Proceedings of the IASTED International Conference
INTELLIGENT SYSTEMS AND CONTROL
November 19-22, 2001, Clearwater (Tampa), Florida, USA

LIMITATIONS OF RELATIONSHIPS CONSTRUCTED FROM COINCIDING DATA

J.H. TER BEKKE and J.A. BAKKER
Faculty of Information Technology and Systems, Delft University of Technology,
Mekelweg 4, 2628 CD Delft, The Netherlands, {j.h.terbekke,j.a.bakker}@its.tudelft.nl

ABSTRACT

Partitioning of data with the *group by* clause is in SQL defined by one or more attributes from a single, possibly derived relation. The language therefore allows groupings defined by compositions of basic and/or referential attributes, i.e. foreign key attributes. However, partitioning applied to a view derived from more than one relation can lead to misinterpretation, ambiguity and/or incomplete results of which examples are shown. We argue that the specification of data aggregated from different tables must solely be based on referential attributes. Consequently, if such attributes are absent, the only way to partition the data reliably is to improve the underlying data model by adding structural aggregation. We illustrate this by simple examples where a semantic framework is used for database design and data manipulation.

Keywords: intelligent databases, SQL, semantic modeling, aggregate query.

1. INTRODUCTION

In relational databases a compound key (a_1, a_2, ...) is normally used to indicate a relationship between the participating component relations $R_1(a_1, ...)$, $R_2(a_2, ...)$ etc. These referential attributes can be used in SQL's *group by* clause; this way of grouping is called referential partitioning. An example can already be found in E.F. Codd's first seminal paper [2], containing the three-attribute compound key relation 'supply' with references to 'supplier', 'project' and 'part' (see also [3]). The reverse situation can also occur. An example is given in this paper: points (indicated by one simple key attribute) are linked to intervals (indicated by a compound key consisting of two simple attributes). Such a reverse coupling of relations requires special consideration of value-based relationships as expressed in SQL's *group by* clause. The following examples are drawn from the area of temporal databases although the general problem of aggregate queries can also be found elsewhere, in the area of data warehouses (see [11]).

There are many examples of temporal aspects in database applications. Often a distinction is made between applications in which absolute values are assigned to the aspect of time (for example actual data in a reservation system) and applications in which the aspect of time has a more relative character (for example historical data in a health care database). In the last mentioned relative approach, the events are often ordered or sequenced.

In temporal databases, the concept of time is conceived of as an ordered sequence of points of a certain granularity suitable for the requirements of the application [7]. A unique point of reference is required in case of absolute time values. This can be a value from a particular calendar connected with an application or something that is culturally related. Within such a calendar another time calendar is needed. This can be based on position-related values.

Often a distinction is made between 'point events' (corresponding to time stamps) and 'duration events' (corresponding to intervals). Special problems can occur when both interpretations of time are used in the same application or query. Sometimes also different granularities play a role. In order to avoid compatibility problems, the system designer often chooses a solution in which the minimal time granularity is selected. Even the selected minimal granularity does not always guarantee a correct solution under all circumstances, however.

In order to illustrate these problems this paper contains a characteristic example and also presents a simple effective solution for the problem using aggregation abstractions. In an earlier paper we already indicated that the generalization and specialization abstractions could play an indispensable role in the registration of historical relationships as those occurring in work flow management and in version/configuration management applications [10].

In the past, much attention was paid to the algebraic aspects of temporal databases (see among others McKenzie and Snodgrass [5]). The present paper considers temporal aspects from another point of view. Special attention is given to the structural aspects of temporal databases. We describe a simple modeling method that can be used to implement temporal aspects reliably (see the semantic concepts in [8] or the overview in [6]). The power of the proposed generic method lies in the fact that it can be applied in each data model that supports aggregation (for example: relational, Entity-Relationship and semantic, etc.). The origin of the solution lies in the application of aggregation-based relationships instead of

relationships constructed from coinciding temporal data.

2. AN ILLUSTRATIVE RELATION

Consider figure 1, which contains a relation that is comparable to an example from literature (see Bettini, Wang, Bertino and Jajodia [1]). The relation could play a role in a hospital database and could be used for patient admissions and patient treatments by medical doctors.

pat#	patient	doc#	start time	end time
1	J. Smith	12	4/2/2001:08:30	4/2/2001:08:40
1	J. Smith	12	4/2/2001:09:01	4/2/2001:09:16
2	A. Brady	14	4/1/2001:23:30	4/1/2001:23:35
2	A. Brady	13	4/2/2001:10:01	4/2/2001:10:21
2	A. Brady	14	4/4/2001:12:35	4/4/2001:12:45
2	A. Brady	15	4/4/2001:13:30	4/4/2001:13:55
2	A. Brady	15	4/5/2001:17:05	4/5/2001:17:20

Figure 1: Patient treatments

Duration events are indicated by two time stamps (month/day/year:hour:minute). This table allows simple relational queries such as single table queries specifying groupings to determine patient's total treatment duration. However, more complex queries, require an extended data structure, which will be shown in section 5. That is the reason why in section 3 we introduce some semantic modeling concepts from the Xplain approach [8]. Section 4 contains a more generic and realistic example of the temporal problem. It describes situations occurring in hospital registrations.

3. SEMANTIC MODELING CONCEPTS

Databases can be based on data models in which three types of abstractions are applied:

- *classification*
  The real world can be described by considering the properties of relevant objects, where a property is defined as a fundamental notion. The abstraction leading to a property (or type) is called classification. Examples of properties are 'name', 'address' and 'city'.
- *aggregation*
  Aggregation is defined as the collection of a certain number of properties into a *type*, which in itself can be regarded as a new property (note the analogy with the mathematical set concept). A property occurring in an aggregation is called an attribute of the type. A type without properties is called *base*. Aggregation allows view independence: we can discuss the obtained type (possibly as a property) without referring to the underlying attributes. For example, the properties 'name', 'address' and 'city' can be aggregated into the type 'patient', whereas 'patient' can also be a property of 'treatment'.
  In abstraction hierarchies, aggregation is indicated by a line connecting the centers of two facing rectangle sides, where the aggregate type is placed above its attributes, indicates aggregation. Other examples of aggregation can be found in figures 2 and 5.
- *generalization*
  The third type of abstraction is generalization: it is defined here as recognizing similar attributes from various but related types and combining these into a new type (note the analogy with the intersection operation from set theory).

4. A CHARACTERISTIC EXAMPLE

Consider a relational database containing patient admissions and treatments by physicians in a hospital. In the database the following tables occur:

- *doctor*
  A distinction must be made between the doctor who is responsible for a patient's admission to hospital and the doctors who treat the patient. It is obvious that the doctor responsible for admission can be one of the doctors who provide medical treatments.
- *patient*
  Patients can be hospitalized to undergo certain medical treatments. Several applications require determination of information per patient. That is why a separate patient table is introduced in the database. This table contains patient data independently of their medical records.
- *treatment*
  For several (medical and financial) reasons it is necessary to register certain treatment details. It is assumed that a treatment takes some time, hence the duration attribute in the data model. The duration of the treatment determines the patient costs. This relation references patient and doctor.
- *admission*
  Patients can be admitted to hospital for a number of treatments. This makes it necessary to introduce a hospital admissions registration as well. Here the physician responsible for admission is found. This relation also references patient and doctor.

This results in the following global relational model for hospital applications:

relation doctor (doctor_id, name, telephone, department)
relation patient (patient_id, name, address, city)
relation treatment (patient_id, doctor_id, date, duration)
relation admission (patient_id, doctor_id, admission_date, release_date)

Converting the relational definitions above into semantic model equivalents is straightforward as only aggregation

relationships occur. Aggregation occurs in the relational model when referential integrity is expressed by means of foreign and primary keys. In the semantic model aggregation occurs whenever a type acts as an attribute. The foregoing relational model (including *base* domains such as 'name') is therefore expressed as follows:

type doctor = name, telephone, department.
type patient = name, address, city.
type treatment = patient, doctor, date, duration.
type admission = patient, doctor, admission_date, release_date.

The aggregation relationships between the types of this semantic model can also be expressed in an abstraction-hierarchy (see figure 2).

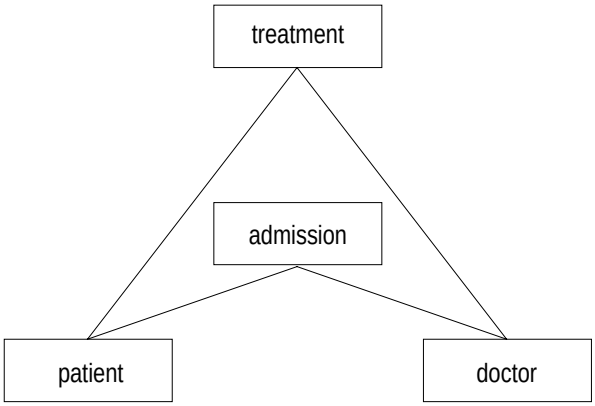

Figure 2: Treatment and admission as independent aggregations

This model allows certain derivations of aggregated information using data partitioning.

Query 1: Determine total treatment time per patient. This information is required including the patient's personal data.

In SQL the required query has the following structure:

select <target>
from <relations>
where <condition>
group by <attributes>;

The <target> of this query consists of attributes and set operations, applied to data from <relations> in the database. A <condition> may be applied to the selected data. When data must first be partitioned to obtain the derivation, this grouping can be given in <attributes>. Groupings are completely based on <attributes>-values; this implies that only non-empty groups can be considered. This comprehensive explanation leads to the following query in SQL:

select p.patient_id, name, address, city, *sum* (duration)
from treatment t, patient p
where t.patient_id = p.patient_id
group by p.patient_id, name, address, city;

From a join of patients and treatments from the respective tables we can derive the required information. The result of this join is comparable to the table in figure 1. The total duration is expressed by *sum* (duration). This query can also be formulated in a semantic query language. Here it has a completely different structure because different language constructs are used for derivation and selection. For derivation the *extend* command adds a new temporary column to a table. This column is used for intermediate results in complex derivations. The general syntax of the relevant *extend* command is as follows:

extend <subtype> *with* <extension> =
    <function> <maintype> [*its* <simple expression>]
    [*where* <simple condition>]
    per <subtype>.

For selection, the *get* command is used. It is similar to a simple *select* statement in SQL. The general syntax of the *get* command is as follows:

get <maintype> *its* <simple expression>
    [*where* <simple condition>].

The query is now expressed as follows:

extend patient *with* totality =
    total treatment *its* duration
    per patient.

get patient *its* name, address, city, totality.

The following reliable SQL-expression is equivalent to the semantic expression above.

create view treatment_time (patient_id, totality) *as*
select patient_id, *sum* (duration)
from treatment
group by patient_id
union
select patient_id, 0
from patient
where patient_id *not in*
    (*select* patient_id
    from treatment);

select p.patient_id, name, address, city, totality
from patient p, treatment_time tt
where p.patient_id = tt.patient_id;

There is a remarkable difference exists between the first and second SQL query formulation. The first formulation (without *union*) is weak in the sense that the resulting list does not contain the patients who did not undergo any medical treatment, while the second formulation (with *union*) does. This makes the last one reliable under *all* circumstances, i.e. it also gives the correct result if conditions are added to the selection criterion (see also [9]). The last query contains a correct referential partitioning.

After this introduction we consider the following query:

Query 2: Determine the total treatment duration per admission.

SQL enables us to join the tables admission and treatment. In this join the following conditions should play a role:

admission_date ≤ treatment_date ≤ release_date.

This approach results in the following data grouping statement in SQL:

```
select a.patient_id, admission_date, release_date,
       sum (duration)
from admission a, treatment t
where a.patient_id = t.patient_id
      and t.date >= a.admission_date
      and t.date <= a.release_date
group by a.patient_id, admission_date, release_date;
```

In the semantic language of Xplain [8] this problem cannot be solved, simply because the structural aggregation relationship between treatment and admission is missing. This brings us to a further investigation of the proposed SQL solution. Does SQL really present a solution or is a reliable solution only suggested? Therefore, let's investigate the following tables in figure 3 containing data according the first data model shown in figure 2.

treatment

treatment_id	patient_id	doctor_id	date	duration
1	1	12	4/2/2001	15
2	2	13	4/2/2001	20
3	2	14	4/4/2001	10
4	2	15	4/5/2001	15
5	3	13	4/7/2001	10

admission

patient_id	doctor_id	admission_date	release_date
1	12	4/2/2001	4/4/2001
2	14	4/1/2001	4/4/2001
2	15	4/4/2001	4/8/2001

Figure 3: Database contents based on the first model

The following remarks can be made about these database contents:

1 It is not possible to group all treatments unambiguously. For example it is unclear to which admission treatment 3 belongs (to admission 2 or admission 3?). From the admission table it is clear that patient 2 was released on the 4th of April and hospitalized again on the same day.

2 Not all treatments are related to admissions. For example treatment 5 is apparently a treatment in an outpatients' department (i.e. concerning an outpatient) for which no admission was required. The model of figure 2 allows this because it does not require any relationship between admission and treatment.

5. DATA MODEL REVISION

The desired information cannot be derived correctly using the above query, in spite of the SQL solution. A finer granularity of time stamps (viz. addition of hours, minutes, seconds, ..) is not practical and does not really solve the problem. A better strategy would be to adapt the model with a structural aggregation relationship between treatment and admission. This solution does not require any synchronization based on arbitrary time stamps and avoids granularity choices; the model becomes in fact time independent and therefore also suitable for a multi-temporal environment such as the Internet. This structural solution is expressed with the following data model:

type doctor = name, telephone, department.
type patient = name, address, city.
type admission = patient, doctor, admission_date, release_date.
type treatment = admission, doctor, date, duration.

This can simply be converted into the following relational definitions:

relation doctor (doctor_id, name, telephone, department)
relation patient (patient_id, name, address, city)
relation admission (admission_id, patient_id, doctor_id, admission_date, release_date)
relation treatment (treatment_id, admission_id, doctor_id, date, duration)

This database could contain the following data (figure 4).

treatment

treatment_id	adm_id	doctor_id	date	duration
1	1	12	4/2/2001	15
2	2	13	4/2/2001	20
3	3	14	4/4/2001	10
4	3	15	4/5/2001	15

admission

adm_id	patient_id	doctor_id	adm_date	rel_date
1	1	12	4/2/2001	4/4/2001
2	2	14	4/1/2001	4/4/2001
3	2	15	4/4/2001	4/8/2001

Figure 4: Database contents based on the improved data model

Be aware that data about polyclinic treatment 5 must be stored in a separate table containing only treatments in the outpatients' department.

The abstraction hierarchy of figure 5 corresponds to these definitions. This structure creates a registration without anomalies: treatments cannot be assigned to an unknown admission or to more than one admission. The dotted line between treatment and patient can now be derived from the given connections between treatment and admission and the connection between admission and patient (i.e. treatment *its* admission *its* patient).

The query to determine total treatment duration per admission can now be formulated in Xplain as follows:

extend admission *with* totality =
    total treatment *its* duration
    per admission.

Now, intrinsic and derived data about admissions can be retrieved:

get admission *its* patient, doctor, admission_date,
        release_date, totality.

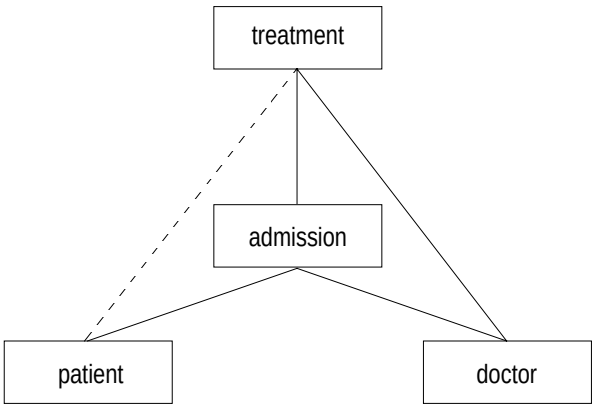

Figure 5. Treatment and admission as dependent aggregations

Conversion of this query expression now results in the following reliable SQL formulation.

```
create view treatment_time (admission_id, totality) as
select a.admission_id, sum (duration)
from treatment t, admission a
where t.admission_id = a.admission_id
group by a.admission_id
union
select admission_id, 0
from admission
where admission_id not in
    (select admission_id
    from treatment);

select patient_id, doctor_id, admission_date, release_date,
      totality
from admission a, treatment_time tt
where a.admission_id = tt.admission.id;
```

6. PRACTICAL GUIDELINES

The described problem is caused by the procedural *join* specification of SQL. Users become uncertain about the correctness and performance of the specific solution. This problem has been tackled here by introducing the orthogonal semantic concepts from the Xplain data language in the development phase of the relational database. The conversion into relational concepts results in unique, consistent, well-performing and portable SQL definitions and SQL queries.

The basic constructs of SQL are easy to learn. However, as users move on to more complex queries like joins or correlated sub-queries, the lack of orthogonality and the abundance of pitfalls can cause confusion about the semantics of SQL specifications. A practical alternative has been developed for this problem. This solution consists of open software which is freely downloadable from the Internet [4] and can even be used in more phases of information systems development.

Xplain does not offer the multiplicity of relationships found in some information modeling literature, but it supports all possible cases with only two relationships: aggregation and specialization (including multiple inheritance). Its visual side does not strain people's cognitive abilities. It just offers easy drawing and a distinction between aggregation and inheritance. Therefore, it is practical to use Xplain when certain exactness has to be introduced during the design phase or the requirements analysis phase. The visual model can be refined from high-level business semantics to the semantic detail necessary for information systems.

CONCLUSION

It has been shown that time-dependent data aggregation in SQL can best be specified using time-independent referential attributes. This means that temporal properties can best be interpreted as basic attributes. One should not use them in join conditions in order to prevent misinterpretations and ambiguities. The consequence is that in absence of the necessary referential attributes, one must add aggregations to the data model in order to achieve correct data aggregation.

REFERENCES

[1] C. Bettini, X. Sean Wang, E. Bertino and S. Jajodia, Semantic assumptions and query evaluation in temporal databases, *Proc. ACM SIGMOD '95*, 257-268.

[2] E.F. Codd, A relational model of data for large shared data banks, *CACM* 13, 6 (1970), 377-387.

[3] C.J. Date, *An introduction to database systems*, 7th edition, Addison-Wesley (2000).

[4] B. de Boer and J.H. ter Bekke, Applying semantic database principles in a relational environment, *Proc. IASTED Int. Symp. Applied Informatics* (AI2001), Innsbruck (2001), ed. M.H. Hanza, 400-405.

[5] L.E. McKenzie and R.T. Snodgrass, Evaluation of relational algebras incorporating the time dimension in databases, *ACM Comp. Surv.* 23, 4 (1991), 501-543.

[6] F.D. Rolland, *The essence of databases*, Prentice Hall (1998).

[7] X. Sean Wang, C. Bettini, A. Brodsky and S. Jajodia, Logical design for temporal databases with multiple granularities, *ACM TODS*, 22, 2 (1997), 115-170.

[8] J.H. ter Bekke, *Semantic data modeling*, Prentice Hall (1992).

[9] J.H. ter Bekke, Can we rely on SQL?, *Proc. 8th Int. DEXA Workshop '97*, Toulouse, France, ed. R.R. Wagner, IEEE Computer Society (1997), 378-383.

[10] J.H. ter Bekke, Semantic modeling of successive events applied to version management, *Proc. Int. Symp. on Cooperative Database Systems for Advanced Applications* (CODAS '96), Kyoto (1996), 32-39.

[11] J. Yang and J. Widom, Incremental computation and maintenance of temporal aggregates, *Proc. 17th Int. Conf. on Data Engineering* (2001), 51-60.