Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

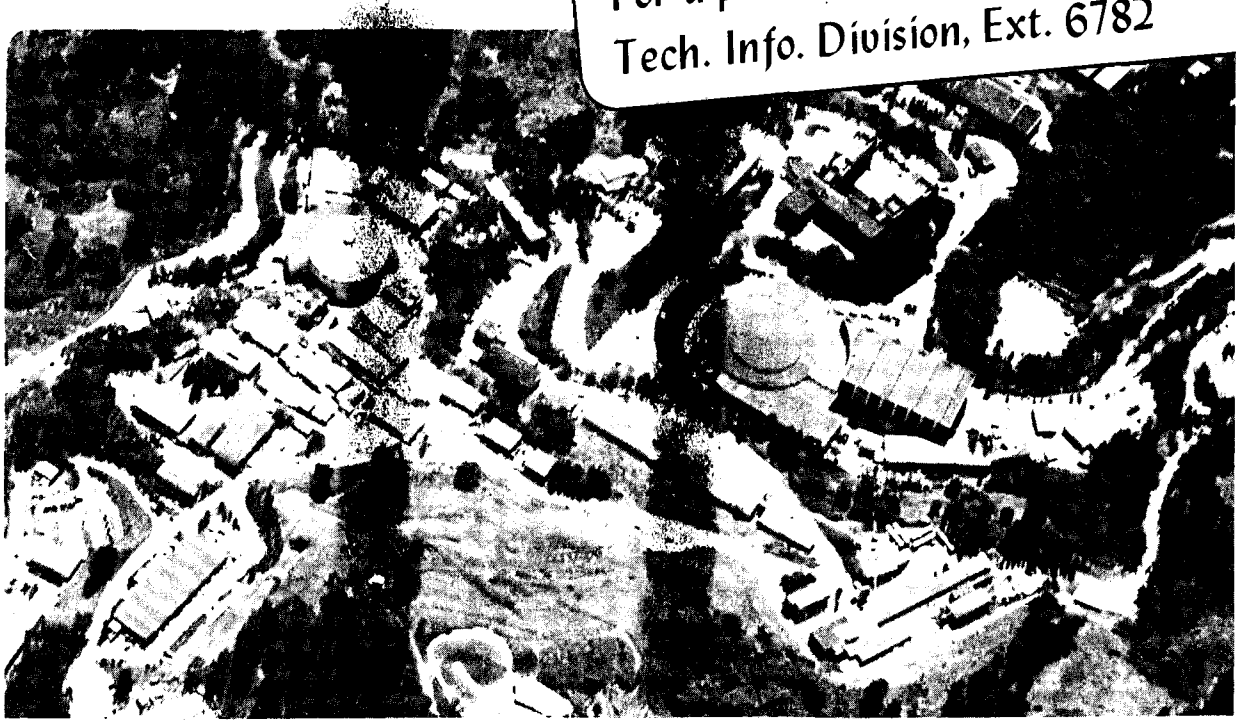
Physics, Computer Science & Mathematics Division

Presented at the ACM/SIGMOD International Conference on Management of Data, University of Michigan, Ann Arbor, MI, April 29-May 1, 1981; and published in the Proceedings

MODELLING SUMMARY DATA

Rowland R. Johnson

March 1981



Prepared for the U.S. Department of Energy under Contract W-7405-ENG-48

DISCLAIMER

Modelling Summary Data*

by
Rowland R Johnson
Computer Science and Applied Math
Lawrence Berkeley Lab
University of California
Berkeley, California 94720

ABSTRACT

Several problems in specifying aggregate functions in relational systems are investigated. We propose a solution to these problems in the form of an extension of the relational data model. In particular we introduce the concept of summary data. The query language STRAND is presented in order to describe retrieval operations on the extended model. STRAND allows a user to formulate queries involving aggregate functions without conceptualizing the query in terms of aggregation. Two example applications, proposal tracking and socio-demographic data bases, are used to illustrate the concepts of the extended model.

1. Introduction

Most relational data bases have some kind of facility to evaluate aggregate functions. In general, there are many aggregate functions that may be applied to a given relation. A consequence of this fact is that users often find it difficult to understand the sematics of aggregation. Furthermore, once a user understands the required aggregation it is often difficult and/or lengthy to express in a query language. This paper considers the semantics of summarized data which is a special case of aggregated data and is found in many applications. With the incorporation of these semantics into the data base model it is

*This work was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, Department of Energy under Contract W-7405-ENG-48.

possible to alleviate the user from having to understand or express aggregations that are required by a query.

In section 2 we investigate some problems of relational systems in terms of specifying queries with aggregation. Section 3 presents a schema specification technique of the relational model that is similar to that of the ER model. The query language STRAND which operates on this type of schema is then described. In section 4 the concept of summary data is presented as an extension of the relational model. Finally, the semantics of STRAND are extended to operate on the extended model. Two example applications; proposal tracking and socio-demographic data are used throughout the paper for illustration purposes.

The concepts that are presented in this paper are embodied in an implemented system that runs under UNIX on a PDP 11/70. The system consists of three components. A front end parser accepts a STRAND expression from a user. A translator that is written in EQUEL[Allman 76] takes the output of the parser and produces a sequence of QUEL statements. Finally, these QUEL statements are submitted to an INGRES DBMS [Stonebraker 76] to be evaluated. Both the proposal tracking and socio-demographic data applications are running under this system.

2. Aggregation in the Relational Model

The ability of relational systems to process queries involving aggregate functions is a powerful tool [Stonebraker 76, Chamberlain 78]. An aggregate function takes the result of a join and divides it into groups. For each of these groups an aggregate procedure (such as count, average or sum) yields a single aggregate value. The semantics of aggregate functions in a system such as INGRES or System R are complex and therefore require a complex query language. In actual usage, however, a complex aggregate function is not required in most cases. The disadvantage is that the simple aggregate function must be specified by the query language in a way that is more complex than necessary.

This argument will be illustrated with an example based on INGRES[Stonebraker 76]. A similar example also exists for System R [Chamberlain 78]. Consider the following relational schema from a socio-demographic application.

STATE

name	federal	census

POPULATION

state	race	sex	total	avginc

The relation STATE represents the fact that states are grouped into federal regions and census regions. Each tuple in the POPULATION relation represents the total number of people and their average incomes in a state for a particular race and sex. The possible values for race and sex are {black, white, hispanic} and {male, female} respectively.

Consider the question "For each federal region and sex classification what are the total number of blacks and their average income?". The corresponding QUEL query is shown in Fig 2.1.

```
range of s is state
range of p is population
retrieve(s.federal,
         p.sex,
         total=sum(p.total
                   by s.federal,p.sex
                   where p.race="black" and s.state=p.state),
         avginc=(sum(p.avginc*p.total
                   by s.federal,p.sex
                   where p.race="black" and s.state=p.state)  /
                 sum(p.total
                   by s.federal,p.sex
                   where p.race="black" and s.state=p.state)))
```

Fig. 2.1.

A user who deals only with simple aggregate functions will find that the syntax of this query is unnecessarily complex in several ways. Since the target-list and the by-list are identical there should be just be one such specification. There are three identical occurrences of the by/where clause specification where one would suffice. In addition, the scopes of these by/where clauses are unclear. The nesting would seem to indicate that each by/where clause is local only to the aggregate function that it appears in. In fact, the query processor recognizes the identity of these by/where clauses and replaces them with a single by/where clause that has global status.

In addition to the above mentioned syntactic considerations this example also illustrates a shortcoming of the relational model itself. Note that this query has the property that the

aggregate functions compute new values for the attributes total and avginc. All such queries with this property will have aggregate functions with the same functional form as this example. That is, the aggregate function for each such query will have the form

```
total=sum(p.total
          <BY/WHERE clause>    ),
avginc=(sum(p.avginc*p.total
          <BY/WHERE clause>    ) /
        sum(p.total
          <BY/WHERE clause>    )))
```

It should be possible to pre-define the functional form of these aggregate functions so that the user is relieved of this task.

The conceptual advantages of allowing pre-defined joins become apparent with this example. There are many queries that can be based on the relations STATE and POPULATION. However, they will all require that a join be done with the name attribute of STATE and the state attribute of POPULATION. By allowing pre-specified joins to be defined in the schema the user may be relieved of this task.

The solution to these problems is two-fold. First, the relational model must be extended in the manner described above. Second, we require a query language that is capable of taking advantage of these extended semantics. In the following sections we first introduce the query language STRAND that operates on a relational schema. We then extend the relational model so that the semantics of aggregate functions may be specified in the schema. Finally, the semantics of STRAND are extended in a

natural way to operate on a schema of the extended model.

3. STRAND

The query language STRAND(Simple To Read and Understand) was first introduced in [Johnson 80]. STRAND will be used in the next section for describing operations on summarized data. In this section a brief review of the syntax and semantics of STRAND will be presented. Historically, STRAND was developed as an ER model[Chen 76] query language. The implementation of STRAND is with a parser that translates STRAND expressions into QUEL statements; the query language for the relational system INGRES. It is a derivative of the earlier version of the ER model query language CABLE[Shoshani 78]. Many of the ideas found in STRAND and CABLE are also found in the query language PML[Shneiderman 80] which operates on a DBTG schema. Basically, these three languages operate on schemas that are represented by a network. That is, they are all capable of 1) forming a chain of objects in the schema; 2) effecting a specified restriction on the retrieval from these objects; and 3) selecting a subset of the attributes of these objects to be displayed.

In this paper STRAND is used on a variation of the relational model that results in a network type schema. There are two basic objects of such a schema; the set and the relationship. A set may be one of two types; an entity set or summary set. The concept of a summary set will be presented in the next section. An entity set is analogous to a relation of the relational model. A relationship represents a pre-defined

join between two sets.

The diagrammatic technique for representing the schema for an enterprise closely resembles that of the ER model. Boxes represent sets and diamonds represent relationships. In proposal tracking, proposals and principal investigators are in a many to many relationship. The same is true for proposals and keywords that describe the major thrust of the proposals. The schema for these semantics is shown in Fig. 3.1

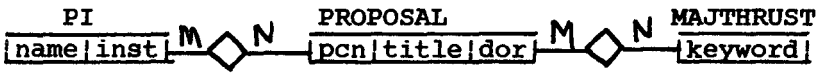


Fig. 3.1 Proposal Tracking Schema

Here, PROPOSAL is an entity set and represents the set of proposals with pcn being the "proposal control number" and dor being the "date of receipt". PI represents the set of principal investigators with inst being the institution with which they are affiliated with. In socio-demographic data bases there are states and counties as shown in Fig. 3.2.

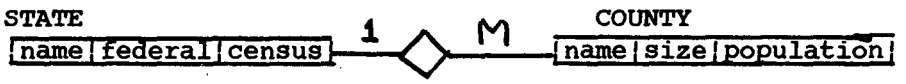


Fig. 3.2 Socio-demographic Schema

In this case STATE and COUNTY are entity sets.

STRAND expressions are constructed from 3 basic operations on sets; projection, restriction, and chaining. Projection and restriction are similar to the projection and restriction operators in the relational model. Chaining is the n-way join of a linear sequence of sets. These sets must be connected with

relationships in the schema.

A STRAND expression consists of a select clause followed by an output clause. The select clause specifies the chaining to be performed and consists of a sequence of "beads", one for each set along the path. The output clause specifies the projection by listing the attributes that are to be displayed. If an attribute does not exist in the output clause then it is removed via projection.

For example, consider the question "Who is the principal investigator and what is the proposal title for each proposal?". The appropriate STRAND query is shown in Fig. 3.3.

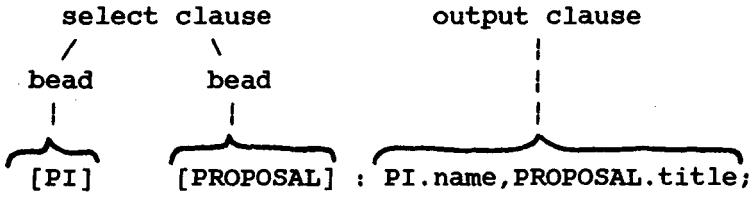


Fig. 3.3 Basic Components of Sample STRAND Query

Restriction of a set is accomplished by allowing a bead to have a qualification clause. For example, the STRAND query for "What proposals have a principal investigator from LBL?" is

```
[PI inst=LBL][PROPOSAL] : PROPOSAL.title;
```

For the query "What counties in Texas have more than 1,000,000 people" we have

```
[STATE name=TEXAS][COUNTY population>1000000]:COUNTY.name
```

Thus far we have presented the basic syntactic structure of STRAND. Additional syntactic constructions and dialectical modifications are possible and often desirable. However, for expository purposes it is useful to include two of these additions. First, it is sometimes inconvenient to have to specify each and every bead in a path. Accordingly, we allow incomplete select clauses that are filled in after parsing and before query processing. The query language PML also has this capability. For example, the question "Which principal investigators at LLL have proposals whose major thrust is conservation?" may be formulated as

```
[PI inst=LLL][MAJTHRUST keyword=conservation]:PI.name;
```

In this case the bead [PROPOSAL] is omitted but can be inferred by the system. Second, an output clause can become quite long. It can be shortened by grouping adjacent attributes belonging to the same set. For example, "What is the name, size and population of all counties in Texas?" would look like

```
[STATE name=Texas][COUNTY]:COUNTY.name,COUNTY.size,
                                    COUNTY.population;
```

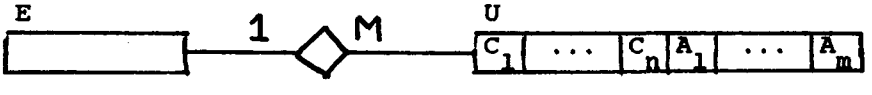
However, the shortened form looks like

```
[STATE name=Texas][COUNTY]:COUNTY.name,size,population;
```

4. Summary Sets

An informal description of summary sets will serve to introduce the formal aspects of summary sets. In many applications users need to manipulate data that has already been aggregated. Population counts in socio-demographic data bases are examples of such data. It would be unacceptable to require a user to aggregate the raw census data in each and every query. In essence, a summary set represents data that has already been aggregated. There are several advantages in using summary sets. First, users do not have to express an aggregate function in every query that is posed. Second, the unaggregated raw data does not have to exist. This is especially useful in socio-demographic data bases where access to the raw data would be a violation of privacy. Third, the storage and manipulation of aggregated data results in a significant reduction in the amount of processing required by a query.

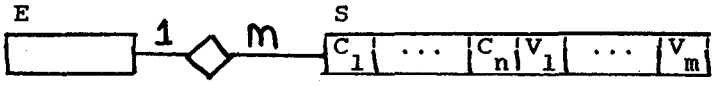
More formally, a summary set is the result of aggregating some underlying entity set. In practice, this underlying entity set does not need to exist, nor do we require that it ever have existed. Its only purpose is to aid in the formal treatment given here. The underlying entity set is in a one to many relationship with some entity set E and is illustrated by the form:



where U is the underlying entity set. We will refer to C_1 ... C_n

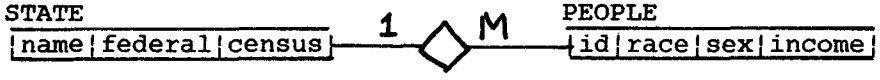
as category attributes and $A_1 \ldots A_m$ are regular attributes.

The grouping for the aggregation that results in the summary set is as follows. Let U^1 and U^2 be entities of U with the values of the category attributes being $u_1^1, u_2^1, \ldots, u_n^1$ and $u_1^2, u_2^2, \ldots, u_n^2$ respectively. U^1 and U^2 are in the same group iff a) U^1 and U^2 are related with same entity of E and b) $u_i^1 = u_i^2$ for i=1,n. Thus, there is a group for each possible combination of values for $C_1 \ldots C_n$ and entity of E. The results of the aggregation are represented in the summary set by summary value attributes. An element of the summary set is called a summary. The form of the resulting summary set is



Here, S is the summary set and $V_1, \ldots, V_m$ are summary value attributes. $C_1, \ldots, C_n$ are the same category attributes as in the underlying entity set.

As an example, consider the underlying entity set PEOPLE in the schema



Here each individual person is represented by an entity in the PEOPLE entity set. Race and sex are category attributes and income is a regular attribute. The summary set POPULATION is obtained from the underlying entity set PEOPLE and is illustrated by the schema

In POPULATION, count and avginc are summary value attributes. Count is simply the number of individuals in each race/sex category. Avginc represents the average income of individuals within a race/sex category. For each race/sex category it is computed by summing all incomes and dividing by the count.

As another example, consider proposal tracking where the funding for a proposal is broken down by type and budget year. The type can be personnel, equipment, or construction. Budget year has the values -1 (previous year), 0 (current year), and 1 (next year). Amt is a summary value attribute that represents the funding amount for each combination of type, byear and entity of PROPOSAL. Fig. 4.1 illustrates these semantics

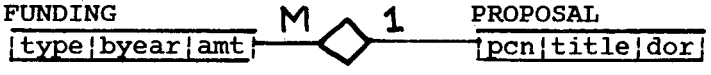


Fig. 4.1 Summary Sets in Proposal Tracking Schema

In applications that require summary sets a user will almost always desire further aggregation of a summary set. We will refer to this operation as summarization. As an example of summarization consider the case where a user may be interested only in population counts and average income broken down by race and not sex. In this particular example the summarization that is required is the following. The new count is obtained by summing the counts over the two sexes. The new average income is derived by a weighted average of the average income for the two

sexes. This summarization is depicted in Fig. 4.2 where the 6 summaries of POPULATION that are related to the entity "Texas" in STATE are shown.

race	sex	count	avginc
b	m	15	12
w	m	32	6
h	m	18	16
b	f	20	18
w	f	10	10
h	f	40	24

==========>

race	count	avginc
b	35	15.43
w	42	6.95
h	58	22.76

Figure 4.2

We now consider the way in which STRAND can be used to express summarization. It will be shown that such a query can be constructed without conceptualizing the query in terms of aggregation. Instead, a user simply expresses the tabular form of the result in terms of the 3 basic operations. That is, there is no need to specify the aggregation procedure or the grouping on which the summarization is based. The aggregate function is specified by the schema and the grouping is determined from the attributes that appear in the output clause. As an illustrative example, the following STRAND expression performs the aggregation depicted in Fig. 4.2.

```
[STATE name=Texas][POPULATION]:POPULATION.race,count,avginc;
```

More formally, the grouping implied by a particular query is an extension of the grouping for the aggregation from the underlying entity set to the summary set. For any two elements v^1 and v^2 in the set that is the result of the chaining operation let $v^1_1, v^1_2, \ldots, v^1_n$ and $v^2_1, v^2_2, \ldots, v^2_n$ be the values of the attributes in the output clause. Then v^1 and v^2 are in the same group iff

$v_i^1 = v_i^2$ for i=1,..,n and v_i not a summary value attribute. This method of specifying groups allows a group to be defined on attributes from different sets. As an example of this consider the query "What is the total funding of proposals dealing with nukes?" The STRAND expression for this query is

```
[MAJTHRUST keyword=nukes][FUNDING]:FUNDING.amt;
```

After the groups have been formed the next step is to evaluate the aggregation procedures on each group to form a single summary. The specification of the aggregation procedures are combined and represented in the schema as a single procedure. This procedure, called the summarization procedure, is part of the description of a summary set. Fig. 4.3. illustrates the general form of the summary procedure.

```
summarization(G)
{
        .
        .
        .
      WHILE(A1,A2,... <-getnextsummary(G))
        {
            .
            .
        }
        .
        .
      RETURN(SVA1,SVA2,...)
}
```

Fig. 4.3.

G is a parameter that represents a group. $A_1, A_2, \ldots$ are variables for holding the values of attributes of the summary set. The function getnextsummary( ) returns a different summary from the group G everytime it is called. When the summaries of the group G have been exhausted a value is returned that causes

the WHILE loop to terminate. Finally, SVA_1,SVA_2,... are variables that hold the new values of the summary value attributes.

The summarization procedure for the summary set POPULATION is

```
summarization(G)
{
        weightedavg <- 0.0
        total <- 0
        WHILE(count,avginc <- getnextsummary(G))
           {    total <- total+count
                weightedavg <- (avginc * count) + weightedavg
           }
        weightedavg <- weightedavg/total
        RETURN(total,weightedavg)
}
```

Thus far we have described the semantics of projection and chaining for STRAND expressions involving summary sets. In the balance of this section we will describe the semantics of the restriction operator for STRAND expressions involving summary sets. In particular, we will consider the order in which restriction and summarization occur.

Restriction on category attributes will be treated in the same manner as restriction on attributes of an entity set. That is, restriction of a category attribute occurs prior to summarization. As an example, consider the query "List the pcn and the current year funding for all proposals.". In this case we want to sum over all three types(i.e. personnel, equipment, and construction) but just for byear=0. Thus we have

```
[FUNDING byear=0][PROPOSAL]:PROPOSAL.pcn,FUNDING.amt;
```

Since byear is a category attribute, restriction on it must take place prior to the summarization.

An example of restriction on a summary value attribute is illustrated by the query "What is the pcn for all proposals with funding greater than 100?". This query looks like

```
[FUNDING amt>100][PROPOSAL]:PROPOSAL.pcn;
```

In the case of a summary value attribute we will adopt the semantics that restriction take place after any summarization. Thus, for this query the total funding amount for a particular proposal is computed and then the test is made to see if that amount is greater than 100.

As a final example, consider the query "Which states have more than 5,000,000 blacks and what is their average income?" and the corresponding STRAND expression

```
[STATE][POPULATION count>5000000,race=black]:
                    STATE.name,POPULATION.avginc;
```

This query restricts both summary value and category attributes. Therefore, restriction both before and after the summarization is required. First, restriction with race=black is performed. Then summarization takes place to form the intermediate summary set

POPULATION	
count	avginc

This intermediate summary set represents the count and average income for blacks of both sexes in each state. Finally, restriction is done with count>5000000 on the intermediate summary set.

5. Summary

The concept of summary data has been introduced. The semantics of summary data has been incorporated into the relational data model. It was shown that summary data exists in many data base applications. The advantage of the extended model is that queries requiring aggregate functions are easy to formulate. In fact, a query can be formulated without conceptualizing the query in terms of aggregation.

As a vehicle for describing operations on this extended model we presented the query language STRAND. In addition, two example applications; proposal tracking and socio-demographic data bases, have been presented.

Acknowledgements

The author would like to thank Susan Eggers, Peter Kreps, Arie Shoshani and Harry Wong for the many times they have read this paper and for their helpful comments. Special thanks go to Mike Stonebraker for his comments and suggestions.

References

[Allman 76] Allman, E., Held, G. and Stonebraker, M. "Embedding a Data Manipulation Language in a General Purpose Programming Language," In Proc. 1976 ACM-SIGPLAN-SIGMOD Conference on Data Abstractions, Salt Lake City, Utah, March, 1976.

[Chamberlain 76] Chamberlain, D. D. et. al. SEQUEL 2: A Unified Approach to Data Definition Manipulation and Control In IBM J. Res. Develop. 20, No. 6, 560-575(1976)

[Chen 76] Chen, P. The Entity-Relationship Model-- Toward a Unified View of Data. In ACM Transactions on Database Systems (March, 1976) 9-36

[Codd 80] Codd E.F. Data Models in Database Management In Proceedings of the Pingree Workshop (June, 1980)

[Johnson 80] Johnson, R. Modelling Summary Data with the Entity Relationship Model Technical Report 10647 Lawrence Berkeley Laboratory (May 1980)

[Shneiderman 80] Shneiderman, B., Thomas, G. Path Expressions for Complex Queries and Automatic Database Program Conversion. In Proceedings of the Sixth International Conference on Very Large Data Bases 33-44

[Scheuermann 79] Scheuermann, P., Schiffner, G., and Weber, H. Abstraction Capabilities and Invariant Properties Modelling Within the Entity-Relationship Approach. In Proceedings of International Conference on Entity-Relationship Approach to Systems Analysis and Design 210-229

[Shoshani 78] Shoshani, A. CABLE: A Language Based on the Entity-Relationship Model Technical Report UCID-8005 Lawrence Berkeley Laboratory (January 1978)

[Smith 77a] Smith and Smith Database Abstractions: Aggregation and Generalization. In ACM Transactions on Database Systems (June 1977) 105-133

[Smith 77b] Smith, J.M. and Smith, D.C.P. Database Abstractions: Aggregation. In Communications of the ACM (June 1977) 405-413

[Stonebraker 76] Stonebraker, M. et. al., The Design and Implementation of INGRES". In TODS 2,3, September 1976.

TECHNICAL INFORMATION DEPARTMENT
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720