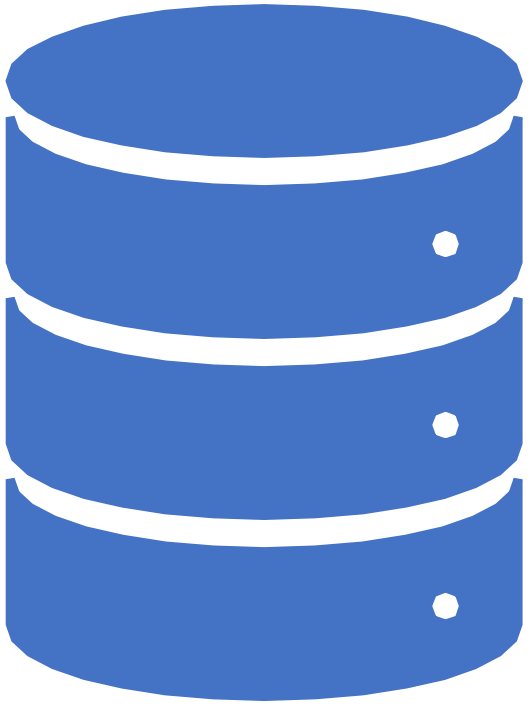
Data Management and Database Design

Role-based access control

Group 20



Presented by,

Ashwin Muthiah Murugappan

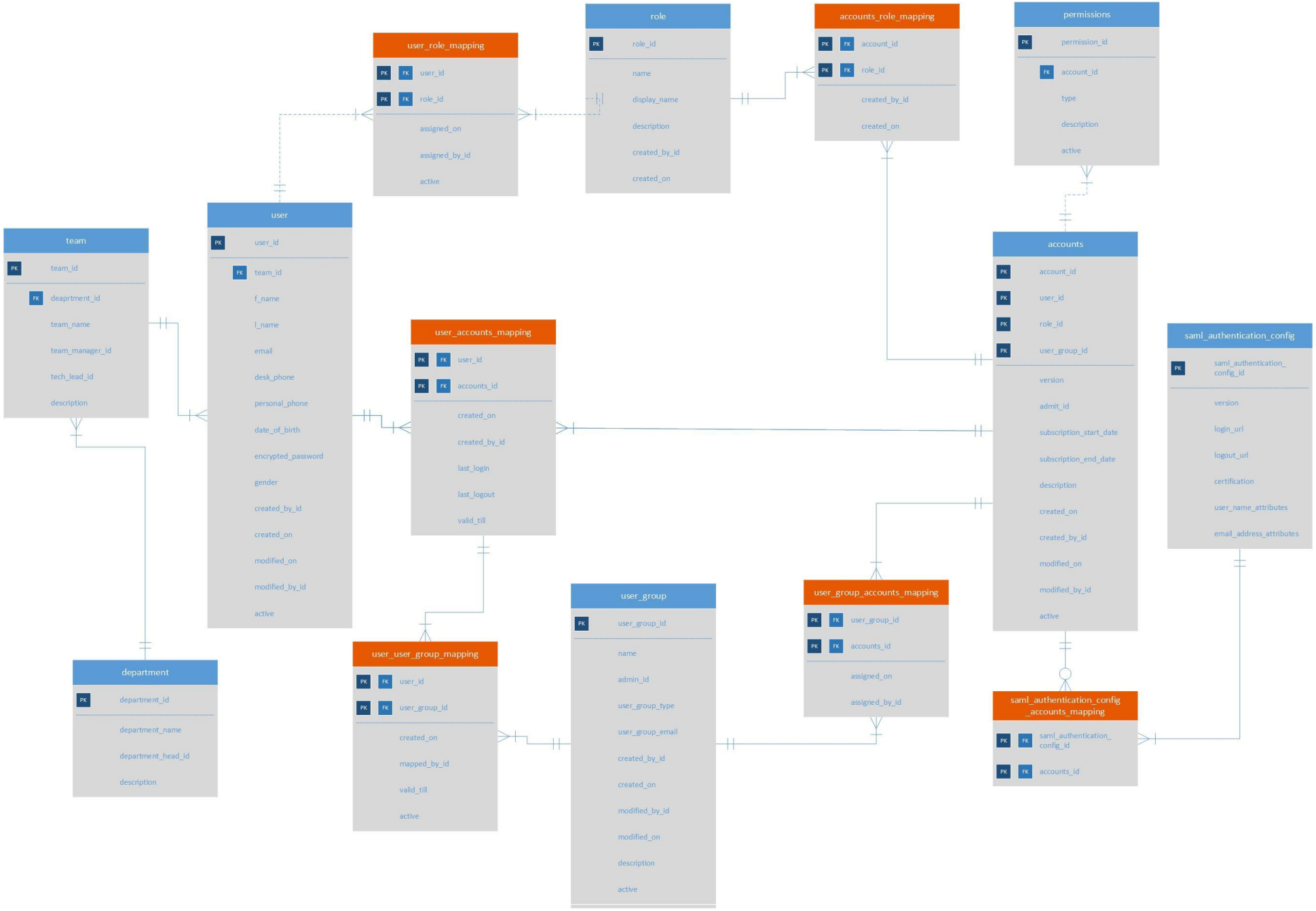
Vijayalakshmi Siddappa

Naga Mahesh Tiruveedhula

Role Based Access Control

- Role-based access control (RBAC) is a method of restricting network access based on the roles of individual users within an enterprise.
- RBAC lets employees have access rights only to the information they need to do their jobs and prevents them from accessing information that doesn't pertain to them.
- In the role-based access control data model, roles are based on several factors, including authorization, responsibility and job competency.
- As such, companies can designate whether a user is an end user, an administrator or a specialist user. In addition, access to computer resources can be limited to specific tasks, such as the ability to view, create or modify files.
- Limiting network access is important for organizations that have many workers, employ contractors or permit access to third parties, like customers and vendors, making it difficult to monitor network access effectively.
- Companies that depend on RBAC are better able to secure their sensitive data and critical applications.
- Benefits - Improves operational efficiency, Enhances compliance, Giving administrators increased visibility, Reduces costs, Decreases risk of breaches and data leakage.

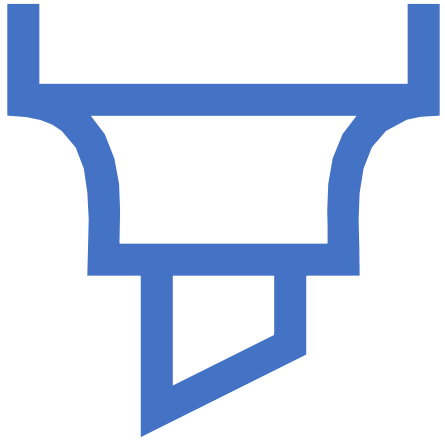
Entity Relationship Diagram



SQL DDL
User - Role - UserRoleMapping

- Let's look at the entities USER , Role and User role mapping .
- The UserRoleMapping entity is an Associative table .
- The Associative Tables are used for Normalizing a many to many relationship between the User and the Role entity.

```
CREATE TABLE dbo.Users (
    UserId INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
    TeamId INT NOT NULL,
    FOREIGN KEY (TeamId) REFERENCES dbo.Team(TeamId),
    FName VARCHAR(30) NOT NULL,
    LName VARCHAR(30) NOT NULL,
    Email VARCHAR(50) NOT NULL,
    DeskPhone INT NOT NULL,
    PersonalPhone INT,
    DateOfBirth DATE,
    EncryptedPassword VARCHAR(225) NOT NULL,
    Gender VARCHAR(30),
    CreatedById INT,
    CreatedOn Date,
    ModifiedOn Date,
    ModifiesById INT,
    Active BIT NOT NULL
);
ALTER TABLE
    dbo.Users ADD CONSTRAINT check_age CHECK (datediff(yy,
    CONVERT(datetime,
    DateOfBirth),
    GETDATE()) < 100);

ALTER TABLE
    dbo.Users ADD CONSTRAINT check_gender CHECK (Gender IN ('M',
    'F',
    'O'));

ALTER TABLE
    dbo.Users ADD CONSTRAINT check_createdate_users CHECK (datediff(dy,
    CONVERT(datetime,
    CreatedOn),
    GETDATE()) >= 0);
```



```
CREATE TABLE dbo.UserRoleMapping(
    UserId INT NOT NULL REFERENCES dbo.Users(UserId),
    RoleID INT NOT NULL REFERENCES dbo.Roles(RoleId),
    AssignedOn Date,
    AssignedById INT,
    Active BIT NOT NULL
    CONSTRAINT PKUserRoleMapping PRIMARY KEY CLUSTERED (UserId,RoleID)
);
ALTER TABLE
dbo.UserRoleMapping ADD CONSTRAINT
check_createdate_UserRoleMapping CHECK
(datediff(dy,CONVERT(datetime,AssignedOn),GETDATE()) >= 0);

ALTER TABLE
dbo.UserRoleMapping ADD CONSTRAINT
check_user_role_mapping_pk CHECK
(dbo.UserRoleMappingCheck(UserId,RoleID) = 1);
```

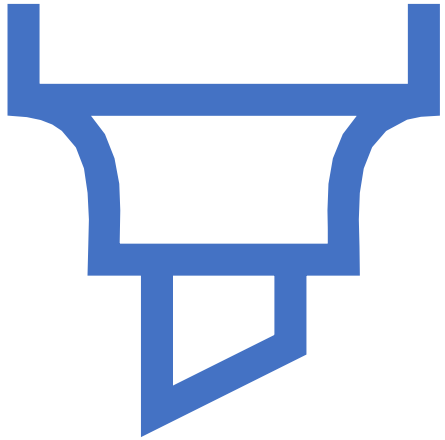
```
CREATE TABLE dbo.Roles(
    RoleId INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
    Name VARCHAR(30) NOT NULL,
    DisplayName VARCHAR(30) NOT NULL,
    CreatedById INT,
    CreatedOn DATE,
    Description VARCHAR(255)
);
ALTER TABLE
    dbo.Roles ADD CONSTRAINT check_createdate_roles
    CHECK (datediff(dy,
    CONVERT(datetime,
    CreatedOn),
    GETDATE()) >= 0);
```

SQL DDL

Role - Accounts - AccountRoleMapping

- Let's look at the entities Role, Accounts and AccountRoleMapping.
- The AccountRoleMapping entity is an Associative table .
- The Associative Tables are used for Normalizing a many to many relationship between the Roles and the Accounts entity.

```
CREATE TABLE
    dbo.Accounts ( AccountId INT NOT NULL IDENTITY UNIQUE,
    UserId INT NOT NULL REFERENCES dbo.Users(UserId),
    RoleId INT NOT NULL REFERENCES dbo.Roles(RoleId),
    UserGroupId INT NOT NULL REFERENCES dbo.UserGroup(UserGroupId),
    Version DECIMAL,
    AdminId INT,
    SubscriptionStartDate DATE,
    SubscriptionEndDate DATE,
    Description VARCHAR(100),
    CreatedOn DATE,
    CreatedById INT,
    ModifiedOn DATE,
    ModifiedById INT,
    Active BIT NOT NULL,
    CONSTRAINT PKACCOUNT PRIMARY KEY CLUSTERED (AccountId,
    UserId,
    RoleId,
    UserGroupId)
);
```



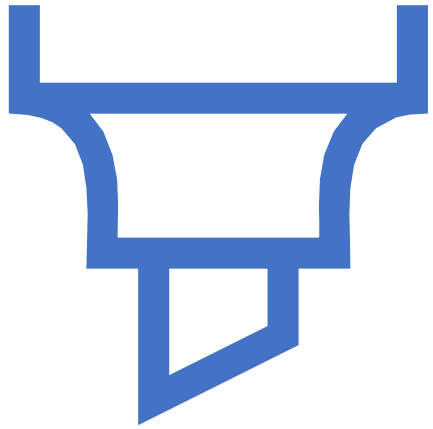
```
CREATE TABLE
    dbo.AccountsRoleMapping( AccountId INT NOT NULL REFERENCES
    dbo.Accounts(AccountId),
    RoleID INT NOT NULL REFERENCES dbo.Roles(RoleId),
    CreatedOn DATE,
    CreatedById INT,
    CONSTRAINT PKAccountsRoleMapping PRIMARY KEY
    CLUSTERED (AccountId,RoleID) );

ALTER TABLE
    dbo.AccountsRoleMapping ADD CONSTRAINT
    check_createdate_AccountsRoleMapping CHECK (datediff(dy,
    CONVERT(datetime,
    CreatedOn),
    GETDATE()) >= 0);
```

```
CREATE TABLE dbo.Roles(
    RoleId INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
    Name VARCHAR(30) NOT NULL,
    DisplayName VARCHAR(30) NOT NULL,
    CreatedById INT,
    CreatedOn DATE,
    Description VARCHAR(255)
);
ALTER TABLE
    dbo.Roles ADD CONSTRAINT check_createdate_roles
    CHECK (datediff(dy,
    CONVERT(datetime,
    CreatedOn),
    GETDATE()) >= 0);
```

SQL DDL
Password Encryption - Stored procedures

- Stored Procedures were used for inserting data into User Entity with Password Encryption .
- HASHBYTES (Transact-SQL) was used to convert the Actual Password into an Encrypted form .

```
CREATE PROCEDURE dbo.AddUsers @pTeamId INT,
@pFName VARCHAR(30),@pLName VARCHAR(30),
@pEmail VARCHAR(50) ,@pDeskPhone INT,
@pPersonalPhone INT,@pDateOfBirth DATE,
@pEncryptedPassword VARCHAR(225),
@pGender VARCHAR(30),@pCreatedById INT,
@pCreatedOn DATE,@pModifiedOn DATE,
@pModifiesById INT,@pActive BIT AS
BEGIN
SET
NOCOUNT ON
INSERT
    INTO
        dbo.Users(TeamId,FName,LName ,Email ,
        DeskPhone,PersonalPhone,DateOfBirth ,
        EncryptedPassword ,Gender,
        CreatedById ,CreatedOn ,ModifiedOn ,
        ModifiesById ,Active )
    VALUES (@pTeamId,
    @pFName,@pLName ,@pEmail ,@pDeskPhone,
    @pPersonalPhone,@pDateOfBirth ,HASHBYTES('SHA2_512',
    @pEncryptedPassword) ,@pGender,@pCreatedById ,
    @pCreatedOn ,@pModifiedOn ,@pModifiesById ,
    @pActive)
END
```

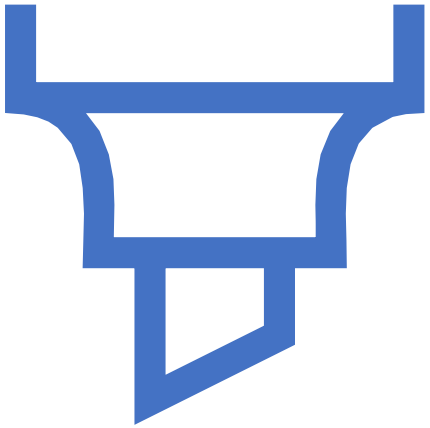


```
DECLARE @pTeamId INT;SET @pTeamId = 4;
DECLARE @pFName VARCHAR(30);SET @pFName = 'obama';
DECLARE @pLName VARCHAR(30);SET @pLName =  'brack';
DECLARE @pEmail VARCHAR(50) ;SET    @pEmail = 'ob@gmail.com';
DECLARE @pDeskPhone INT;SET @pDeskPhone = 1234457898;
DECLARE @pPersonalPhone INT;SET @pPersonalPhone = 1237;
DECLARE @pDateOfBirth DATE;SET  @pDateOfBirth = CONVERT(datetime,'08/25/1997');
DECLARE @pEncryptedPassword VARCHAR(225);SET    @pEncryptedPassword = 'duckky';
DECLARE @pGender VARCHAR(30);SET    @pGender ='M';
DECLARE @pCreatedById INT;SET   @pCreatedById = 1;
DECLARE @pCreatedOn Date;SET    @pCreatedOn = CONVERT(datetime,'07/26/2019');
DECLARE @pModifiedOn Date;SET   @pModifiedOn = CONVERT(datetime,'07/26/2019');
DECLARE @pModifiesById INT;SET  @pModifiesById = 1;
DECLARE @pActive BIT ;SET   @pActive = 1;
EXEC dbo.AddUsers @pTeamId,@pFName,@pLName ,@pEmail ,@pDeskPhone,@pPersonalPhone,
@pDateOfBirth ,@pEncryptedPassword,@pGender,@pCreatedById ,
@pCreatedOn ,@pModifiedOn ,@pModifiesById ,@pActive;
```

SQL DDL
Function - Check Constrains

- A user-defined function (UDF) in SQL Server is a programming construct that accepts parameters, does work that typically makes use of the accepted parameters, and returns a type of result.
- Here, the function Accepts the Role Id and User Id and checks if they are present in the User and Role tables Respectively.
- If present the function returns or else it returns 1.

```
CREATE FUNCTION UserRoleMappingCheck(@UserId INT,
@RoleId INT) RETURNS BIT AS
BEGIN
DECLARE @UserCount SMALLINT = 0 SELECT
    @UserCount = COUNT(1)
FROM
    dbo.Users
WHERE
    UserId = @UserId;

DECLARE @RoleCount SMALLINT = 0 SELECT
    @RoleCount = COUNT(1)
FROM
    dbo.Roles
WHERE
    RoleId = @RoleId;

RETURN (
CASE
    WHEN @UserCount > 0
    AND @RoleCount > 0 THEN 1
    ELSE 0
END )
END;
```

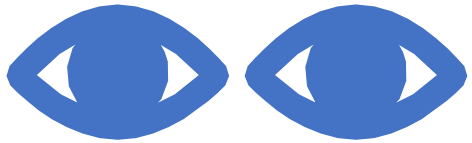


```
CREATE TABLE dbo.UserRoleMapping(
    UserId INT NOT NULL REFERENCES dbo.Users(UserId),
    RoleID INT NOT NULL REFERENCES dbo.Roles(RoleId),
    AssignedOn Date,
    AssignedById INT,
    Active BIT NOT NULL
    CONSTRAINT PKUserRoleMapping PRIMARY KEY CLUSTERED (UserId,RoleID)
);
ALTER TABLE
dbo.UserRoleMapping ADD CONSTRAINT
check_createdate_UserRoleMapping CHECK
(datediff(dy,CONVERT(datetime,AssignedOn),GETDATE()) >= 0);

ALTER TABLE
dbo.UserRoleMapping ADD CONSTRAINT
check_user_role_mapping_pk CHECK
(dbo.UserRoleMappingCheck(UserId,RoleID) = 1);
```

Views

- In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table.
- The fields in a view are fields from one or more real tables in the database.
- This View on the right is used to report the Number of accounts per user.

	UserId	NumberOfAccounts
1	1	1
2	2	2
3	3	0
4	4	0
5	5	1
6	6	0
7	7	3
8	8	0
9	9	0
10	10	4
11	11	1
12	12	0
13	13	0
14	14	1
15	15	0
16	16	1
17	17	0
18	18	3
19	19	2
20	20	2

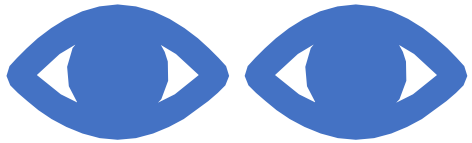


```
CREATE VIEW V_NumAccountsPerUser AS SELECT
    A.UserId,
    COUNT(B.AccountId) AS NumberOfAccounts
FROM
    dbo.Users A
LEFT JOIN dbo.Accounts B ON
    A.UserId = B.UserId
GROUP BY
    A.UserId;

SELECT
    *
FROM
    V_NumAccountsPerUser;
```

Views

- This view takes in a particular user and a particular account and specifies all the permissions for that user for that account.
- For this example we have used the UserId = 3 and AcccountId = 3.

```sql
CREATE VIEW V_UserAccountPermissions AS SELECT
    A.UserId,
    A.AccountId,
    P.PermissionType,
    P.Description,
    (CASE
        WHEN P.Active = 0 THEN 'Inactive'
        ELSE 'Active'
    END) AS Permission_status
FROM
    Accounts A
INNER JOIN Permissions P ON
    P.AccountId = A.AccountId;

SELECT
    *
FROM
    V_UserAccountPermissions
WHERE
    UserId = 3
    AND AccountId = 3;
```

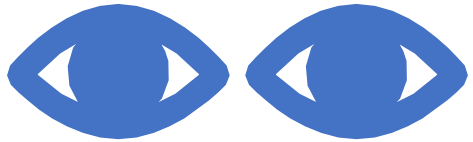


	UserId	AccountId	PermissionType	Description	Permission_status
1	2	1	write	write only access	Active
2	2	2	read	read only access	Inactive
3	7	3	read	read only access	Active
4	1	4	write	write only access	Active
5	5	5	all	full permission	Active
6	10	6	read	read only access	Inactive
7	7	7	all	full permission	Active
8	7	8	read	read only access	Active
9	18	9	write	write only access	Active
10	10	10	write	write only access	Inactive
11	11	11	read	read only access	Active
12	10	12	read	read only access	Active
13	10	13	all	full permission	Inactive
14	14	14	read	read only access	Active
15	18	15	read	read only access	Active
16	16	16	all	full permission	Inactive
17	19	17	write	write only access	Inactive
18	18	18	read	read only access	Active
19	19	19	all	full permission	Active
20	20	20	read	read only access	Inactive
21	7	3	modify	modify only access	Inactive
22	7	3	execute	execute only access	Active
23	7	3	execute	execute only access	Active
24	5	5	execute	execute only access	Inactive
25	5	5	execute	execute only access	Inactive
26	5	5	modify	modify only access	Active
27	5	5	execute	execute only access	Active
28	5	5	modify	modify only access	Inactive

Views

- This view gives the number of male and female employees for each role.
- We can see that ratio of female is higher in the sales related role.

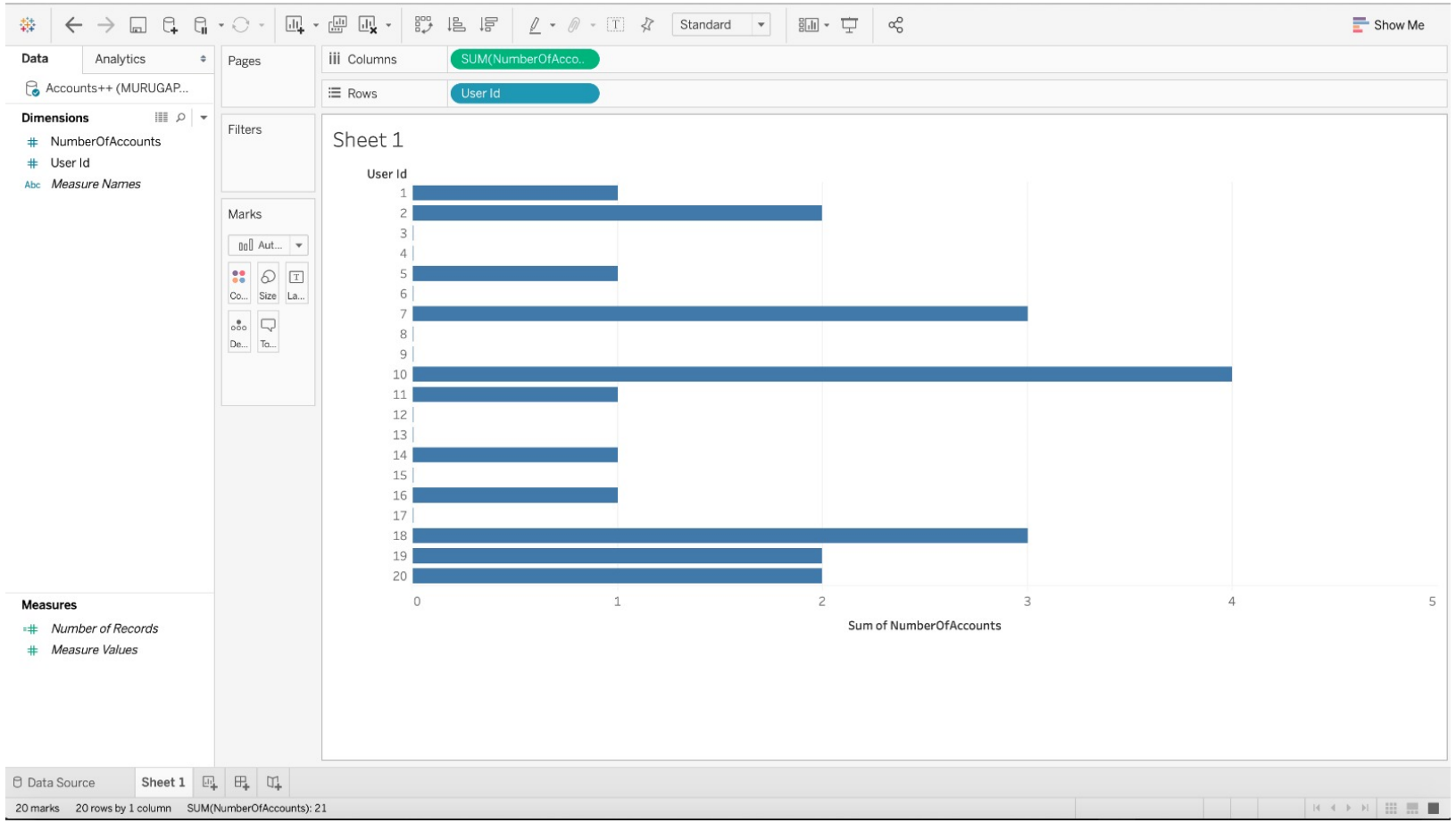
```
CREATE VIEW V_GenderRatioByRole AS WITH temp AS (
    SELECT C.RoleId AS RoleId,
    C.Name AS RoleName,
    A.UserId AS UserId,
    A.Gender AS Gender
FROM
    dbo.Users A
JOIN dbo.UserRoleMapping B ON
    A.UserId = B.UserId
JOIN dbo.Roles C ON
    B.RoleId = C.RoleId ) SELECT
    RoleName,
    COUNT(UserId) AS Employees,
    COUNT( CASE WHEN Gender = 'M' THEN 1 END ) AS Male,
    COUNT( CASE WHEN Gender = 'F' THEN 1 END ) AS Female
FROM
    temp
GROUP BY
    RoleName;
```



	RoleName	Employees	Male	Female
1	Accountant	2	1	1
2	BusinessIntelligence	2	1	1
3	DataAnalyst	2	1	1
4	HumanResources	3	3	0
5	OnlinePayments	4	3	1
6	SalesPerson	3	1	2
7	SoftwareEngineer	2	2	0
8	SupplyChainEngineer	3	2	1

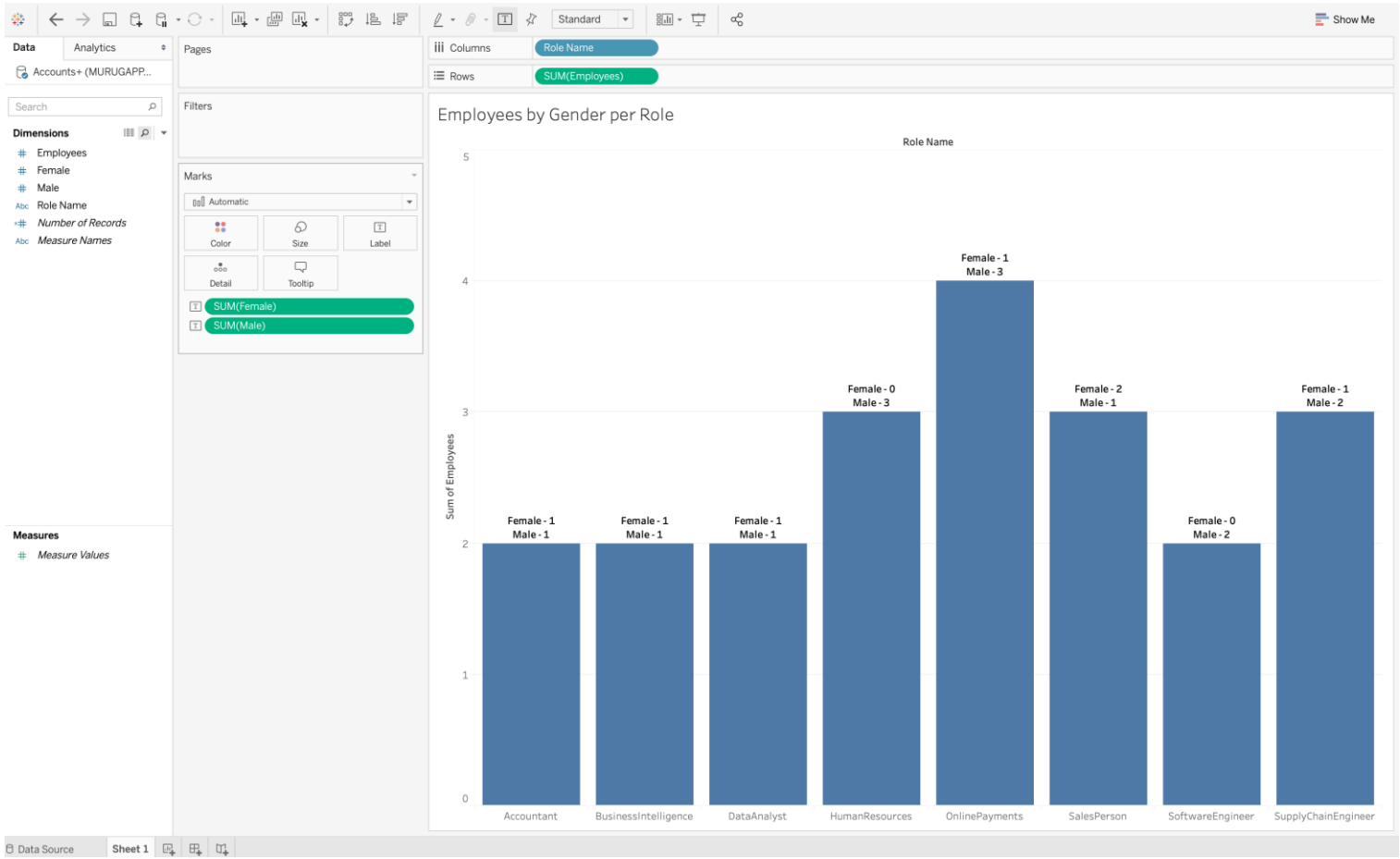
Reports and Visualization

- Report shows number of accounts each user has.
- Sum of NumberOfAccounts is on the X-axis
- UserId is on the Y-Axis



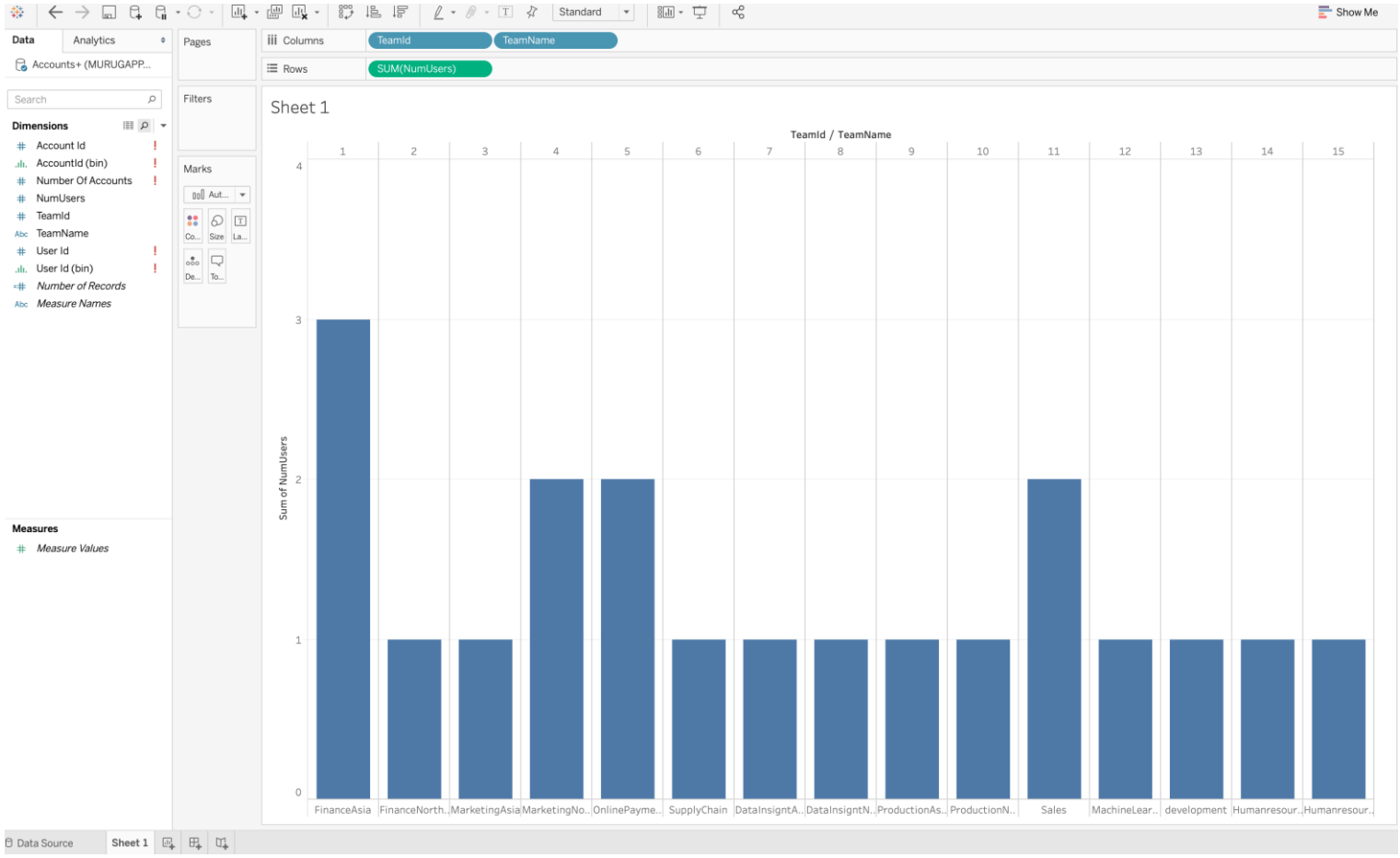
Reports and Visualization

- Report primarily shows number of employees for each role.
- RoleName on the x-axis
- SumOfEmployees on the Y-axis
- Additionally, we can also get the count of male and female employees in each role.



Reports and Visualization

- Report shows number of users/employees each team has.
- TeamId/Team Name on the x-axis.
- Sum of NumUsers on the Y-Axis.



Any questions ?

Thank you