Database Design – Job Management System

Final Project for Advanced Database Management System | Group 5 | ISM6218.901F21.94685 Advanced Database Management

USF UNIVERSITY OF SOUTH FLORIDA.

Team Members

Akshitha Reddy Suram
Kumar Amrit Vankadara
Santhosh Kumar Nalgonda
Vishwanath Sri Harsha Kotturi
Yashwanth Kumar Kante

Summary of Contents

Topic Area	Description	Group Member	Weight
Database Design	This part should include a logical database design (for the relational model), using normalization to control redundancy and integrity constraints for data quality.	Amrit Kumar Vankadara/ Santhosh Kumar /Yash Kumar Kante	25%
Query Writing	This part is another chance to write SQL queries, explore transactions, and even do some database programming for stored procedures.	Santhosh Kumar Nalgonda	25%
Performance Tuning	In this section, you can capitalize and extend your prior experiments with indexing, optimizer modes, partitioning, parallel execution and any other techniques you want to further explore.	Vishwanath Sri Harsha Kotturi/Amrit Kumar	25%
DBA Scripts	Here you are free to explore any other topics of interest. Suggestions include DBA scripts, database security, interface design, data visualization, data mining, and NoSQL databases.	Akshitha Reddy Suram/Santhosh Kumar Nalgonda	25%

Purpose

This document describes the process that has been used to create a database system that could potentially be used as a Job Management System. The details of the entities and attributes that have been tracked in the database are discussed. This document also describes the processes involved in managing database integrity, data generation and loading, query writing to fetch data, performance tuning using indexes and parallelism. Additional topics such as dba scripts have also been discussed in this paper.

Narrative

Schools are the cradle of knowledge and the fountain that feeds students. It's mandatory to have the best management system in place for it to function the way it should and pass across relevant information to students. For it to function accordingly, a proper job management system is one of the basic but significant things that should be in place. The Job Management System provides a single interface for users to manage on-campus student employment. This will enable an improved communication, streamlining of tasks, better accessibility. The benefits are endless.

The job management system is for managing career building events and jobs for their students. Three important aspects in this system are – Job posting and applications managements, posting of events and enrolment management, Managing information for Faculty, Employers and Students. This system aims to increase automation and reduce manual effort by giving features like viewing jobs and events, applying for jobs and events, adding, and updating information of all end users, accepting rejecting applications.

The system consists of many students, faculty, and employers. Each student has been assigned a unique student ID. Additional information pertaining to each student such as their full name, DOB, contact details, email, address, student type, major, expected graduation date, current GPA, profile URL are stored. Similarly, each faculty is given a unique staff ID and details like staff name, date of birth, email, contact, address, staff type, and staff status are stored. The employer also has a unique employer ID and additional details like Employer name, industry type, employer type, address, and website are stored.

An employer can post multiple jobs and each job has a unique job ID. A students can apply to multiple jobs. Similarly, each faculty can post multiple events where each event can be enrolled by multiple students. The details of the jobs applied, and events enrolled by the students are stored.

Entities Identified to be Tracked

- Students
- Employers
- Faculty
- Jobs
- Events
- Applied Jobs
- Enrolled Events

Entities with Attributes Nested

- **Students**
    - Student Id
    - Student Name
    - DOB
    - Email Id
    - Contact Info
    - Address
    - Student Type
    - Majors
    - Expected Graduation
    - Current GPA
    - Profile URL

- **Employers**
    - Employer ID
    - Employer name
    - Industry Category
    - Employer Category
    - Address
    - Website

- **Faculty**
    - Faculty ID
    - Faculty Name
    - DOB
    - Email
    - Phone number
    - Address
    - Staff Role
    - Staff Status

- **Jobs**
    - Job ID
    - Job Name
    - Description
    - Job Type
    - Location

    - Application Deadline
    - Job URL
    - Poster ID

- **Events**
    - Event ID
    - Event Name
    - Description
    - Start Date
    - End Date
    - Primary Acceptor
    - Secondary Acceptor
    - host ID

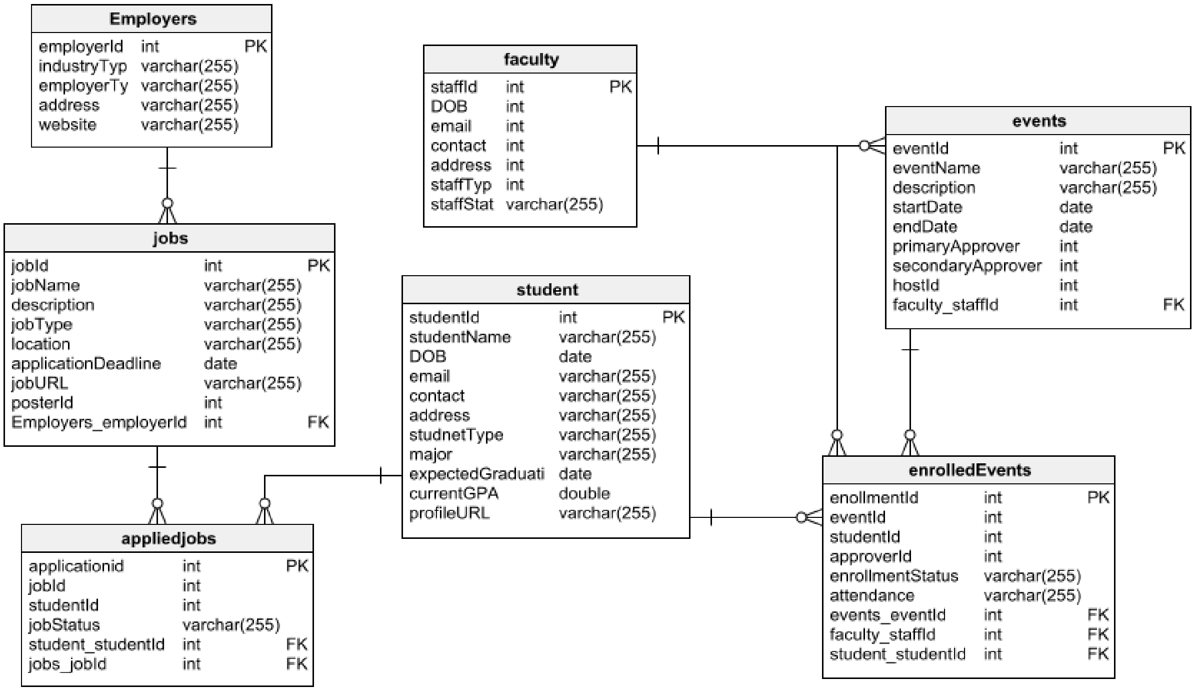
- **Applied Jobs**
    - Application ID
    - Job ID
    - Student ID
    - Job Status

- **Enrolled Events**
    - Enrolled ID
    - Event ID
    - Student ID
    - Approver ID
    - Enrollment Status
    - Attendance

Business Rules

- An employer can post multiple jobs.
- A student can apply to multiple jobs
- A faculty can post multiple events
- An event can be enrolled by multiple students
- A student can attend multiple events
- Job status should be updated in applied jobs table
- Employer can be of type Private/Government
- Student and Faculty can have status Active/Inactive

Entity Relationship Diagram representing Database Design



Employers		
employerId	int	PK
industryTyp	varchar(255)	
employerTy	varchar(255)	
address	varchar(255)	
website	varchar(255)	

faculty		
staffId	int	PK
DOB	int	
email	int	
contact	int	
address	int	
staffTyp	int	
staffStat	varchar(255)	

events		
eventId	int	PK
eventName	varchar(255)	
description	varchar(255)	
startDate	date	
endDate	date	
primaryApprover	int	
secondaryApprover	int	
hostId	int	
faculty_staffId	int	FK

jobs		
jobId	int	PK
jobName	varchar(255)	
description	varchar(255)	
jobType	varchar(255)	
location	varchar(255)	
applicationDeadline	date	
jobURL	varchar(255)	
posterId	int	
Employers_employerId	int	FK

student		
studentId	int	PK
studentName	varchar(255)	
DOB	date	
email	varchar(255)	
contact	varchar(255)	
address	varchar(255)	
studnetType	varchar(255)	
major	varchar(255)	
expectedGraduati	date	
currentGPA	double	
profileURL	varchar(255)	

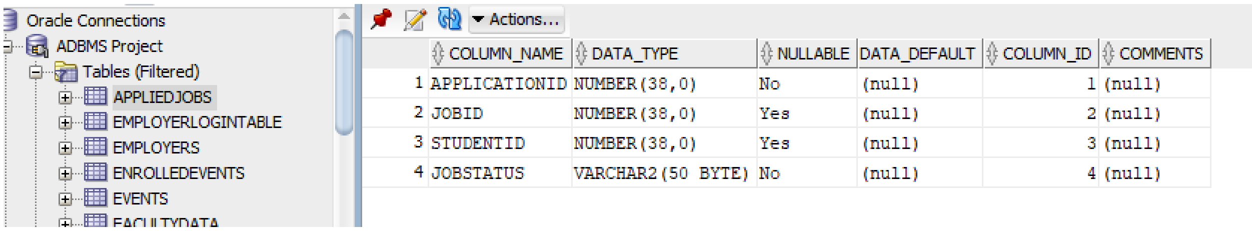
appliedjobs		
applicationid	int	PK
jobId	int	
studentId	int	
jobStatus	varchar(255)	
student_studentId	int	FK
jobs_jobId	int	FK

enrolledEvents		
enollmentId	int	PK
eventId	int	
studentId	int	
approverId	int	
enrollmentStatus	varchar(255)	
attendance	varchar(255)	
events_eventId	int	FK
faculty_staffId	int	FK
student_studentId	int	FK

Vertabelo

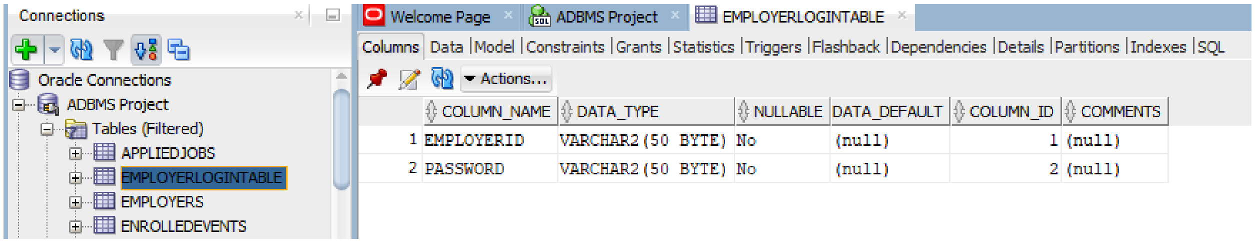
Table Views

Applied Jobs: This table collects all details about the jobs applied by students such as Application ID, Job ID, Student ID and Job Status.



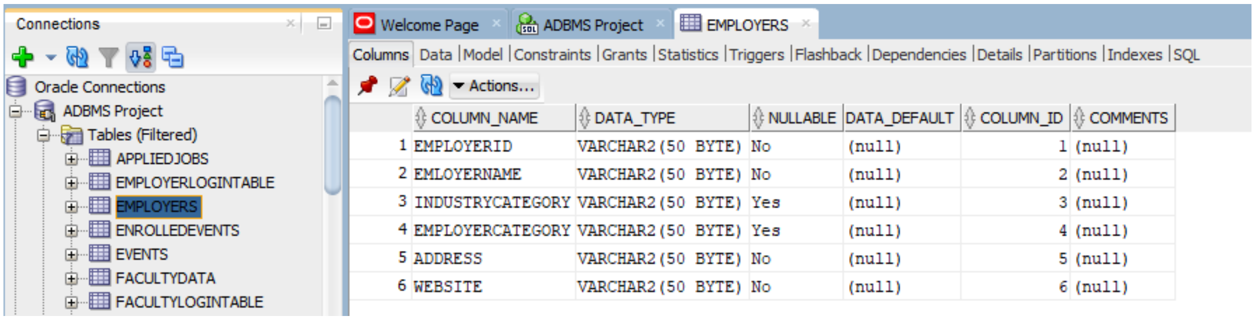
	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	APPLICATIONID	NUMBER(38,0)	No	(null)	1	(null)
2	JOBID	NUMBER(38,0)	Yes	(null)	2	(null)
3	STUDENTID	NUMBER(38,0)	Yes	(null)	3	(null)
4	JOBSTATUS	VARCHAR2(50 BYTE)	No	(null)	4	(null)

Employer Login Table: This table contains details about the employer login details namely Employer ID and Password.



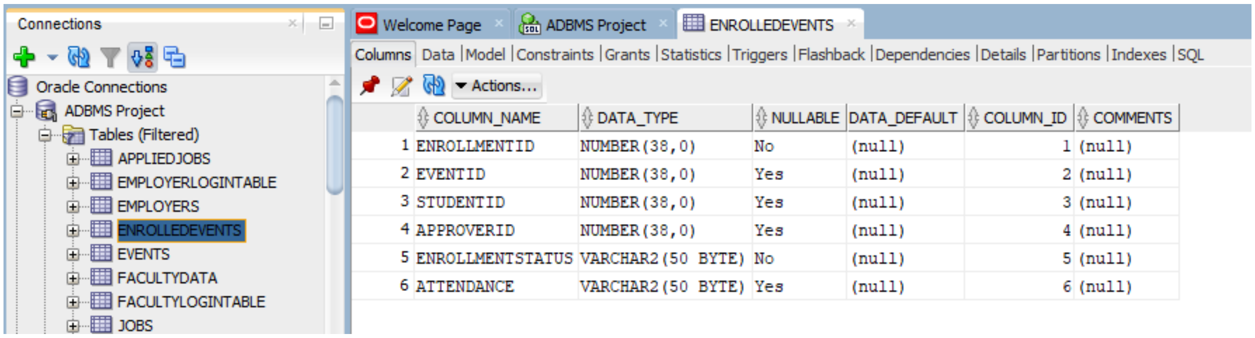
	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	EMPLOYERID	VARCHAR2(50 BYTE)	No	(null)	1	(null)
2	PASSWORD	VARCHAR2(50 BYTE)	No	(null)	2	(null)

Employers Table: This table contains all the details of the employers like Employer ID, Employer name, Industry Category, Employer Category, Address and Website.



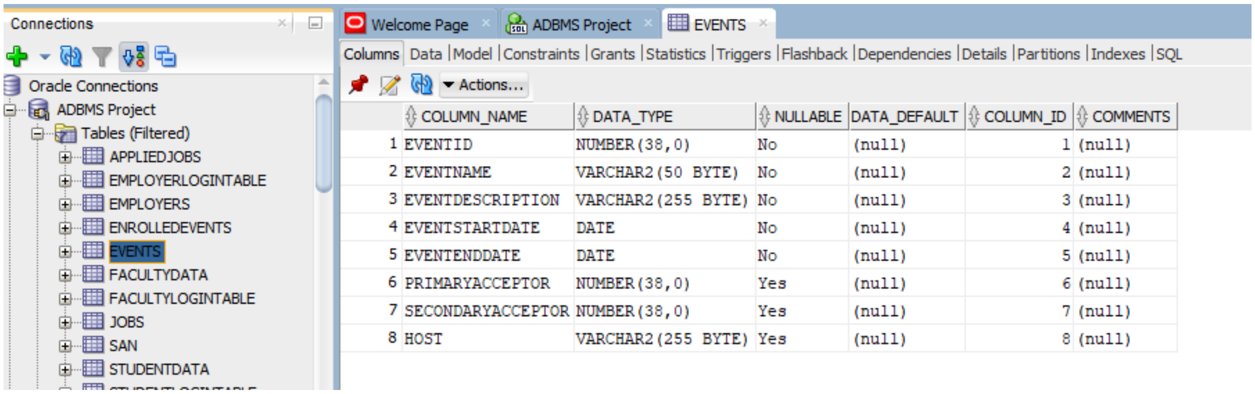
	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	EMPLOYERID	VARCHAR2(50 BYTE)	No	(null)	1	(null)
2	EMLOYERNAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	INDUSTRYCATEGORY	VARCHAR2(50 BYTE)	Yes	(null)	3	(null)
4	EMPLOYERCATEGORY	VARCHAR2(50 BYTE)	Yes	(null)	4	(null)
5	ADDRESS	VARCHAR2(50 BYTE)	No	(null)	5	(null)
6	WEBSITE	VARCHAR2(50 BYTE)	No	(null)	6	(null)

Enrolled Events Table: This table contains event enrollment details such as enrollment id, event id, student id, approved id, enrollment status, and attendance.



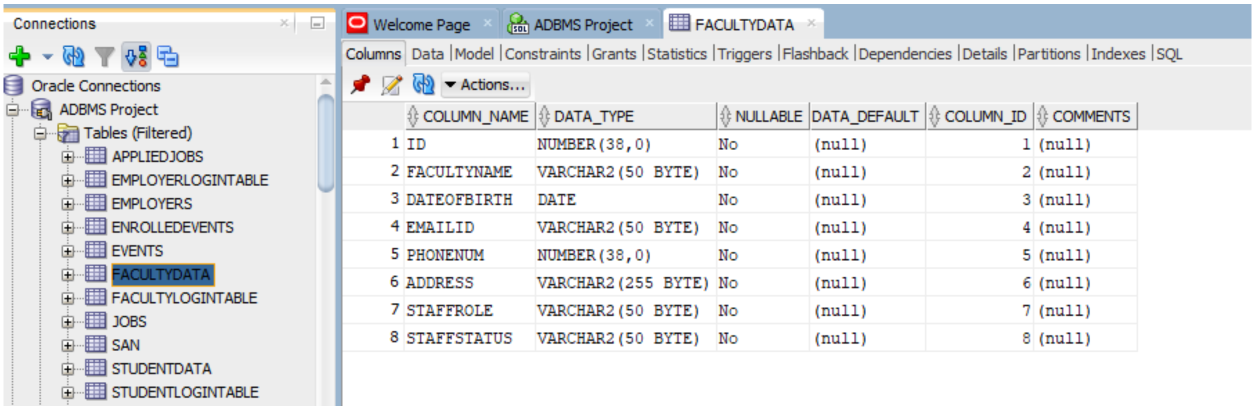
	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ENROLLMENTID	NUMBER(38,0)	No	(null)	1	(null)
2	EVENTID	NUMBER(38,0)	Yes	(null)	2	(null)
3	STUDENTID	NUMBER(38,0)	Yes	(null)	3	(null)
4	APPROVERID	NUMBER(38,0)	Yes	(null)	4	(null)
5	ENROLLMENTSTATUS	VARCHAR2(50 BYTE)	No	(null)	5	(null)
6	ATTENDANCE	VARCHAR2(50 BYTE)	Yes	(null)	6	(null)

Events Table: This table contains event details such as event id, event name, event description, event start date, event data, primary acceptor, secondary acceptor, and host.



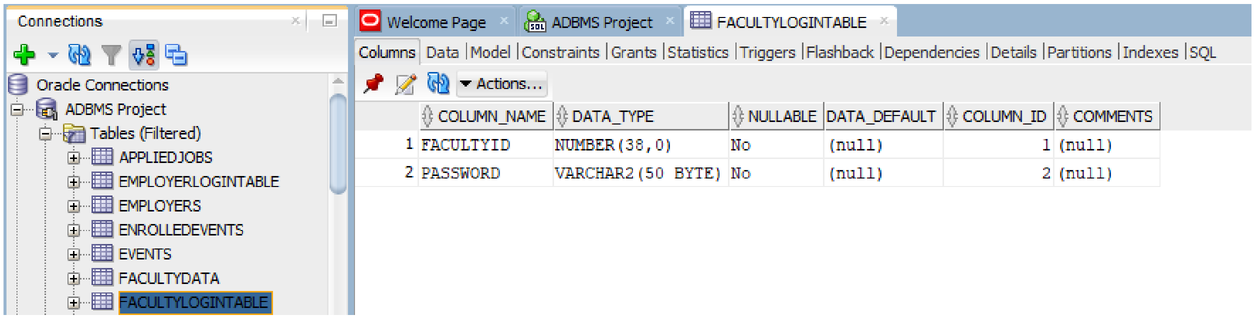
	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	EVENTID	NUMBER(38,0)	No	(null)	1	(null)
2	EVENTNAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	EVENTDESCRIPTION	VARCHAR2(255 BYTE)	No	(null)	3	(null)
4	EVENTSTARTDATE	DATE	No	(null)	4	(null)
5	EVENTENDDATE	DATE	No	(null)	5	(null)
6	PRIMARYACCEPTOR	NUMBER(38,0)	Yes	(null)	6	(null)
7	SECONDARYACCEPTOR	NUMBER(38,0)	Yes	(null)	7	(null)
8	HOST	VARCHAR2(255 BYTE)	Yes	(null)	8	(null)

Faculty Data Table: This table contains faculty data like Faculty id, faculty name, date of birth, email ID, phone number, address, staff role, and staff status.



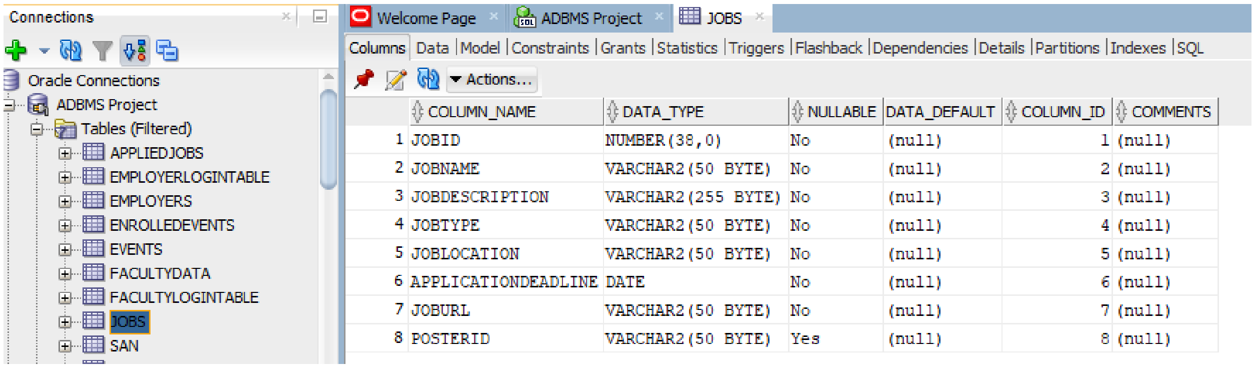
	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(38,0)	No	(null)	1	(null)
2	FACULTYNAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	DATEOFBIRTH	DATE	No	(null)	3	(null)
4	EMAILID	VARCHAR2(50 BYTE)	No	(null)	4	(null)
5	PHONENUM	NUMBER(38,0)	No	(null)	5	(null)
6	ADDRESS	VARCHAR2(255 BYTE)	No	(null)	6	(null)
7	STAFFROLE	VARCHAR2(50 BYTE)	No	(null)	7	(null)
8	STAFFSTATUS	VARCHAR2(50 BYTE)	No	(null)	8	(null)

Faculty Login Table: This table contains faculty login data namely faculty id and password.



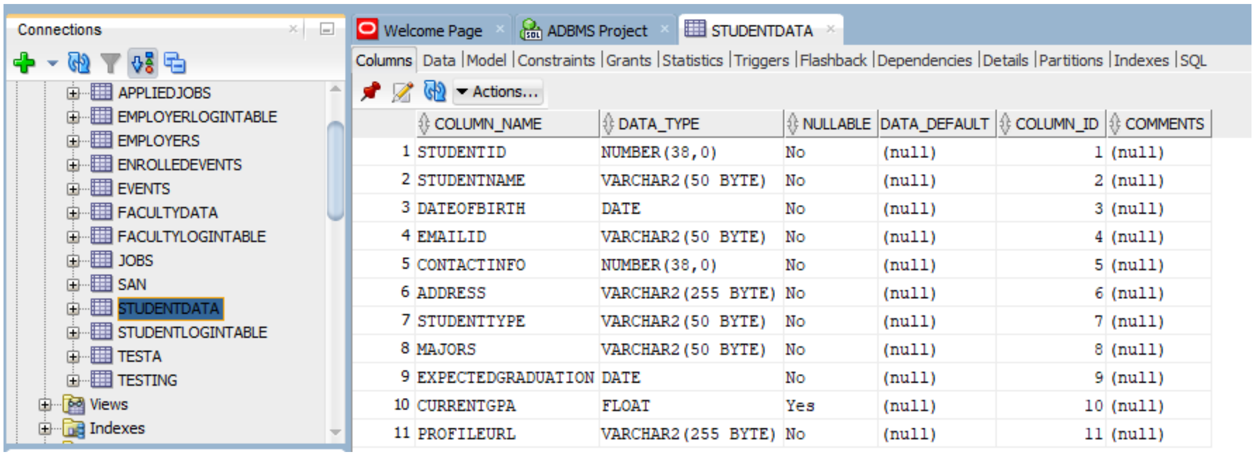
	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	FACULTYID	NUMBER(38,0)	No	(null)	1	(null)
2	PASSWORD	VARCHAR2(50 BYTE)	No	(null)	2	(null)

Jobs Table: This table contains jobs data namely job id, job name, job description, job type, job location, application deadline, job url and poster id.



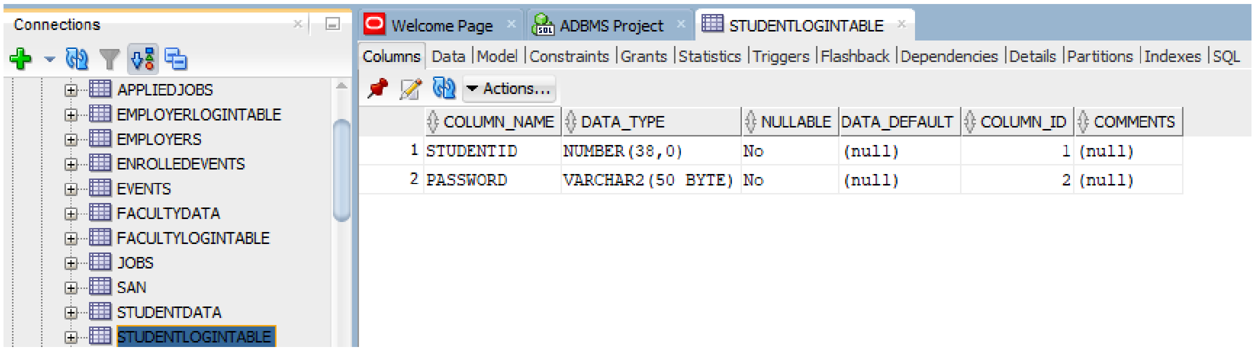
	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	JOBID	NUMBER(38,0)	No	(null)	1	(null)
2	JOBNAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	JOBDESCRIPTION	VARCHAR2(255 BYTE)	No	(null)	3	(null)
4	JOBTYPE	VARCHAR2(50 BYTE)	No	(null)	4	(null)
5	JOBLOCATION	VARCHAR2(50 BYTE)	No	(null)	5	(null)
6	APPLICATIONDEADLINE	DATE	No	(null)	6	(null)
7	JOBURL	VARCHAR2(50 BYTE)	No	(null)	7	(null)
8	POSTERID	VARCHAR2(50 BYTE)	Yes	(null)	8	(null)

Student Data Table: This table contains student data namely student id, student name, date of birth, email id, contact info, address, student type, majors, expected graduation, current gpa, and profile url.



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	STUDENTID	NUMBER(38,0)	No	(null)	1	(null)
2	STUDENTNAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	DATEOFBIRTH	DATE	No	(null)	3	(null)
4	EMAILID	VARCHAR2(50 BYTE)	No	(null)	4	(null)
5	CONTACTINFO	NUMBER(38,0)	No	(null)	5	(null)
6	ADDRESS	VARCHAR2(255 BYTE)	No	(null)	6	(null)
7	STUDENTTYPE	VARCHAR2(50 BYTE)	No	(null)	7	(null)
8	MAJORS	VARCHAR2(50 BYTE)	No	(null)	8	(null)
9	EXPECTEDGRADUATION	DATE	No	(null)	9	(null)
10	CURRENTGPA	FLOAT	Yes	(null)	10	(null)
11	PROFILEURL	VARCHAR2(255 BYTE)	No	(null)	11	(null)

Student Login Table: This table contains student login data namely student id and password.



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	STUDENTID	NUMBER(38,0)	No	(null)	1	(null)
2	PASSWORD	VARCHAR2(50 BYTE)	No	(null)	2	(null)

Data Synthesis

The data for the project has been synthesized using a combination of an online tool named Mockaroo and Microsoft Excel. Some of the prominent functions that were used in Excel include,

- VLOOKUP
- INDEX
- ROWS
- RAND and
- RANDBETWEEN

The tabulation below provides a summary of the data housed in the tables,

Table Name	**Columns**	**# of constraints**	**# of Records**
APPLIEDJOBS	4	4	233
EMPLOYERS	6	4	657
ENROLLEDEVENTS	6	6	239
EVENTS	8	6	513
FACULTYDATA	8	8	533
JOBS	8	8	780
STUDENTDATA	11	10	1200
EMPLOYERLOGINTABLE	2	3	200
FACULTYLOGINTABLE	2	3	200
STUDENTLOGINTABLE	2	3	200

Data Integrity

Data Integrity refers to the consistency and maintenance of the data through the life cycle of the database. In a database, data integrity can be ensured through the implementation of Integrity Constraints in a table. Integrity constraints help apply business rules to the database tables. The constraints can either be at a column level or a table level. Some of the most common constraints are NOT NULL – Prevents a column from having a NULL value.

- NOT NULL – Prevents a column from having a NULL value.

- PRIMARY KEY – Uniquely identifies each row or record in table.

- FOREIGN KEY – Uniquely identifies a column that references a PRIMARY KEY in another table.

- UNIQUE – Prevents a column from having duplicate values.

- CHECK – Checks for values that satisfy a specific condition as defined by the user.

Listed below are the constraints that were created for our database development project along with their purpose,

1. Primary Acceptor Foreign Key Used to grant or approve students registration for jobs

```
-- table containing events information
drop table events;

CREATE TABLE events(
eventId INT PRIMARY KEY,
eventName VARCHAR (50) NOT NULL,
Eventdescription VARCHAR (255) NOT NULL,
EventstartDate DATE NOT NULL,
EventendDate DATE NOT NULL,
primaryAcceptor INT,
secondaryAcceptor INT,
host varchar(255),
CONSTRAINT FK_events FOREIGN KEY (primaryAcceptor) REFERENCES facultyData(ID)
);

-- table containing enrolled events information

CREATE TABLE enrolledEvents(
enrollmentId INT PRIMARY KEY NOT NULL ,
eventId INT,
StudentId INT,
approverId INT,
enrollmentStatus VARCHAR (50) NOT NULL,
attendance VARCHAR (50),
CONSTRAINT FK_approver FOREIGN KEY (approverId) REFERENCES facultyData(ID),
CONSTRAINT FK_student FOREIGN KEY (StudentId) REFERENCES studentData(StudentId),
CONSTRAINT FK_enrolledEvent FOREIGN KEY (eventId) REFERENCES events(eventId)
);


-- table containing jobs information

CREATE TABLE jobs(
jobId INT PRIMARY KEY,
jobName VARCHAR (50) NOT NULL,
jobdescription VARCHAR (255) NOT NULL,
jobType VARCHAR (50) NOT NULL,
joblocation VARCHAR (50) NOT NULL,
applicationDeadline DATE NOT NULL,
jobURL VARCHAR (50) NOT NULL,
posterId VARCHAR(50),
CONSTRAINT FK_jobs FOREIGN KEY (posterId) REFERENCES employers (employerId)
);


-- table containing applied jobs information

CREATE TABLE appliedjobs(
applicationId INT PRIMARY KEY,
jobId INT,
studentId INT,
jobStatus VARCHAR (50) NOT NULL,
CONSTRAINT FK_appJobs FOREIGN KEY (jobId) REFERENCES jobs (jobId),
CONSTRAINT FK_jobStudent FOREIGN KEY (studentId) REFERENCES studentData (StudentId)
);
```

2) Different Not Null constraints have been implemented on different columns (jobstatus, jobName, enrollmentStatus etc) as those column values should not be null.

Queries / Sproc

1. Check application status for all students who applied for job

```
Select appj.applicationId,jb.jobName,stud.studentName,stud.email,stud.contact,stud.major,stud.expectedGraduation,stud.currentGPA,appj.jobStatus

FROM (select * from appliedjobs) appj JOIN (select * from student) stud on appj.studentId = stud.studentId

JOIN (select * from jobs) jb on appj.jobId=jb.jobId;
```

2. Check total number of students enrolled per course

```
select count(*) as cnt,major from student group by major;
```

3. Check total number of students placed per course

```
select count(stud.studentName) as cnt,stud.major

FROM (select * from appliedjobs) appj JOIN (select * from student) stud on appj.studentId = stud.studentId where appj.jobStatus='accepted'

group by stud.major;
```

4. View detailed information of students and jobs who applied for that job when a employer is logged in

```
select appj.applicationId,appj.jobId,jb.jobName,stud.studentName,stud.dob,stud.email,stud.contact,stud.major,stud.expectedGraduation,stud.currentGPA,stud.profileURL,appj.jobStatus

FROM (select * from appliedjobs) appj JOIN (select * from student) stud on appj.studentId = stud.studentId

JOIN (select * from jobs) jb on appj.jobId=jb.jobId where jb.posterId=santosh;
```

5. Display employer id whether their industry category exists or not

```
SELECT b.employerid,a.industrycategory FROM

employerlogintable b LEFT JOIN employers a ON (a.employerid=b.employerid)

ORDER BY 1;
```

6.Display student and student names based on their order of majors

```
SELECT studentID, StudentName,studenttype FROM studentdata ORDER BY majors;
```

7. Display students whose student ids are greater than specific ids in the database

```
SELECT studentid FROM studentdata
 where studentid>=2589
```

8. Display emailing addresses for all students grouped on the basis of their majors for announcement purposes.

```
select email,contact from student GROUP BY major;
```

9. Display students who are not subscribed or applied to any jobs

```
SELECT studentid FROM studentdata
MINUS SELECT studentid FROM appliedjobs
```

10. Display employer id whether their industry category exists only

```
SELECT b.employerid,a.industrycategory FROM
employerlogintable b inner JOIN employers a ON (a.employerid=b.employerid)
ORDER BY 1;
```

11. Stored Procedure for selecting the data

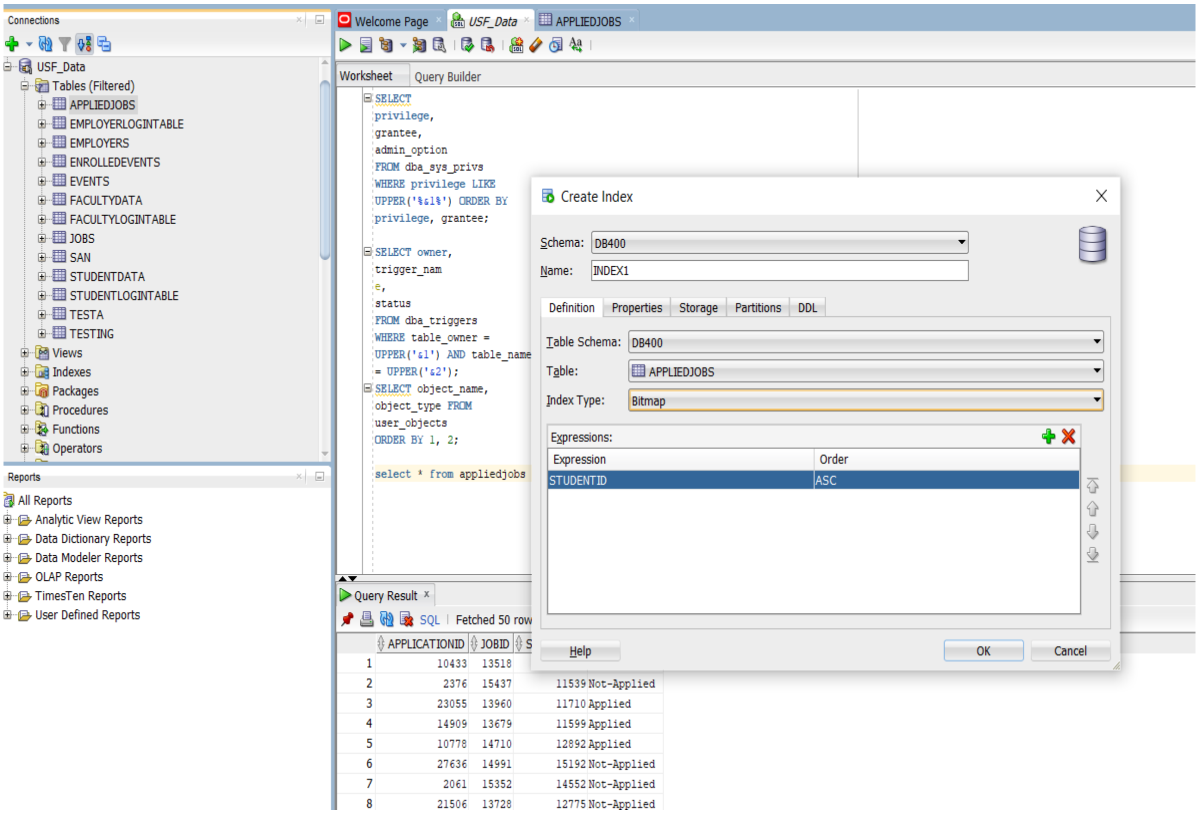
```
create or replace PROCEDURE ABV_TOP
IS
ABV_VAL VARCHAR2(30);
BEGIN
SELECT studentid INTO ABV_VAL FROM studentdata FETCH FIRST 1 ROWS ONLY;
DBMS_OUTPUT.PUT_LINE(ABV_VAL);
END ABV_TOP;
```

Performance Tuning

INDEX

An index is used to increase the overall performance of queries. Indexing does this by reducing the data pages that has to be visited or scanned every time a query is run.

When we create index, by default the primary key creates a clustered index. In SQL Server, a clustered index determines the physical order of data in a table. There can be only one clustered index per table.



```
SELECT
privilege,
grantee,
admin_option
FROM dba_sys_privs
WHERE privilege LIKE
UPPER('%&1%') ORDER BY
privilege, grantee;

SELECT owner,
trigger_nam
e,
status
FROM dba_triggers
WHERE table_owner =
UPPER('&1') AND table_name
= UPPER('&2');
SELECT object_name,
object_type FROM
user_objects
ORDER BY 1, 2;

select * from appliedjobs
```

Create Index

Schema: DB400

Name: INDEX1

Definition | Properties | Storage | Partitions | DDL

Table Schema: DB400

Table: APPLIEDJOBS

Index Type: Bitmap

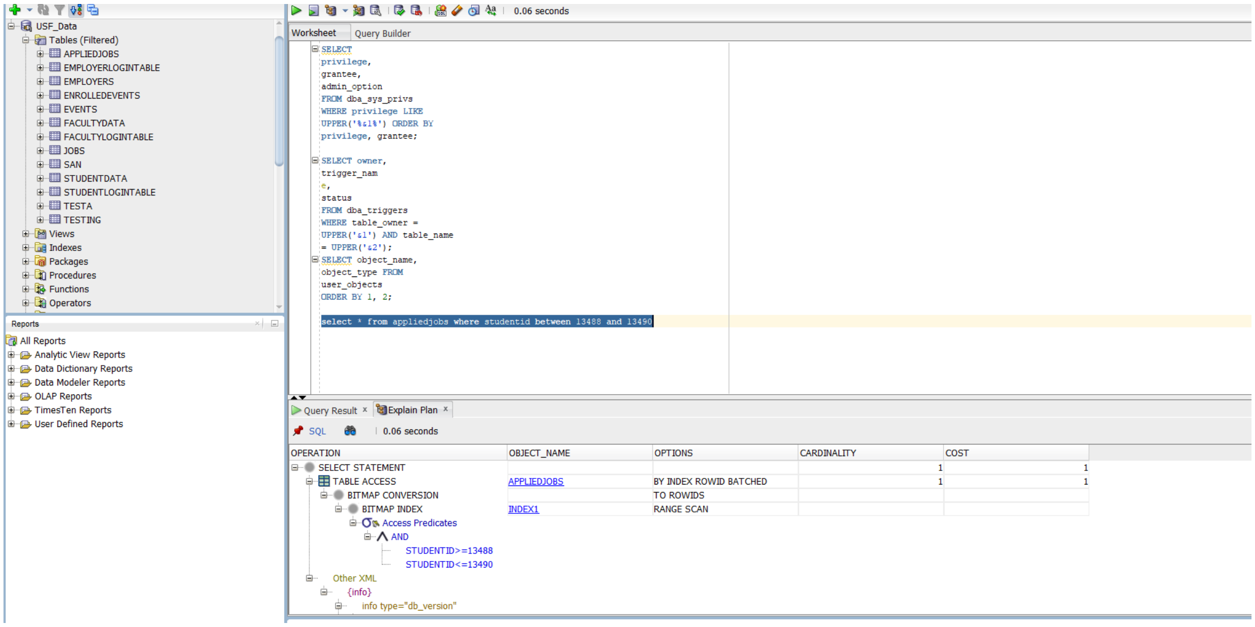
Expressions:

Expression	Order
STUDENTID	ASC

Fetched 50 row

	APPLICATIONID	JOBID	S	
1	10433	13518		
2	2376	15437	11539	Not-Applied
3	23055	13960	11710	Applied
4	14909	13679	11599	Applied
5	10778	14710	12892	Applied
6	27636	14991	15192	Not-Applied
7	2061	15352	14552	Not-Applied
8	21506	13728	12775	Not-Applied

Execution Plan of Index:



```
SELECT
privilege,
grantee,
admin_option
FROM dba_sys_privs
WHERE privilege LIKE
UPPER('%&1%') ORDER BY
privilege, grantee;

SELECT owner,
trigger_nam
e,
status
FROM dba_triggers
WHERE table_owner =
UPPER('&1') AND table_name
= UPPER('&2');
SELECT object_name,
object_type FROM
user_objects
ORDER BY 1, 2;

select * from appliedjobs where studentid between 13488 and 13490
```

0.06 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	1
TABLE ACCESS	APPLIEDJOBS	BY INDEX ROWID BATCHED	1	1
BITMAP CONVERSION		TO ROWIDS		
BITMAP INDEX	INDEX1	RANGE SCAN		
Access Predicates				
AND				
STUDENTID>=13488				
STUDENTID<=13490				
Other XML				
{info}				
info type="db_version"				

<u>Query :</u> select * from appliedjobs where studentid between 13488 and 13490

Parallelism:

SELECT /*+ PARALLEL (4) */ a.studentid,b.studentname

FROM

appliedjobs a, studentdata b WHERE a.studentid = b.studentid

- Given two sets of parallel execution servers S1 and S2 for the query plan in the above query the execution proceeds as follows: each server set (S1 and S2) has four execution processes because of the PARALLEL hint in the query that specifies the DOP.

- The set S1 first scans the table appliedjobs and sends rows to SS2, which builds a hash table on the rows.

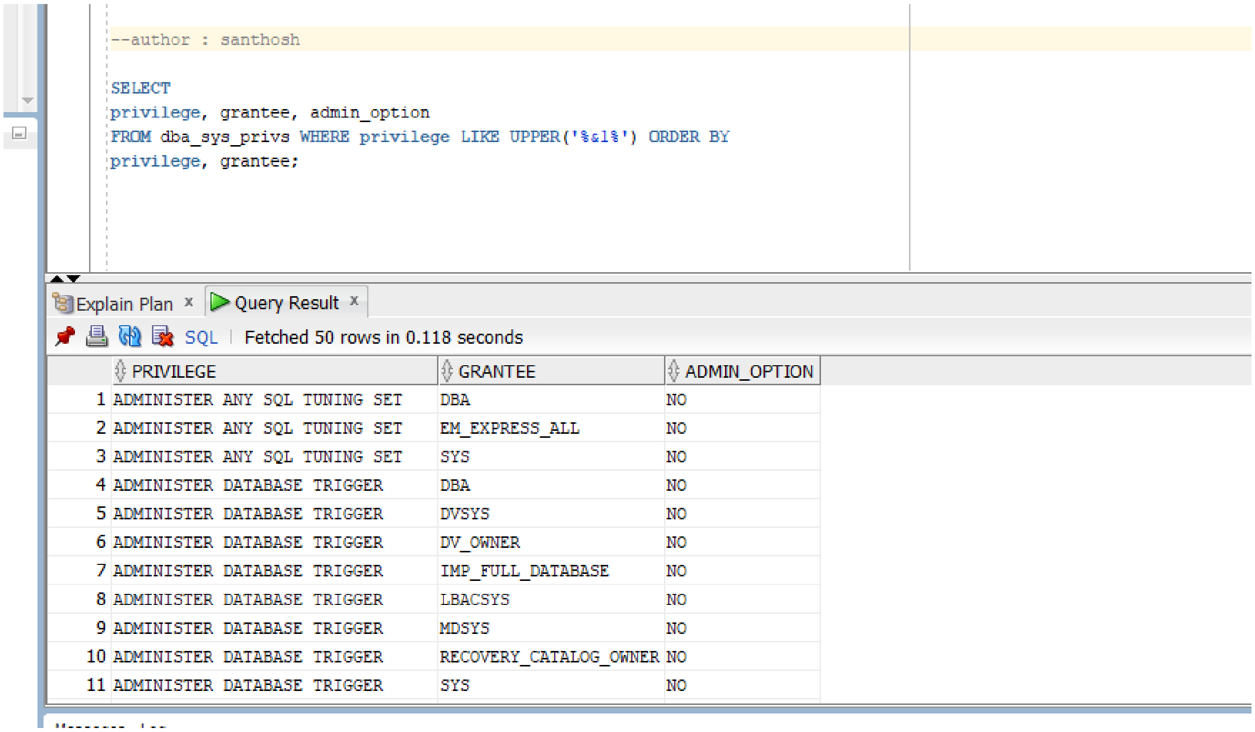
- After S1 has finished scanning the entire appliedjobs table, it scans the studentdata table in parallel.

- It sends its rows to servers in s2, which them finish the hash join in parallel. After S1 has scanned the studentdata table in parallel and sent the rows to S2, This is how two server sets run concurrently to achieve inter-operation parallelism across various operators in the query tree.

DBA SCRIPTS

```
SELECT
privilege, grantee, admin_option
FROM dba_sys_privs WHERE privilege LIKE UPPER('%&1%') ORDER BY
privilege, grantee;
```

Description: The above sql which access dba_sys_privs and displays the users granted the specified system privilege.



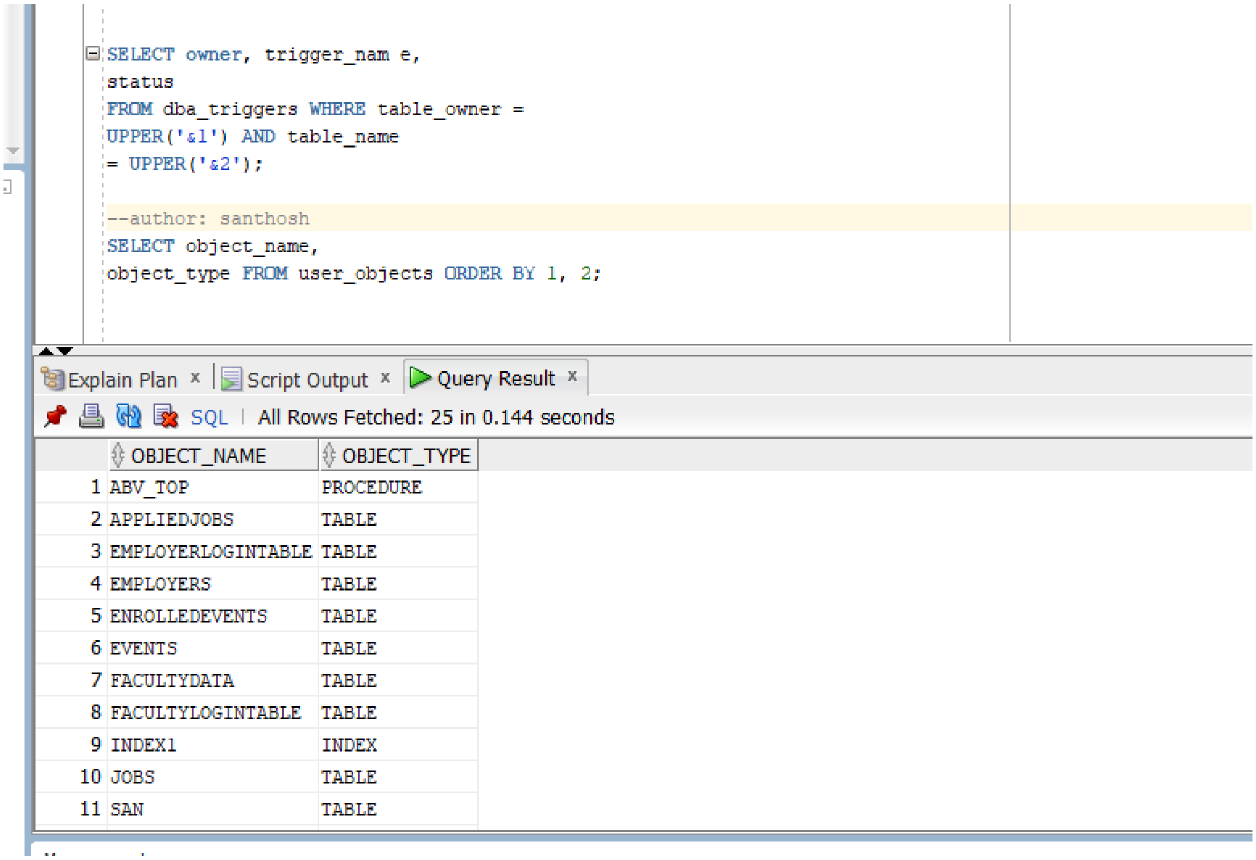
```
--author : santhosh

SELECT
privilege, grantee, admin_option
FROM dba_sys_privs WHERE privilege LIKE UPPER('%&1%') ORDER BY
privilege, grantee;
```

Fetched 50 rows in 0.118 seconds

	PRIVILEGE	GRANTEE	ADMIN_OPTION
1	ADMINISTER ANY SQL TUNING SET	DBA	NO
2	ADMINISTER ANY SQL TUNING SET	EM_EXPRESS_ALL	NO
3	ADMINISTER ANY SQL TUNING SET	SYS	NO
4	ADMINISTER DATABASE TRIGGER	DBA	NO
5	ADMINISTER DATABASE TRIGGER	DVSYS	NO
6	ADMINISTER DATABASE TRIGGER	DV_OWNER	NO
7	ADMINISTER DATABASE TRIGGER	IMP_FULL_DATABASE	NO
8	ADMINISTER DATABASE TRIGGER	LBACSYS	NO
9	ADMINISTER DATABASE TRIGGER	MDSYS	NO
10	ADMINISTER DATABASE TRIGGER	RECOVERY_CATALOG_OWNER	NO
11	ADMINISTER DATABASE TRIGGER	SYS	NO

```
SELECT object_name,
object_type FROM user_objects ORDER BY 1, 2;
```

Description: The above sql which access user_objects and displays object name and object type for current user.



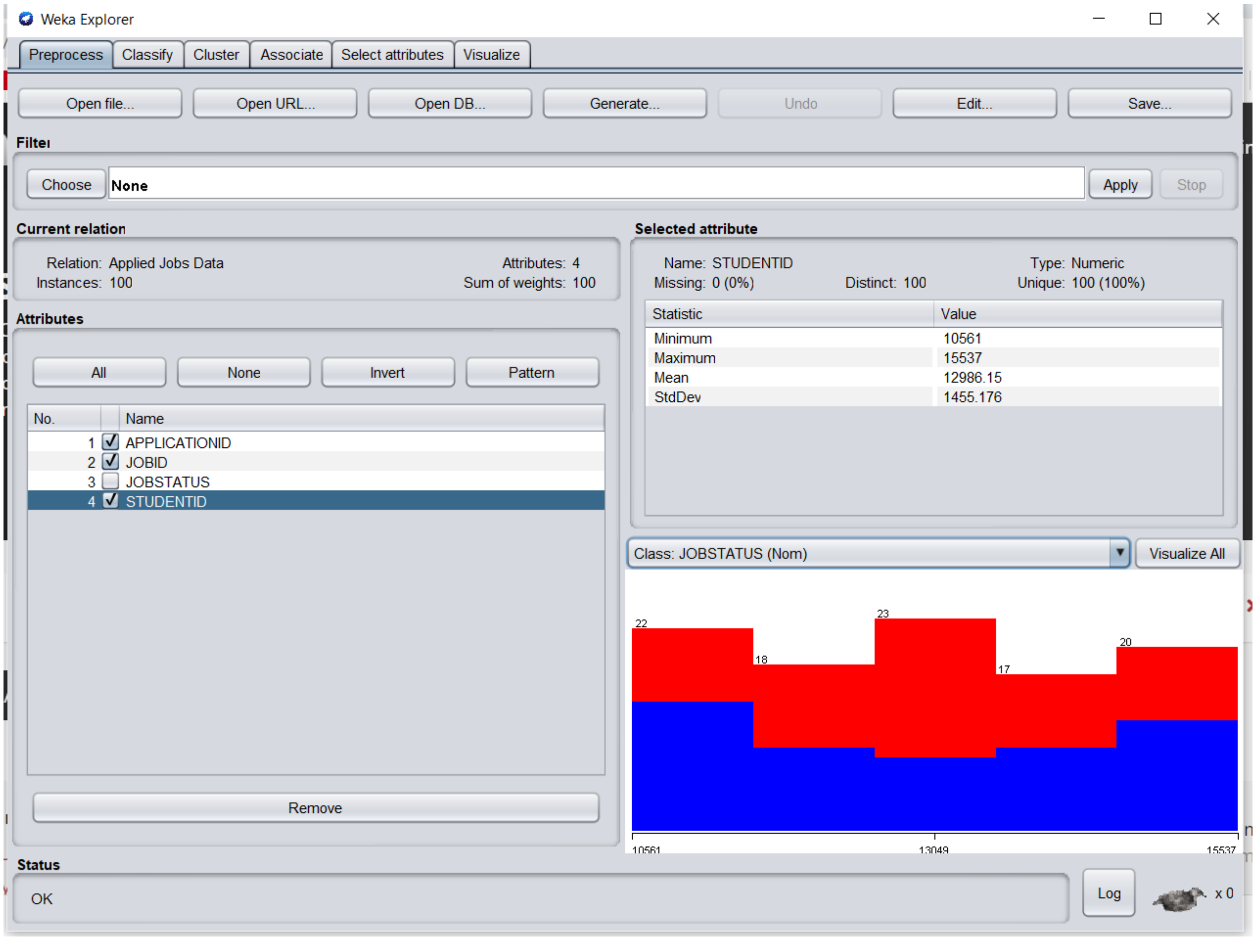
```
SELECT owner, trigger_nam e,
status
FROM dba_triggers WHERE table_owner =
UPPER('&1') AND table_name
= UPPER('&2');

--author: santhosh
SELECT object_name,
object_type FROM user_objects ORDER BY 1, 2;
```

All Rows Fetched: 25 in 0.144 seconds

	OBJECT_NAME	OBJECT_TYPE
1	ABV_TOP	PROCEDURE
2	APPLIEDJOBS	TABLE
3	EMPLOYERLOGINTABLE	TABLE
4	EMPLOYERS	TABLE
5	ENROLLEDEVENTS	TABLE
6	EVENTS	TABLE
7	FACULTYDATA	TABLE
8	FACULTYLOGINTABLE	TABLE
9	INDEX1	INDEX
10	JOBS	TABLE
11	SAN	TABLE

Data Mining in Weka

Description: we have also loaded few of our datasets in weka and performed feature engineering using some data mining tools available in weka and below screenshot shows the gradient distribution of datapoints for our data base.



Filter

Choose None Apply Stop

Current relation

Relation: Applied Jobs Data Attributes: 4
Instances: 100 Sum of weights: 100

Selected attribute

Name: STUDENTID Type: Numeric
Missing: 0 (0%) Distinct: 100 Unique: 100 (100%)

Statistic	Value
Minimum	10561
Maximum	15537
Mean	12986.15
StdDev	1455.176

Attributes

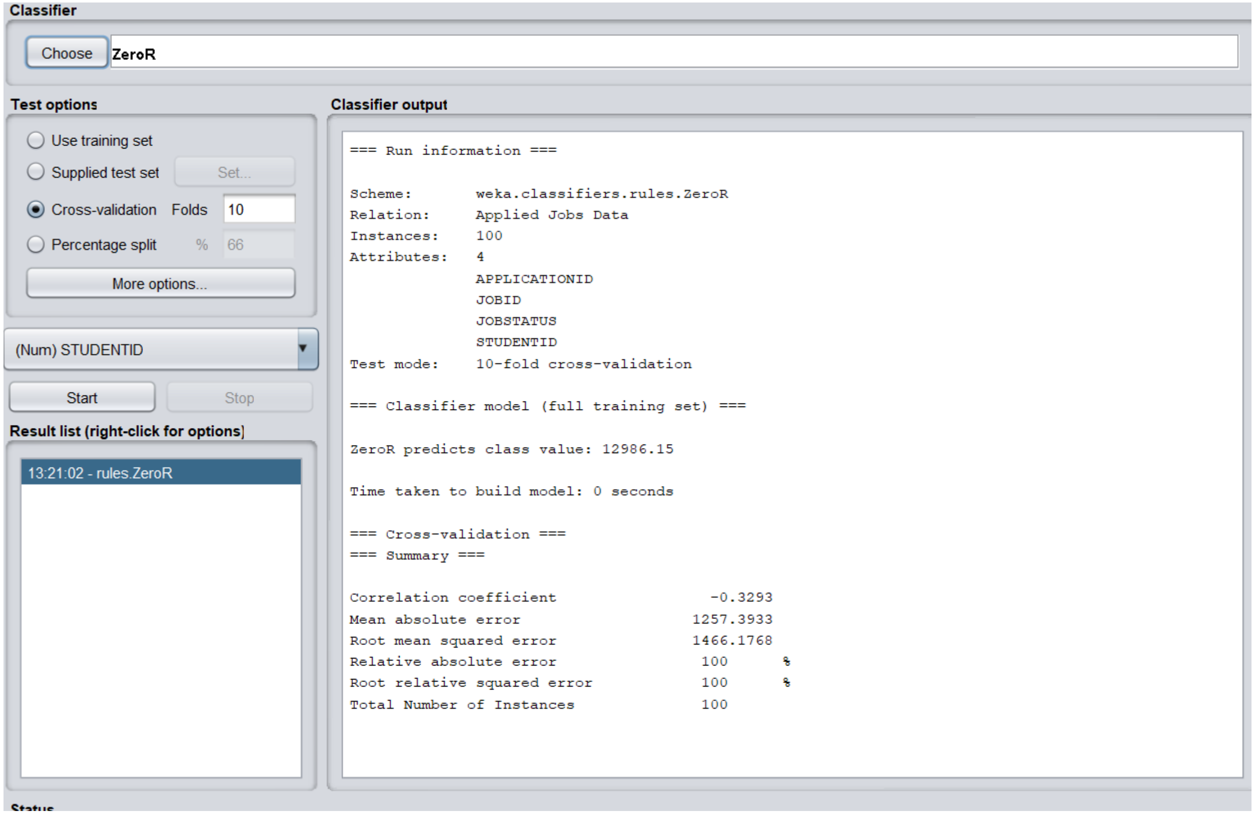
No.		Name
1	☑	APPLICATIONID
2	☑	JOBID
3	□	JOBSTATUS
4	☑	STUDENTID

Class: JOBSTATUS (Nom)

Status

OK

Classification:
We have also performed classification in Weka using algorithms like zeroR for one of our datasets to know the mean square error and standard deviations in the dataset as well



Classifier

ZeroR

Test options

● Cross-validation Folds 10

(Num) STUDENTID

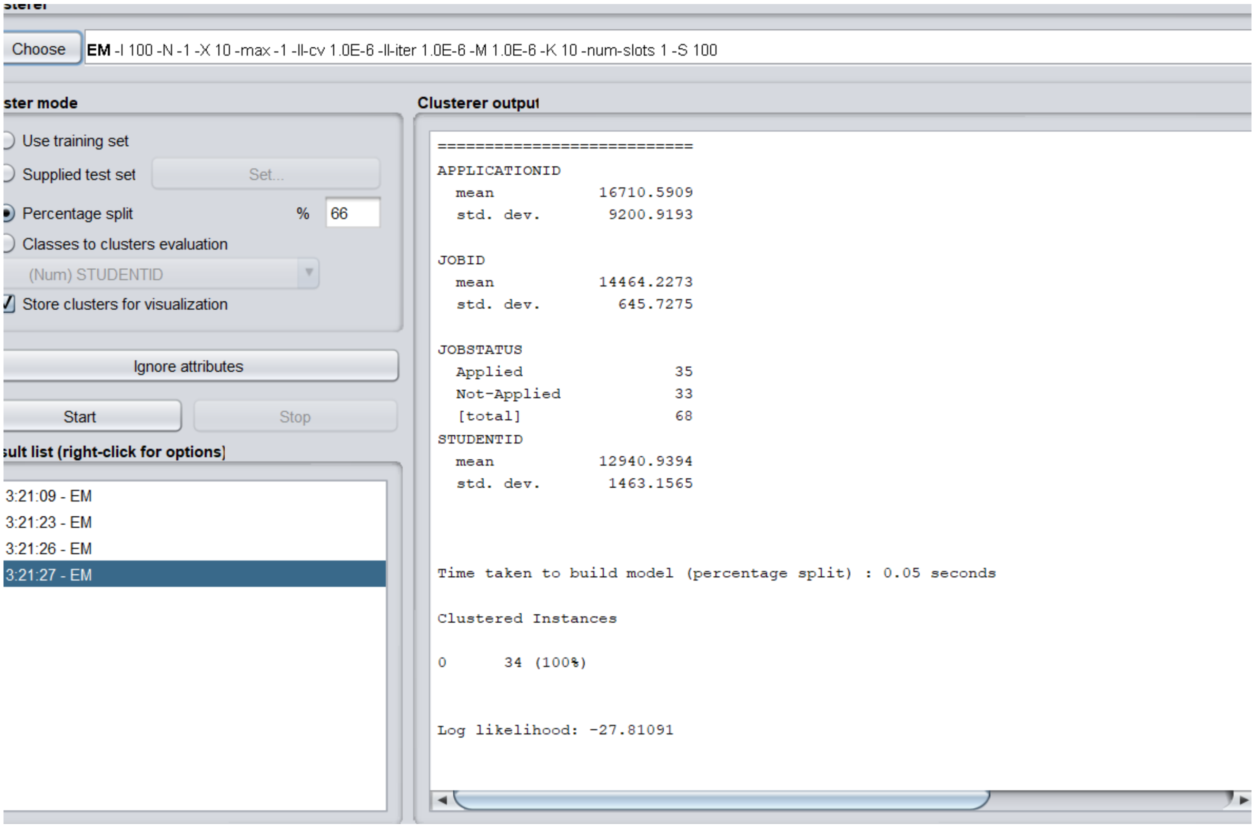
Result list (right-click for options)

13:21:02 - rules.ZeroR

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.rules.ZeroR
Relation:     Applied Jobs Data
Instances:    100
Attributes:   4
              APPLICATIONID
              JOBID
              JOBSTATUS
              STUDENTID
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

ZeroR predicts class value: 12986.15

Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                 -0.3293
Mean absolute error                   1257.3933
Root mean squared error               1466.1768
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances              100
```

Clustering:
This is performed inorder to know the similar instances of data attributes in our data



EM -I 100 -N -1 -X 10 -max -1 -ll-cv 1.0E-6 -ll-iter 1.0E-6 -M 1.0E-6 -K 10 -num-slots 1 -S 100

● Percentage split % 66

☑ Store clusters for visualization

```
===========================
APPLICATIONID
  mean          16710.5909
  std. dev.      9200.9193

JOBID
  mean          14464.2273
  std. dev.       645.7275

JOBSTATUS
  Applied               35
  Not-Applied           33
  [total]               68
STUDENTID
  mean          12940.9394
  std. dev.      1463.1565



Time taken to build model (percentage split) : 0.05 seconds

Clustered Instances

0       34 (100%)


Log likelihood: -27.81091
```