)

VALUES(@question)

--Now return the @@identity so the calling code knows where

--the data ended up

RETURN @@identity

END

GO

--Create Stored Procedure for adding a new metric

CREATE PROCEDURE add\_new\_metric(@metric varchar(30),@description varchar(150) = NULL,@direction varchar(5) = 'Up',

@population varchar(25) = '',@data\_steward varchar(25)='Data Science',

@reference varchar(50) = "http://log.blog.docs") AS

BEGIN

--Adding the row using an INSERT Statement

INSERT INTO metric (metric,description, direction,population,data\_steward,reference)

VALUES(@metric,@description, @direction,@population,@data\_steward,@reference)

--Now return the @@identity so the calling code knows where

--the data ended up

RETURN @@identity

END

GO

--Create Stored Procedure for mapping roles to questions

CREATE PROCEDURE add\_new\_rbq\_mapping(@role\_id int, @bq\_id int) AS

BEGIN

--Adding the row using an INSERT Statement

INSERT INTO role\_business\_question\_list (role\_id,business\_question\_id)

VALUES(@role\_id,@bq\_id)

--Now return the @@identity so the calling code knows where

--the data ended up

RETURN @@identity

END

GO

--Create Stored Procedure for mapping questions to metrics

CREATE PROCEDURE add\_new\_mbq\_mapping(@metric\_id int, @bq\_id int) AS

BEGIN

--Adding the row using an INSERT Statement

INSERT INTO metric\_business\_question\_list (metric\_id,business\_question\_id)

VALUES(@metric\_id,@bq\_id)

--Now return the @@identity so the calling code knows where

--the data ended up

RETURN @@identity

END

GO

--Create Stored Procedure for deleting a user

CREATE PROCEDURE delete\_user(@user\_id int,@name varchar(25)) AS

BEGIN

--Deleting the user using a DELETE Statement. Requiring both name and ID to

--ensure deletion is the desired action.

DELETE FROM [user]

WHERE name=@name and user\_id = @user\_id

END

GO

--Create Stored Procedure for deleting a role-to-question mapping

CREATE PROCEDURE delete\_rbq\_mapping(@role\_id int,@bq\_id int) AS

BEGIN

--Deleting the user using a DELETE Statement. Requiring both ID's

--ensure deletion is the desired action on the right fields.

DELETE FROM role\_business\_question\_list

WHERE role\_id=@role\_id and business\_question\_id = @bq\_id

END

GO

--Create Stored Procedure for deleting a metric-to-question mapping

CREATE PROCEDURE delete\_mbq\_mapping(@metric\_id int,@bq\_id int) AS

BEGIN

--Deleting the user using a DELETE Statement. Requiring both ID's

--ensure deletion is the desired action on the right fields.

DELETE FROM metric\_business\_question\_list

WHERE metric\_id=@metric\_id and business\_question\_id = @bq\_id

END

GO

--Creating Views

/\*

View for Count of roles, users per Business Question

View for List of business questions asked by a given roles

View for averages of metrics

View for metric, value, benchmark, forecast

View for establishing available ranges for each metric

\*/

--

--View 1 - How many roles and users are associated with the available business questions?

--Creating View for Counts of roles, users

CREATE VIEW bq\_role\_user\_counts AS (

SELECT question

, dbo.bq\_count\_roles(bq.business\_question\_id) as role\_count

, dbo.bq\_count\_users(bq.business\_question\_id) as user\_count

FROM business\_question bq

)

GO

--View 2 - What are the questions asked by a given business role?

--Creating View for List of Business Questions

CREATE VIEW business\_questions\_by\_role AS (

SELECT role\_name

, question as questions\_asked

FROM business\_question bq

JOIN role\_business\_question\_list rbq ON rbq.business\_question\_id=bq.business\_question\_id

JOIN [role] r ON rbq.role\_id=r.role\_id

--WHERE r.role\_name='Executive'

)

GO

--View 3 - What are the rolling averages for each metric, listed by their associated business question?

--Creating View 3

CREATE VIEW company\_performance AS (

SELECT question

,m.metric

, AVG(value) as overall\_value

, dbo.max\_benchmark(m.metric\_id) as overall\_benchmark

, AVG(dbo.max\_benchmark(m.metric\_id) - value) as difference\_from\_benchmark

, dbo.next\_forecast(m.metric\_id) as next\_forecast

FROM metric\_value mv

JOIN metric m ON m.metric\_id=mv.metric\_id

JOIN metric\_business\_question\_list mbq ON m.metric\_id=mbq.metric\_id

JOIN business\_question bq on bq.business\_question\_id=mbq.business\_question\_id

GROUP BY question,metric,m.metric\_id

)

GO

--View 4 - How did our actuals compare to our forecast?

--Creating View 4

CREATE VIEW forecast\_actual\_comparison AS (

SELECT m.metric

, AVG(f.forecast) as forecast\_average

, AVG(mv.value) actual\_average

,AVG(mv.value - f.forecast)\*100 as difference\_from\_forecast

FROM metric m

JOIN metric\_value mv ON m.metric\_id=mv.metric\_id

JOIN forecast f ON f.metric\_id = m.metric\_id AND mv.period = f.period

GROUP BY metric

)

GO

--View 5 -Which employees are responsible for which business questions?

-- Creating View 5

CREATE VIEW bq\_employee\_lookup AS (

SELECT

u.name

,bq.question

FROM business\_question bq

JOIN role\_business\_question\_list rbq ON rbq.business\_question\_id=bq.business\_question\_id

JOIN [role] r ON rbq.role\_id=r.role\_id

JOIN [user] u ON u.role\_id=r.role\_id

)

GO

## SQL DML

### INSERT Statements

--Insert into role table

INSERT INTO [role]

(role\_name)

VALUES

('Executive'),

('Manager'),

('Analyst'),

('Staff')

--End role table insertion

GO

--Insert into user table

INSERT INTO [user]

(name,role\_id)

VALUES

('Haleigh Musslewhite',1),

('Barbi Barbary',2),

('Ignace Veldens',3),

('Schuyler Beldum',4),

('Pauli Wherry',1),

('Caryl Jarrell',2),

('Brodie Auten',3),

('Alane Poveleye',4),

('Skylar Claw',1),

('Corny Immings',2),

('Aluin Rayer',4),

('Aluino Sheraton',2),

('Krystyna Crocetti',3),

('Ayn Jecks',4),

('Karrah Eisenberg',2),

('Cullie Gehrts',4),

('Arv Busby',4),

('Thorny Arp',4),

('Olly Savory',4),

('Clarie Prose',4)

--End role table insertion

GO

--Insert into business question table

INSERT INTO business\_question

(question)

VALUES

('At what rate are we generating new leads?'),

('How is the sales funnel performing?'),

('How many visits does our website get?'),

('How are email campaigns performing?'),

('How do our sales forecasts compare to management targets?'),

('What is our total cost of acquisition?'),

('How satisfied are our clients?')

--End business question table insertion

GO

--Insert into role/business question list table

INSERT INTO role\_business\_question\_list

(business\_question\_id,role\_id)

VALUES

(1,1),(1,2),(1,3),(1,4),(2,1),(2,2),(2,3),(3,2),(3,3),(3,4),(4,2),(4,3),(4,4),(5,1),(5,3),(6,1),(6,3),(7,1),(7,3),(7,4)

--End role/business question list table insertion

GO

--Insert into metric table

INSERT INTO metric

(metric,description,direction,population,data\_steward,reference)

VALUES

('Lead Generation Rate','Velocity of new leads into the sales funnel','Up','Leads','Sales Operations','https://referencepage.io'),

('Sales Funnel Performance','Rate of Sales made relative to previous year','Up','','Sales Operations','https://bloglines.com/dui/proin/leo/odio.xml?partu'),

('Quarterly Website Visits','# of Website visits in a quarter','Up','Web Visitors','Web Sales','http://oakley.com/felis/sed/interdum/venenatis.xml'),

('Website Visit Return Rate','# of users who return to the website','','Web Visitors','Web Sales','https://purevolume.com/morbi.json?accumsan=neque&t'),

('Forecast-to-Target Ratio','Ratio of our forecasted performance relative to company targets','Down','','Sales Analytics','https://storify.com/id/consequat/in/consequat/ut/n'),

('TCA Rate','Rate of Total Cost of acquiring a sale','Down','','Sales Analytics','http://amazonaws.com/ut/massa.jpg?semper=consectet'),

('Client Satisfaction','% of clients who report they are satisfied with their product','Up','',NULL,'https://bluehost.com/elementum/nullam/varius/nulla'),

('Email Open Rate','% of sent emails that were opened','Up',NULL,'','https://parallels.com/posuere/cubilia/curae/donec/'),

('Email Click Rate','% of sent emails where user clicked the link','Up','Email List','',NULL)

--End metric table insertion

GO

--Insert into metric/business question list table

INSERT INTO metric\_business\_question\_list

(business\_question\_id,metric\_id)

VALUES

(1,1),

(2,2),

(3,3),

(3,4),

(4,8),

(4,9),

(5,5),

(6,6),

(7,7)

--End metric/business question list table insertion

GO

--Insert into metric value table

INSERT INTO metric\_value

(value,period,metric\_id)

VALUES

(0.51,'2019-01-01 05:00:00',1),

(0.99,'2019-02-01 05:00:00',1),

(0.57,'2019-03-01 05:00:00',1),

(0.51,'2019-04-01 05:00:00',1),

(0.69,'2019-05-01 05:00:00',1),

(0.75,'2019-06-01 05:00:00',1),

(0.14,'2019-07-01 05:00:00',1),

(0.93,'2019-08-01 05:00:00',1),

(0.71,'2019-09-01 05:00:00',1),

(0.55,'2019-01-01 05:00:00',2),

(0.96,'2019-02-01 05:00:00',2),

(0.67,'2019-03-01 05:00:00',2),

(0.78,'2019-04-01 05:00:00',2),

(0.62,'2019-05-01 05:00:00',2),

(0.33,'2019-06-01 05:00:00',2),

(0.64,'2019-07-01 05:00:00',2),

(0.26,'2019-08-01 05:00:00',2),

(0.14,'2019-09-01 05:00:00',2),

(0.46,'2019-01-01 05:00:00',3),

(0.93,'2019-02-01 05:00:00',3),

(0.87,'2019-03-01 05:00:00',3)

--(truncated)

--End metric value table insertion

GO

--Insert into benchmark table

INSERT INTO benchmark

(benchmark,valid\_from,valid\_to,metric\_id)

VALUES

(0.6,'2019-01-01 05:00:00',NULL,1),

(0.8,'2019-01-01 05:00:00',NULL,2),

(0.5,'2019-01-01 05:00:00',NULL,3),

(0.4,'2019-01-01 05:00:00','2019-05-31 05:00:00',4),

(0.6,'2019-06-01 05:00:00',NULL,4),

(0.65,'2019-01-01 05:00:00',NULL,5),

(0.72,'2019-01-01 05:00:00',NULL,6),

(0.60,'2019-01-01 05:00:00','2019-06-30 05:00:00',7),

(0.70,'2019-01-01 05:00:00',NULL,7),

(0.75,'2019-01-01 05:00:00',NULL,8),

(0.38,'2019-01-01 05:00:00',NULL,9)

--End benchmark table insertion

GO

--Insert into forecast table

INSERT INTO forecast

(forecast,period,valid\_from,valid\_to,metric\_id)

VALUES

(0.91,'2019-01-01 05:00:00','2019-01-01 05:00:00',NULL,1),

(0.12,'2019-04-01 05:00:00','2019-01-01 05:00:00',NULL,1),

(0.56,'2019-07-01 05:00:00','2019-01-01 05:00:00',NULL,1),

(0.96,'2019-10-01 05:00:00','2019-01-01 05:00:00',NULL,1),

(0.21,'2019-01-01 05:00:00','2019-01-01 05:00:00',NULL,2),

(0.72,'2019-04-01 05:00:00','2019-01-01 05:00:00',NULL,2),

(0.02,'2019-07-01 05:00:00','2019-01-01 05:00:00',NULL,2)

--(truncated )

--End benchmark table insertion

GO

--Demonstrate Stored Procedures at work:

--Updating Existing Metrics

EXEC update\_metric\_value 1, '2019-08-01 05:00:00', 0.63

--Adding a new forecast

DECLARE @new\_forecast\_id INT

EXEC @new\_forecast\_id = add\_new\_forecast 1, 0.85, '2020-01-01 05:00:00'

SELECT \* FROM forecast WHERE forecast\_id = @new\_forecast\_id

GO

--Adding a new user

DECLARE @new\_user\_id INT

EXEC @new\_user\_id = add\_new\_user 'Wendy Williams', 'Manager'

SELECT \* FROM [user] WHERE user\_id = @new\_user\_id

GO

--Adding a new role

DECLARE @new\_role\_id INT

EXEC @new\_role\_id = add\_new\_role 'Senior Staff'

SELECT \* FROM [role] WHERE role\_id = @new\_role\_id

GO

--Adding a new question

DECLARE @new\_question\_id INT

EXEC @new\_question\_id = add\_new\_question 'How is our employee productivity?'

SELECT \* FROM business\_question WHERE business\_question\_id = @new\_question\_id

GO

--Adding a new metric

DECLARE @new\_metric\_id INT

EXEC @new\_metric\_id = add\_new\_metric 'Employee Productivity Rate'

SELECT \* FROM metric WHERE metric\_id = @new\_metric\_id

GO

--Add new role-to-business-question mapping

DECLARE @new\_mapping\_id INT

EXEC @new\_mapping\_id = add\_new\_rbq\_mapping 5,8

SELECT \* FROM role\_business\_question\_list WHERE role\_business\_question\_id = @new\_mapping\_id

GO

--Add new metric-to-business-question mapping

DECLARE @new\_mapping\_id INT

EXEC @new\_mapping\_id = add\_new\_mbq\_mapping 10, 8

SELECT \* FROM metric\_business\_question\_list WHERE metric\_business\_question\_id = @new\_mapping\_id

GO

/\*

Could perform these actions, but the intention is to use a GUI to update

benchmarks and forecasts.

--Add new forecasts for 2020

EXEC add\_new\_forecast VALUES

--Add new benchmark for the new employee productivity rate

EXEC add\_new\_benchmark VALUES

\*/

--Deleting a user, where in this instance we've terminated Wendy's employment.

EXEC delete\_user 21, 'Wendy Williams'

/\*

Including these two queries to demonstrate that the mappings can be deleted at will.

--Deleting a metric-to-question mapping

EXEC delete\_mbq\_mapping 10, 8

--Deleting a role-to-question mapping

EXEC delete\_rbq\_mapping 5, 8

\*/

/\*

There are only a select few use cases for deletion in this database - most values

should only be updated to ensure the business can track what has been done

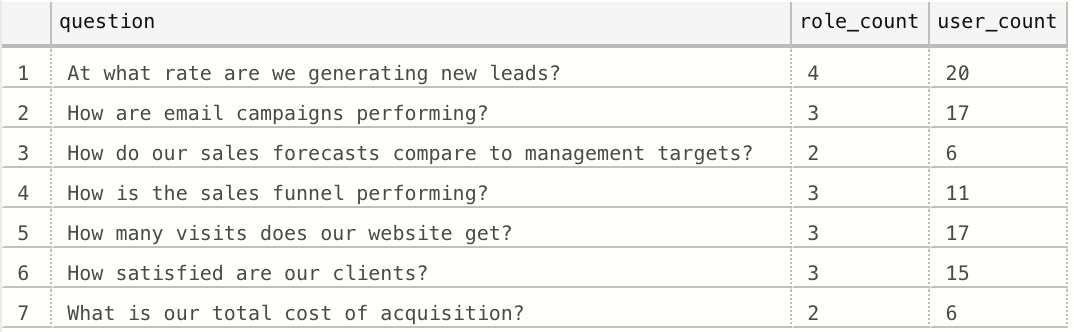
before. Historical data is \*important\*.

\*/

## Answers to Data Questions (SQL Select Statements)

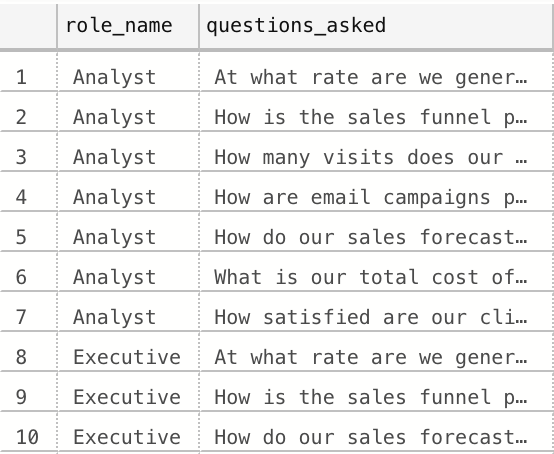
**--1. How many roles and users are linked to a given business question?**

SELECT \* FROM bq\_role\_user\_counts



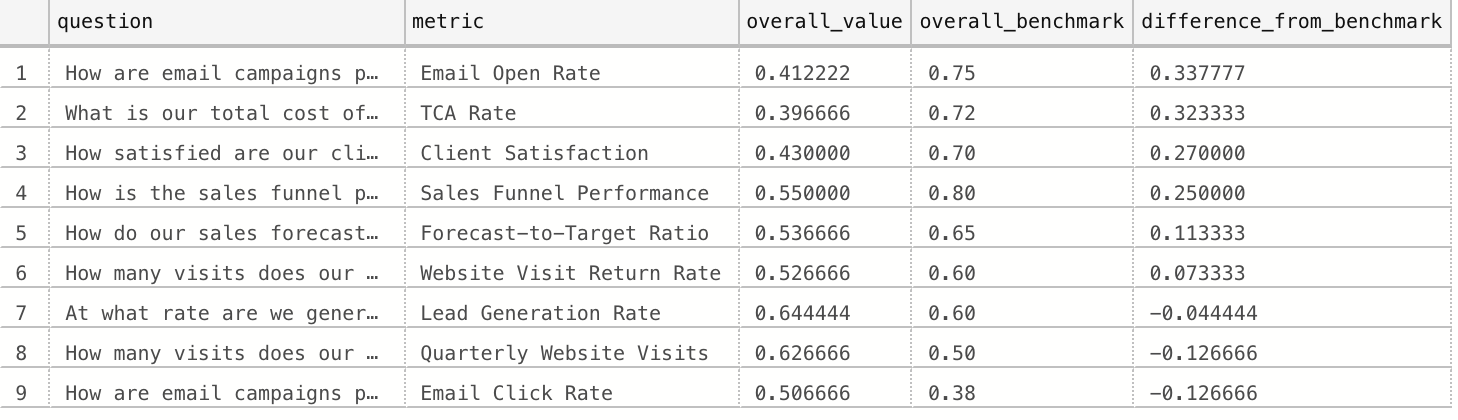
**--2. Which business questions does a particular role need to ask?**

SELECT \* FROM business\_questions\_by\_role ORDER BY role\_name;



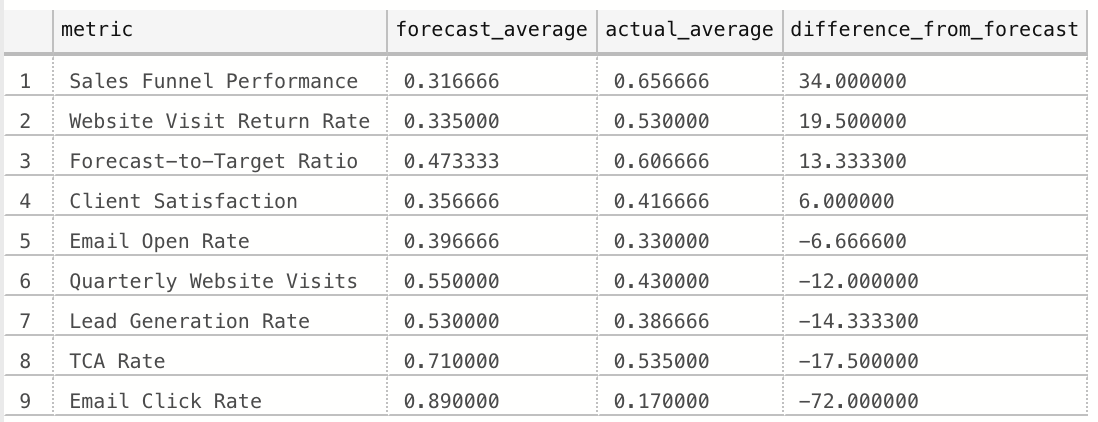
**3. What is our current company performance, based on the questions we care about, and relative to our benchmarks?**

SELECT \* FROM company\_performance ORDER BY difference\_from\_benchmark DESC;



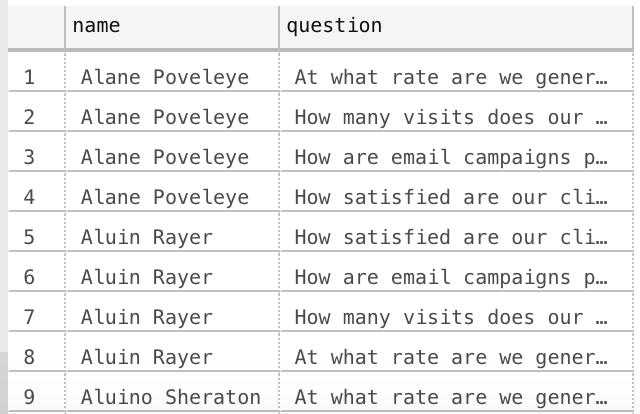
**4. For each of our metrics, how did our actuals compare with our forecasts?**

SELECT \* FROM forecast\_actual\_comparison ORDER BY difference\_from\_forecast DESC;



**5. Which employees (users) can answer a given business question?**

SELECT \* FROM bq\_employee\_lookup ORDER BY name;



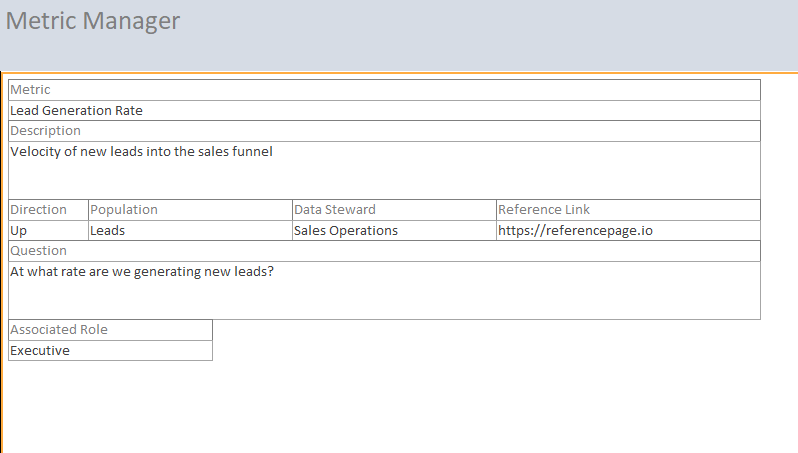
## 

## 

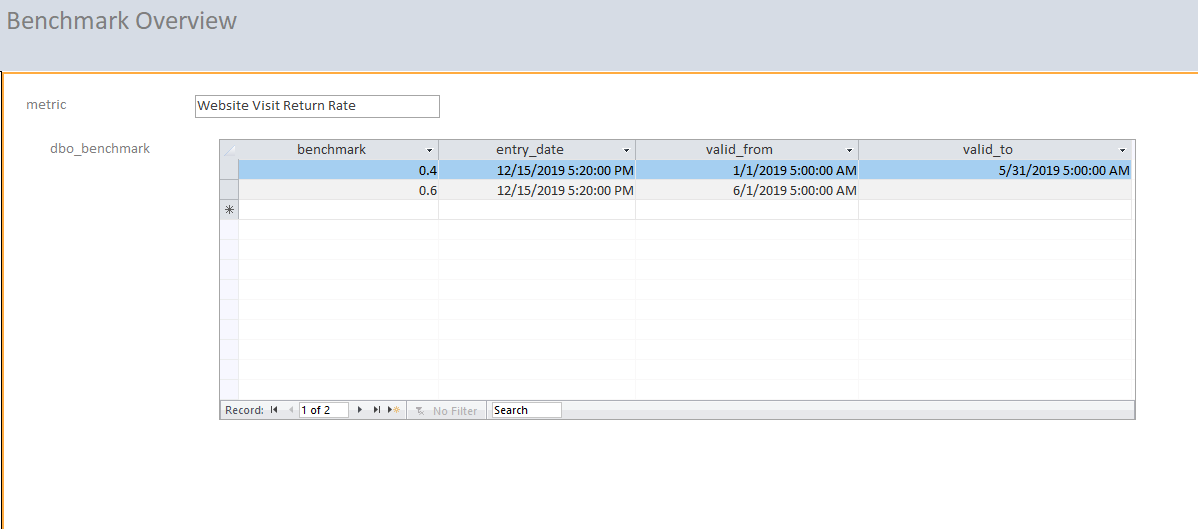
## 

## GUI Prototype

Form 1 - The Metrics Manager. This view allows users to update their assigned metrics, while also having a filter so a user only receives the metrics for which they are responsible.

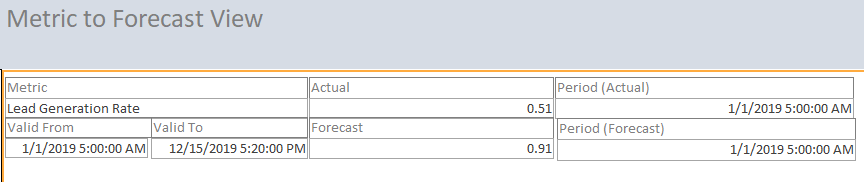


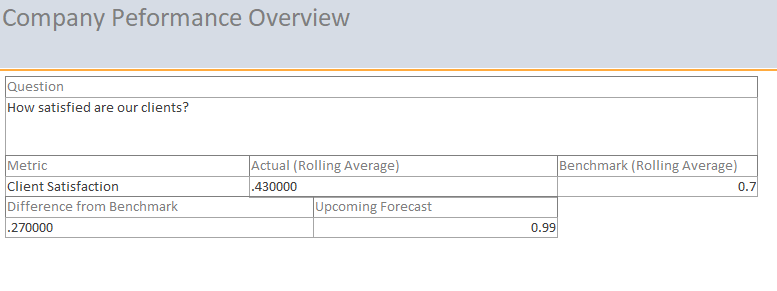
Form 2 - Benchmark Overview. This allows users to update the database with new benchmarks while keeping the former benchmark data.



Form 3 - Metric to Forecast View

This view allows users to directly compare actual values with the forecasts created and also allows them to update accordingly.



Form 4 - Performance Overview - Provides a metric-level overview of the company’s performance respective to the user’s role and question.

## Reflection

I came into this project assuming that database creation and implementation was extremely complex and difficult to abstract and automate. What I’ve discovered is that an excellently laid-out design actually frees you substantially later on to allow your database to be both developer and user-friendly. In my personal work I’ve only ever worked with data warehouses like Amazon Redshift clusters. Knowing that I can set up something more small-scale and begin to solve problems all on my own at a low cost is really quite incredible.

The next time I’m asked to take build a database for a client with no/ill-designed data solutions, I will begin the project by thinking of what it would take to develop a solution that would allow me to hand it off with little-to-no onboarding required. I understand now that it’s possible to build a strong framework that’s easy to maintain, even with simple database design tools. The most important aspect is drafting an intelligent design that adheres to the principles we learned in this class.