***Indexing***

> Both Hashes are using Maps of C++ which internally create Red-Black Tree. Thus, both access and update are in constant time i.e log n.

>*Column Wise* :-

Pros :

1. Can apply ITA
2. Can Easily construct and index views at runtime
3. Other Ranking techniques are based on column Indexing
4. Index files would be smaller
5. Systematic and readable indexes
6. Better for advanced search by a power user aware of exact column values

Cons :

1. Cannot have a general search box
2. Has to define a predefined structure for querying i.e like advanced search for power user.
3. No way of indexing views by primary keys even if other columns of views indexed from before.
4. Index are unreadable

>*Tuple Retrieval* :-

1. We are not writing select queries.
2. We assign our own TIDs to tuples and for constant time access we create a Hashtable for tuple and its starting byte from the start, along with the size of tuple.

>*ITA :-*

***Ranking*** *(Profiling)*

> Main aim is solving empty tuple problem

>*IDF* :-

1. On basis of Database
2. Lesser value for values appearing more often
3. IDF = log( total tuples / no of tuples in which value occurs ).

>*QF* :-

1. On basis of Workload
2. QF ~ IDF except QF is for workload while IDF for database.

>*QFIDF* :-

1. Ranking by combining both of above as QF\*IDF.
2. Jakard’s coefficient
3. Will have very less collisions i.e occurrences of tuples with same similarity score

***Scope of Work***

>*Numerical data in columns (Range)*:-

1. Will get maximum similarity score from the entire range

>*Many Answer Problem (Tie break between tuples with same score)* :-

>’*IN’ Queries (For Proximity Search to find trends)*:-

1. Works on basis of workload patterns of ‘IN’ queries relating unrelated data in a column.

>*View creation and Indexing*

***Not Accounting For***

>*Multiple related tables* :-

>*Pattern Matching (Suffix Tree)*:-

***Comments***

>A query may not necessarily have all columns.

>Not considering Term Frequency (TF) as not processing a generic document

***To Dos :-***

* Exclude column present in user query during QF Calculation for tie-break.
* Account for No Workload.
* Capital and small letters.
* Recreating column\_cum\_cell index and basing QF calculation on it to avoid sequence match.
* The Apple apple problem.
* Workload update with each user query i.e do file writing and workload index update after certain number of user queries.
* Datatype identification to do idf for non-categorical data.
* Prefix match
* Make Workload analysis faster
* Serialization of indexes object into hard-disk.