CSE 601- DataMining and BioInformatics

Project-1

Uttara Asthana-50168804

Contents

1.INTRODUCTION:

Data warehousing technology was originally developed and used in business context in support of management decisions. Online Analytical Processing gives users a multidimensional view of the data in the data warehouse and facilitates the analysis of data. There is a huge amount of genomic and clinical data being generated every day in the field of biomedicine. The size and complex nature of the dataset gathered makes datamining operations and handling of the data extremely challenging. To overcome this, data warehousing technology is beginning to be used in biomedicine and other biological sciences fields.

Though the concepts of data warehousing and OLAP have been successful in business decisions and applications, applying this as it is to biological and biomedical data will be impractical.

In our project, we have implemented a clinical and genomic data warehouse based on biostar as well as our proposed schema in earlier assignment using MYSQL as the relational database.

The input data for schema design was provided as flat text files under the directory /projects/azhang/cse601/Data_For_Project1/. This data comprises of 5 data spaces rendered in the form of a star schema. Each data space has a fact table and connected dimension tables.

Though the concepts of data warehousing [generally star schema and snow flake schema] and OLAP have been successful in business decisions and applications, applying this as it is to biological and biomedical data will be impractical.

<u>Need for a new Schema:</u>

Star and Snowflake schema design work well with business data that have relationship between a business fact and dimension [n: 1].

In our case, for biomedical data given the number of dimensions will be large and the fact table which is a combination of foreign keys grows unmanageable in size. Another thing that needs to be considered is that as the dimensions evolve, adding a new dimension to the start table mandates re-computing the fact table. **Clinical data values may be incomplete** resulting in the fact table containing a lot of null values for foreign keys. This results in inconsistency.

The idea is to implement a schema applicable even in case of complex biomedical data. In this project we have implemented a combination of biostar and bioflake [our proposed schema] for better query performance.

2. <u>Dataset</u>

The dataset we have used contains modeling clinical and genomic data at the conceptual level. Based on the diversity and nature of the biomedical dataset, the design of the data warehouse may include several modeling data spaces. We have based our model on the following six dataspaces: clinical dataspace, sample dataspace, microarray and proteomic data space, and experiment dataspace and gene space.

a. The Clinical Dataspace has patient, drug, disease and test as its entities.
b. The Sample Dataspace has sample, marker, assay and term as its entities.
c. The Microarray and Proteomic Dataspace has probe, measure Unit and microarray_fact as its entities.
d. The Gene Dataspace has gene, go, cluster, domain, promoter as its entities.
e. The Experiment Dataspace has experiment, project, platform, norm, person, protocal, publication as its entities.

We have modeled our schema on the dataset given in the paper [1]

3. Implementation Details:

Relational Database Used: MySQL6.1

Other tools/ programming Languages used: R ,PHP and Google chart API

We have used MySQL as the relational database management system for creating our schema. This is an open source tool. We have populated the dataset as part of Part I in MySQL. The queries 1, 2, 3 in part II of the project description are implemented using Structured Query Language [SQL].

In Part II, from Query 4 onwards, we perform statistical operations on the retrieved queries. MySQL does not have inbuilt functions to calculate t-statistics, F-statistics. To alleviate this problem, we have connected our schema to R using the 'RMySQL' package in R. The calculations are performed in R.

In Part III 1, we have used the data warehouse and statistical operations to find the informative genes given a specific disease. To achieve this, we have used MySQL to find the patients with a disease and all others without it. Then we have written a R script to calculate t-statistics for the expression values between Group A and Group B. Based on the p-value we classify the gene as informative or not.

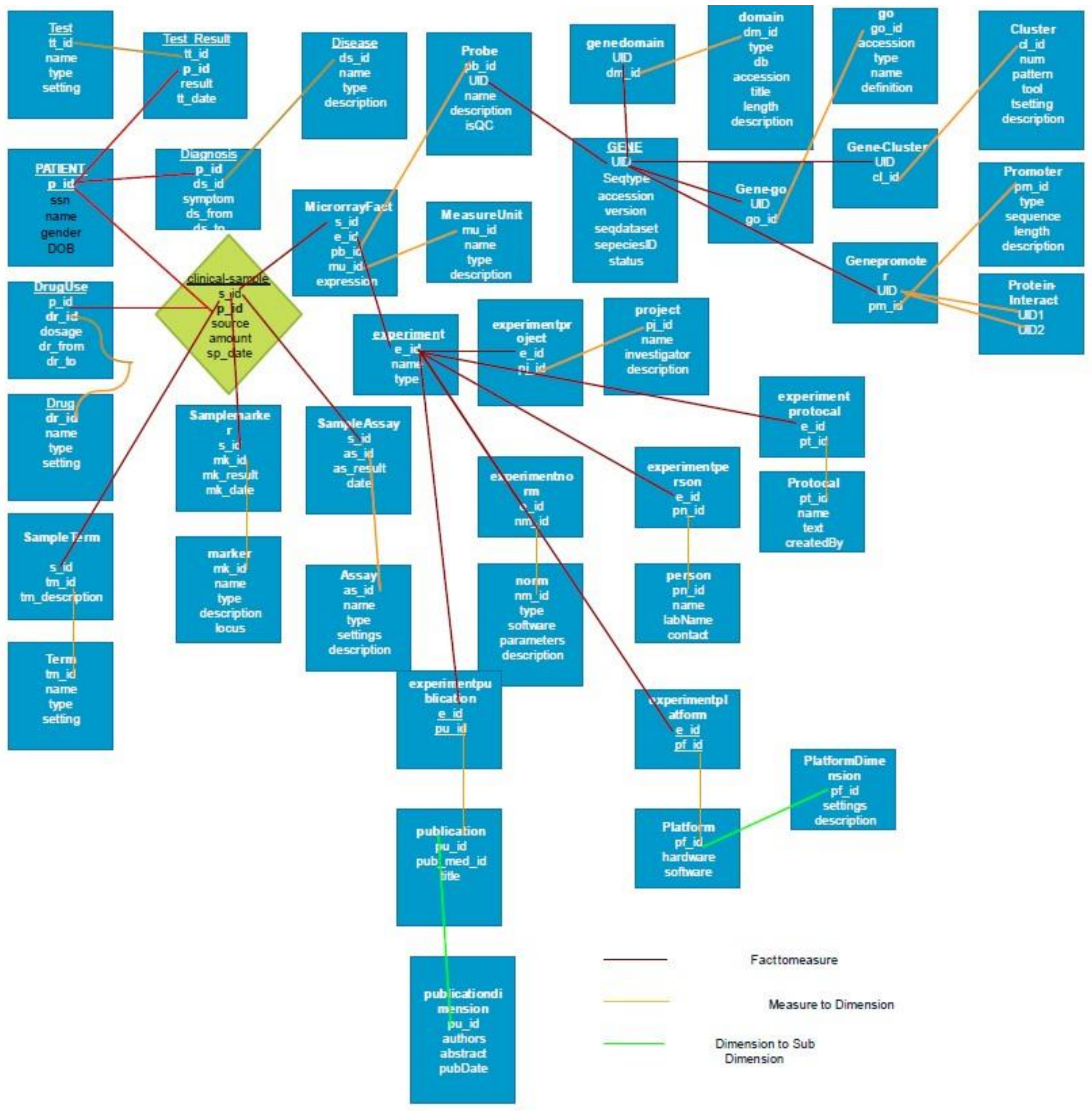
Based on the informative genes obtained we classify new patients as with or without a particular disease. The information of the test cases is provided in the test_samples.txt file of the input dataset. This classification is achieved using an R script too. We have developed the front end using PHP. Through our UI, we provide the user flexibility to find information similar to the standard sample Queries by choosing parameters of his/her choice. We have used Google chart API's for visualizing the results obtained for further use and data analysis.

PART 1:

3.1 A] IMPLEMENTATION OF DATA WAREHOUSE AND POPULATING SCHEMA

The definition of the five data spaces provided to us in the **data set is based on star schema and fact constellation schema**. Though this **data ware house design has its own benefits it is not adaptable to future changes , say adding/ deleting any dimension in the schema would demand update in the fact tables** which have the links to multiple dimension tables in the given data space. The fact tables act like a central entity for the given data spaces, this justifies its comparatively greater table size; hence making updates in it would come with a large over- head. However this kind of data ware house design are more efficient and optimized for large data sets and the Queries in such system are usually less complex. Hence **we have followed this schema design logic where Query complexity and read time is a priority. Besides we have also implemented a hybrid of bio star and snow flake schema .Our design is based on the schema design logic proposed by us in assignment 1 , due to its complexity we have modified our proposed design so as to suite our data requirements and used it for some basic queries .** This will also make the data warehouse more flexible for future use, from the point of view of incorporating new data or integrating the existing model with the newly based changes. Detailed description is given in the later parts of the report as to which tables, or partition tables created for a particular Query were utilized to add efficiency to Query operation.

<u>**Following is the design of the schema implemented by us,**</u>


Test tt_id name type setting
Test_Result tt_id p_id result tt_date
Disease ds_id name type description
Probe pb_id UID name description isQC
genedomain UID dm_id
domain dm_id type db accession title length description
go go_id accession type name definition
Cluster cl_id num pattern tool tsetting description
PATIENT p_id ssn name gender DOB
Diagnosis p_id ds_id symptom ds_from ds_to
MicrorrayFact s_id e_id pb_id mu_id expression
MeasureUnit mu_id name type description
GENE UID Seqtype accession version seqdataset sepeciesID status
Gene-go UID go_id
Gene-Cluster UID cl_id
Promoter pm_id type sequence length description
Genepromoter UID pm_id
Protein-Interact UID1 UID2
DrugUse p_id dr_id dosage dr_from dr_to
clinical-sample s_id p_id source amount sp_date
experiment e_id name type
experimentproject e_id pj_id
project pj_id name investigator description
experiment protocal e_id pt_id
Drug dr_id name type setting
Samplemarker s_id mk_id mk_result mk_date
SampleAssay s_id as_id as_result date
experimentnorm e_id nm_id
experimentperson e_id pn_id
Protocal pt_id name text createdBy
SampleTerm s_id tm_id tm_description
marker mk_id name type description locus
Assay as_id name type settings description
norm nm_id type software parameters description
person pn_id name labName contact
Term tm_id name type setting
experimentpublication e_id pu_id
experimentplatform e_id pf_id
PlatformDimension pf_id settings description
publication pu_id pub_med_id title
Platform pf_id hardware software
publicationdimension pu_id authors abstract pubDate
Facttomeasure
Measure to Dimension
Dimension to Sub Dimension

DataSpace Name	Table Name
Clinical Dataspace	Clinical Sample
Clinical Dataspace	Disease
Clinical Dataspace	Drug
Clinical Dataspace	Patient
Clinical Dataspace	Test
Clinical Dataspace	DrugUse
Clinical Dataspace	Diagnosis
Clinical Dataspace	TestResult
Sample Dataspace	Samplemarker
Sample Dataspace	Marker

Sample Dataspace	Sampleassay
Sample Dataspace	Assay
Sample Dataspace	Term
Sample Dataspace	sampleterm
Microarrayfact Dataspace	probe
Microarrayfact Dataspace	MeasureUnit
Microarrayfact Dataspace	Microarray_fact
Gene Dataspace	gene
Gene Dataspace	go
Gene Dataspace	cluster
Gene Dataspace	domain
Gene Dataspace	promoter
Gene Dataspace	Gene-go
Gene Dataspace	Gene-cluster
Gene Dataspace	Gene-domain
Gene Dataspace	Gene-promoter
Experiment Dataspace	experiment
Experiment Dataspace	project
Experiment Dataspace	platform
Experiment Dataspace	norm
Experiment Dataspace	person
Experiment Dataspace	protocal
Experiment Dataspace	publication
Experiment Dataspace	Experiment-project
Experiment Dataspace	Experiment-platform
Experiment Dataspace	Platform-Dimension
Experiment Dataspace	Experiment-norm
Experiment Dataspace	Experiment-person
Experiment Dataspace	Experiment-publication
Experiment Dataspace	Publication-dimension

As we can see **diagnosis drug use, test result are measure tables in the clinical data space connecting to the dimension tables disease, drug and test respectively**. For example, the measure table diagnosis has a patient id as foreign key which connects it to the patient table that acts as a central entity in the clinical sample data space and also had ds_id as the foreign key which establishes its relation with the disease dimension table. All the measure table in clinical data space have the primary key of patient table as foreign key in them thus establishing their relationship with the central entity and have the primary key of the dimension table they are connected to as the foreign key. The clinical sample table connects the Clinical data space to the sample data space. As the **disease name is frequently queried we have also included it in the diagnosis measure table so as to decrease the read time (Data denormalization).** Similarly samplemarker, sampleassay, sampleterm are the measure tables connecting to the dimension tables marker, assay and term respectively. A star schema is simply followed in the microarray data space, the microarray_fact table is directly connected to the probe and measureUnit dimension tables. The microarray_fact table connects further to the experiment table which acts as the central entity for the experiment data space. Experimentplatform, experimentnorm, experimentperson, experimentprotocal and experimentpublication act as the measure tables for the platform, norm, person and publication dimension respectively. The probe dimension table in the microarray data space star schema is connected to the gene table which is the central entity for the gene data space. Genego, genecluster, genedomain and genepromoter are the measure tables for the platform, norm, person, protocal and publication dimension table. We have also implemented sub dimensioning in the experiment data space for publication and platform tables so as to save memory and store non-frequently queried data.

3.2 B.] PARTITION CREATED FOR DATA CLEANING

For a quick look up we have created **the pasadi partition table and also the pasadi_nonull partition table. These table have p_id (patient_id), s_id (sample id) and ds_id (ds_id) and disease name columns.** Thus this partition stores the ds id associated with a given sample of patient. We have created and populated this partition table during the initial data base design in order to improve the Query performance. Thus by creating this partition we are improving the query complexity involved, besides in this table we have also taken care of the incomplete data. There are missing entries in one table which can be inferred from other tables in the given data set. For example if we consider the clinical_fact table we have 7 records for the patient with p_id = “6413” as follows,

```
"P_ID" "DS_ID" "SYMPTON" "DS_FROM_DATE" "DS_TO_DATE" "DR_ID" "DOSAGE" "DR_FROM_DATE"
"DR_TO_DATE" "TT_ID" "RESULT" "TT_DATE" "S_ID"
"6413" "5" "Sympton of Breast tumor" "1/15/2000" "12/31/2003" "80573" "10" "1/15/2000" "1/15/2001" "null"
"null" "null" "null"
"6413" "null" "null" "null" "null" "22055" "1" "1/15/2001" "1/15/2002" "null" "null" "null" "null"
"6413" "null" "null" "null" "null" "84608" "4" "1/15/2002" "1/15/2003" "null" "null" "null" "null"
"6413" "null" "null" "null" "null" "null" "null" "null" "null" "745" "Result of test" "1/15/2000" "null"
"6413" "null" "null" "null" "null" "null" "null" "null" "null" "796" "Result of test" "1/15/2001" "null"
"6413" "null" "null" "null" "null" "null" "null" "null" "null" "817" "Result of test" "1/15/2002" "null"
"6413" "null" "null" "null" "null" "null" "null" "null" "null" "null" "null" "null" "13834"
```

For the given small data set it is verified that each patient has a single disease id and sample id associated with it .
Thus in case of records like,

P_id	ds_id	s_id
6413	5	null
6413	null	null
6413	null	1384
6414	4	null
6414	null	null

6414 null 1385

We have handled such cases by filling up the incomplete data. We have handled this by creating two partition tables pasadi table and pasadi_no null table . pasadi table contains the entries of type where there are null values along with the corrected tuple.

The table pasadi would contain records like,

P_id ds_id s_id disease name

```
6413   5       null disease5
6413 null null
6413 null 1384
6513  5       1384    disease5
6414   4         null   disease4
6414   null     null
6414   null   1385
5414    4      1385 disease4
```

Where as the pasadi_no null would contain only those records which do not have the null vale,

P_id ds_id s_id disease name

```
6413   5        1384   disease5
6414   4        1385   disease4
```

The reason for creating separate table is in case where we want to query disease id not like 5 the pasadi table would return patient 6413 due to the second tuple which is incorrect. At the same time we cannot ignore null values in some cases and hence have retained the pasadi partition.

For example, select * from pasadi table where p_id =6413

Would return,

P_id ds_id s_id disease name

```
6413   5       null  disease5
6413 null null
6413 null 1384
6513  5       1384 disease5
```

If say we want to query for ds_id 5, the tuple returned should be the last tuple and care is taken that the tuples containing null vale in ds_id are not retuned is handled during query time. However for cases where we want to query for ds_id not 5 then we have to use the pasadi_no null table

SQL Query used for creating the partition table pasadi :

```
create table sys.pasadi select distinct sys.clinical_fact.p_id, s_id, sys.clinical_fact.ds_id, name from sys.clinical_fact left join sys.disease on sys.disease.ds_id=sys.clinical_fact.ds_id union all select distinct  sys.clinical_fact.p_id, s_id, sys.clinical_fact.ds_id, name from sys.clinical_fact right join sys.disease on sys.disease.ds_id=sys.clinical_fact.ds_id
```

SQL Query used for creating the partition table pasadi_nonull :

```
create table sys.pasadi_nonull select * from sys.pasadi where ds_id not like 'null'
```

3.3 Part II: Regular and statistical OLAP operations

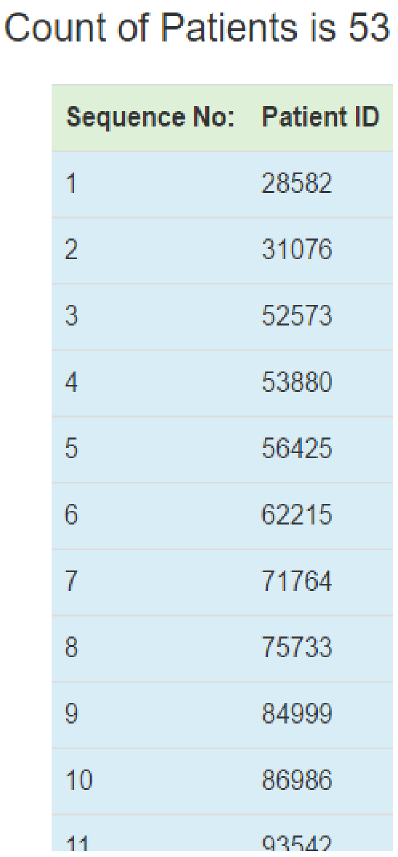
Query 1: List the number of patients who had “ tumor” (disease description), “ leukemia” (disease type) and “ ALL” (disease name), separately

We have used our hybrid schema to retrieve the results for this query.

Description type Tumor:

SQL Query: select count(distinct a.p_id) from sys.diagnosis as a inner join sys.disease as b on b.ds_id=a.ds_id where b.descriptionl like 'tumor'

Result:



Count of Patients is 53

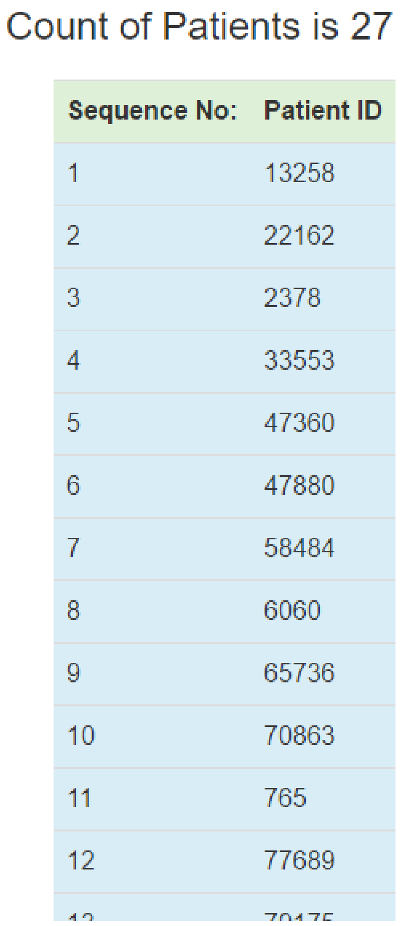
Sequence No:	Patient ID
1	28582
2	31076
3	52573
4	53880
5	56425
6	62215
7	71764
8	75733
9	84999
10	86986
11	93542

An inner join on the diagnosis measure table and the disease table where the description attribute has the value tumor, would result the patient ids which have tumor in their disease description. We have used the count() Sql function to count the number of patient id’s the join operation returns.

Disease type leukemia:

SQL Query: select count(distinct a.p_id) from sys.diagnosis as a inner join sys.disease as b on b.ds_id=a.ds_id where b.type like 'leukemia'

Result:



Count of Patients is 27

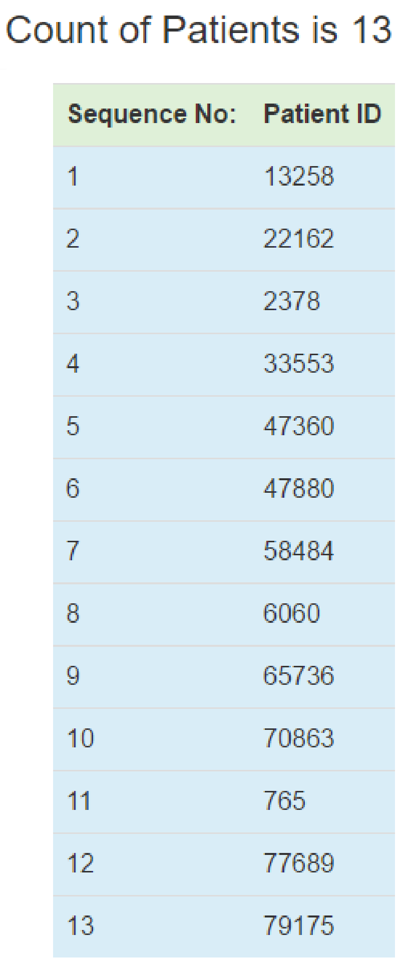
Sequence No:	Patient ID
1	13258
2	22162
3	2378
4	33553
5	47360
6	47880
7	58484
8	6060
9	65736
10	70863
11	765
12	77689
13	79175

An inner join on the diagnosis measure table and the disease table where the type attribute has the value leukemia, would result the patient ids which have leukemia as their disease type. We have used the count() Sql function to count the number of patient id's the join operation returns.

Disease name ALL:

SQL Query: select count(distinct a.p_id) from sys.pasadi as a where a.name like 'ALL'

Result:

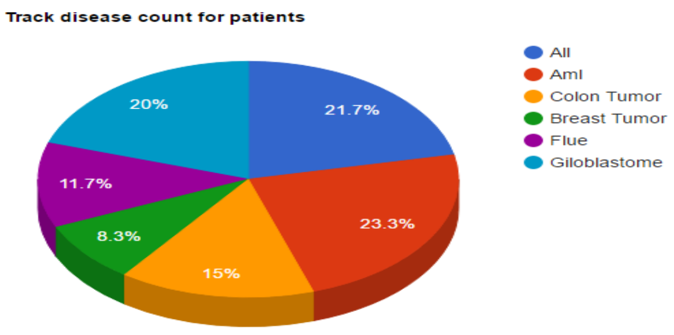


Count of Patients is 13

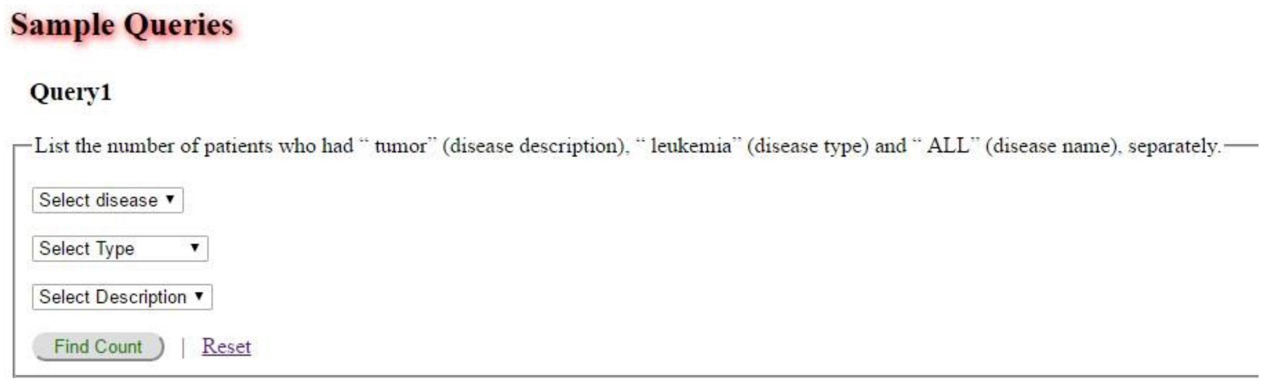
Sequence No:	Patient ID
1	13258
2	22162
3	2378
4	33553
5	47360
6	47880
7	58484
8	6060
9	65736
10	70863
11	765
12	77689
13	79175

We have used the partition table ‘pasadi’ to retrieve the patient id’s who had disease ALL. SQL function count is used to return the number of such patients.

Visual Analytics: Pie chart for count of patients associated with each disease



We have modified our UI for Query 1 to display the number of patients associated with any disease name, type or Description



Sample Queries

Query1

List the number of patients who had " tumor" (disease description), " leukemia" (disease type) and " ALL" (disease name), separately.

Query complexity:

In the above Queries the **third Query (for ALL) will have the least complexity O(N)** where N is the size of the table 'pasadi' as it is simply a select statement from table and as all N records would have to be searched. The **first two queries** are inner joins and hence would be more complex. As for these queries both the tables have index on the joined column the complexity would be around **O(M+N) where M,N** are the size of the two tables on whom the join is implemented.

Query 2: List the types of drugs which have been applied to patients with " tumor".

We have used our hybrid schema to retrieve the results for this query.

SQL Query:

We have attempted to implement this query in two ways, we have stored the result obtained from first Query so as to optimize the Query run time process. We have stored the results for patients having tumor as their disease description in table 'tumor_pid' and used it to retrieve the patient id with disease description tumor.

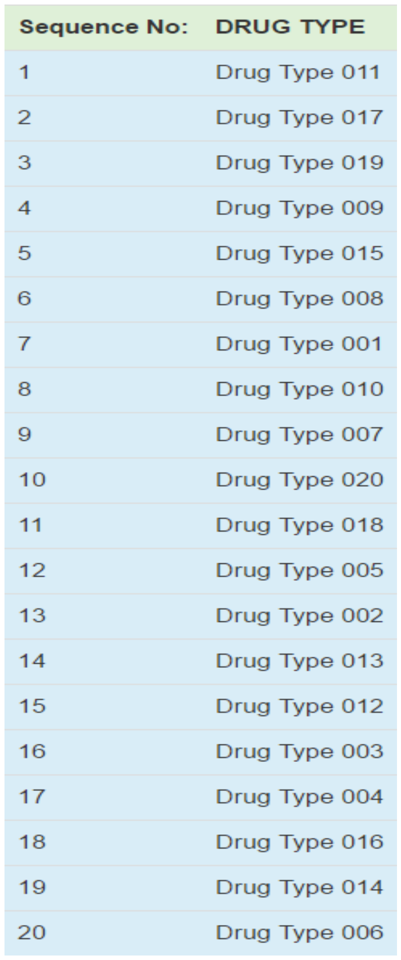
select distinct type from sys.drug as e inner join (select distinct dr1_id from sys.druguse as d inner join sys.tumor_pid as c on c.p_id=d.drugp_id) as f where f.dr1_id=e.dr_id

OR

Besides these query results can also be derived from our hybrid schema as follows,

select distinct type from sys.drug as e inner join (select distinct dr1_id from sys.druguse as d inner join (select distinct a.p_id from sys.diagnosis as a inner join sys.disease as b on b.ds_id=a.ds_id where b.descriptionl like 'tumor') as c on c.p_id=d.drugp_id) as f where f.dr1_id=e.dr_id

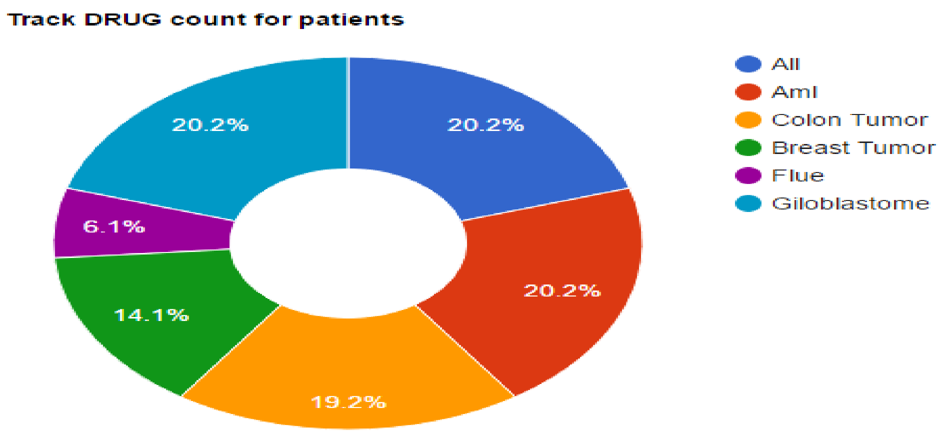
Result:



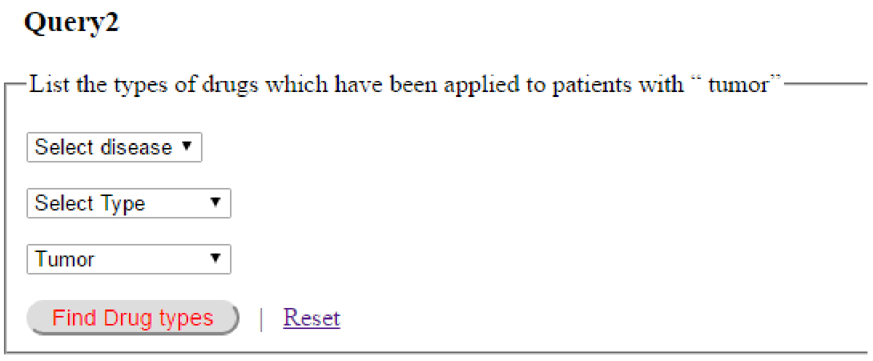
Sequence No:	DRUG TYPE
1	Drug Type 011
2	Drug Type 017
3	Drug Type 019
4	Drug Type 009
5	Drug Type 015
6	Drug Type 008
7	Drug Type 001
8	Drug Type 010
9	Drug Type 007
10	Drug Type 020
11	Drug Type 018
12	Drug Type 005
13	Drug Type 002
14	Drug Type 013
15	Drug Type 012
16	Drug Type 003
17	Drug Type 004
18	Drug Type 016
19	Drug Type 014
20	Drug Type 006

We first implemented an iner join on diagnosis and disease table to get the patient id's of those having tumor as their disease description, then performed an inner join of these selectin on druguse table for patient id's and later retrieved the grug type of those patient id's from the drug table with the help of drug use.

Visual Analytics: Drug count associated with each patient id.



We have modified this to display drug types associated with any disease name, type or description in our UI.



Query2

List the types of drugs which have been applied to patients with " tumor"

Query Complexity:

As the Query consists of an inner join on an index common to both the tables the Query complexity would be 0(M+N) where M,N is the size of the two tables respectively.

Query 3: For each sample of patients with " ALL", list the mRNA values (expression) of probes in cluster id " 00002" for each experiment with measure unit id = " 001".

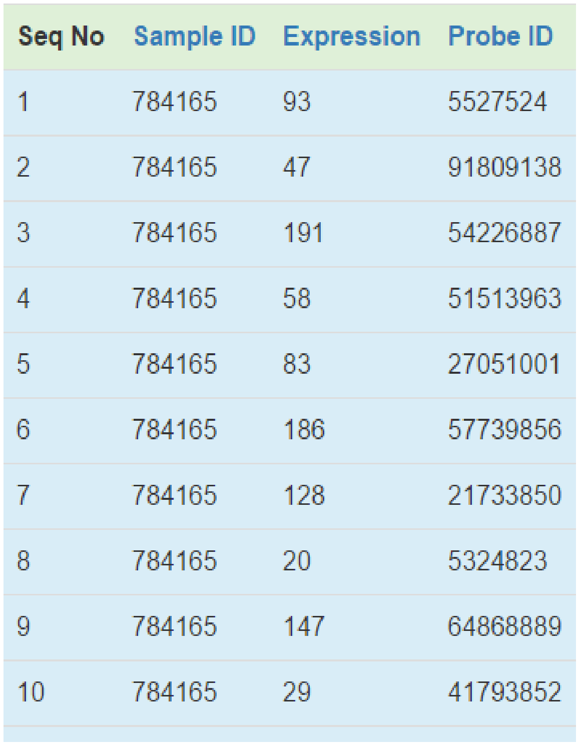
SQL Query:

1.) create table sys.q3testpb select pb_id from sys.probe as b inner join (select distinct uid from sys.gene_fact where cl_id like '2') as a on a.uid=b.uid

2.) create table sys.query3_resultrevised select distinct b.s_id, a.expression, a.pb_id from sys.pasadi as b inner join (select distinct ma_s_id, pb_id, expression from sys.microarray_fact where pb_id in (select pb_id from sys.q3testpb) and mu_id like '1') as a on a.ma_s_id=b.s_id and b.ds_id like '2'

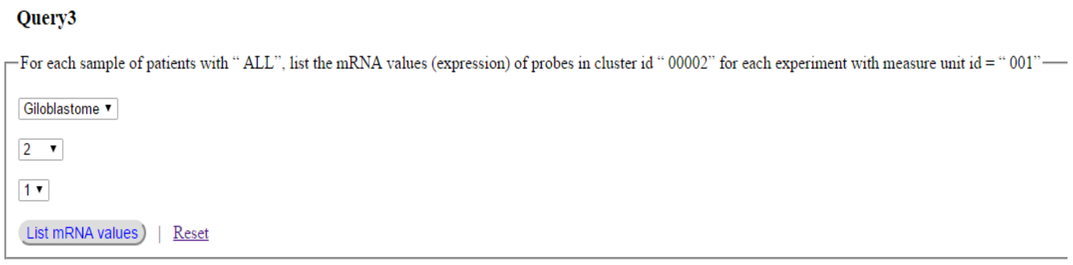
Initially in table q3testpb we selected the probes which had cluster id ‘2’. For those probe id’s we then retrieve the expression values for those tuples from micro array gene_fact we had measurement unit id as 1 and whose samples had disease ALL. Samples which were detected by ALL disease were filtered out by performing an inner join of microarray_fact with the pasadi partition table created. We have used the fact table and the partition table created by us to retrieve the results for this query. The reason for this is using our hybrid schema would requies to traverse many measure tables for this query as we need information from 3 data spaces. Using the partition table pasadi (p_id,s_id,ds_id) and the fact table would help to optimize and reduce the Query complexity/ run time.

Result:



Seq No	Sample ID	Expression	Probe ID
1	784165	93	5527524
2	784165	47	91809138
3	784165	191	54226887
4	784165	58	51513963
5	784165	83	27051001
6	784165	186	57739856
7	784165	128	21733850
8	784165	20	5324823
9	784165	147	64868889
10	784165	29	41793852

We have modified this to display the MRna expression values associated with any cluster id, measurement unit id and given any disease.



Query3

For each sample of patients with “ ALL”, list the mRNA values (expression) of probes in cluster id “ 00002” for each experiment with measure unit id = “ 001”

Giloblastome ▾

2 ▾

1 ▾

Query 4: For probes belonging to GO with id = “ 0012502”, calculate the t statistics of the expression values between patients with “ ALL” and patients without “ ALL”

SQL Query:

- GROUP A: Query for patient having disease ALL where Go-id =12502

create table sys.q4test1 select distinct c.ma_s_id, c.expression, c.pb_id from sys.microarray_fact as c where c.pb_id in (select a.pb_id from sys.probe as a where a.uid in (select distinct uid from sys.gene_fact where go_id like '12502'))

create table sys.query4_result_revised_part1 select ma_s_id, expression, pb_id from sys.q4test1 where ma_s_id in (select s_id from sys.pasadi where ds_id like '2')

- GROUP B: Query for patient not having disease ALL where Go-id=12502

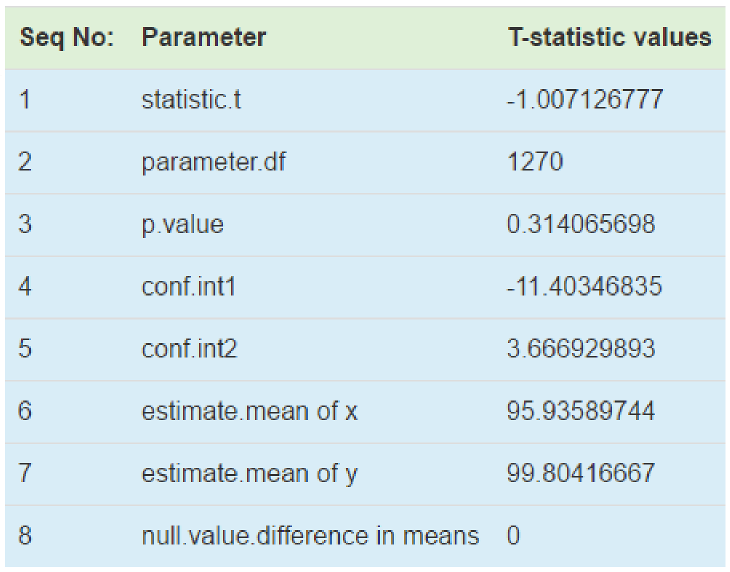
create table sys.q4test select a.pb_id from sys.probe as a where a.uid in (select distinct uid from sys.gene_fact where go_id like '12502')

create table sys.query4_resultrevised select distinct b.pb_id, a.expression , a.ma_s_id from sys.q4test as b inner join (select distinct ma_s_id, expression, pb_id from sys.microarray_fact where ma_s_id in (select distinct s_id from sys.pasadi_nonull where ds_id not like '2')) as a on b.pb_id=a.pb_id

In this part we are expected to calculate the t-statistics between the expression values associated between GROUP A and GROUP B for each patient.

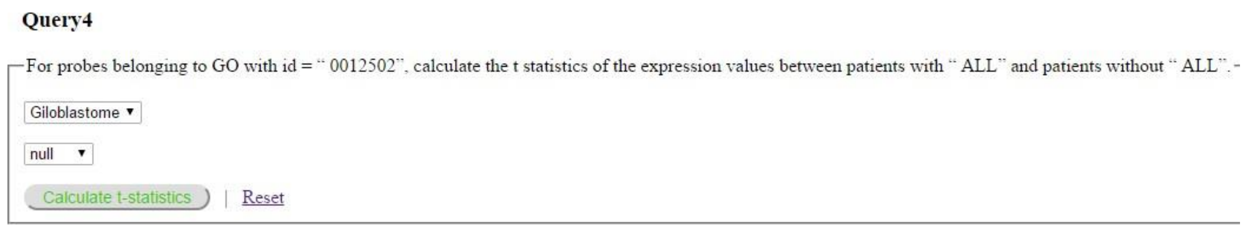
So we got the expression values for those sample id's for probes having Go-id=12502. Filter out those expression values for the sample id's which have disease ALL. We have used the pasadi partition table to filter out the expression values for those sample with disease ALL. Similarly we found out the expression values for those patients not having disease ALL having the same GO id. In this case also in order to avoid traversal of different tables , we have used the pasadi (p_id, ds_id and ds_id) partition table and the fact table. We have used the t test for unequal sample size and equal variance by setting the third parameter in the T function var.equal =T. This specifies that we have set the parameter of variance equal=TRUE.

Query Result:



Seq No:	Parameter	T-statistic values
1	statistic.t	-1.007126777
2	parameter.df	1270
3	p.value	0.314065698
4	conf.int1	-11.40346835
5	conf.int2	3.666929893
6	estimate.mean of x	95.93589744
7	estimate.mean of y	99.80416667
8	null.value.difference in means	0

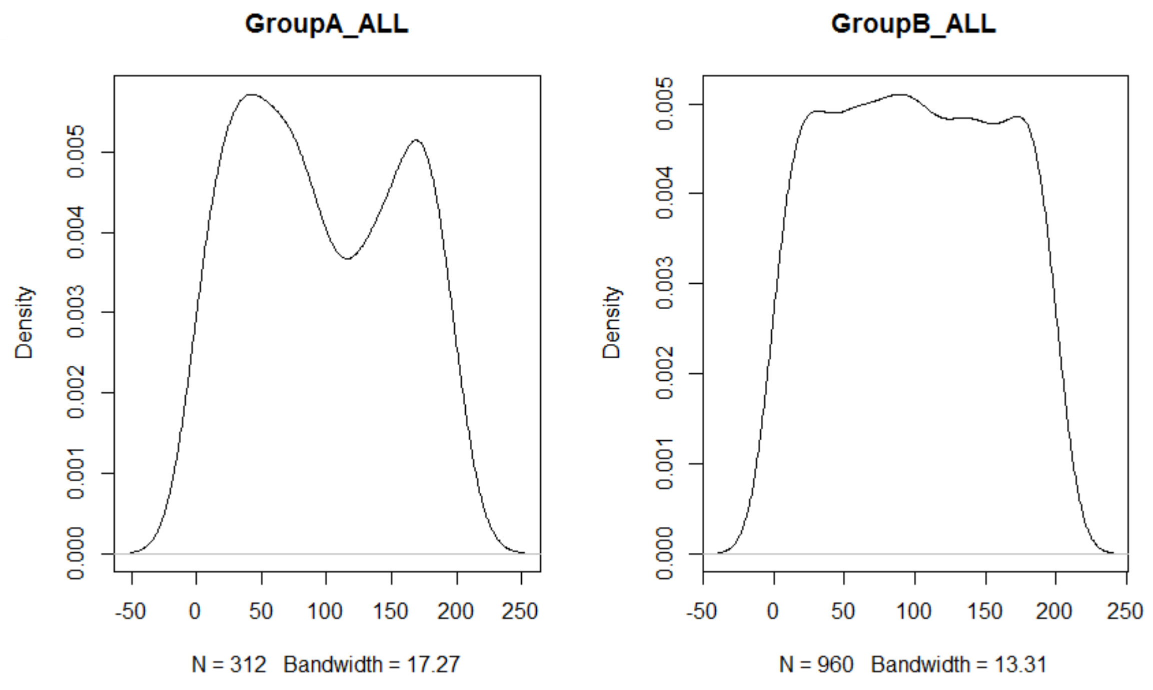
We have modified our UI to provide the option of calculating the t- statistics between the expression values of patients given any disease and Go id.

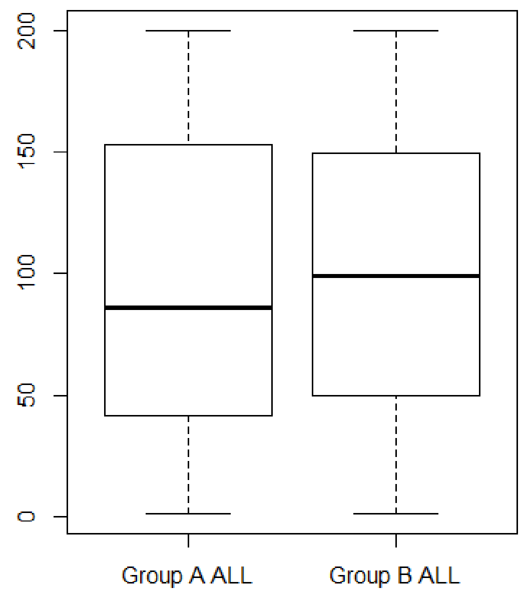


Query4

For probes belonging to GO with id = " 0012502", calculate the t statistics of the expression values between patients with " ALL" and patients without " ALL".

Visual Analysis:





Query 5: For probes belonging to GO with id=" 0007154", calculate the F statistics of the expression values among patients with " ALL", " AML", "colon tumor" and " breast tumor".

SQL Queries:

1.) selecting all probes with go id = 7154

create table sys.q5test select distinct c.ma_s_id, c.expression, c.pb_id from sys.microarray_fact as c where c.pb_id in (select a.pb_id from sys.probe as a where a.uid in (select distinct uid from sys.gene_fact where go_id like '7154'))

2.) Selecting expression values for ALL patients with Go id=7154

create table sys.query5_groupA select distinct ma_s_id, expression, pb_id from sys.q5test where ma_s_id in (select s_id from sys.pasadi where ds_id like '2')

3.) Selecting expression values for AML patients with Go id=7154

create table sys.query5_groupB select distinct ma_s_id, expression, pb_id from sys.q5test where ma_s_id in (select s_id from sys.pasadi where ds_id like '3')

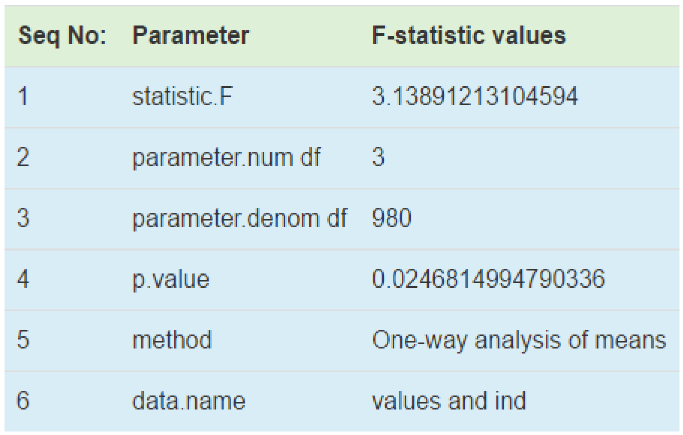
4.) Selecting expression values for colon tumor patients with Go id=7154

create table sys.query5_groupC select distinct ma_s_id, expression, pb_id from sys.q5test where ma_s_id in (select s_id from sys.pasadi where ds_id like '4')

5.) Selecting expression values for Breast tumor with Go id=7154 create table sys.query5_groupD select distinct ma_s_id, expression, pb_id from sys.q5test where ma_s_id in (select s_id from sys.pasadi where ds_id like '5')

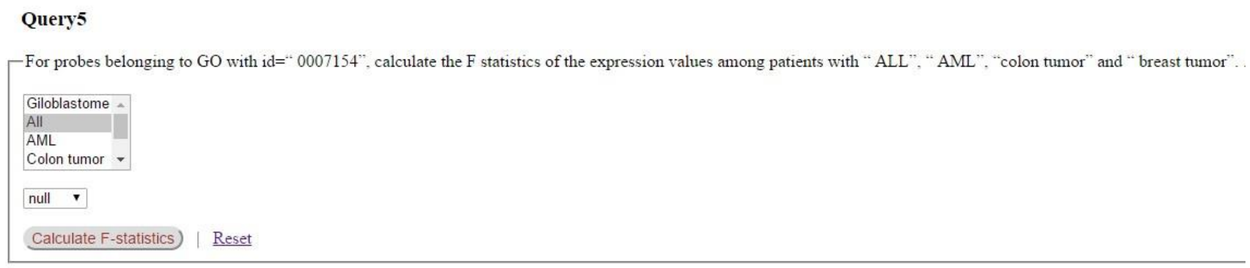
For this in the first step we have selected the expression values for probes belonging to Go id 7154. Then we have filtered out the expression values for those which have disease 'ALL' i.e id 2. We have used the pasadi (p_id, s_id, dds_id) partition table to filter out those samples that have the disease ALL. Similarly we have filtered out the expression values for patients having AML, Breast tumor and colon tumor. We have calculated F statistics of assuming variance of the selected groups is equal.

Results



Seq No:	Parameter	F-statistic values
1	statistic.F	3.13891213104594
2	parameter.num df	3
3	parameter.denom df	980
4	p.value	0.0246814994790336
5	method	One-way analysis of means
6	data.name	values and ind

We have modified the UI to calculate F statistics between any combination of groups.



Query5

For probes belonging to GO with id=" 0007154", calculate the F statistics of the expression values among patients with " ALL", " AML", "colon tumor" and " breast tumor". .

Query 6: For probes belonging to GO with id=" 0007154", calculate the average correlation of the expression values between two patients with " ALL", and calculate the average correlation of the expression values between one " ALL" patient and one " AML" patient

SQL Query:

1.) create table step1
select distinct pb_id from probe as b inner join
(select distinct uid from gene_fact where go_id like '7154') as a on b.uid=a.uid

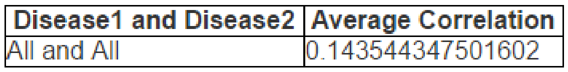
2.) create table step4 select distinct expression, ma_s_id , b.pb_id from microarray_fact as b where pb_id in (select pb_id from (select distinct pb_id from probe as b inner join (select distinct uid from gene_fact where go_id like '7154') as a on b.uid=a.uid) as c)

3.) create table q6groupa
select distinct * from step4 inner join (select s_id, p_id from pasadi where ds_id like '2') as b on b.s_id = ma_s_id

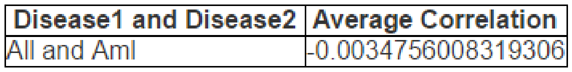
4.) create table q6groupb
select distinct * from step4 inner join (select s_id, p_id from pasadi where ds_id like '2') as b on b.s_id = ma_s_id

Initially we filter out those probes which have their Go id=7154, we then found out the corresponding expression values and sample id's for the filtered probes. We then found out the patients associated with these sample id's and expression values where the sample id has 'ALL' disease

Query Result:



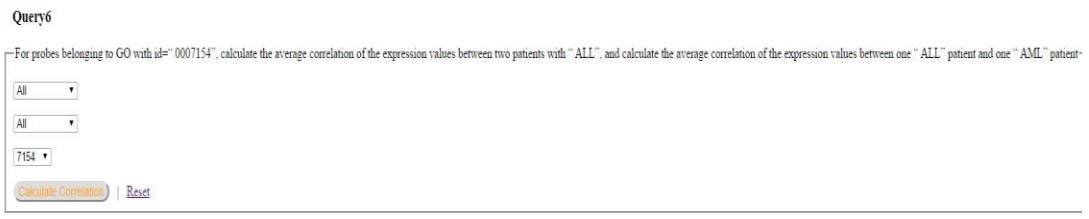
Disease1 and Disease2	Average Correlation
All and All	0.143544347501602



Disease1 and Disease2	Average Correlation
All and Aml	-0.0034756008319306

Table Q6 is the only input for finding out the correlation amongst ALL patients.

We have modified the UI to compute correlation with respect to any disease



Query6

For probes belonging to GO with id=" 0007154", calculate the average correlation of the expression values between two patients with " ALL", and calculate the average correlation of the expression values between one " ALL" patient and one " AML" patient

3.4 PART 3: KNOWLEDGE DISCOVERY

1.] QUERY: Given a specific disease, find the informative genes.

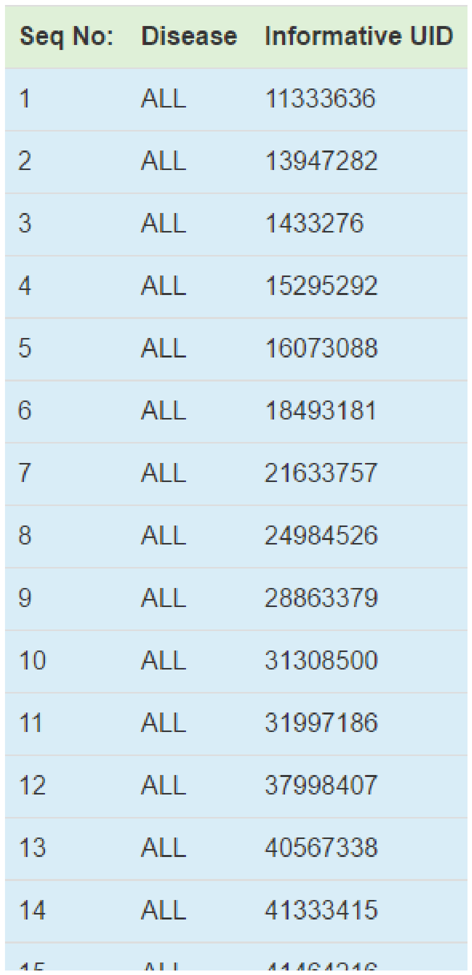
SQL QUERY:

```
    1.) create table all_ds
select distinct expression, ma_s_id, pb_id from sys.microarray_fact where
ma_s_id in (select distinct s_id from sys.pasadi where ds_id like '2')

    2.) create table all_no
select distinct expression, ma_s_id, pb_id from sys.microarray_fact where
ma_s_id in (select distinct s_id from sys.pasadi_nonull where ds_id not like '2')


    3.) create table sys.q3part1grpa_revised
select distinct uid, a.pb_id, a.expression, a.ma_s_id from sys.probe as b inner join
all_ds as a
on a.pb_id=b.pb_id

    4.) create table sys.q3part1grpb_revised
select distinct uid, a.pb_id, a.expression, a.ma_s_id from sys.probe as b inner join
sys.all_no as a
on a.pb_id=b.pb_id
```

Initially we create a table all_ds, where we filter out the expression values for those samples that are detected with 'ALL' disease and are not detected with 'ALL' diseasee. We have used the partition table pasadi and pasadi_nonull for this purpose. This reduces the complexity in filtering out the required expression values. In order to calculate t statistics between each gene, in step 3 and 4 we found ot the uid's corresponding to the expression values for 'ALL' and noALL.We have calculated the t statistics for unequal sample size and equal variance.
In addition to this we have also pre computed the values of informative genes with respect to any disease. Thus now the classificaltion for any new patient for any disease can be done more efficiently, as we have already stored the required data in our Data Warehouse. This eliminates

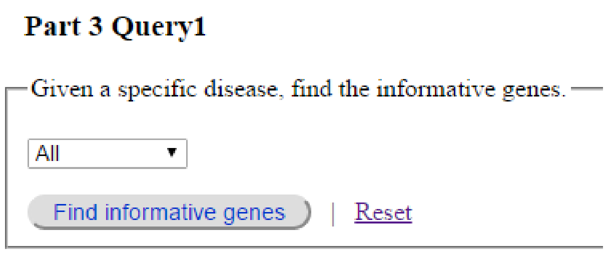
the overhead associated with modifying/adding or deleting data from Data Ware house thus improving Query performance.

Result:



Seq No:	Disease	Informative UID
1	ALL	11333636
2	ALL	13947282
3	ALL	1433276
4	ALL	15295292
5	ALL	16073088
6	ALL	18493181
7	ALL	21633757
8	ALL	24984526
9	ALL	28863379
10	ALL	31308500
11	ALL	31997186
12	ALL	37998407
13	ALL	40567338
14	ALL	41333415
15	ALL	[illegible]

We have modified our UI to find out the informative genes with respect to any disease.

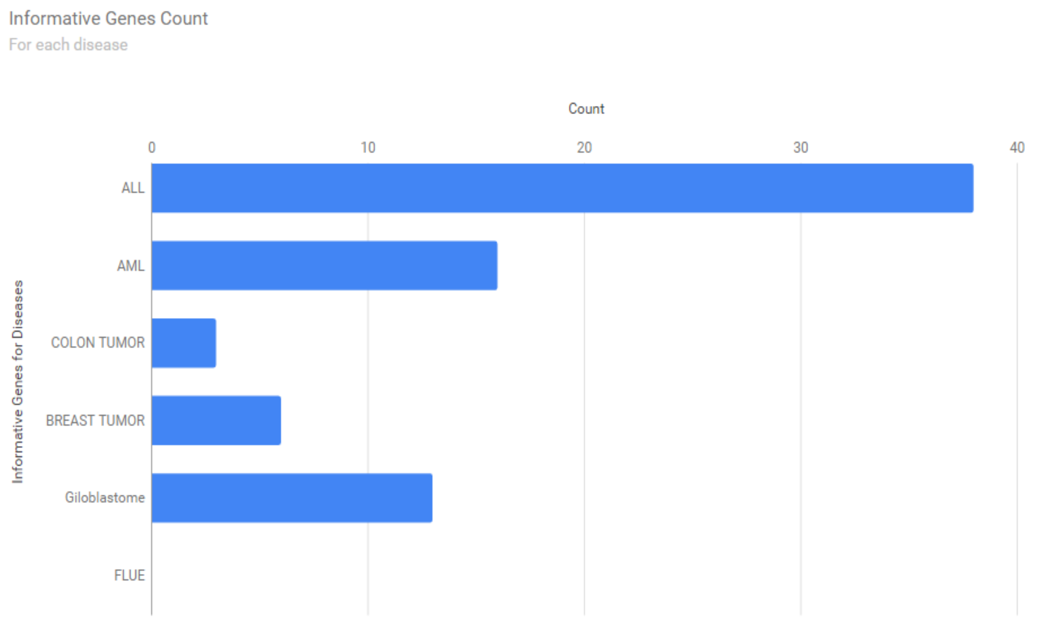


Part 3 Query1

Given a specific disease, find the informative genes.

All

Visual Analytics: Displays the count of informative genes with respect to each disease.



2.**] Query:** Use informative genes to classify a new patient (five test cases in test_samples.txt are given in the data).

SQL Query:

1.) create table filtered_samples_all select distinct * from sys.test_samples where UID in (select distinct uid from informative_genes where ds_name like 'All')

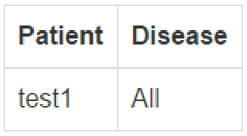
2.) create table sys.part3q2groupa
select distinct uidexp.p_id, sys.informative_genes.uid,sys.informative_genes.ds_name, uidexp.expression, uidexp.pb_id
from sys.informative_genes
inner join
(select distinct pro.uid, sapaex.expression , sapaex.p_id, sapaex.pb_id from sys.probe as pro
inner join
(select a.ma_s_id, b.p_id, a.pb_id, a.expression from sys.microarray_fact as a
inner join
(select distinct s_id, p_id from pasadi_nonull where ds_id like '2') as b on a.ma_s_id=b.s_id) as sapaex
on sys.sapaex.pb_id=pro.pb_id)
as uidexp on uidexp.uid=sys.informative_genes.uid and informative_genes.ds_name like 'All'

3.) create table sys.part3q2groupb
select distinct uidexp.p_id, sys.informative_genes.uid,sys.informative_genes.ds_name, uidexp.expression, uidexp.pb_id
from sys.informative_genes
inner join

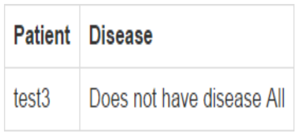
```
(select distinct pro.uid, sapaex.expression , sapaex.p_id, sapaex.pb_id from sys.probe as pro
inner join
(select a.ma_s_id, b.p_id, a.pb_id, a.expression from sys.microarray_fact as a
inner join
(select distinct s_id, p_id from pasadi_nonull where ds_id not like '2') as b on a.ma_s_id=b.s_id) as sapaex
on sys.sapaex.pb_id=pro.pb_id )
as uidexp on uidexp.uid=sys.informative_genes.uid and informative_genes.ds_name like 'All'
```

Initially we find out the probe id's,patient id and the expression values associated with those sample id's that have 'ALL' disease. We have used the partition table pasadi_nonull for this. We then find out the uid associated with the given probe values. Later we filter those to obtain the expression values and patient id's of only those tuples whose uid was computed informative previously. Similarly, we get the required list of patient id's and expressions for patients without ALL also. For each patient in both the groups correlation is calculated between the expression values and then t test is applied on this correlation. Based on this information the patient is classified to have or not have that particular disease.

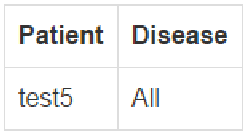
Query Result:



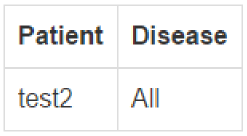
Patient	Disease
test1	All



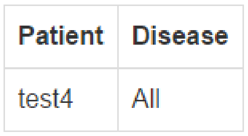
Patient	Disease
test3	Does not have disease All



Patient	Disease
test5	All

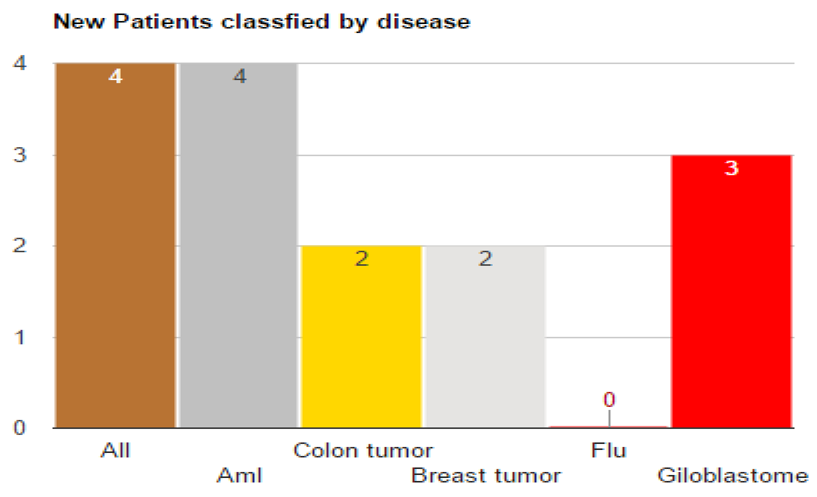


Patient	Disease
test2	All

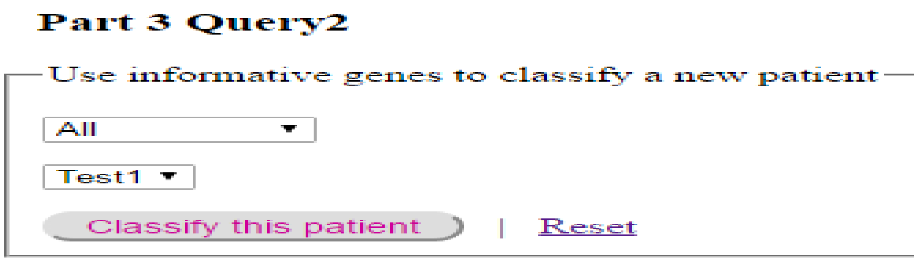


Patient	Disease
test4	All

Visual Analytics:

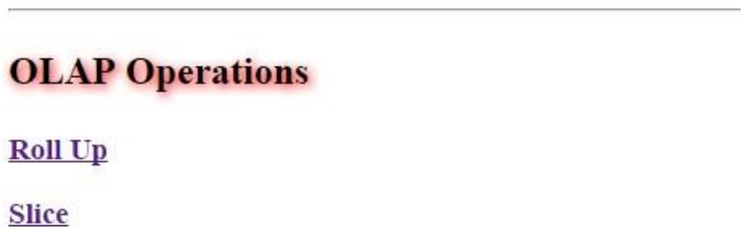


We have modified our UI to classify the new patient for any given disease.

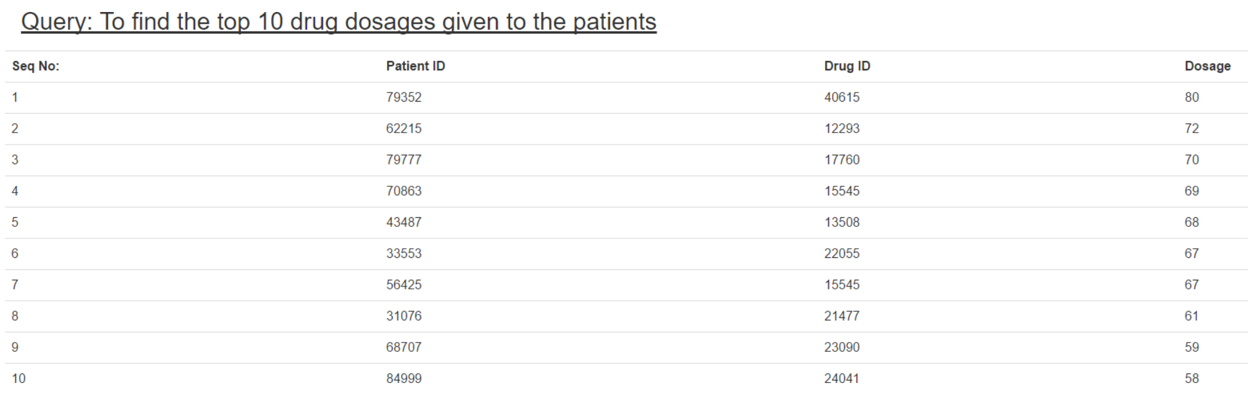


Extra OLAP Operations:

We have also implemented some extra OLAP operations in order for meaningful data visualization.

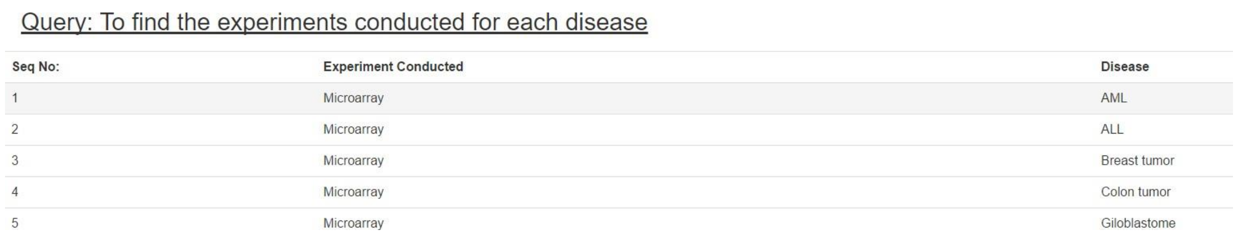


We have found the top 10 drug dosages given to the patients and also listed their dosages respectively



Query: To find the top 10 drug dosages given to the patients

Seq No:	Patient ID	Drug ID	Dosage
1	79352	40615	80
2	62215	12293	72
3	79777	17760	70
4	70863	15545	69
5	43487	13508	68
6	33553	22055	67
7	56425	15545	67
8	31076	21477	61
9	68707	23090	59
10	84999	24041	58



Query: To find the experiments conducted for each disease

Seq No:	Experiment Conducted	Disease
1	Microarray	AML
2	Microarray	ALL
3	Microarray	Breast tumor
4	Microarray	Colon tumor
5	Microarray	Giloblastome

3.5 Reducing Query complexity:

We have taken several steps to improve upon Query complexity. In cases like part 2 Query 2 we have **stored the intermediate results in order to improve upon the Query run time. In order to make our Data Ware house efficient we have pre computed the results for important/ frequently required data. S**ay for example we have stored the informative genes with respect to each disease. Besides we are also reducing the Query complexity in terms of big 0. **We have avoided the use of nested SQL which is infamous for having large complexity 0(MN) where M ,N are the number of tuples in both the tables respectively. We have used join statements which roughly have a Query complexity close to 0(M+N).**

4. Conclusion:

The regular data warehouse schema design techniques will fail on numerous fronts while dealing with biomedical data. 'Biostar' as well as our proposed schema handles the complex nature of the input biomedical data. Using our schema designed in MySQL we have implemented and verified that our data warehouse for the input dataspace supports regular and statistical OLAP operations. It can be extended to include additional dataspaces of the same nature. Knowledge discovery and statistical operations have been achieved through the use of R scripts on the queried data from MySQL.

We have used a concept similar to materialized view to significantly reduce querying time for larger data sets and also added extra features of charts associated with some Queries in order to visualize data for getting meaningful information and also performed OLAP operations for some new Query sets.

5. References

[1]. http://www.cse.buffalo.edu/faculty/azhang/cse601/IJBRA.pdf

[2] https://www.researchgate.net/publication/262400722_BioStar_a_data_warehouse_schema_for_integrating_clinical_and_genomic_data_from_HIV_patients

[3] http://www.google.com/url?q=http%3A%2F%2Feric.univ-lyon2.fr%2Fpublications%2Ffiles%2Frapport-

fawad06.pdf&sa=D&sntz=1&usg=AFQjCNFyBBCNWlQWPhE9OwXd-UNzRUsuhg