Normalization

**Database normalization** is the process of organizing the [fields](http://en.wikipedia.org/wiki/Field_(computer_science)) and [tables](http://en.wikipedia.org/wiki/Table_(database)) of a [relational database](http://en.wikipedia.org/wiki/Relational_database) to minimize [redundancy](http://en.wikipedia.org/wiki/Data_redundancy). Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them.

The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database using the defined relationships.

In original data, we had sample information for every sample (patient) in every FPKM table like gene\_fpkm, isoform\_fpkm and tss\_fpkm.

Initially we had data corresponding to 17 patients and for every patient, we had information about that patient in the column. But the basic problem with this design was that in future, if we increase the number of patients or collect more data about different patients, it will be very difficult to add columns for every patient.

Secondly, the sample (patient) information was repeated for every patient in every fpkm table (gene\_fpkm, isoform\_fpkm and tss\_fpkm) pertaining to each sample\_id. Hence we thought of performing the normalization to avoid the redundant information in every table.

In order to perform normalization, we removed the 17 samples information in each fpkm table and created another table with title ‘*sample*’ that stores the each patient information with primary key *sample\_id*. For instance,

|  |
| --- |
| Sample Table |
| sample\_id (PK) |
| patient\_id |
| sample\_extraction |
| condition |
| sample\_type |
| sample\_date |

Next, for each fpkm (gene\_fpkm, tss\_fpkm, isoform\_fpkm) table, we referred ‘*sample\_id’* of ‘*sample*’ table as foreign key. For example, in gene\_fpkm table

|  |
| --- |
| gene\_fpkm Table |
| gene\_fpkm\_id (PK) |
| tracking\_id |
| class\_code |
| nearest\_ref\_id |
| gene\_id (FK – gene\_info.id) |
| tss\_id (FK – tss\_info\_id) |
| chromosome |
| start |
| end |
| length |
| coverage |
| sample\_id (FK – sample.sample\_id) |
| sample\_fpkm |
| sample\_fpkm\_lo |
| sample\_fpkm\_high |
| sample\_status |

Similarly, in other isoform\_fpkm table also, we had ‘*sample\_id*’ as foreign key to ‘*sample*’ table *sample\_id* column. And any sample information can be retrieved from ‘*sample*’ table corresponding to sample\_id

|  |
| --- |
| isoform\_fpkm Table |
| iso\_fpkm\_id (PK) |
| isoform\_id (FK – isoform\_info.id) |
| class\_code |
| nearest\_ref\_id |
| gene\_id (FK – gene\_info.id) |
| tss\_id (FK – tss\_info\_id) |
| chromosome |
| start |
| end |
| length |
| coverage |
| sample\_id (FK – sample.sample\_id) |
| sample\_fpkm |
| sample\_fpkm\_lo |
| sample\_fpkm\_high |
| sample\_status |

Similarly, in other tables like ‘*fasta\_clipper*’, ‘*fastq\_makser*’ and ‘*fastq\_trimmer*’ , ‘sample\_id’ is referred as Foreign Key for ‘sample’ table ‘sample\_id’ column.

Sample (Patient) information is repeated in various tables hence we thought of normalizing it and hence created ‘*sample*’ table with reference *sample\_id* column.

Indexing

A **database index** is a [data structure](http://en.wikipedia.org/wiki/Data_structure) that improves the speed of data retrieval operations on a [database table](http://en.wikipedia.org/wiki/Table_(database)) at the cost of additional writes and the use of more storage space to maintain the extra copy of data. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed.

Indexes can be created using one or more [columns of a database table](http://en.wikipedia.org/wiki/Column_(database)), providing the basis for both rapid random [lookups](http://en.wikipedia.org/wiki/Lookup) and efficient access of ordered records.

For original data we had initially, we did not have any index on the data and hence it was very challenging how to put index on the columns to make search faster and query data in more optimized manner.

When we designed the database, for every table irrespective of the data in the table and data structure, we defined the primary key as *auto\_increment Integer* so that while performing the search on the table with primary key, the column in indexed and the query can perform faster and give results in short time. For instance,

‘Fasta\_clipper’ table has ‘Clipper\_ID’ int(11) NOT NULL AUTO INCREMENT as its primary key. So by default, this key becomes indexed and query can perform faster. For example,

|  |  |
| --- | --- |
| Table | Primary Key |
| gene\_fpkm | gene\_fpkm\_id int(11) NOT NULL AUTO INCREMENT |
| isoform\_fpkm | iso\_fpkm\_id int(11) NOT NULL AUTO INCREMENT |
| tss\_fpkm | tss\_fpkm\_id int(11) NOT NULL AUTO INCREMENT |
| fastq\_trimmer | trimmer\_id int(11) NOT NULL AUTO INCREMENT |
| sample | sample\_id int(11) NOT NULL AUTO INCREMENT |
| fastq\_masker | masker\_id int(11) NOT NULL AUTO INCREMENT |
| splicing | splicing\_id int(11) NOT NULL AUTO INCREMENT |

So, the primary key is indexed so that we can perform the query based on primary key faster. In this way, primary key is defined as ‘INT (11) AUTO INCREMENT NOT NULL in every table to make this field indexed.

Optimization and Tuning

**Database tuning** describes a group of activities used to optimize and homogenize the performance of a [database](http://en.wikipedia.org/wiki/Database). It usually overlaps with [query](http://en.wikipedia.org/wiki/Query_language) tuning, but refers to design of the database files, selection of the [database management system](http://en.wikipedia.org/wiki/Database_management_system) (DBMS) application, and configuration of the database's environment ([operating system](http://en.wikipedia.org/wiki/Operating_system), [CPU](http://en.wikipedia.org/wiki/CPU), etc.).

Database tuning aims to maximize use of system resources to perform work as efficiently and rapidly as possible. Most systems are designed to manage their use of system resources, but there is still much room to improve their efficiency by customizing their settings and configuration for the database and the DBMS.

In this project, we try to optimize the performance of the query and tune the database. As previously defined in normalization technique also, we did the vertical decomposition of the original data for every sample in each column, which would be very tedious and impossible to expand in case the number of sample (patients) increase in future.

So in order to optimize the performance of query, we did the vertical decomposition of the original data in horizontal mechanism. For instance, initially for gene\_fpkm table we had the table as –

**Gene\_fpkm Table**

CREATE TABLE Gene\_FPKM ( Gene\_Test\_ID INT NOT NULL auto\_increment, Iso\_Test\_ID Int(11), Tss\_Tracking\_ID Int(11), tracking\_id varchar(32), class\_code varchar(32), nearest\_ref\_id varchar(32), gene\_id varchar(32), gene\_short\_name varchar(32), tss\_id varchar(32), chromosome integer(11), start int(11), end int(11), length int(11), coverage double,

q0\_FPKM double, q0\_FPKM\_lo double, q0\_FPKM\_hi double, q0\_status varchar(32), q1\_FPKM double, q1\_FPMK\_lo double, q1\_FPKM\_hi double, q1\_status varchar(32),

………..

………..

………..

qN\_FPKM double, qN\_FPKM\_lo double, qN\_FPKM\_hi double, qN\_status varchar(10), PRIMARY KEY (Gene\_Test\_ID), FOREIGN KEY (Iso\_Test\_ID) REFERENCES Isoform\_FPKM(Iso\_Test\_ID), FOREIGN KEY( TSS\_Tracking\_ID) REFERENCES TSS\_FPKM(TSS\_Tracking\_ID));

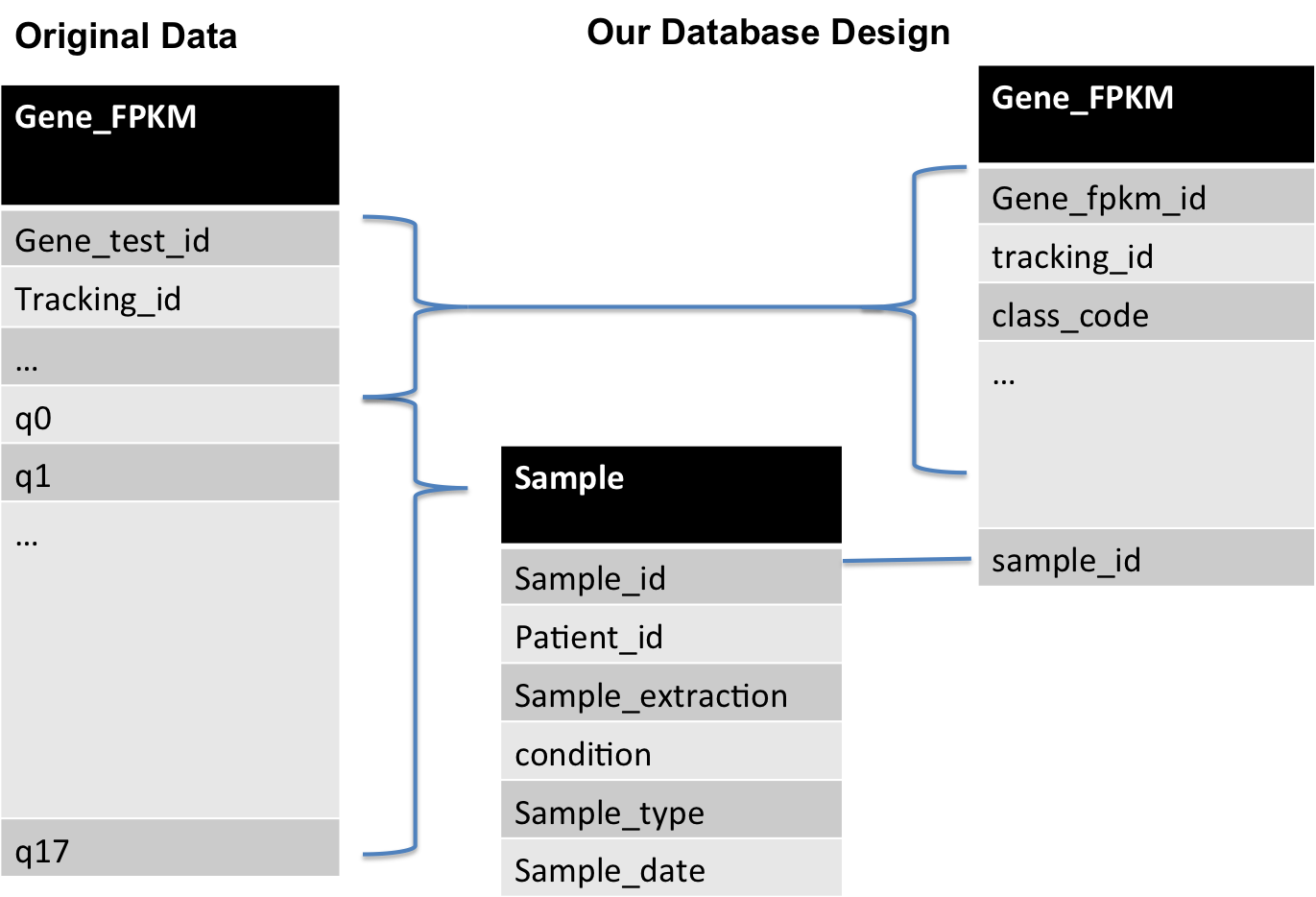
If you see the above query for **gene\_fpkm** original table, we had q0\_FPKM, q0\_FPKM\_lo, q0\_FPKM\_high, q0\_status. Here q0 defines the Sample 1 (Patient 1). Similarly, this query goes on till qN where N=17 since we had the data for 17 patients. Hence this table will have 17 \* 4 = 68 columns, which is not a good database design.

Secondly, in future when the number of patients increases, we will have to add more columns corresponding to the number of patients. Hence, we opted for vertical decomposition of the table in horizontal mechanism. For instance, we decompose the above table in ‘*sample*’ and ‘*gene\_fpkm’ table* like

|  |
| --- |
| sample Table |
| sample\_id (PK) |
| patient\_id |
| sample\_extraction |
| condition |
| sample\_type |
| sample\_date |

|  |
| --- |
| gene\_fpkm Table |
| gene\_fpkm\_id (PK) |
| tracking\_id |
| class\_code |
| nearest\_ref\_id |
| gene\_id (FK – gene\_info.id) |
| tss\_id (FK – tss\_info\_id) |
| chromosome |
| start |
| end |
| length |
| coverage |
| sample\_id (FK – sample.sample\_id) |
| sample\_fpkm |
| sample\_fpkm\_lo |
| sample\_fpkm\_high |
| sample\_status |

We tried to show this vertical and horizontal decomposition via this diagram –



Hence, it is obvious from the above design diagram that if we follow this decomposition, the query will perform faster and will be much efficient.