Lecture Notes
Date: 19th October 2011

Prepared by:

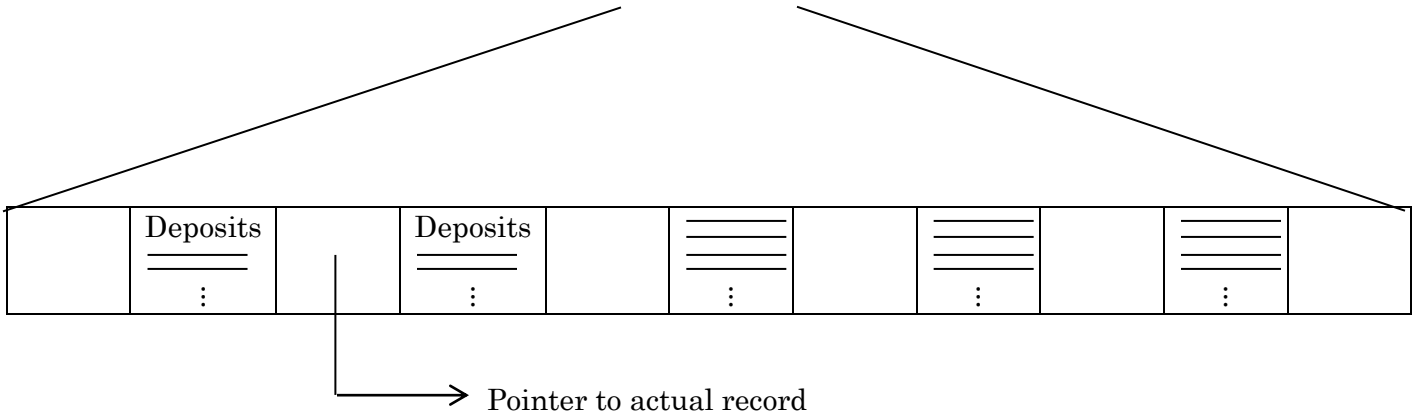
Ajay Mehra (08012303)
Raunak Singhi (08012316)

- Code of a file for DBMS implementation consists of a very low percentage (around 2%) of query optimization. The remaining code consists of the following components:
  - Exception handling which forms a major component
  - Indexing
  - Access control
  - Multithreading
  - Transaction Management
  - Networking
  - Normalization
  - Parser

- Union and Intersection:
  - Union (∪) : If the schemas used are the same, the union of two files $F_1$and F_2 is done by simply appending the two files.
  - Intersection (∩) : For the same schema, the method for taking the intersection of two files F_1 and F_2 depends on the size of the two files :
    i) If $F_1$and F_2 are small enough to fit in the main memory, one can use the brute force approach by comparing the records sequentially and obtaining $F_1 \cap F_2$ by listing the common records.
    ii) If either one or both of $F_1$and F_2 are large, i.e. they don't fit into the main memory, then one should use hashing and match hash values for finding the intersection.

- Aggregate values: Aggregate values include count, sum, average, max and min.

  Consider the example of a bank database and say, one wishes to find the average of all deposits made.
  In this case, deposits (in general, any important attribute) are stored through the tree indexing mechanism as shown below and the average is calculated.



Another option is to keep the mean, median and similar values at hand for possible future queries.

- Views: A specific relation which is not a part of the logical model, but is made visible to a user as a virtual relation, is called a view.
  The reasons for using views include:
  - Security concerns: Security considerations may require that certain data be hidden from users and that all information should not exposed.
  - Ease of writing SQL queries
  - A personalized set of relations can be created which is better matched to the user's needs.

Depending upon the situation, any of the following 2 types of views may be required:

i. Materialized views: For these views, the database systems allow view relations to be stored permanently, and make sure that the view is kept up to date, if the actual relations used in the view definition change.
   The ideal situation for using the materialized view would be if:
   a) One uses the view frequently or demands a fast response to the view based queries.
   b) The base table doesn't change frequently (as overhead for updating will be less).

ii. Creating views on the fly: In this case whenever a view relation appears in a query, it is replaced by the stored view definition and thus, the view relation is recomputed every time the query is evaluated.
   Such views would be preferred when:
   a) One uses view less frequently.
   b) Base table changes frequently.

The intermediate cases (when one uses view frequently and the base table changes frequently, and when one uses view less frequently and the base table doesn't change frequently) involve a tradeoff between the extremes and the selection of the view type depends upon the situation.

- Deletion of a record: The main concerns involved during the deletion of a record are:
  - Defragmentation of the database file (specifically if the record size is big)
  - Foreign key problems: All records dependent on the primary key of the deleted record have to be modified.

- Adding a new attribute: For adding a new attribute, usually one has to rewrite the whole file.