

# **FINAL PROJECT REPORT**

**ISM 6208 – Data Warehousing**

**Submitted by Group Alpha**

Uma Srikanth Reddy Koduru	U94125452
Sindhura Alla	U06443828
Medha Alla	U16177536
Krishna Sai Chaluvadi	U48971923
Vashishta	

**Major in**

**BUSINESS ANALYTICS AND INFORMATION SYSTEMS**

**Under the guidance of**

**DR. DONALD BERNDT**



**MUMA COLLEGE OF BUSINESS  
UNIVERSITY OF SOUTH FLORIDA**

## Table of Contents

<b>Introduction .....</b>	<b>4</b>
<b>Part 1: Executive Summary.....</b>	<b>4</b>
<b>Part 2: Problem Statement.....</b>	<b>4</b>
<b>Part 3: Data Collection and Preparation.....</b>	<b>5</b>
Section 3.1 Exploratory Data Analysis (EDA) .....	5
<b>Part 4: Datawarehouse Dimensional Modelling.....</b>	<b>6</b>
Section 4.1 Requirement gathering.....	6
Section 4.2 Grain Identification.....	6
Section 4.3 Dimensions and Facts.....	6
<b>Part 5: Data Cube VS OLTP Design.....</b>	<b>8</b>
Section 5.1 Star Schema.....	8
Section 5.2 Comparison with OLTP Design .....	8
Section 5.3 Integrity Constraint .....	9
<b>Part 6: Analytics.....</b>	<b>13</b>
Section 6.1 Online Analytical Processing.....	13

<b>Part 7: Performance Tuning.....</b>	<b>19</b>
Section 7.1 Aggregation.....	19
Section 7.2 Indexing.....	21
Section 7.3 Partitioning.....	23
<b>Part 8: Other Topics .....</b>	<b>26</b>
Data Visualization.....	26
<b>References .....</b>	<b>30</b>

## **Introduction**

The dataset used in this project is from the U.S. Small Business Administration (SBA) and It consists of historical data from 1987 to 2014. It has around 7 million rows and 28 columns. In the Data Warehousing course (ISM 6208) a final project was done under Dr. Berndt with this dataset. But the scope and the approach for this project are completely different. In this project an OLTP system has been designed and implemented as a Data Warehouse.

## **Part 1: Executive Summary**

As the data is huge, it becomes appropriate to use big data techniques. So, the data has been cleaned using Apache Spark (PySpark) and loaded into Oracle SQL Developer for constructing the Data Warehouse. Various analytical queries have been written for analyzing the data using the OLTP Cube design (Star schema). The data is further migrated to Tableau Desktop and the same star schema is built for visualizing the data.

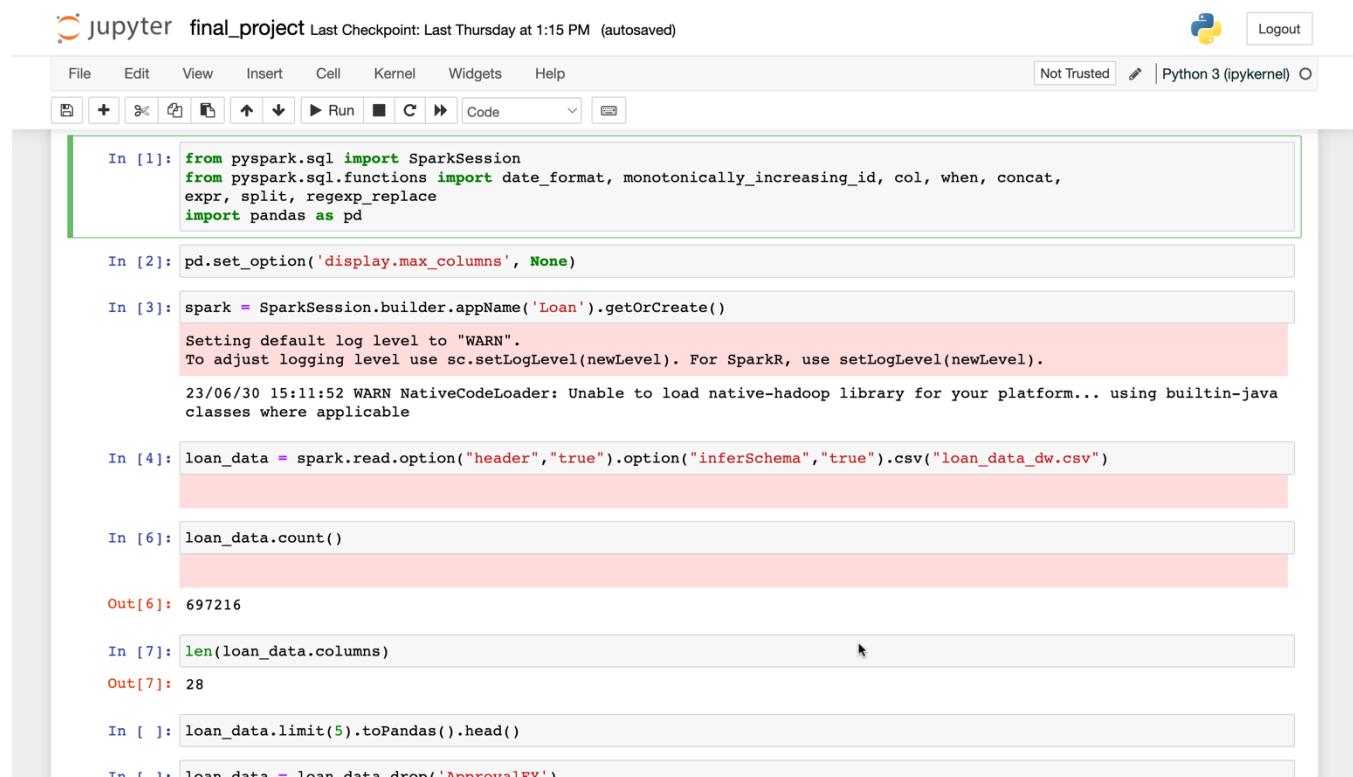
## **Part 2: Problem Statement**

- What are the top and outstanding loans?
- What is the industry wise employee generation?
- Is there any correlation between employment generated and retained based on industries and years?

## Part 3: Data Collection and Preparation

### Section 3.1 Exploratory Data Analysis (EDA)

As the data has millions of records it became obvious to use big data tech like Apache Spark instead of using a pandas data frame for cleaning the data. First the data is loaded into a spark object and all the existing columns are analyzed. It is observed that there are many columns which have 0 and 1 as code for yes or no and approved or not approved. So, following the Kimball's best practices in data warehousing the values are converted to elaborate texts (used in the dimensional tables). Some of the columns in the fact tables contain the comma marks which are read as texts in oracle, so these columns are transformed using regexp\_replace function and converted into an 'int' data type in the spark program. The column names have also been changed to make it into a meaningful one (used during slicing dicing operations)



The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** jupyter final\_project Last Checkpoint: Last Thursday at 1:15 PM (autosaved), Python 3 (ipykernel) O
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Not Trusted, Logout
- Cells:**
  - In [1]:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import date_format, monotonically_increasing_id, col, when, concat,
expr, split, regexp_replace
import pandas as pd
```
  - In [2]:

```
pd.set_option('display.max_columns', None)
```
  - In [3]:

```
spark = SparkSession.builder.appName('Loan').getOrCreate()
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/06/30 15:11:52 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```
  - In [4]:

```
loan_data = spark.read.option("header", "true").option("inferSchema", "true").csv("loan_data_dw.csv")
```
  - In [6]:

```
loan_data.count()
```
  - Out[6]: 697216
  - In [7]:

```
len(loan_data.columns)
```
  - Out[7]: 28
  - In [ ]:

```
loan_data.limit(5).toPandas().head()
```
  - In [ ]:

```
loan_data = loan_data.drop(['Approved?'])
```

## **Part 4: Datawarehouse Dimensional Modelling**

### **Section 4.1 Requirement Gathering**

Every system is designed based on the end goal and end user in mind. The present model is designed for a data engineer as the end user (writing analytical queries for obtaining the results).

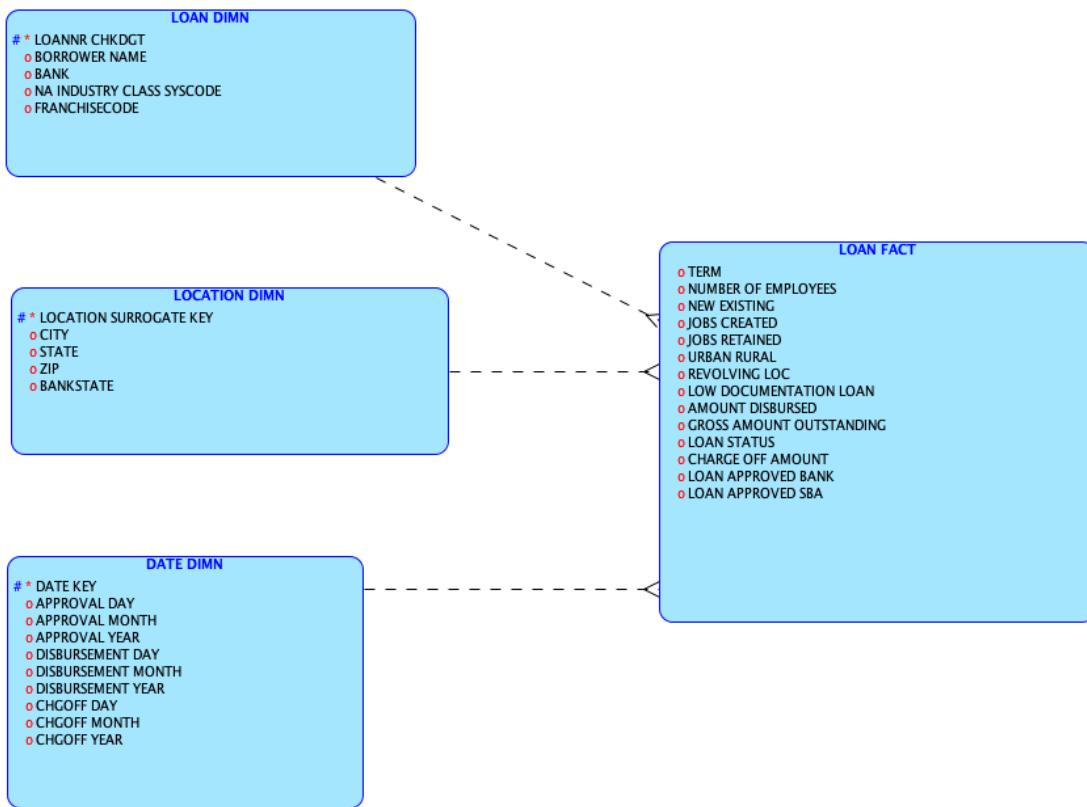
### **Section 4.2 Grain Identification**

A very large grain may not be helpful in answering the questions in the problem statement and so a very fine grain (with seconds as time stamp) eats away the computing resources. So a grain which has a year, loan number and city is identified for the present data warehouse. This approach may be helpful in future to run some more deeper analytical queries without compromising the system performance.

### **Section 4.3 Dimensions and Facts**

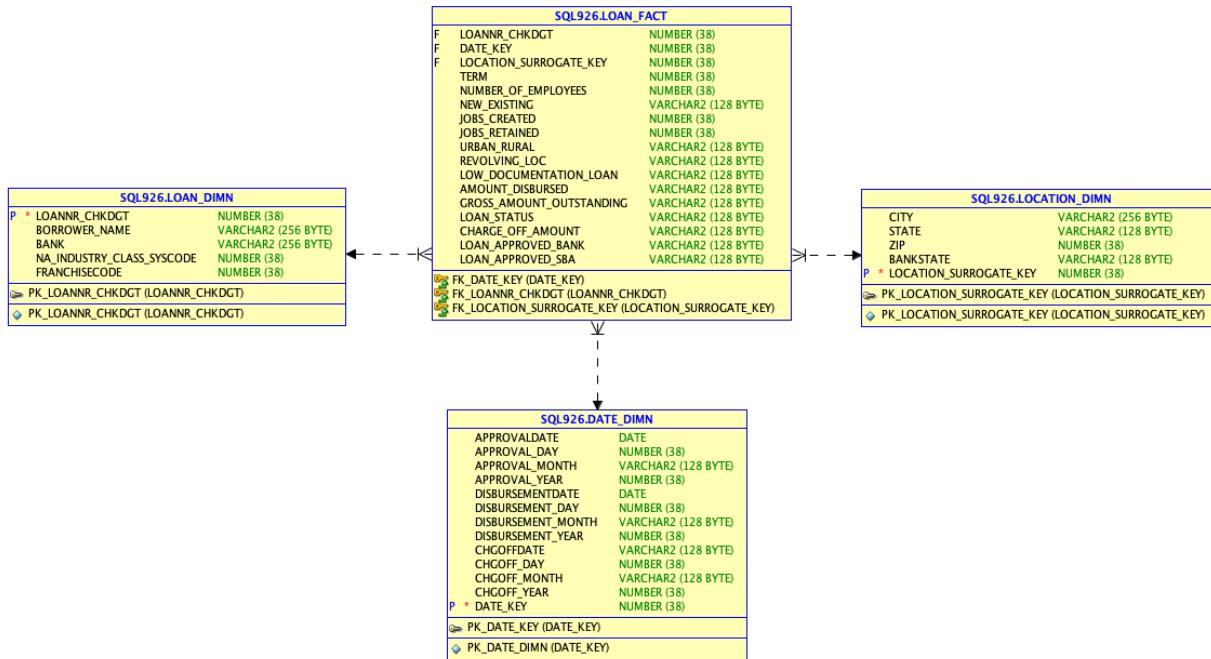
As a best practice first dimensions are identified. The dimensions identified are Location, Date and Loan dimension. A surrogate key is generated for each of the dimensions. In case of the date dimension various date columns (approval, disbursement, charged off) are concatenated to bring a unique values column for each row of the data. In the case of location dimension a unique number is generate for each row.

A single fact table is considered for this data warehouse with additive fact columns ad flag columns (transformed prior using Apache spark).



## Part 5: Data Cube vs OLTP design

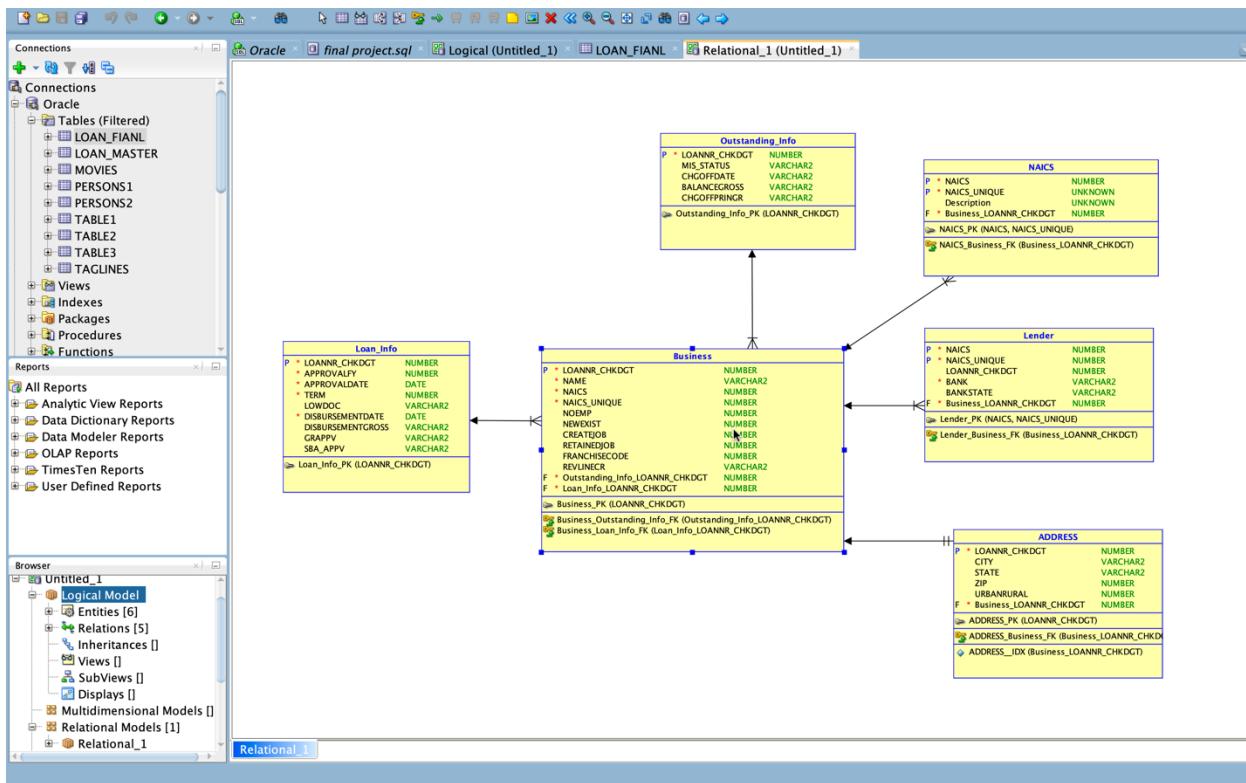
### Section 5.1 Star Schema



The fact table is surrounded by the three dimensions and these fact and dimensions are related to each other by one-to-many relations.

### Section 5.2 Comparison with OLTP

There is no concept of fact and dimension in the OLTP design rather the data is divided into tables based on the normalization constraints in mind and business use case. This design is not suitable for running extensive analytical queries as it would involve complex join operations.



## Section 5.3 Integrity Constraints

The dimension tables are associated with an individual primary key (surrogate keys) despite some of the dimensions are having a natural key. These primary keys are referenced to the fact table using the foreign key for the respective primary key column of dimension in the fact table. Null values and redundant data is removed before assigning a particular column as primary key.

The code for this operation is given below.

```
/*Deleting the null keys from the column to make it into a primary key*/
DELETE FROM DATE_DIMN
WHERE DATE_KEY IS NULL;

/*Deleting duplicate values and preserving one value among them with lowest rowid*/
DELETE FROM DATE_DIMN
WHERE DATE_KEY IN (
    SELECT DATE_KEY
    
```

```

FROM DATE_DIMN
GROUP BY DATE_KEY
HAVING COUNT(*) > 1
)
AND ROWID NOT IN (
  SELECT MIN(ROWID)
  FROM DATE_DIMN
  GROUP BY DATE_KEY
  HAVING COUNT(*) > 1
);
/*Adding the primary key constraint*/
ALTER TABLE DATE_DIMN
ADD CONSTRAINT PK_DATE_KEY PRIMARY KEY (DATE_KEY);

```

/\*The above mentioned process is followed for the LOAN\_DIMN table\*/

```

DELETE FROM LOAN_DIMN
WHERE LOANNR_CHKDGT IS NULL;

DELETE FROM LOAN_DIMN
WHERE LOANNR_CHKDGT IN (
  SELECT LOANNR_CHKDGT
  FROM LOAN_DIMN
  GROUP BY LOANNR_CHKDGT
  HAVING COUNT(*) > 1
)
AND ROWID NOT IN (
  SELECT MIN(ROWID)
  FROM LOAN_DIMN
  GROUP BY LOANNR_CHKDGT
  HAVING COUNT(*) > 1
);

```

```

ALTER TABLE LOAN_DIMN
ADD CONSTRAINT PK_LOANNR_CHKDGT PRIMARY KEY (LOANNR_CHKDGT);

```

/\*The above mentioned process is followed for the LOCATION\_DIMN table\*/

```
DELETE FROM LOCATION_DIMN  
WHERE LOCATION_SURROGATE_KEY IS NULL;
```

```
DELETE FROM LOCATION_DIMN  
WHERE LOCATION_SURROGATE_KEY IN(  
    SELECT LOCATION_SURROGATE_KEY  
    FROM LOCATION_DIMN  
    GROUP BY LOCATION_SURROGATE_KEY  
    HAVING COUNT(*)>1)
```

```
AND ROWID NOT IN(  
    SELECT MIN(ROWID)  
    FROM LOCATION_DIMN  
    GROUP BY LOCATION_SURROGATE_KEY  
    HAVING COUNT(*)>1);
```

```
ALTER TABLE LOCATION_DIMN  
ADD CONSTRAINT PK_LOCATION_SURROGATE_KEY PRIMARY KEY(LOCATION_SURROGATE_KEY);
```

/\*Deleting the rows with null in loan number from Loan\_Fact table\*/

```
DELETE FROM LOAN_FACT  
WHERE LOANNR_CHKDG IS NULL;
```

/\*Deleting the row in fact table for which the referenced rows do not match with the primary key in the dimension tables\*/

```
DELETE FROM LOAN_FACT  
WHERE DATE_KEY NOT IN (SELECT DATE_KEY FROM DATE_DIMN);  
/*Adding the foreign key constraint*/  
ALTER TABLE LOAN_FACT  
ADD CONSTRAINT FK_DATE_KEY FOREIGN KEY(DATE_KEY) REFERENCES DATE_DIMN(DATE_KEY);
```

/\*Data cleaning by deleting the attributes that do not contribute to the business problem\*/

```
DELETE FROM LOAN_FACT  
WHERE LOANNR_CHKDG NOT IN (SELECT LOANNR_CHKDG FROM LOAN_DIMN);
```

```
ALTER TABLE LOAN_FACT
ADD CONSTRAINT FK_LOANNR_CHKDG FOREIGN KEY(LOANNR_CHKDG) REFERENCES
LOAN_DIMN(LOANNR_CHKDG);
```

```
/**/
```

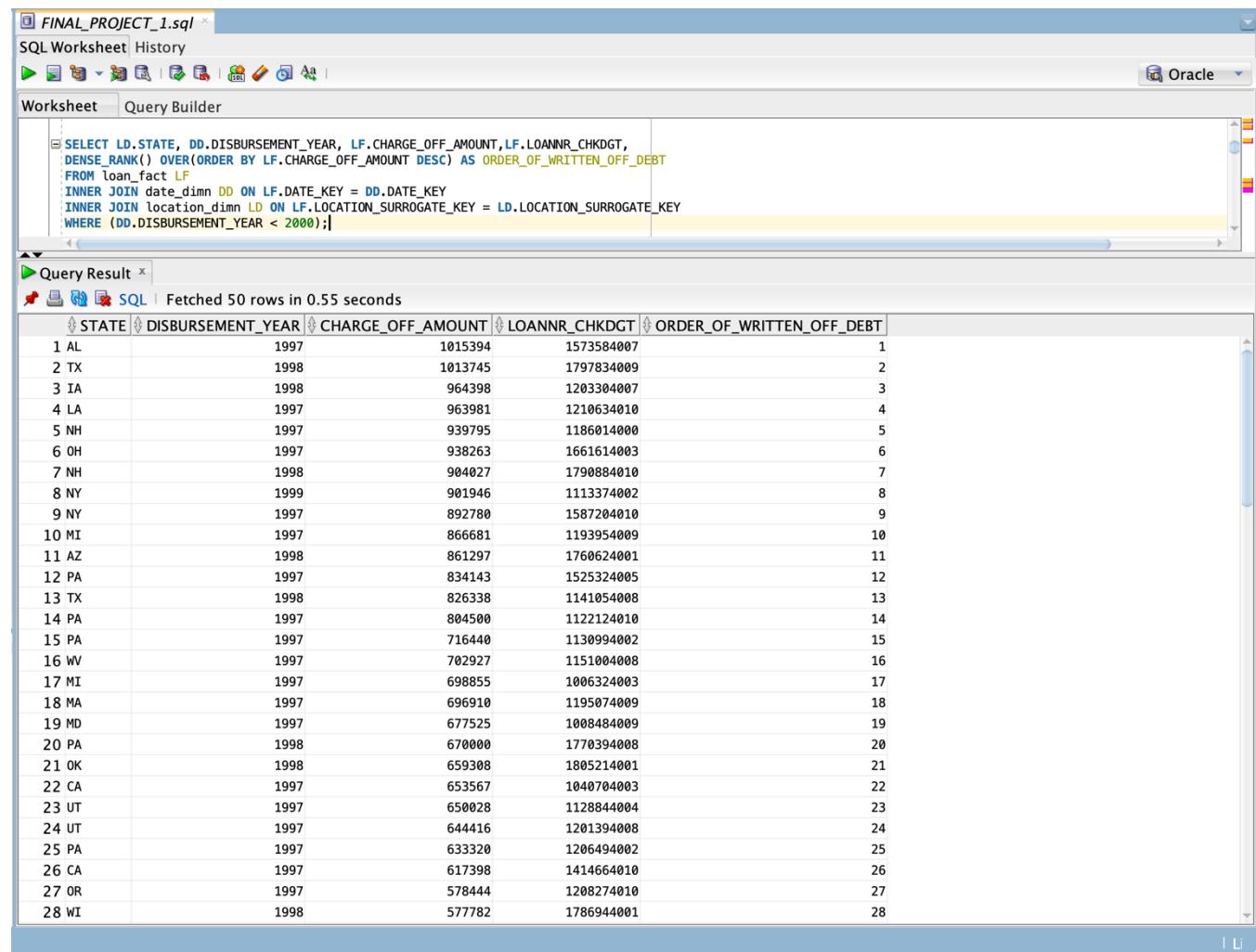
```
DELETE FROM LOAN_FACT
WHERE LOCATION_SURROGATE_KEY NOT IN (SELECT LOCATION_SURROGATE_KEY FROM
LOCATION_DIMN);
```

```
ALTER TABLE LOAN_FACT
ADD CONSTRAINT FK_LOCATION_SURROGATE_KEY FOREIGN KEY(LOCATION_SURROGATE_KEY)
REFERENCES LOCATION_DIMN(LOCATION_SURROGATE_KEY);
```

## Part 6: Analytics

### Section 6.1 Online Analytical Processing

DENSE RANK:



The screenshot shows a SQL developer interface with a query editor and a results viewer. The query editor contains the following SQL code:

```
FINAL_PROJECT_1.sql
SQL Worksheet History
Query Worksheet Query Builder
SELECT LD.STATE, DD.DISBURSEMENT_YEAR, LF.CHARGE_OFF_AMOUNT, LF.LOANNR_CHKDGDT,
DENSE_RANK() OVER(ORDER BY LF.CHARGE_OFF_AMOUNT DESC) AS ORDER_OF_WRTTEN_OFF_DEBT
FROM loan_fact LF
INNER JOIN date_dimm DD ON LF.DATE_KEY = DD.DATE_KEY
INNER JOIN location_dimm LD ON LF.LOCATION_SURROGATE_KEY = LD.LOCATION_SURROGATE_KEY
WHERE (DD.DISBURSEMENT_YEAR < 2000);
```

The results viewer displays a table with the following data:

STATE	DISBURSEMENT_YEAR	CHARGE_OFF_AMOUNT	LOANNR_CHKDGDT	ORDER_OF_WRTTEN_OFF_DEBT
1 AL	1997	1015394	1573584007	1
2 TX	1998	1013745	1797834009	2
3 IA	1998	964398	1203304007	3
4 LA	1997	963981	1210634010	4
5 NH	1997	939795	1186014000	5
6 OH	1997	938263	1661614003	6
7 NH	1998	904027	1790884010	7
8 NY	1999	901946	1113374002	8
9 NY	1997	892780	1587204010	9
10 MI	1997	866681	1193954009	10
11 AZ	1998	861297	1760624001	11
12 PA	1997	834143	1525324005	12
13 TX	1998	826338	1141054008	13
14 PA	1997	804500	1122124010	14
15 PA	1997	716440	1130994002	15
16 WV	1997	702927	1151004008	16
17 MI	1997	698855	1006324003	17
18 MA	1997	696910	1195074009	18
19 MD	1997	677525	1008484009	19
20 PA	1998	670000	1770394008	20
21 OK	1998	659308	1805214001	21
22 CA	1997	653567	1040704003	22
23 UT	1997	650028	1128844004	23
24 UT	1997	644416	1201394008	24
25 PA	1997	633320	1206494002	25
26 CA	1997	617398	1414664010	26
27 OR	1997	578444	1208274010	27
28 WI	1998	577782	1786944001	28

In the above analytical query, the loan numbers (Loannr\_Chkdgt) are ranked based on highest written off debt by the banks using the dense\_rank function. The loan numbers are also shown how much loan they have taken from the banks by state and year.

## LEAD:

The screenshot shows an Oracle SQL Worksheet interface. The title bar says "FINAL\_PROJECT\_1.sql". The top menu includes "SQL Worksheet", "History", and "Oracle". The main area has tabs for "Worksheet" and "Query Builder", with "Worksheet" selected. The query window contains the following SQL code:

```
SELECT a.NA_INDUSTRY_CLASS_SYSCODE
FROM (
  SELECT LF.JOBS_RETAINED, DD.DISBURSEMENT_YEAR AS "YEAR", LOAN_DIMN.NA_INDUSTRY_CLASS_SYSCODE,
  LEAD(LF.JOBS_RETAINED) OVER(PARTITION BY LOAN_DIMN.NA_INDUSTRY_CLASS_SYSCODE ORDER BY DD.DISBURSEMENT_YEAR DESC) AS Jobs_Retained_Previous
  FROM loan_fact LF
  INNER JOIN date_dimn DD ON DD.DATE_KEY = LF.DATE_KEY
  INNER JOIN location_dimn LD ON LF.LOCATION_SURROGATE_KEY = LD.LOCATION_SURROGATE_KEY
  INNER JOIN LOAN_DIMN ON LF.LOANNR_CHKDT = LOAN_DIMN.LOANNR_CHKDT
) a
WHERE a.JOBS_RETAINED > a.Jobs_Retained_Previous
GROUP BY a.NA_INDUSTRY_CLASS_SYSCODE
HAVING COUNT(*) = 2;
```

The "Query Result" tab is selected, showing a table with one column "NA\_INDUSTRY\_CLASS\_SYSCODE" containing 24 rows of data:

NA_INDUSTRY_CLASS_SYSCODE
112111
113110
113210
114210
115113
115115
211111
212311
221210
221320
233220
234990
235710
235940
235950
237120
311111
311119
311422
311611
311823
311830
311911
313221

At the bottom of the interface, status bars show "Line 36 Column 21", "Insert", "Modified", and "Unix/Mac: LF".

The above lead query returns the industries where the jobs retained are more than that of at least two consecutive previous years.

## CORRELATION:

The screenshot shows an Oracle SQL Worksheet interface. The top window is titled 'FINAL\_PROJECT\_1.sql' and contains a SQL query. The bottom window is titled 'Query Result' and displays the results of the query.

```
SELECT a.*,
CASE
    WHEN CORRELATION > 0.75 THEN 'Very High Positive Correlation'
    WHEN CORRELATION > 0.5 THEN 'High Positive Correlation'
    WHEN CORRELATION > 0 THEN 'Low Positive Correlation'
    WHEN CORRELATION = 0 THEN 'No Correlation'
    WHEN CORRELATION > -0.5 THEN 'Low Negative Correlation'
    WHEN CORRELATION > -0.75 THEN 'High Negative Correlation'
    WHEN CORRELATION <= -0.75 THEN 'Very Low Negative Correlation'
    ELSE 'Unknown'
END AS CORRELATION_CATEGORY
FROM (
    SELECT LD.NA_INDUSTRY_CLASS_SYS_CODE, location_dimm.STATE, ROUND(CORR(LF.JOBS_CREATED, LF.JOBS_RETAINED) OVER (PARTITION BY LD.NA_INDUSTRY_CLASS_SYS_CODE),
    FROM loan_fact LF
    INNER JOIN loan_dimm LD ON LF.LOANNR_CHKDTG = LD.LOANNR_CHKDTG
    INNER JOIN location_dimm ON location_dimm.LOCATION_SURROGATE_KEY = LF.LOCATION_SURROGATE_KEY
)a;
```

Query Result

NA_INDUSTRY_CLASS_SYS_CODE	STATE	CORRELATION	CORRELATION_CATEGORY
47	112111 WY	-0.14	Low Negative Correlation
48	112111 MO	-0.14	Low Negative Correlation
49	112111 ID	-0.14	Low Negative Correlation
50	112111 NE	-0.14	Low Negative Correlation
51	112111 TX	-0.14	Low Negative Correlation
52	112111 UT	-0.14	Low Negative Correlation
53	112111 MS	-0.14	Low Negative Correlation
54	112111 SD	-0.14	Low Negative Correlation
55	112111 TX	-0.14	Low Negative Correlation
56	112111 NE	-0.14	Low Negative Correlation
57	112120 WI	(null)	Unknown
58	112120 IA	(null)	Unknown
59	112210 IA	0.76	Very High Positive Correlation
60	112210 OK	0.76	Very High Positive Correlation
61	112210 IA	0.76	Very High Positive Correlation
62	112210 NE	0.76	Very High Positive Correlation
63	112210 IA	0.76	Very High Positive Correlation
64	112210 MO	0.76	Very High Positive Correlation
65	112210 IA	0.76	Very High Positive Correlation
66	112210 AR	0.76	Very High Positive Correlation

In the above query for each of the NAICS (Industry code) the correlation between the jobs created and retained are shown by state. Each of the correlation categories are shown using case statement in the sql query.

## FIRST VALUE, LAST VALUE:

The screenshot shows an Oracle SQL Worksheet interface. The query in the worksheet window is:

```
SELECT LF.LOANNR_CHKDTG, LF.LOAN_STATUS, LF.GROSS_AMOUNT_OUTSTANDING,
FIRST_VALUE(LD.CITY) OVER(PARTITION BY LD.STATE ORDER BY LF.GROSS_AMOUNT_OUTSTANDING RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Most_Outstanding_Loan_City,
LAST_VALUE(LD.CITY) OVER(PARTITION BY LD.STATE ORDER BY LF.GROSS_AMOUNT_OUTSTANDING RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Least_Outstanding_Loan_City
FROM LOAN_FACT LF
INNER JOIN LOCATION_DIMN LD ON LF.LOCATION_SURROGATE_KEY = LD.LOCATION_SURROGATE_KEY
WHERE LF.TERM > 36;
```

The results are displayed in a table titled "Query Result". The columns are:

LOANNR_CHKDTG	LOAN_STATUS	GROSS_AMOUNT_OUTSTANDING	MOST_OUTSTANDING_LOAN_CITY	LEAST_OUTSTANDING_LOAN_CITY
1	1245735001 Paid in Full	0 ANCH	WRANGELL	WRANGELL
2	1215355004 Paid in Full	0 ANCH	WRANGELL	WRANGELL
3	1178914010 Paid in Full	0 ANCH	WRANGELL	WRANGELL
4	1155605004 Charged Off	0 ANCH	WRANGELL	WRANGELL
5	1134455008 Paid in Full	0 ANCH	WRANGELL	WRANGELL
6	1115885002 Paid in Full	0 ANCH	WRANGELL	WRANGELL
7	1118785002 Paid in Full	0 ANCH	WRANGELL	WRANGELL
8	1123505004 Paid in Full	0 ANCH	WRANGELL	WRANGELL
9	1123705005 Paid in Full	0 ANCH	WRANGELL	WRANGELL
10	1127255009 Paid in Full	0 ANCH	WRANGELL	WRANGELL
11	1128205007 Paid in Full	0 ANCH	WRANGELL	WRANGELL
12	1128485010 Paid in Full	0 ANCH	WRANGELL	WRANGELL
13	1056575002 Charged Off	0 ANCH	WRANGELL	WRANGELL
14	1039555005 Paid in Full	0 ANCH	WRANGELL	WRANGELL
15	1046405004 Paid in Full	0 ANCH	WRANGELL	WRANGELL
16	1050755007 Paid in Full	0 ANCH	WRANGELL	WRANGELL
17	1051295002 Paid in Full	0 ANCH	WRANGELL	WRANGELL
18	1054215006 Charged Off	0 ANCH	WRANGELL	WRANGELL
19	1035295002 Paid in Full	0 ANCH	WRANGELL	WRANGELL
20	1018426007 Paid in Full	0 ANCH	WRANGELL	WRANGELL
21	1199544005 Paid in Full	0 ANCH	WRANGELL	WRANGELL
22	1204415003 Paid in Full	0 ANCH	WRANGELL	WRANGELL
23	1769876005 Paid in Full	0 ANCH	WRANGELL	WRANGELL
24	1776006000 Paid in Full	0 ANCH	WRANGELL	WRANGELL
25	1783196009 Paid in Full	0 ANCH	WRANGELL	WRANGELL
26	1743525000 Paid in Full	0 ANCH	WRANGELL	WRANGELL
27	1743735004 Charged Off	0 ANCH	WRANGELL	WRANGELL
28	1753824004 Paid in Full	0 ANCH	WRANGELL	WRANGELL

At the bottom of the worksheet window, status indicators show "Line 115 Column 1", "Insert", and "Modified Unix/Mac: LF".

In the above query the city with highest and lowest outstanding loan amount for a loan greater than 3 years (36 months) are depicted. Window function (range between unbounded preceding and unbounded following) is used.

## NTILE:

The screenshot shows an Oracle SQL Developer interface. The top window is titled 'FINAL\_PROJECT\_1.sql' and contains a SQL query. The bottom window is titled 'Query Result' and displays the results of the query.

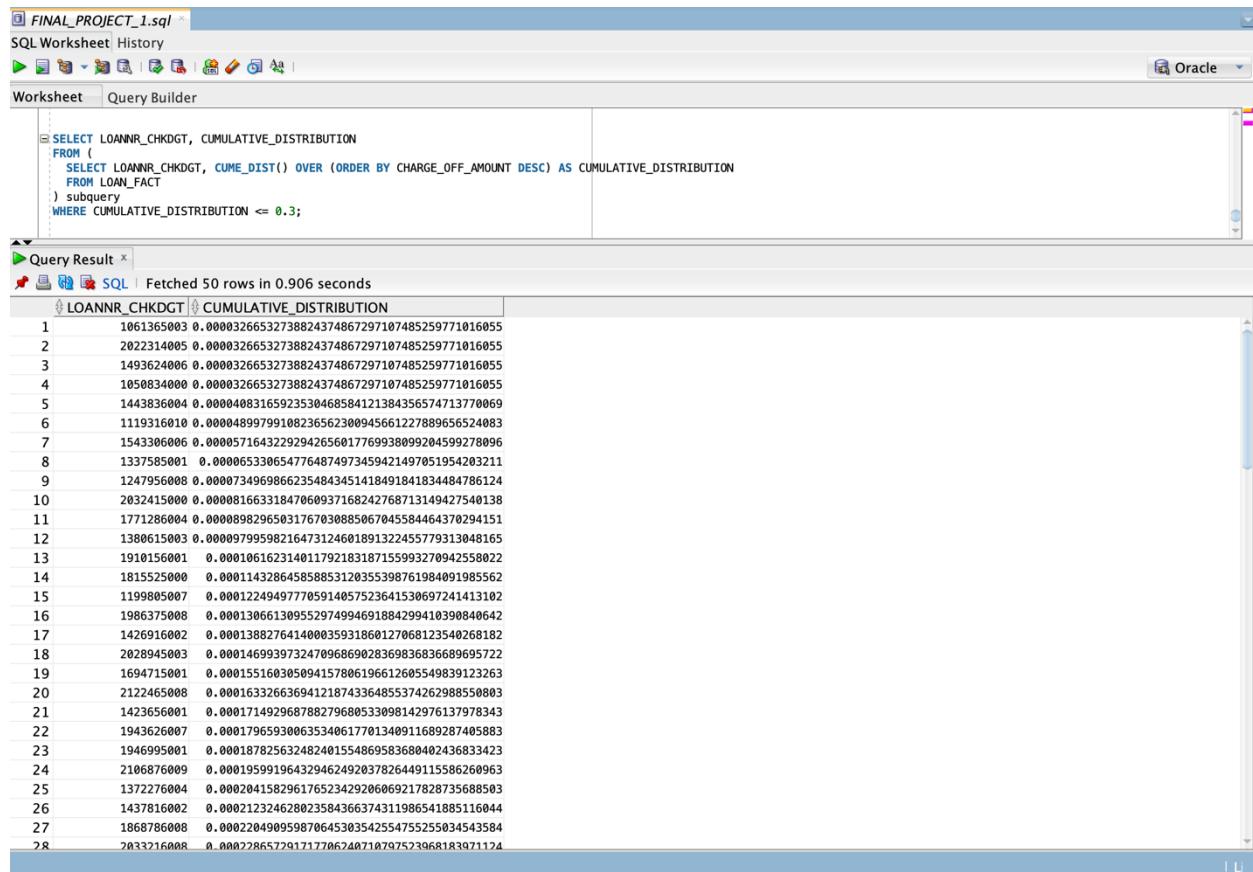
```
SELECT a.DISBURSEMENT_YEAR, a.LOANNR_CHKDG,
CASE
WHEN a.BUCKET = 1 THEN 'High Disbursement'
WHEN a.BUCKET = 2 THEN 'Medium Disbursement'
WHEN a.BUCKET = 3 THEN 'Low Disbursement'
END
AS DISBURSMENT_TYPE
FROM
(SELECT LF.NUMBER_OF_EMPLOYEES, LF.LOANNR_CHKDG, LF.NEW_EXISTING, LF.AMOUNT_DISBURSED, DD.DISBURSEMENT_YEAR,
NTILE(3) OVER(ORDER BY LF.AMOUNT_DISBURSED DESC) AS BUCKET
FROM LOAN_FACT LF INNER JOIN DATE_DIMN DD ON LF.DATE_KEY = DD.DATE_KEY
)a;
```

The 'Query Result' window shows a table with 50 rows. The columns are DISBURSEMENT\_YEAR, LOANNR\_CHKDG, and DISBURSMENT\_TYPE. The data is as follows:

	DISBURSEMENT_YEAR	LOANNR_CHKDG	DISBURSMENT_TYPE
1	2006	2126736010	High Disbursement
2	2006	1634685003	High Disbursement
3	1999	2021004004	High Disbursement
4	2004	1016505009	High Disbursement
5	2006	1412085005	High Disbursement
6	2008	1049226000	High Disbursement
7	2006	1754216003	High Disbursement
8	1997	1652014009	High Disbursement
9	1998	1885504004	High Disbursement
10	2006	1999146004	High Disbursement
11	1998	2080264001	High Disbursement
12	1998	1515104005	High Disbursement
13	2006	1866326006	High Disbursement
14	1998	1594814003	High Disbursement
15	2006	1954055001	High Disbursement
16	1997	1663294007	High Disbursement
17	1997	1024254010	High Disbursement
18	2006	2095705001	High Disbursement
19	2006	1948976006	High Disbursement
20	2006	1765615004	High Disbursement
21	1998	1591324003	High Disbursement
22	2006	1016936005	High Disbursement
23	2005	1304675008	High Disbursement
24	1998	1810384000	High Disbursement

In the above analytical query NTILE is used to create buckets for 3 equal sizes based on amount disbursed (High, Medium, and Low).

## CUMULATIVE DISTRIBUTION:



The screenshot shows an Oracle SQL Worksheet interface. The title bar says "FINAL\_PROJECT\_1.sql". The toolbar includes standard icons for opening, saving, and executing queries. The connection dropdown shows "Oracle". The main area has two tabs: "Worksheet" (selected) and "Query Builder". The "Worksheet" tab contains the following SQL code:

```
SELECT LOANNR_CHKDT, CUMULATIVE_DISTRIBUTION
FROM (
  SELECT LOANNR_CHKDT, CUME_DIST() OVER (ORDER BY CHARGE_OFF_AMOUNT DESC) AS CUMULATIVE_DISTRIBUTION
  FROM LOAN_FACT
) subquery
WHERE CUMULATIVE_DISTRIBUTION <= .3;
```

The "Query Result" tab is selected, showing the output of the query. The results are a list of 50 rows, each containing a loan number and its cumulative distribution value. The first few rows are:

LOANNR_CHKDT	CUMULATIVE_DISTRIBUTION
1061365003	0.000326653273824374867297107485259771016055
2022314005	0.000326653273824374867297107485259771016055
1493624006	0.0003266532738824374867297107485259771016055
1050834000	0.0003266532738824374867297107485259771016055
1443836004	0.0004983165923530468584121384356574713770869
1119316010	0.0004899799108236562300945661227889656524083
1543360006	0.000571643229294265601776993809204599278096
1337585001	0.000653306547764874973459421497051954203211
1247956008	0.000734969866235484345141849184183448786124
2032415000	0.0008166331847060937168242768713149427540138
1771286004	0.0008982965031767030885067045584464370294151
1380615003	0.000979958216473124601891322455779313048165
1910156001	0.001061623140117921831871559327942558022
1815525000	0.00114328645858853120353398761984091985562
1519980007	0.0012249497705914057523641530697241413102
1986375008	0.00130661309552974994691884299418390846462
1426916002	0.000138827641400035931860127068123540268182
2028945003	0.00014699397324709686902836983683669695722
1694715001	0.000155160305094157806196612605549839123263
2022465008	0.000163326636941218743364855374262988550803
1423656001	0.00017149296878279680533099142976137978343
221943626007	0.000179659300635340617701340911689287405883
231946995001	0.000187825632482401554869583680402436833423
242106876009	0.000195991964329462492037826449115586260963
251372276004	0.0002041582961765234292066921782735688503
261437816002	0.000212324628023584366374311986541885116044
271868760008	0.000220490959870645303542554755255034543584
282033216008	0.0002278657791717706748718797523968183071124

Using cumulative distribution in the query the top 30 percent loan numbers (based on written off amount) in the existing loans are shown.

## Part 7: Performance Tuning

### Section 7.1 Aggregation

CUBE:

The screenshot shows an Oracle SQL Worksheet interface. The top bar has tabs for 'FINAL\_PROJECT\_1.sql' and 'final\_project.sql'. The main area is titled 'Worksheet' and contains a 'Query Builder' window. The query uses the CUBE function to calculate subtotal values for each combination of NA\_Industry\_Class\_Syscode and Disbursement Year, along with a grand total for all rows.

```
SELECT LD.NA_INDUSTRY_CLASS_SYSCODE, DD.DISBURSEMENT_YEAR, SUM(LF.NUMBER_OF_EMPLOYEES) AS Total_Employees
FROM loan_dimm LD
INNER JOIN loan_fact LF ON LD.LOANNR_CHKDGT = LF.LOANNR_CHKDGT
INNER JOIN date_dimm DD ON LF.DATE_KEY = DD.DATE_KEY
WHERE LD.NA_INDUSTRY_CLASS_SYSCODE IS NOT NULL
GROUP BY CUBE(LD.NA_INDUSTRY_CLASS_SYSCODE, DD.DISBURSEMENT_YEAR)
ORDER BY DD.DISBURSEMENT_YEAR, LD.NA_INDUSTRY_CLASS_SYSCODE DESC;
```

```
SELECT LD.STATE, DD.DISBURSEMENT_YEAR, LF.CHARGE_OFF_AMOUNT,
DENSE_RANK() OVER(ORDER BY LF.CHARGE_OFF_AMOUNT DESC) AS ORDER_OF_WRITTEN_OFF_DEBT
FROM loan_fact LF
```

The 'Query Result' tab shows the output of the query. The results are as follows:

NA_INDUSTRY_CLASS_SYSCODE	DISBURSEMENT_YEAR	TOTAL_EMPLOYEES
1	(null)	1995
2	235310	1995
3	(null)	1997
4	928120	1997
5	924110	1997
6	923130	1997
7	923120	1997
8	922130	1997
9	921190	1997
10	814110	1997
11	813930	1997
12	813920	1997
13	813910	1997
14	813110	1997
15	812930	1997
16	812332	1997
17	812331	1997
18	812320	1997
19	812310	1997
20	811490	1997
21	811310	1997
22	811192	1997
23	811122	1997
24	811121	1997
25	811118	1997
26	811112	1997

In the above group by cube query the subtotal of number of employees for each of the NA\_Industry\_Syscode and for each of the disbursement year are calculated. Along with that a grand total of total number of employees is calculated.

## ROLLUP:

The screenshot shows the Oracle SQL Worksheet interface. The top bar has tabs for 'FINAL\_PROJECT\_1.sql' and 'final\_project.sql'. The main area is titled 'Worksheet' and contains a SQL query window with the following code:

```
SELECT LD.STATE, LD.CITY, SUM(LF.AMOUNT_DISBURSED) AS Total_Amount_Disbursed
FROM location_dimn LD
INNER JOIN loan_fact LF ON LD.LOCATION_SURROGATE_KEY = LF.LOCATION_SURROGATE_KEY
INNER JOIN date_dimn DD ON DD.DATE_KEY = LF.DATE_KEY
WHERE DD.DISBURSEMENT_YEAR > 2000
GROUP BY ROLLUP(LD.STATE, LD.CITY)
ORDER BY Total_Amount_Disbursed DESC;
```

Below the code is a 'Query Result' window showing the output:

STATE	CITY	TOTAL_AMOUNT_DISBURSED
26 NY	NEW YORK	79588606
27 FL	MIAMI	78228897
28 RI	(null)	77596026
29 CA	LOS ANGELES	76725285
30 OR	(null)	71239545
31 ID	(null)	67723866
32 TN	(null)	59406181
33 NV	(null)	56186648
34 TX	HOUSTON	54481987
35 KY	(null)	53562242
36 NY	BROOKLYN	53538559
37 IL	CHICAGO	52466706
38 IA	(null)	49742286
39 LA	(null)	49674744
40 KS	(null)	47179183
41 OK	(null)	46834156
42 MS	(null)	41859329
43 MT	(null)	38983620
44 AL	(null)	38353956
45 ME	(null)	37233805
46 TX	DALLAS	33996934
47 CO	DENVER	32171639
48 AZ	PHOENIX	31527519
49 NV	LAS VEGAS	30470967
50 CA	SAN DIEGO	30306398
51 NM	(null)	29171271
52 SC	(null)	29136192
53 VT	(null)	28676644

The bottom status bar shows the path 'fy/Documents/Data Warehousing.nosync/Final Project/FINAL\_PROJECT\_1.sql' and the text 'Line 33 Column 1 | Insert | Unix/Mac: LF'.

This rollup query displays subtotal of amount disbursed and grand total of amount disbursed. The subtotals are calculated based on the group by clause of state and city. In other words, the total amount disbursed per state is calculated as subtotals and per each city within the respective state is calculated. Also, the grand total i.e., total amount disbursed for all the states is calculated.

## Section 7.2 Indexing

As we are dealing with star schema, the star transformation is enabled. Already the primary keys in dimension tables contain b-tree indexes (created on primary key by default) so to run the query faster where the cardinality is low (example: Flag column of Urban\_Rural has values of either Urban, Rural or Undefined) a bitmap index is applied on the columns of Urban\_Rural on Loan\_Fact table and State column in Location\_Dimension Table.

The screenshot shows the Oracle SQL Worksheet interface. The top window displays a SQL query:SELECT LF.JOBS\_RETAINED, LF.URBAN\_RURAL, LD.State  
FROM LOAN\_FACT LF  
INNER JOIN LOCATION\_DIMN LD  
ON LF.LOCATION\_SURROGATE\_KEY = LD.LOCATION\_SURROGATE\_KEY  
WHERE LF.URBAN\_RURAL = 'Rural' AND LD.STATE = 'FL';The bottom window shows the Explain Plan for this query. The plan details the execution steps, including a MERGE JOIN operation, a TABLE ACCESS operation on the LOCATION\_DIMN table (using an INDEX BY INDEX ROWID), and a SORT operation. The cost of the query is listed as 3182.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			7036	3182
MERGE JOIN			7036	3182
TABLE ACCESS	LOCATION_DIMN	BY INDEX ROWID	5436	591
INDEX	PK_LOCATION_SURROGAT...	FULL SCAN	90745	191
JOIN			11670	2591
SORT				
Access Predicates	LF.LOCATION_SURROGATE_KEY=LD.LOCATION_SURROGATE_KEY			
Filter Predicates	LF.URBAN_RURAL='Rural'			
TABLE ACCESS	LOAN_FACT	FULL	11670	2463
Filter Predicates	LD.STATE='FL'			

Other XML information includes db\_version (12.1.0.2), parse\_schema ("SQL26"), dynamic\_sampling (note="y", 2), plan\_hash\_full (2123242274), plan\_hash (2611877751), and plan\_hash\_2 (2123242274).

The cost of the query involving both the columns in the join query is 3182.

FINAL\_PROJECT\_1.sql x final\_project.sql x

SQL Worksheet History 1.37100005 seconds Oracle

Worksheet Query Builder

```

ALTER SESSION SET STAR_TRANSFORMATION_ENABLED = TRUE;
CREATE BITMAP INDEX State_bitmap_idx ON LOCATION_DIMN (State);
CREATE BITMAP INDEX UrbanRural_bitmap_idx ON LOAN_FACT (URBAN_RURAL);

SELECT LF.JOBS_RETAINED, LF.URBAN_RURAL, LD.State
FROM LOAN_FACT LF
INNER JOIN LOCATION_DIMN LD
ON LF.LOCATION_SURROGATE_KEY = LD.LOCATION_SURROGATE_KEY
WHERE LF.URBAN_RURAL = 'Rural' AND LD.STATE = 'FL';

```

Query Result Explain Plan 1.371 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			7036	494
HASH JOIN			7036	494
Access Predicates	LF.LOCATION_SURROGATE_KEY=LD.LOCATION_SURROGATE_KEY			
Nested Loops			7036	494
Nested Loops			7036	494
STATISTICS COLLECTOR				
VIEW	index\$_join\$_002		5436	193
Filter Predicates	LD.STATE='FL'			
HASH JOIN				
Access Predicates	ROWID=ROWID			
BITMAP CONVERS		TO ROWIDS	5436	2
BITMAP INDEX STATE_BITMAP_IDX		SINGLE VALUE		
Access Predicates	LD.STATE='FL'			
INDEX	PK_LOCATION_SURROGAT...	FAST FULL SCAN	5436	238
BITMAP CONVERSION		TO ROWIDS		
BITMAP INDEX URBANRURAL_BITMAP_IDX		SINGLE VALUE		
Access Predicates	LF.URBAN_RURAL='Rural'			
TABLE ACCESS	LOAN_FACT	BY INDEX ROWID	1	301
Filter Predicates	LF.LOCATION_SURROGATE_KEY=LD.LOCATION_SURROGATE_KEY			
TABLE ACCESS	LOAN_FACT	BY INDEX ROWID BATCHED	11670	301

Line 100 Column 1 | Insert | Unix/Mac: LF

It is observed adding bitmap indexes in the flag columns of Fact table and Low cardinality columns in the dimensional has significantly improve the query cost bringing it down to 494 from 3182.

## Section 7.3 Partitioning

A query on DATE\_DIMN table is run that involves the Disbursement\_Year column and the query cost is shown below

The screenshot shows the Oracle SQL Worksheet interface. The top window displays a SQL query:SELECT DATE\_KEY, CHGOFF\_YEAR, DISBURSEMENT\_YEAR  
FROM DATE\_DIMN  
WHERE DISBURSEMENT\_YEAR > 2000;The bottom window shows the Explain Plan for this query:| OPERATION | OBJECT\_NAME | OPTIONS | CARDINALITY | COST |
| --- | --- | --- | --- | --- |
| SELECT STATEMENT |  |  |  |  |
| TABLE ACCESS | DATE\_DIMN | FULL | 851 | 102 |
| Filter Predicates |  |  |  |  |
| DISBURSEMENT\_YEAR>2000 |  |  |  |  |
| Other XML |  |  |  |  |
| {info} |  |  |  |  |
| info type="db\_version" |  |  |  |  |
| 12.1.0.2 |  |  |  |  |
| info type="parse\_schema" |  |  |  |  |
| "SQL92" |  |  |  |  |
| info type="plan\_hash\_full" |  |  |  |  |
| 3399113637 |  |  |  |  |
| info type="plan\_hash" |  |  |  |  |
| 3669072844 |  |  |  |  |
| info type="plan\_hash\_2" |  |  |  |  |
| 3399113637 |  |  |  |  |
| {hint} |  |  |  |  |
| FULL(@"SEL\$1" "DATE\_DIMN"@SEL\$1") |  |  |  |  |
| OUTLINE\_LEAF(@"SEL\$1") |  |  |  |  |
| ALL\_ROWS |  |  |  |  |
| DB\_VERSION('12.1.0.2') |  |  |  |  |
| OPTIMIZER\_FEATURES\_ENABLE('12.1.0.2') |  |  |  |  |
| IGNORE\_OPTIM\_EMBEDDED\_HINTS |  |  |  |  |

Line 151 Column 32 | Insert | Modified | Unix/Mac: LF

Now a new table is created DATE\_DIMN1 with range partitioned column on Disbursement\_Year, now the query cost is as shown below. It is observed that the query cost has increased to our surprise despite partitioning the column.

Oracle FINAL\_PROJECT\_1.sql DATE\_DIMN

SQL Worksheet History 0.317 seconds

Worksheet Query Builder

```

CREATE TABLE DATE_DIMN1 (
    DATE_KEY          NUMBER(38,0),
    APPROVAL_DAY      NUMBER(38,0),
    APPROVAL_MONTH    VARCHAR2(26 BYTE),
    APPROVAL_YEAR     NUMBER(38,0),
    DISBURSEMENT_DAY  NUMBER(38,0),
    DISBURSEMENT_MONTH VARCHAR2(26 BYTE),
    DISBURSEMENT_YEAR NUMBER(38,0),
    CHGOFF_DAY        NUMBER(38,0),
    CHGOFF_MONTH      VARCHAR2(26 BYTE),
    CHGOFF_YEAR       NUMBER(38,0)
)
PARTITION BY RANGE (DISBURSEMENT_YEAR) (
    PARTITION P1 VALUES LESS THAN (1950),
    PARTITION P2 VALUES LESS THAN (2000),
    PARTITION P3 VALUES LESS THAN (2050)
);

INSERT INTO DATE_DIMN1
SELECT *
FROM DATE_DIMN;

ALTER TABLE DATE_DIMN1
ADD CONSTRAINT PK_DATE_DIMN1 PRIMARY KEY (DATE_KEY);

SELECT DATE_KEY, CHGOFF_YEAR, DISBURSEMENT_YEAR
FROM DATE_DIMN1
WHERE DISBURSEMENT_YEAR > 2000 and CHGOFF_YEAR > 2000;

```

SQL Tuning Advisor Explain Plan 0.317 seconds

OPTIONS	PARTITION_START	PARTITION_STOP	PARTITION_ID	CARDINALITY	COST
SINGLE		3	3	1	72 138
FULL		3	3	2	72 138

Line 181 Column 1 Insert Modified Unix/Mac: LF

So Sql Tuning advisor is run to get the following recommendations.

Oracle FINAL\_PROJECT\_1.sql DATE\_DIMN

SQL Worksheet History 0.317 seconds

Worksheet Query Builder

SQL Tuning Advisor 2.717 seconds

Tuning Task Name: staName8166  
Tuning Task Owner: SQL926  
Workload Type: SQL Statement  
Scope: COMPREHENSIVE  
Completion Status: COMPLETED

```
SELECT DATE_KEY, CHGOFF_YEAR, DISBURSEMENT_YEAR
FROM DATE_DIMN1
WHERE DISBURSEMENT_YEAR > 2000 and CHGOFF_YEAR > 2000
```

Findings	Recommendations	Rationale
The execution plan of this statement can be improved by creating one or more indices.	Consider running the Access Advisor to improve the physical schema design or creating the recommended index.	Creating the recommended indices significantly improves the execution plan of this statement. However, it might be preferable to run "Access Advisor" using a representative SQL workload as opposed to a single statement. This will allow to get comprehensive index recommendations which takes into account index maintenance overhead and additional space consumption.
The execution plan of this statement can be improved by creating one or more indices.	Consider running the Access Advisor to improve the physical schema design or creating the recommended index.	Creating the recommended indices significantly improves the execution plan of this statement. However, it might be preferable to run "Access Advisor" using a representative SQL workload as opposed to a single statement. This will allow to get comprehensive index recommendations which takes into account index maintenance overhead and additional space consumption.
Table "SQL926"."DATE_DIMN1" was not analyzed.	Consider collecting optimizer statistics for this table.	The optimizer requires up-to-date statistics for the table in order to select a good execution plan.

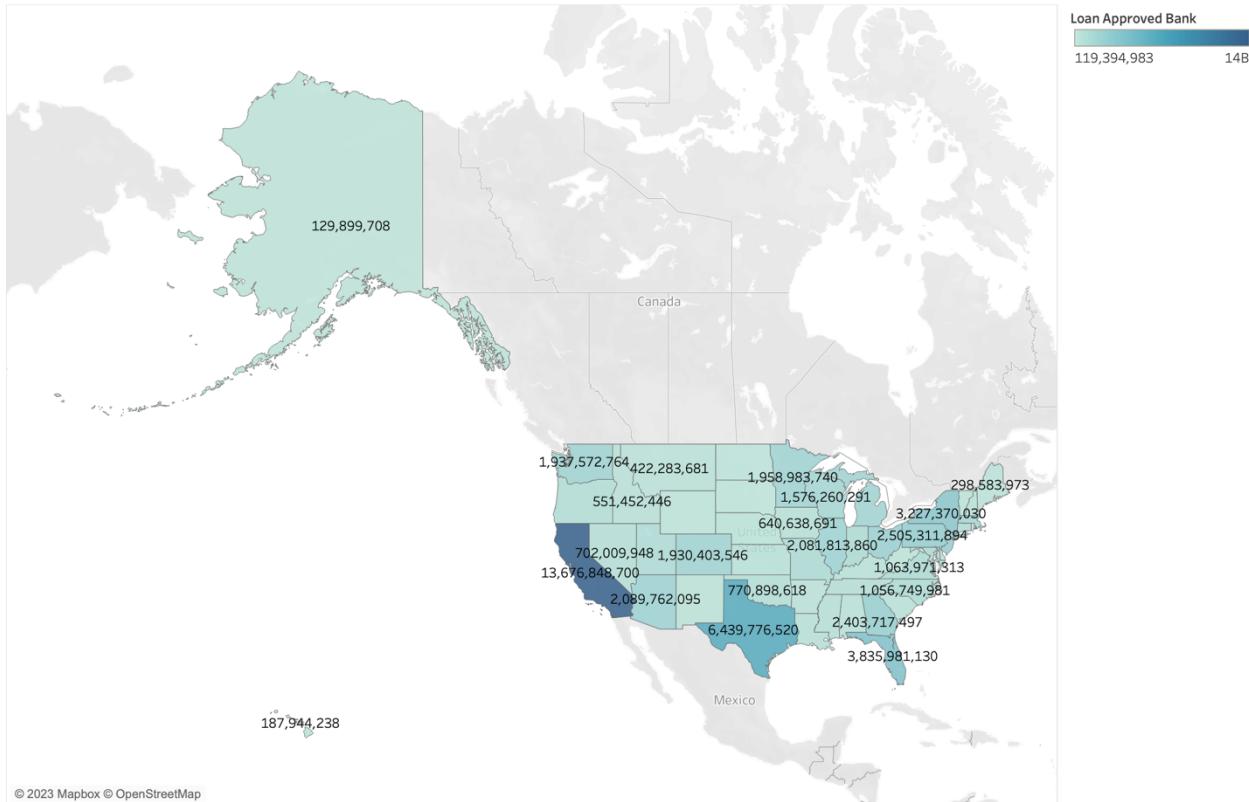
Line 181 Column 1 | Insert | Modified | Unix/Mac: LF

## Part 8: Visualization of the problem statement

### Section 8.1 Data Visualization

- The parameters taken are Borrower city and borrower state from the dimension table of location which is mapped with the loan amount approved column of the fact table loan\_Fact.
- This visualization helps us understand which state has the highest loan amount approved. i.e., the more loan seekers.
- This can be associated with the jobs generated and retained in those states as the proven source of income must be declared for any loan to be approved.

Region wise loan approved

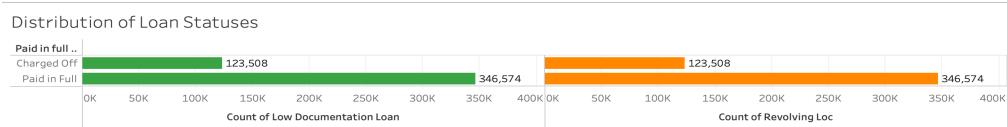


Map based on Longitude (generated) and Latitude (generated). Color shows sum of Loan Approved Bank. Details are shown for State. The view is filtered on State, which excludes Null.

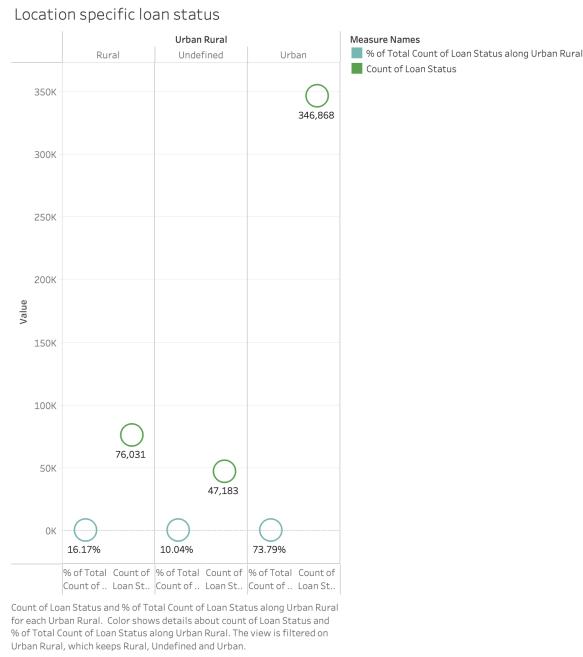
The below visualization portrays businesses with a number of employees of more than 20 within the state of Florida and segmented by each city in the state.

## Distribution of loan statuses

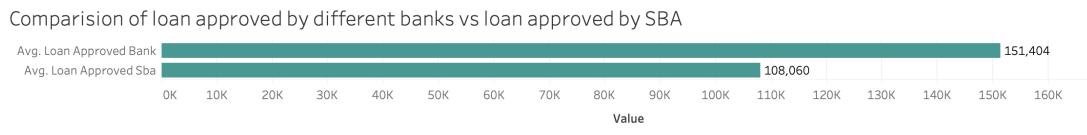
- Charged off loan indicates that the amount at which the loan can be considered defaulted and paid in full loan indicates that the amount has been paid off. We have compared these payments with regard to low documentation loans and revolving lines of credit.



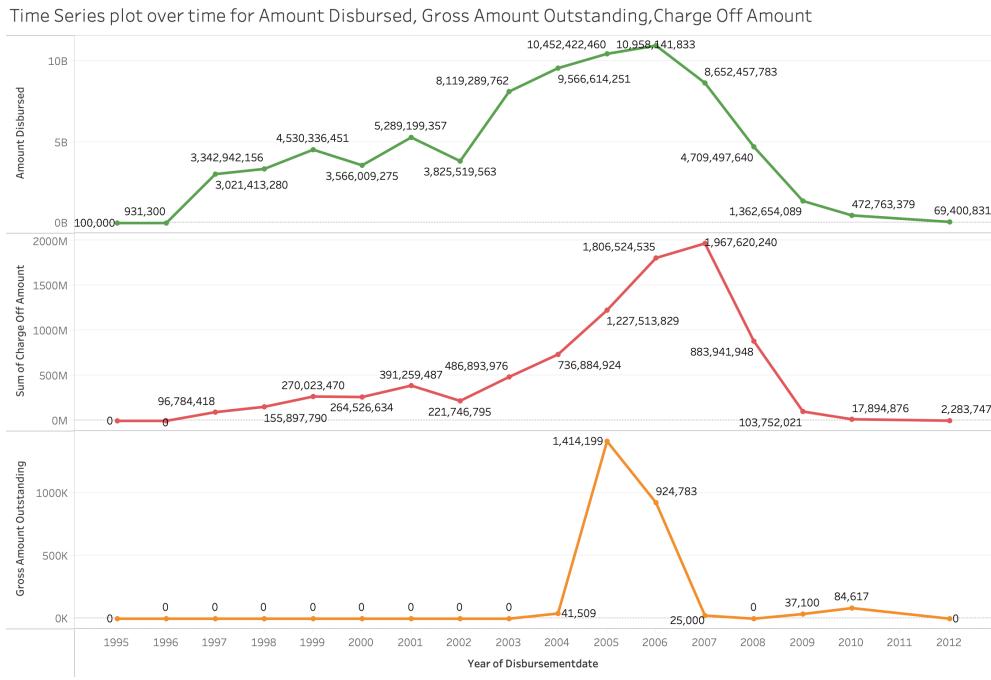
- Based on the urban and rural parameters the count of loans approved was calculated.
- It's observed that the loan applications are more in urban rather than rural areas. Our data didn't have enough parameters to classify the undefined area (we were missing the zip codes, to classify the data), indicating that the urban areas have more job holders compared to that of the rural areas.



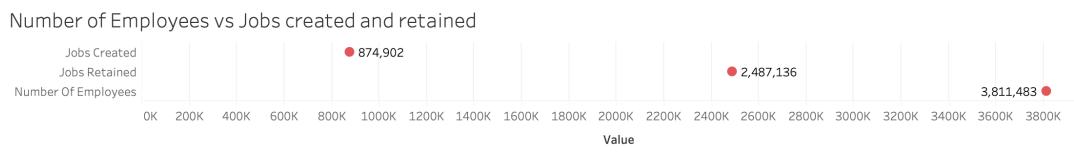
The below visualization illustrates the comparison of bank-approved amounts and SBA-approved amounts for each company in such a manner that the SBA provides an assurance of approved amounts to the bank. It is possible for the lender to easily determine the amount of loan that has been approved through this visualization, and therefore they could decide on opting in for loan either through bank or through SBA.

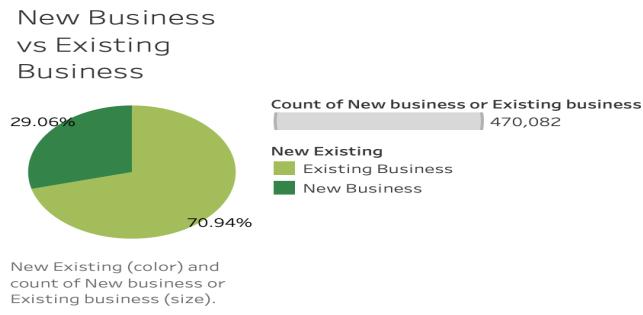


This Visualization helps us understand the number of loans disbursed over the years 1995-2012, the gross outstanding amount, and the charge-off amount. We can understand that the amount disbursed has either been paid off full in most instances i.e., the charge off amount has tallied, due to which the gross amount outstanding has been 0 in most of the cases. We have used the attributes from the date dimension and the loan fact table to achieve this visualization.



The below visualization helps us understand the number of jobs generated by different businesses by taking loans for running their business. The job retention rate is also proportionate to the existing business markets. All the new businesses have contributed to the creation of additional jobs. Despite this the total jobs created and retained are way less than the total available employees for work.





## References

- Journal of Statistics Education - “Should This Loan be Approved or Denied?”: A Large Dataset with Class Assignment Guidelines - Min Li, Amy Mickel & Stanley Taylor
- The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling by Ralph Kimball (Author), Margy Ross (Author)
- Advance Database Management 2023 (Spring Semester) – Final Report by group 4