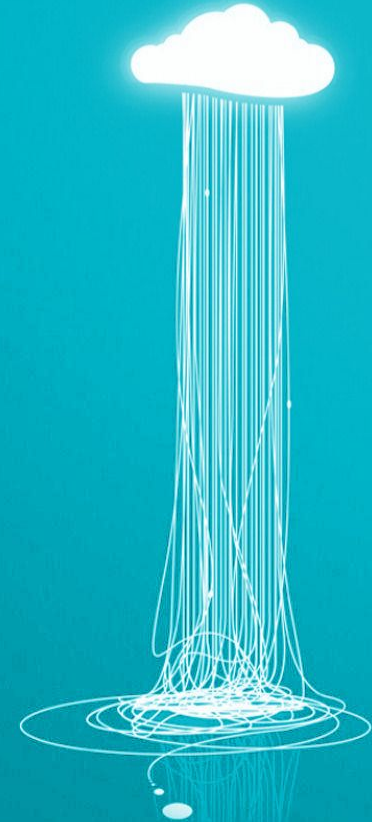


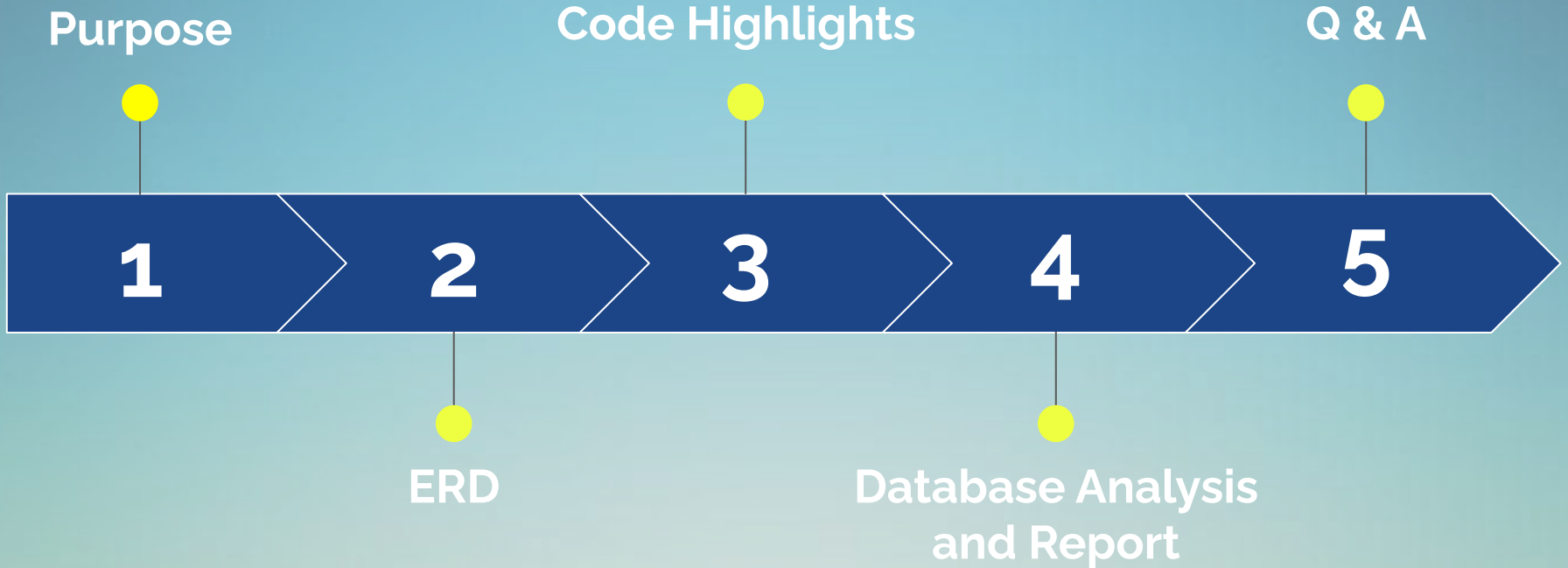
Database for a Cloud Computing Platform



Presented by Team 01

Kirtikumar Waykos
Kruthi Nymisha Kona
Ramya Hebbar
Vishal Baliga

Agenda



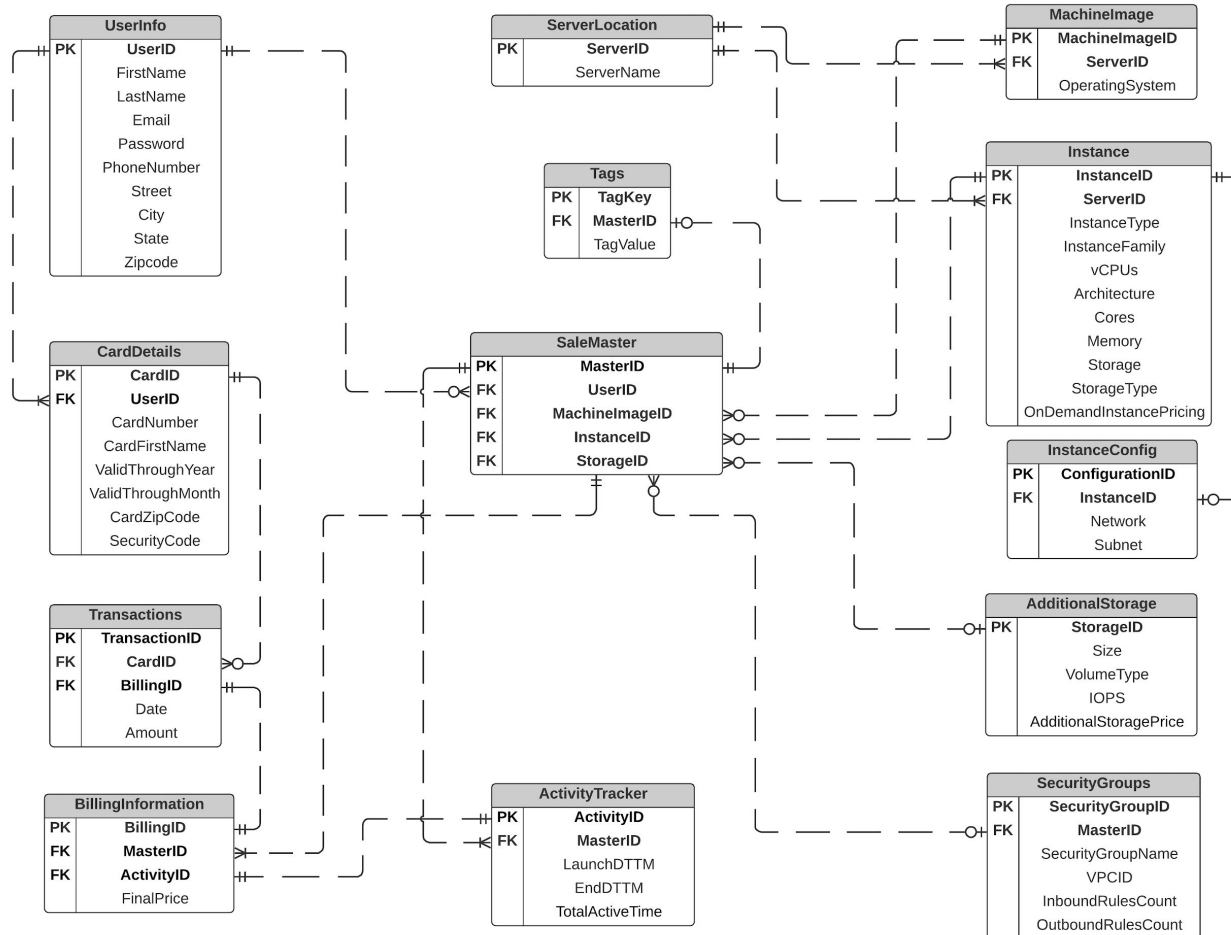
Database Purpose

Delivery model of the business - PaaS

- Monitor and maintain the cloud resources of the company
- Allows customers to configure their virtual machines according to their requirements - with additional costs
- Provides customers the option of saving and retrieving previously used virtual machines
- Offer Encryption and safe storage of customer's Card Details and Password
- Automation of the billing process with instant transactions
- Allow users and the company to track and analyze resource usage

ERD FOR CLOUD COMPUTING PLATFORM

Team 01



User Details

Infrastructure

Billing and Transactions

Code Highlights

```
CREATE TABLE dbo.ServerLocation ( ServerID VARCHAR(4) NOT NULL PRIMARY KEY, ServerLocation VARCHAR(50) );
```

```
CREATE PROCEDURE InsertServerData @ServerNumber int,  
@ServerLocation VARCHAR(50) AS BEGIN DECLARE @ServerID VARCHAR(4);
```

```
SELECT  
    @ServerID = 'S' + RIGHT('00' + CAST (@ServerNumber AS VARCHAR(3)), 3)  
INSERT  
    INTO  
        ServerLocation  
VALUES (@ServerID, @ServerLocation);  
END
```

```
EXEC InsertServerData 1, 'US-East (N. Virginia)';  
EXEC InsertServerData 2, 'US-East (Ohio)';  
EXEC InsertServerData 3, 'US-West (N. California)';  
EXEC InsertServerData 4, 'US-West (Oregon)';  
EXEC InsertServerData 5, 'Asia Pacific (Mumbai)';  
EXEC InsertServerData 6, 'Europe (Stockholm)';  
EXEC InsertServerData 7, 'Europe (Milan)';  
EXEC InsertServerData 8, 'Europe (Frankfurt)';  
EXEC InsertServerData 9, 'Europe (Ireland)';  
EXEC InsertServerData 10, 'Europe (Paris)';
```

ABC ServerID	ABC ServerLocation
S001	US-East (N. Virginia)
S002	US-East (Ohio)
S003	US-West (N. California)
S004	US-West (Oregon)
S005	Asia Pacific (Mumbai)
S006	Europe (Stockholm)
S007	Europe (Milan)
S008	Europe (Frankfurt)
S009	Europe (Ireland)
S010	Europe (Paris)

- Use of stored procedure to populate PK with prefix for ServerLocation entity
- Similar procedures created for MachineImage and SecurityGroups entities

Data Transfer

Input file(s)
Configure input files or directories

Input files

Source	Target
Instance.csv	CloudComputingDatabase_Team01.dbo.l...

Importer settings

Name	Value
Extension	csv,tsv,txt
Encoding	utf-8
Column delimiter	,
Header position	top
Quote char	"

< Back Next > Start Cancel

- Instances entity populated using SQL Data Import Wizard

	123 InstanceID	ABC InstanceType	ABC InstanceFamily	ABC ServerID	123 vCPUs	ABC Architecture	123 Cores
1	1	2xlarge	a1	S001	8	arm64	8
2	2	4xlarge	a1	S001	16	arm64	16
3	3	large	a1	S001	2	arm64	2
4	4	medium	a1	S001	1	arm64	1


```
CREATE TABLE CardDetails (CardID INT IDENTITY NOT NULL PRIMARY KEY,
UserID int references UserInfo(UserID),
CardNumber VARBINARY(500),
CardFirstName VARCHAR(30),
ValidThroughMonth TINYINT check(ValidThroughMonth>=1 and ValidThroughMonth<=12),
ValidThroughYear nvarchar(4) check(ValidThroughYear>=year(CURRENT_TIMESTAMP)),
CardZipCode varchar(7),
SecurityCode int,
);
```

- Check constraints to ensure no invalid entries

- Encryption using symmetric key

```
---Creating a certificate for Encryption
CREATE CERTIFICATE CardEncryptCertificate
ENCRYPTION BY PASSWORD = '0123456789'
WITH SUBJECT = 'Encrypting Sensitive Card Info',
EXPIRY_DATE = '20211031';
```

```
---Creating a SymmetricKey
CREATE SYMMETRIC KEY CCEncrypt
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE CardEncryptCertificate ;
```

```
---Opening a SymmetricKey
OPEN SYMMETRIC KEY CCEncrypt
DECRYPTION BY CERTIFICATE CardEncryptCertificate
WITH PASSWORD='0123456789';
```

```
INSERT into CardDetails values(1001,EncryptByKey(Key_GUID(N'CCEncrypt'),
convert(varbinary, '1234-2345-3456-4567')), 'William',
06, '2025', '07125', 982);
```

	123 CardID	123 UserID	CardNumber
1	22	1,001	& v 'TJ _je Ek~ h É Ô ;O pZ"... [68]

---CREATE TRIGGER TO POPULATE BillingInformation AFTER ActivityTracker IS POPULATED

CREATE TRIGGER FinalPrice **ON**

ActivityTracker

AFTER

INSERT AS

BEGIN

SET

NOCOUNT **ON**;

DECLARE @finalPrice **Float**;

DECLARE @TotatlActivityTime **Float**;

DECLARE @InstancePrice **FLOAT**;

DECLARE @ActivityID **INT**;

DECLARE @MasterID **INT**;

DECLARE @AdditionalStorage **float**;

SELECT

@TotatlActivityTime = at2.TotalActiveTime ,

@InstancePrice = i.OnDemandInstancePricing,

@AdditionalStorage =ad.AdditionalStoragePrice

FROM ActivityTracker at2

LEFT JOIN SaleMaster sm **On**

at2.MasterId = sm.MasterID

LEFT JOIN AdditionalStorage ad **On**

sm.StorageID = ad.StorageID

LEFT JOIN Instances i **On**

sm.InstanceID = i.InstanceID ;

SELECT

@MasterId = ActivityTracker.MasterId

@ActivityID = ActivityTracker.ActivityID

FROM ActivityTracker;

INSERT INTO BillingInformation (MasterID, ActivityID, FinalPrice)

VALUES (@MasterId , @ActivityID , (((@TotatlActivityTime / 60) * @InstancePrice)+@AdditionalStorage));

END

- Trigger created on ActivityTracker
- Fired after each successful user activity
- Final price computed and stored


```

---TRIGGER TO POPULATE Transactions
CREATE TRIGGER PopulateTransaction
on BillingInformation
AFTER
INSERT AS
BEGIN
    SET
        NOCOUNT ON;
    DECLARE @TransactionID INT;
    DECLARE @BillingID INT;
    DECLARE @CardID INT;
    DECLARE @Date1 DATE;
    DECLARE @FinalPrice FLOAT

    SELECT @CardID = cd.CardID
    FROM BillingInformation bi
    LEFT JOIN SaleMaster sm ON bi.MasterID = sm.MasterID
    LEFT JOIN UserInfo ui ON sm.UserID = ui.UserID
    LEFT JOIN CardDetails cd ON ui.UserID = cd.UserID ;

    SELECT @Date1 = Actr.EndDTTM
    FROM BillingInformation bi
    LEFT JOIN ActivityTracker Actr
    ON bi.MasterID = Actr.MasterID
    AND bi.ActivityID = Actr.ActivityID ;

    SELECT
        @BillingID = bi.BillingID,
        @FinalPrice = bi.FinalPrice
    FROM
        BillingInformation bi
    ;

    INSERT INTO Transactions ( BillingID, CardID, [Date], Amount )
    VALUES ( @BillingID , @CardID , @Date1, dbo.calculateBillingPrice(@FinalPrice));

END

```

```









--- CREATING A FUNCTION TO CALCULATE BILLING PRICE(FINAL PRICE+TAX)
CREATE FUNCTION dbo.calculateBillingPrice(
@FinalPrice float
)
returns float
as begin
    DECLARE @Taxes float;
    set @Taxes=0.08;
    return @FinalPrice+@FinalPrice*@Taxes
end;

```

- Trigger created on BillingInformation
- Invoke a function to calculate the final prices with taxes

Database Reports and Analysis

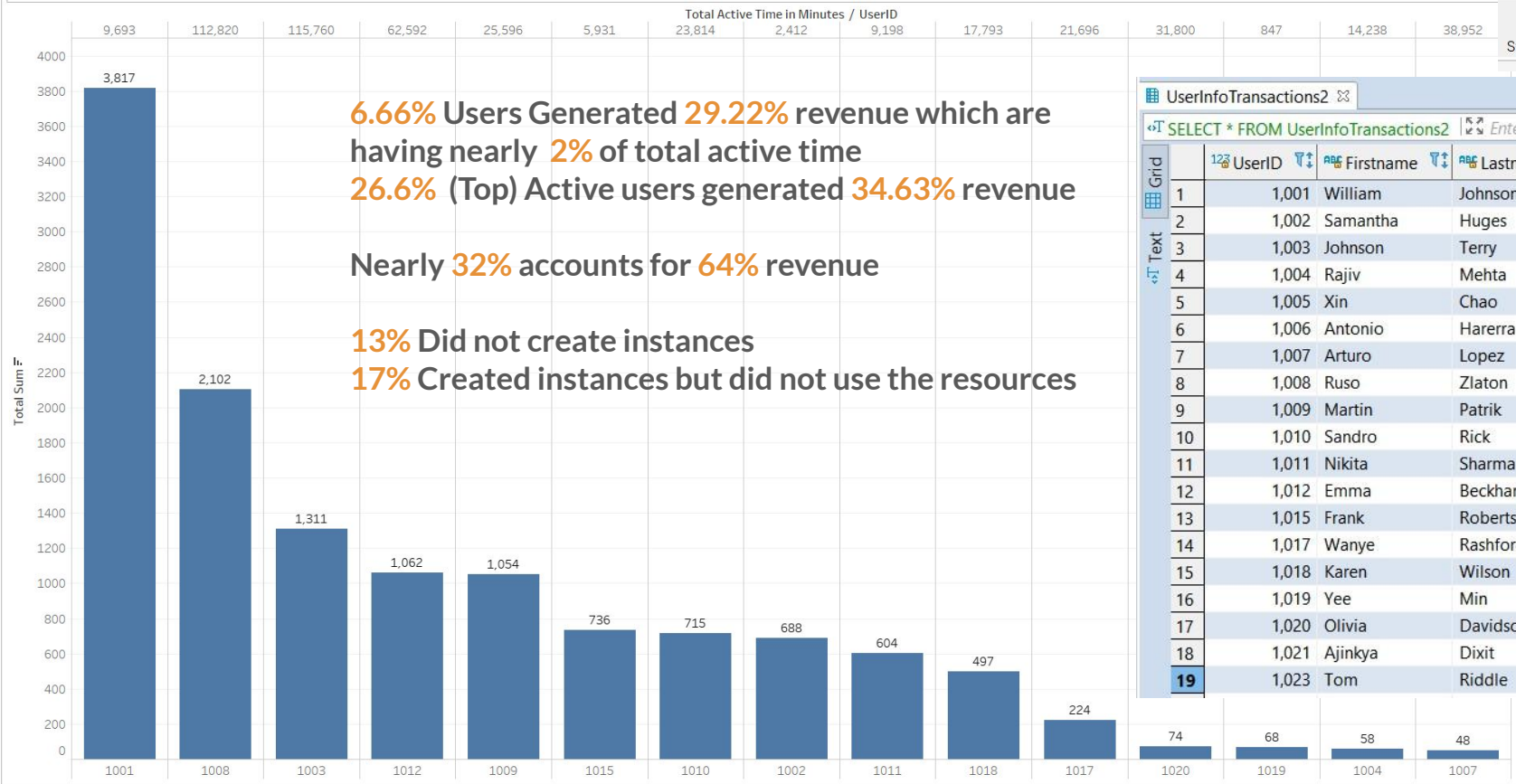
Horizontal report showing subscribed image ID per user:

	 UserID 	 FirstName 	 LastName 	 Subscribed Image ID 
1	1,001	William	Johnson	200, 210, 225, 234, 241, 248, 257, 264
2	1,002	Samantha	Huges	201, 211, 228, 251
3	1,003	Johnson	Terry	202, 212
4	1,004	Rajiv	Mehta	203, 213
5	1,005	Xin	Chao	204, 214, 227, 235, 250, 258
6	1,006	Antonio	Harerra	205
7	1,007	Arturo	Lopez	206, 216, 240, 263
8	1,008	Ruso	Zlaton	207, 217
9	1,009	Martin	Patrik	208
10	1,010	Sandro	Rick	209, 229, 252
11	1,011	Nikita	Sharma	218, 226, 249
12	1,012	Emma	Beckham	219, 223, 233, 239, 242, 246, 256, 262, 265, 266
13	1,015	Frank	Robertson	215
14	1,017	Wanye	Rashford	260
15	1,018	Karen	Wilson	245, 261

Revenue Generated per user and Total Active Time

Count:	15
SUM(Total Active Time in Minutes)	
Sum:	493,142
Average:	32,876.13
Standard deviation:	36,640
SUM(Total Sum)	
Sum:	13,059
Average:	871
Standard deviation:	992

Revenue generated per User



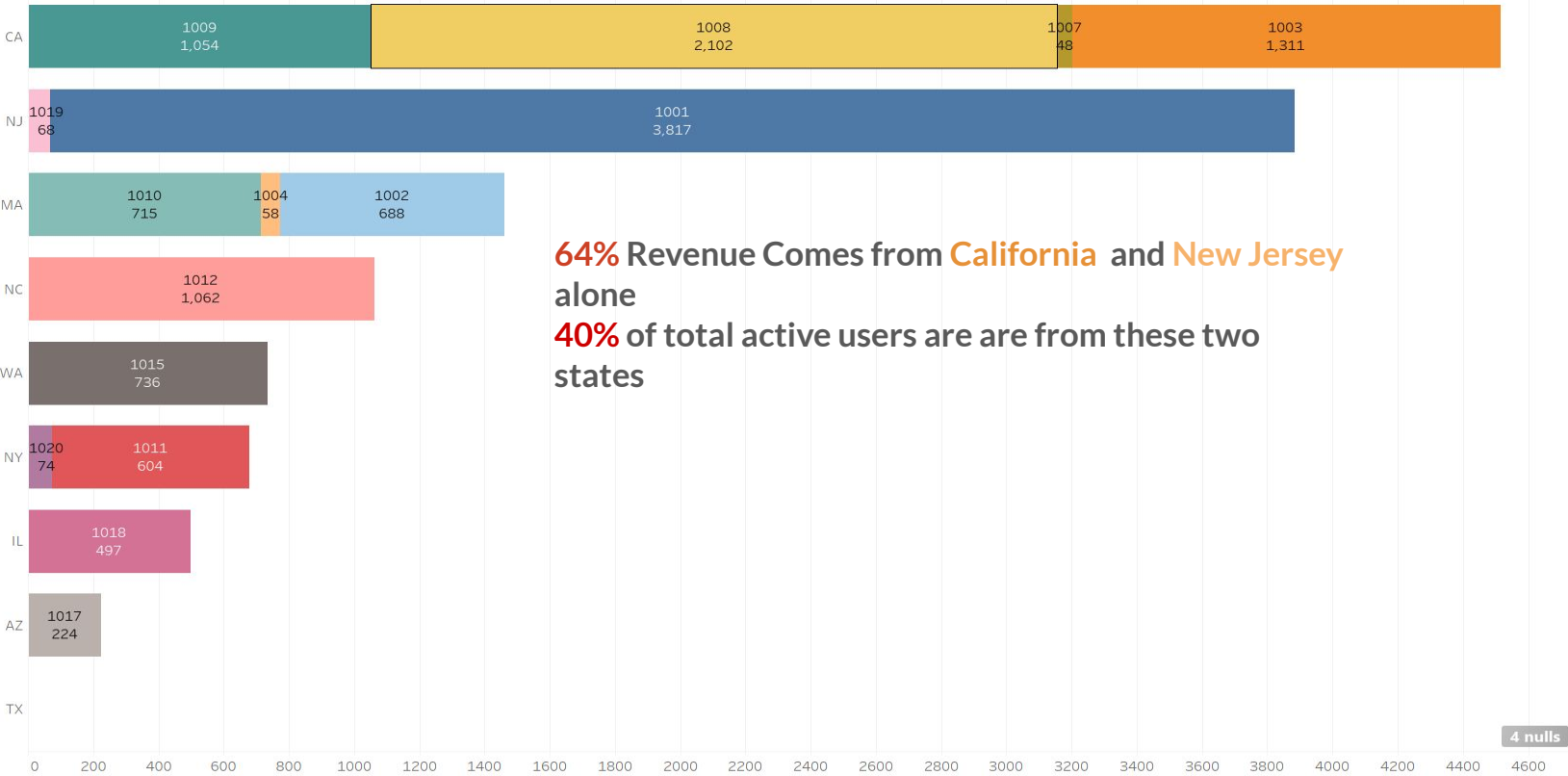
6.66% Users Generated 29.22% revenue which are having nearly 2% of total active time
26.6% (Top) Active users generated 34.63% revenue
Nearly 32% accounts for 64% revenue
13% Did not create instances
17% Created instances but did not use the resources

UserInfoTransactions2				
SELECT * FROM UserInfoTransactions2				
Grid	UserID	Firstname	Lastname	TotalSum
1	1,001	William	Johnson	3,816.822384
2	1,002	Samantha	Huges	688.345776
3	1,003	Johnson	Terry	1,310.890392
4	1,004	Rajiv	Mehta	57.915648
5	1,005	Xin	Chao	[NULL]
6	1,006	Antonio	Harerra	[NULL]
7	1,007	Arturo	Lopez	48.03516
8	1,008	Ruso	Zlaton	2,101.958208
9	1,009	Martin	Patrik	1,053.549504
10	1,010	Sandro	Rick	715.345344
11	1,011	Nikita	Sharma	604.161504
12	1,012	Emma	Beckham	1,062.494064
13	1,015	Frank	Robertson	735.892128
14	1,017	Wanye	Rashford	224.2116
15	1,018	Karen	Wilson	497.384064
16	1,019	Yee	Min	68.369616
17	1,020	Olivia	Davidson	74.0556
18	1,021	Ajinkya	Dixit	[NULL]
19	1,023	Tom	Riddle	[NULL]

Revenue generated by State and User Distribution

Statewise Revenue Distribution

State =



64% Revenue Comes from California and New Jersey alone
40% of total active users are from these two states

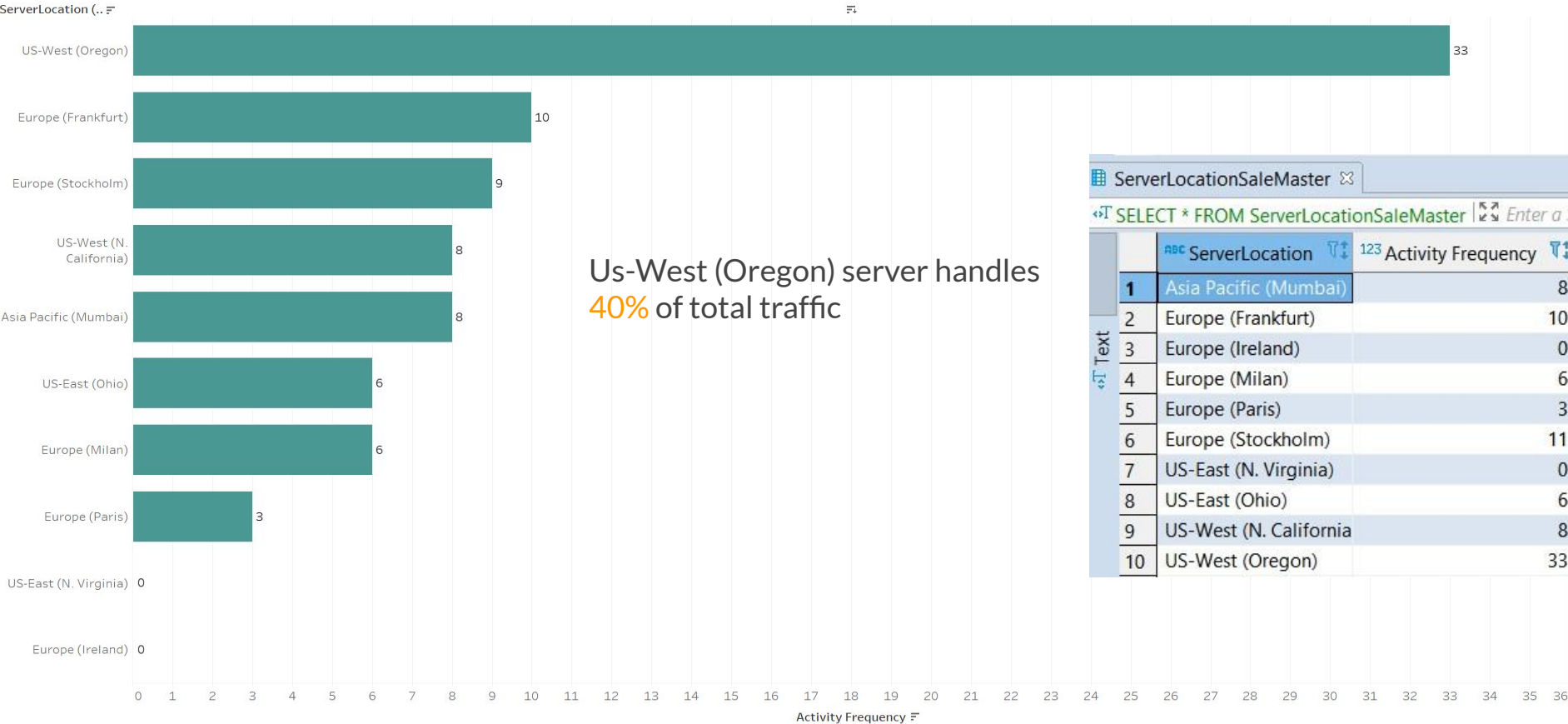
UserID
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1015
1017
1018
1019
1020
1021
1023

4 nulls

Total Sum ₹

Activity Frequency per Server

Activity Frequency Per Server



ServerLocationSaleMaster

SELECT * FROM ServerLocationSaleMaster

	ServerLocation	Activity Frequency
1	Asia Pacific (Mumbai)	8
2	Europe (Frankfurt)	10
3	Europe (Ireland)	0
4	Europe (Milan)	6
5	Europe (Paris)	3
6	Europe (Stockholm)	11
7	US-East (N. Virginia)	0
8	US-East (Ohio)	6
9	US-West (N. California)	8
10	US-West (Oregon)	33

Thank You!