Information Retrieval from an Incomplete Data Cube

Curtis Dyreson
Department of Computer Science
James Cook University
Townsville, Queensland, Australia
curtis@cs.jcu.edu.au

Abstract

A *complete* data cube is a data cube in which every aggregate value in the multidimensional space is stored or can be computed. An *incomplete* data cube is a data cube in which points in the multidimensional space are missing and cannot be computed. This paper describes an incomplete data cube design. An incomplete data cube is modeled as a federation of *cubettes*. A cubette is a complete subcube within the incomplete data cube. The incomplete cube is built piecemeal by giving a concise, high-level specification of each cubette. An efficient algorithm to retrieve an aggregate value from the incomplete data cube is described. When a value cannot be retrieved because it is missing, alternatives at a lower precision that can be retrieved are identified. When a value can be partially computed (i.e., some of the values lower in the hierarchy are missing, but some are present) a measure of the completeness of the result is supplied along with the partially aggregated value. The design also includes an algorithm that removes redundant cubettes and an algorithm to increase the retrieval power of the federation through the creation of *virtual* cubettes.

Proceedings of the 22nd VLDB Conference
Mumbai(Bombay), India, 1996

1 Introduction

People often depend on summary data to make decisions. For instance, a regional sales officer will decide on the effectiveness of an advertising campaign by relying exclusively on summary data such as the volume of sales during the campaign. People also make summaries to conserve limited storage capacities. For example, once the sales data has been summarised, the unsummarised data can be compressed, moved to backup storage, or deleted.

A *data cube* is a popular organisation for summary data. A data cube is a multidimensional space of *aggregate* data.[1] Located at each point in the cube is a set of aggregate values. Each value in the set is the result of computing an aggregate function, such as **max** or **sum**.

The aggregate values in the cube are organised in a hierarchy. Values higher in the hierarchy are aggregations of values lower in the hierarchy. At the base of the hierarchy are aggregates computed on a subset of data chosen from some underlying relation(s) or view(s). The coordinates of points in the base are precise units of measurement in each dimension. Typical dimensions are time and space. As the next level higher-up in the hierarchy are points that have coordinates which are less precise units of measurement. A point in the hierarchy at this level depends upon all the points in the base that represent more precise measurements.

Example 1.1 [a sales data cube]
Consider a data cube that summarises 1995 sales data for a multinational corporation. The dimensions of the cube are 'time' of sale and 'location' of sale. The time dimension has coordinates for **hours**, **days**, **months**, and **years** in 1995; while the location has coordinates for **stores**, **cities**, **states**, **countries**, and

[1] 'Cube' is a misnomer since the cube is actually an N-dimensional space, but since 'cube' is the generally accepted term we will continue to (mis)use it in this paper.

trade_zones where the corporation's stores are located. The aggregate data is the number of items sold and the total amount paid for those items. In this cube, the number of items sold in California in January 1995 and the total amount of those sales would be located at the following coordinates: 'time=January 1995, location=California'. Since the number of items sold in California in January 1995 is a further aggregation of the number of items sold in California from 1 January 1995 through 31 January 1995, this point in the cube is higher in the hierarchy than the points corresponding to any day in January.
∎

The data cube is designed to support quick and easy "drill-down" and "roll-up" on the summary data. Drill-down is an operation that increases the precision of the summary data being viewed, while roll-up decreases that precision. For instance, suppose that the data manager is looking at monthly sales figures and notices that sales in January were much lower than those in the other months. To pinpoint the problem, the manager drills-down to look at daily sales in January.

A cube can be implemented using an *eager* strategy or a *lazy* strategy [Wid95]. The eager strategy precomputes and stores every point within the cube. In database terms, the eager strategy 'materialises' a set of aggregate views on the underlying relations. The advantage of the eager strategy is that points can be quickly fetched from the cube during query evaluation. The primary disadvantage is high storage cost. The size of the cube is the number of points in the N-dimensional space. In the example sales cube, assuming that there are 3200 stores and that the data is collected over a single year, the cube occupies nearly two hundred megabytes. The lazy strategy defers computing points in the cube to query evaluation time. Points in the cube are computed entirely from the underlying relations during query execution. The disadvantage of the lazy strategy is that it increases the cost of query evaluation. There are also 'semi-eager' strategies that materialise 'hot spots' above the base to improve query evaluation performance including a 'near-optimal' semi-eager strategy [HRU95].

Data cubes can be further classified as either *complete* or *incomplete*. In a complete data cube, every point in the cube is materialised (an eager strategy) or can be computed (a lazy strategy). But in an incomplete data cube, aggregate values at some of the points are not only missing, they cannot be computed from the underlying relations or from other values in the cube.

We conjecture that incomplete data cubes will be useful in the following situations.

- A common use of a data cube is to summarise historical data. Once the data is summarised it can be archived. Most users have a "telescoping" view of historical data. While historical data close to the current perspective of the user needs to be precise, summaries of data farther in the past can be imprecise. For instance, the sales manager needs an hourly breakdown of sales in the current year, but for previous years, a less precise daily summary will serve. If the historical data is archived, the portion of the data cube below "days" in previous years is absent, hence the data cube is incomplete.

- Data cubes will be used to summarise data from log files, flat files, and other sources that are expensive to summarise. When the cost of summarising data is excessive, users will eschew lazy strategies in favour of eager or semi-eager strategies. And because fully materialised data cubes occupy too much space, users will use semi-eager strategies to materialise only the parts of a cube that are of interest and leave the rest incomplete. For example, suppose that the sales cube summarises a log file of sales transactions instead of a sales relation. To search this large log file and retrieve data during query evaluation imposes a heavy burden on system resources, so the data cube administrators decide to use an incomplete data cube and package requests for more data in an overnight cron job.

- A data warehouse collects data from a variety of sources. It is reasonable to assume that not every source records data using the same system of measurements. For instance, when warehousing data from different stores, one store might measure sales on each item by the hour while another records only sales made each day. The same store may even change how it measures data during its lifetime. In a complete data cube, when the data is integrated at the warehouse, the different systems of measurements must be combined to create a unified view of the data. This process is sometimes called 'data scrubbing' [Wid95]. But in an incomplete data cube, the data can be inserted, unscrubbed and unchanged, at the appropriate location in the cube. In the above example, for the store that records only daily sales, the points in the cube "below" days are missing.

- Incomplete data cubes can seamlessly integrate several complete data cubes into a single larger data cube. An incomplete data cube is effectively just a collection of complete subcubes.

- Different security levels can be enforced in a data cube by denying access to portions of a complete cube, which results in the user viewing the cube as an incomplete data cube.

This paper describes methods for retrieving information from an incomplete data cube. Information retrieval in an incomplete data cube differs from that in a complete data cube in two ways. First, it requires that a concise, high-level description of the information in the cube exists and can be quickly searched during query evaluation. We anticipate that incomplete data cubes will contain a large number of materialised views. An important problem, common to semi-eager strategies in complete cubes, is how to determine which materialised view or views can be used to satisfy a query. This problem is complicated by the fact that when a new materialised view is added to an incomplete data cube, the increase in the amount of complete information in the incomplete data cube is greater than that contained in or derivable from the materialised view alone. Second, when the requested information is missing, mechanisms for helping the user deal with the missing information are needed. For instance, perhaps a partial answer could be returned when available.

This paper describes an incomplete data cube design. The design treats an incomplete data cube as a federation of *cubettes*. A cubette is a complete subcube within the incomplete data cube. The first section of the paper develops a realistic framework for systems and units of measurement. The framework is used in the following section to concisely specify the information content of a cubette. We then present an algorithm for retrieving information from the incomplete data cube. The time and space cost of the algorithm is analysed and shown to be reasonable. However, the query mechanism does not utilise all the information in the data cube, so we introduce a method to "extend" cubettes by creating *virtual* cubettes to support more queries. We also show how redundant cubettes can be eliminated resulting in modest space savings. When information cannot be retrieved because it is missing, alternatives at a lower precision that can be retrieved are identified. When a query can be partially answered (i.e., some of the values lower in the hierarchy are missing, but some are present) a measure of the completeness of the result is supplied along with the partially aggregated value. Finally, we discuss related and future work.

2 Measures

This section develops a simple, but realistic framework for the measurement of data that is used throughout the paper. Much of the framework should be familiar to most readers.

A *measure* is a system of measurement [Mer74]. We use the term 'measure' instead of the similar terms that are used by subdisciplines within the database research community. For example, in the temporal database community, measures are referred to as *granularities* [JCE+94], in the spatial database community as *subdivisions* [KEG93], in the statistical database community as *cluster*, *category*, or *classification attributes* [RS90, Sat91, Su83], and in SQL as *scale* [MS93]. Although the data modelling requirements of the various subdisciplines differ, the primary concept is the same. To the wider scientific community, the unifying concept is a system of measurement.

A *unit* (of measurement) is a subset chosen from the domain of interest. In this paper, we will treat all domains as sets of strings, without loss of generality. For instance, although all times belong to the temporal domain, we view each element in that domain as the *name* of some time, rather than the time itself.

A set of disjoint units, chosen from the same domain, forms a *measure*. A measure sometimes partitions a domain, but not always since the measure might not cover the domain. For example, the unit '`28/Mar/1995:17:50:24`' belongs to the measure of Universal Coordinated Time (UTC) seconds. UTC seconds does not partition the underlying time-line since UTC seconds are undefined prior to 1972. But UTC seconds is a perfectly good system of measurement since each second is disjoint, that is, each represents a unique portion of the time-line. In this paper, we will denote a unit using the notation u_m, where m is the measure to which that unit belongs.

Measures are sometimes related in that a measurement in one system is more *precise* than another. We formalise this concept below.

Definition 2.1 [precision]
A measure, m, is at least as *precise* as another, m', if

$$(\bigcup m') \cap (\bigcup m) \neq \emptyset \wedge \forall u_m[(u_m \cap (\bigcup m') \neq \emptyset) \Rightarrow (\exists u_{m'}[u_m \subseteq u_{m'}])]$$

■

Precision is a relationship between non-disjoint measures. A measure is said to be at least as precise as another when every unit in the more precise measure is either part of a single unit in the less precise measure or is not a part of any unit in the less precise measure. We will use precision to rank measures in terms of how finely they can locate objects in a domain. Consider the pair of measures seconds and years. Seconds is at least as precise as years since every second belongs to some year. But years is not as precise as seconds

since each year is composed of a number of seconds. A measurement in years less precisely locates an object in the underlying (temporal) domain.

Not every measure is defined over the same portion of the underlying domain. *Cover* relates systems of measurements defined over the same portion of the underlying domain.

Definition 2.2 [cover]
A measure, m, is said to *cover* another, m', if

$$\bigcup m' \subseteq \bigcup m$$

■

In the spatial domain, the measure of countries covers that of continents since every continent is composed of some number of countries (we assume that Antarctica is both a country and a continent).

The distinct concepts of cover and precision are related in the following definition.

Definition 2.3 [finer]
A measure, m, is a *finer* measure than a measure m', if m is not m', m covers m', and m is at least as precise as m'.
■

So if a measure both covers and is at least as precise as another, then the covering, precise measure is defined to be finer than the other. Seconds is a finer measure than years because it both covers the same portion of the time-line as years and is more precise than years.

We note that a consequence of two measures being in a finer than relationship is that every unit in one measure can be decomposed into a set of units at the finer measure. Said differently, every unit is the union of some number of units at a finer measure, for every finer measure.

The *measure graph* depicts the minimal set of "finer than" relationships between measures. Each node in the graph is a measure. Each edge in the graph represents a finer than relationship, but not all such relationships are represented by edges. There is an edge from measure m to measure m' if m is finer than m' and there is no other measure x such that m is finer than x and x is finer than m'. The full set of finer than relationships is obtained by taking the transitive closure of the measure graph. The measure graph is a directed, acyclic graph since the finer relation is asymmetric and irreflexive.

An example graph for the measures in the 'location' domain is shown in Figure 1. The finest measure is `stores`. It is finer than `trade_zones` (e.g., NAFTA), `cities`, `states`, and `countries`. Some measures, such as `cities` and `states`, are unrelated since neither is finer than the other (some cities span state borders and cities do not cover states).

2.1 Units and measures in multidimensional space

Units and measures in a data cube are chosen from a domain in multidimensional space. If we assume that the dimensions are independent (the standard data cube assumption), then the framework does not have to be changed to handle more than one dimension since the multiple dimensions combine to form a single domain that behaves exactly like any other measurement domain. We adopt the following naming convention for units and measures in a multidimensional domain. The name of each unit is an ordered tuple of unit names superscripted with the name of that domain, one chosen from each dimension. An example unit in the sales cube domain is

$$(1995^{TIME}_{\texttt{years}}, \texttt{California}^{LOCATION}_{\texttt{states}}).$$

When the context is clear we will drop the domain names. Similarly, the name of a measure in the multidimensional domain is an ordered tuple of measure names superscripted with the name of that domain, one chosen from each dimension, e.g.,

$$(\texttt{years}^{TIME}, \texttt{states}^{LOCATION}).$$

An easy method of determining whether a measure is finer than another is by checking whether a finer relationship holds for every dimension. So (`months`,`states`) is not finer than (`days`,`countries`), since `months` are not finer than `days`, but both measures are finer than (`years`,`countries`).

3 Cubettes

An incomplete cube is a federation of complete subcubes, which we call cubettes. Unfortunately this means we will continue to (mis)use the term 'cube' and perpetuate the myth that a cubette is 'cube-like.' In fact, a cubette is a multidimensional hierarchy as discussed in the next section.

3.1 Cubette specification

A cubette specification consists of a *cubette unit* and a *cubette measure*. The cubette unit is the top node in a multidimensional hierarchy. The hierarchy extends from a base in the multidimensional plane given by the cubette measure. Intuitively, the cubette unit is the portion of the domain over which the aggregate data in the cubette is maintained, while the cubette measure is the precision of that data. We will denote

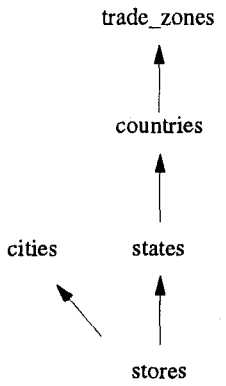

Figure 1: A spatial measure graph

a cubette with cubette unit, u_x, and cubette measure, m, as $u_x@m$ (literally, u_x at m). To be a valid cubette specification m must either be the same as or more precise than x, but m does not have to be finer than x.

Example 3.1 [cubette specifications]
To create a sales data cube, the user might issue the following cubette specifications:

```
(January 1995,United States)@(days,states)
(January 1995,Canada)@(days,countries)
(1995,Canada)@(months,countries)
(1995,California)@(days,states)
(1995,United States)@(months,states)
```

The specification `(January 1995,United States)@(days,states)` is a cubette that has aggregate values for every combination of day in January 1995 and state in the United States. The resulting incomplete cube is shown in Figure 2. In the figure, the cubettes are the regions enclosed by the dashed lines, the base of each cubette is shaded dark grey, and the other areas enclosed by a cubette are shaded light grey. Incomplete areas are represented by a question mark. Note that some of the cubettes overlap. We will ignore the issue of overlapping specifications in this paper. ∎

The cubette specifications are stored with the measure graph since the cubettes are typically accessed by traversing the measure graph. The specification $u_x@m$ would be stored in a list of specifications at node m in the graph.

3.2 The cubette store

The *cubette store* is the data collection, internal to a cubette, that stores the aggregate data in the cubette. At the logical level, a cubette supports the following operations:

- **new**($u_x@m$) – Create and populate a cubette with data drawn from some set of underlying relation(s), view(s), or other source data.
- **aggregate**($u_x@m$) – Compute and return the aggregate value(s) for the input cubette specification.
- **delete** – Remove this cubette, freeing the associated space.

At the physical level, each cubette could be implemented using a lazy strategy with only the base of the cubette, those units that are in the multidimensional plane of the cubette measure, materialised, or an eager strategy, with every point within the cubette materialised. Both eager and lazy cubettes can be mixed in an incomplete data cube, resulting in a 'semi-eager' incomplete cube. Internal to the cubette, the materialised data might be stored in an array, a sparse array, a hash table, or any other appropriate data structure. Since the physical implementation is hidden, the cubettes could be distributed across a network. We do not present protocols for handling distributed cubettes in this paper, we only note that separating the functionality permits this option.

Example 3.2 [a lazy SQL implementation]
A cubette can be implemented by mapping the operations to the appropriate SQL queries. Below we give example operations and the corresponding SQL code. We will assume that the cube is derived from a `SALES` relation which records the id, hour, store, and amount of each item sold, and that `CONVERT_m_to_x` relations that map units in measure `m` to measure `x` are available.

- **new**(`(1995,California)@(days,states)`) maps to a table creation query.

```
CREATE TABLE 1995_CAL_BY_DAYS_STATES(
    days CHAR(20), states CHAR(20),
    count NUMBER, sum NUMBER);
INSERT INTO 1995_CAL_BY_DAYS_STATES
SELECT C.days, C.states, COUNT(*), SUM(amount)
FROM SALES AS S,
    CONVERT_HOURS_STORES_TO_DAYS_STATES as C,
    CONVERT_YEARS_STATES_TO_HOURS_STORES as D
WHERE D.years = '1995' AND D.hours = S.hour
      AND D.states = 'California' AND
      D.stores = S.store AND S.hour = C.hours
      AND S.store = C.stores
GROUP BY C.days, C.states;
```

The table is created by first finding all the units in the measure of `(hours,stores)` that belong to the unit `(1995,California)`. All sales tuples in those units are retrieved from `SALES` and then

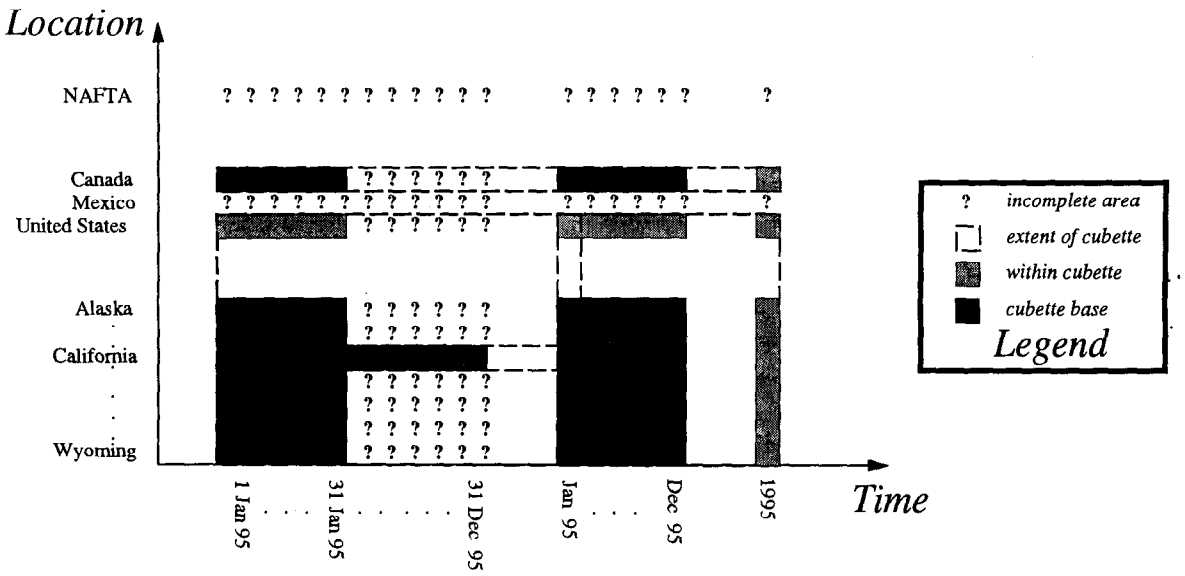

Figure 2: Cubettes in the sales data cube

"converted" to the measure of (days,states), where they are grouped and the appropriate aggregates computed on the groups. Alternatively, a materialised view could be used.

- **aggregate**((January 1995,California)@(months,states)) requests the aggregate values for January 1995.

```
SELECT D.months, D.states, SUM(count),
       SUM(amount)
FROM   1995_CAL_BY_DAYS_STATES AS V,
    CONVERT_MONTHS_STATES_TO_DAYS_STATES as D
WHERE  D.months = 'January 1995' AND
       D.states = 'California' AND
       D.days = V.days AND
       D.states = V.states
GROUP BY D.months, D.states;
```

In this query, the days in January are selected from the materialised view, they are grouped into months (there is only one group), and a **sum** aggregate is used to compute both the count and the amount for each group.

- **delete** simply drops the table.

```
DROP TABLE 1995_CAL_BY_DAYS_STATES;
```

■

3.3 Retrieval from an incomplete data cube

Although a cubette is created just once, it can be queried many times. A query can be thought of as a cubette that is computed by consulting the federation rather than the underlying data collection. A query consists of a *query unit* and a *query measure*, which are analogous to the cubette unit and measure.

The important semantic difference between a cubette and a query is that a cubette is complete, whereas only the data in the federation is available to a query. Consequently, when a query is processed it must be determined whether the query can be answered from the store of information.

Definition 3.3 [query satisfiability]
A query, $u_x@m$, can be satisfied if there exists a cubette, $u_{x'}@m'$, such that m' is finer than or the same as m and $u_x \subseteq u_{x'}$.
■

The definition states that a query can be satisfied if there exists a cubette with a finer measure than the query measure and a cubette unit that covers the query unit. Intuitively, a cubette at a finer measure can always be aggregated to present the data at a coarser measure. So if the cubette covers the same portion of the domain as the query, then the query can be satisfied.

Example 3.4 [querying the sales cube]
The sales officer queries the data cube from Example 3.1 to find data on sales in January 31 for the United States. The query (January 31,United States)@(days,countries) falls within a cubette that has a measure finer than the query measure, (January 1995,United States)@(days,states), so the query can be satisfied. But a similar query for

Mexico, (January,Mexico)@(days,countries), can not be satisfied.
∎

The above definition gives a sufficient condition for query satisfiability, but not a necessary condition. Some relationships exist between particular units that are not captured by the general framework of units and measures developed in Section 2. For example, **weeks** are not finer than **months** since certain weeks are in two different months, however there happens to be exactly four weeks in the month of February 1987. An incomplete data cube that stores weekly aggregate data for those four weeks has enough information to satisfy a query for monthly aggregate data for February 1987. This would not be detected using the definition given above since weeks are not finer than months. We expect these accidental relationships to be rare in practice, and expensive to detect, so we sacrifice completeness for query evaluation efficiency.

Below we give an efficient algorithm to satisfy a query.

Algorithm 3.5 [query evaluation]
We assume that the query is $u_x@m$.

1. Make a hash table, H, of all the units, u_y such that $u_x \subseteq u_y$, i.e., all the units above u_x in the multidimensional hierarchy.
2. Traverse the measure graph starting at m visiting all finer measures. For each cubette specification $u_{x'}@m'$ at a visited node, do the following.
   (a) We know that m' meets the query satisfiability constraints, it is finer than or the same as m.
   (b) If $u_{x'} \in H$ then the query is satisfied by this cubette, compute the query from the cubette store and exit.
   (c) Otherwise, this cubette does not satisfy the query, continue.

∎

Overall, the algorithm has a time cost of $O(E*G)+O(N)+O(E)+C$ where N is the number of cubettes, E is the number of edges in the measure graph, G is the cost of "converting" u_x to measure y, and C is the cost of computing the query from the cubette store once the appropriate cubette has been found. Traversing the measure graph above m to build H costs $O(E*G)$. Traversing the measure graph below m costs $O(E)$ to visit each node. There are at most N specifications distributed among the nodes so the loop costs $O(N)$ overall, assuming that the hash table lookup in H is constant. In practice we expect $N >> (E+G+C)$, so the cost of computing a query is dominated by the cost of iterating through the cubette specifications at the measures below m.

The key to the algorithm's efficiency is that the cubette specification concisely describes a region in the multidimensional hierarchy and that the test to determine whether the query falls within that region is just a single hash table lookup.

3.4 Redundant cubettes

A redundant cubette is a cubette than can be computed by another cubette in the federation. For example, the cubette '$\mathbf{1995}_{\mathbf{years}}$**@months**' is redundant if the cubette '$\mathbf{1995}_{\mathbf{years}}$**@days**' exists since the monthly count for 1995 can be computed from the daily count. Redundant cubettes waste space, but are easily detected and eliminated. A redundant cubette can be detected by removing the cubette from the list of cubette specifications and posing it as a query. If the query can be satisfied, then the cubette is redundant and can be eliminated.

3.5 Virtual cubettes

A virtual cubette is a cubette in which the base aggregate data is contained in other cubettes. A virtual cubette can be thought of as a placeholder, a specification of how to aggregate data contained in other cubettes. A virtual cubette supports the following operations:

- **new**($u_x@m, \{u_{x'_1}@m'_1, \ldots, u_{x'_n}@m'_n\}$) – The base of the virtual cubette is derived from the set of cubette specifications.
- **aggregate**($u_x@m$) – Compute and return the aggregate value(s) for the input cubette specification using the derived base.
- **delete** – Remove this virtual cubette.

Like a cubette, a virtual cubette can be implemented using either a lazy or an eager strategy.

Example 3.6 [a lazy SQL implementation]
A virtual cubette can be implemented by mapping the operations to the appropriate SQL queries. This particular implementation maps the virtual cubette specification to a view. Below we give example operations and the corresponding SQL code. We will assume that **CONVERT_m_to_x** relations that map units in measure **m** to measure **x** are available.

- **new**(Q, $\{U_1, \ldots, U_{12}\}$) where
  Q = (**1995**, **California**)@(**months**, **states**),
  U_1 = (**1/1995**, **California**)@(**months**, **states**),

..., and
U_{12} = (12/1995, California)@(months, states) maps to the following view creation query.

```
CREATE VIEW 1995_CAL_BY_MONTHS_STATES AS
SELECT D.months, D.states,
       SUM(count), SUM(amount)
FROM 1_1995_CAL_BY_MONTHS_STATES AS JAN,
   2_1995_CAL_BY_MONTHS_STATES AS FEB,
              ...
   12_1995_CAL_BY_MONTHS_STATES AS DEC,
   CONVERT_YEARS_STATES_TO_MONTHS_STATES as D
WHERE  (D.years = '1995' AND
        D.states = 'California' AND
        D.months = JAN.months AND
        D.states = JAN.states) OR
           ...
       (D.years = '1995' AND
        D.states = 'California' AND
        D.months = DEC.months AND
        D.states = DEC.states)
GROUP BY D.months, D.states;
```

The view is created by retrieving all the units in the measure of (months,states) that belong to the unit (1995,California) from the underlying cubettes, grouping the units, and then computing the appropriate aggregates on the groups. This view could be materialised to improve performance.

- The other operations, **aggregate** and **delete**, are the same as those in the cubette SQL implementation.

∎

Virtual cubettes are created when a cubette specification is added to the federation of cubettes. In some cases, the specification allows the incomplete data cube to compute more aggregate data than is given by the new specification. When a new cubette is added, it may be the case that the new cubette "extends" or is "extended by" other cubettes.

Definition 3.7 [cubette extension]
A cubette, $u_x@m$, *extends* a set of cubettes, $\{u_{x'_1}@m'_1, \ldots, u_{x'_n}@m'_n\}$, if there exists some specification, $u_\tau@z$, such that

- every measure in the set $\{m, m'_1, \ldots, m'_n\}$ is finer than or the same measure as z,
- $u_\tau \subseteq (u_x \cup (\bigcup\{u_{x'_1}, \ldots, u_{x'_n}\}))$,
- and, $u_\tau@z \notin \{u_x@m, u_{x'_1}@m'_1, \ldots, u_{x'_n}@m'_n\}$.

∎

The notion of extension is best explained by an example.

Example 3.8 [cubette extension]
Suppose that we add the cubette (January 1995,Mexico)@(days,countries) to the sales data cube. This specification is extended by the cubettes that cover the other countries for days in January to cover the NAFTA trade zone for days in January as shown in Figure 3. By extending the cubette to cover NAFTA, a query for (January 1995, NAFTA)@(months,trade_zones) can be satisfied. Without extending the cubette, this query could not be satisfied.
∎

So cubette extension is a method of increasing the utilisation of the information in the federation of cubettes. Below we give an algorithm to compute an extension when a new cubette, $u_x@m$, is inserted. The extension creates virtual cubettes.

Algorithm 3.9 [cubette extension]

1. Compute all the measures that are coarser than m, $T = \{\tau_1, \ldots, \tau_n\}$.
2. Compute all the units that cover u_x, $UT = \{(u_\tau, \{u_x@m\}) \mid \tau \in T \wedge u_x \cap u_\tau \neq \emptyset\}$. Each element in UT is an ordered pair consisting of a unit and a set of cubette specifications such that the cubette units intersect the unit and the cubette measures are finer than or the same as the unit's measure. Initially, the set of cubettes contains only $u_x@m$ because it has yet to be determined which other cubettes satisfy these requirements.
3. Repeat the following until all cubettes have been tried.
   (a) Choose an untried cubette, $u'_x@m'$.
   (b) Determine which units this cubette could possibly extend or be extended by. For each (u_τ, S) in UT such that m' is finer than or the same as τ and $u_\tau \bigcap u'_x \neq \emptyset$, add $u'_x@m'$ to the set S.
4. Now determine if the set of candidate cubettes actually extends a set of cubettes. For every (u_τ, S) in UT such that $u_\tau \subseteq (\bigcup\{u_y \mid u_y@m' \in S\})$, add the *virtual* cubette $u_\tau@z$ where z is the coarsest cubette measure in S and S is the set of cubettes from which the virtual cubette is constructed.
5. Eliminate redundant virtual cubettes.

∎

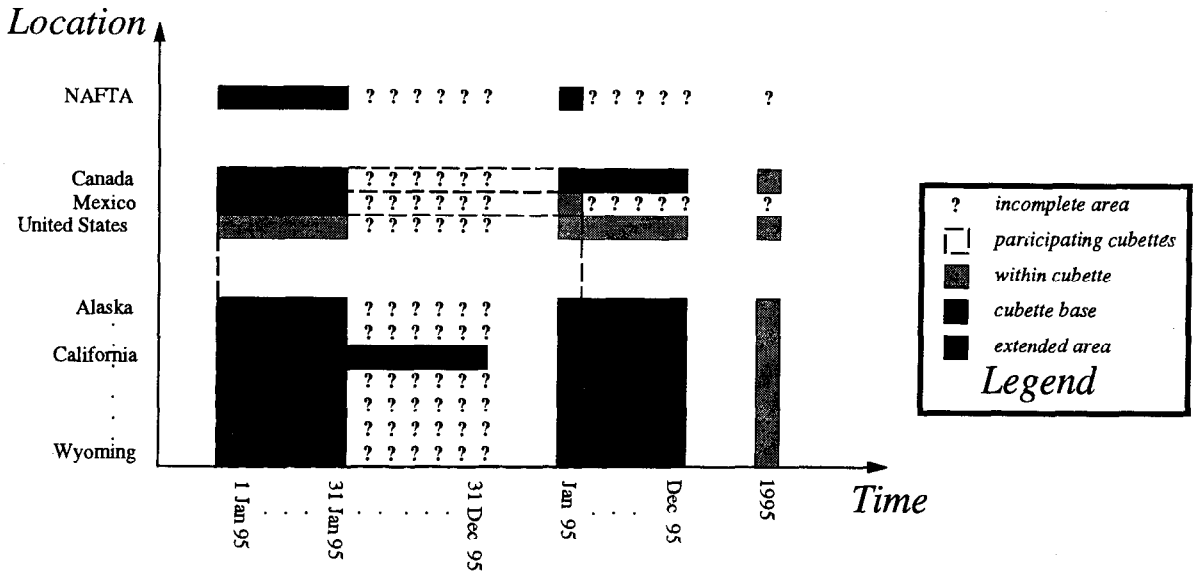

Figure 3: Extending the cubettes to cover NAFTA

The time cost of this algorithm is dominated by the repeat loop which costs $O(N*E*U)$ and the final step to eliminate redundant virtual cubettes which costs $O(N*N)$, where N is the number of cubettes, E is the number of edges in the measure graph, and U is the sum of the sizes of all the units that are supersets of u_β. The expensive step in the algorithm is step (b) in the repeat loop. We assume that this is implemented by precomputing a table of all the units that are in supersets of u_β, a table of size U. If a cubette has a finer measure and a cubette unit that is in the superset table, we mark in the table that the corresponding unit is covered. This step costs $O(E*U)$. At most $O(E)$ virtual cubettes are created (one for every measure).

While the cost of creating virtual cubettes is substantial, the cost is incurred infrequently. Cubette extension happens only when a new cubette specification is added. The payoff is that it keeps the cost of retrieving information low. The alternative is to extend cubettes during query evaluation, but this would substantially raise the cost of retrieving information from an incomplete data cube.

3.6 Removing or updating a cubette

A cubette can be updated or deleted at any time. However, when a cubette is either updated or deleted, all the virtual cubettes that use that cubette must also be updated or deleted. In the case of update, if the virtual cubettes are not materialised then no changes need to be made to any virtual cubette when updates are made to the aggregate values in a cubette. But in general, updating or deleting a cubette is an $O(E+N)$ operation, since it requires a search through the measure graph and the list of cubette specifications to change or delete the affected virtual cubettes.

4 Handling Unsatisfied Queries

When a query fails there are several mechanisms that can be used to restructure the query so that some useful information can be returned to the user.

4.1 Suggesting alternative queries

Most users will be unaware of all the data in an incomplete data cube. So when a user queries the cube, they might request information that is not in the cube. To improve the utility of a negative response to the request for information, an incomplete data cube has the capacity to supply an ordered list of *alternative* queries, which the cube can satisfy. An alternative query is a query at a less precise measure with the same query unit.

Below we give an algorithm to compute a list of alternative queries. The algorithm is a variation of the query evaluation algorithm which explores measures coarser than the query measure rather than finer measures.

Algorithm 4.1 [alternative queries]
We assume that the query is $u_x@m$.

1. Make a hash table, H, of all the units, u_y such that $u_x \subseteq u_y$, i.e., all the units above u_x in the multidimensional hierarchy.

2. Traverse the measure graph starting at m visiting all coarser measures. For each cubette specification $u_{x'}@m'$ at a visited node, do the following.

(a) We know that m' is less precise than m.

(b) If $u_{x'} \in H$ then the alternative query, $u_x@m'$ can be satisfied. Add this query to the list of alternative queries. Order the list by the distance in the measure graph from m to m' (assume that each edge has the same weight).

■

The cost of this algorithm is essentially the same as that for query evaluation.

Example 4.2 [alternatives]
In the example sales data cube depicted in Figure 3, the query `(1995, United States)@(days, states)` can not be satisfied. However, the query can be answered at the coarser measures of `(months,states)`, `(months,countries)`, `(years,states)`, and `(years,countries)`.
■

4.2 Partial query results

A query that requests aggregate data for a single point, i.e., $u_m@m$, wants the result of an aggregate computed over a subset of the multidimensional domain. But it may be the case that only part of that subset is contained in the incomplete data cube. Such a query can be "partially satisfied" by aggregating over the portion of the subset that is in the incomplete cube.

Definition 4.3 [partial query satisfiability]
A query, $u_m@m$, is *partially* satisfied if there exists cubettes, $u_{x'_1}@m'_1, \ldots, u_{x'_n}@m'_n$, such that every measure in the set $\{m'_1, \ldots, m'_n\}$ is finer than m and every unit in the set $\{u_{x'_1}, \ldots, u_{x'_n}\}$ intersects u_m.
■

In addition to the capacity to partially satisfy a query, an incomplete data cube has enough information to quantify the completeness of the partially computed result.

Definition 4.4 [completeness of a partial result]
The *completeness* of a partial result for a query $u_m@m$ is the percentage, with respect to the underlying domain(s), of u_m that is covered by the partial result.
■

For instance, information on January through June 1995 can satisfy approximately 50% of a request for information on 1995 since one-half of the unit 1995 in the temporal domain is covered by the first six months of that year. Other notions of completeness, such as the percentage of subunits used to compute the result, could also be defined.

Below we give an algorithm for computing a partial result from an incomplete data cube. The algorithm does a breadth-first search on the multidimensional hierarchy below the query. It computes as many subpoints within that hierarchy as possible.

Algorithm 4.5 [partial results]
We assume that the query is $u_m@m$.

1. Traverse the measure graph starting at m visiting all finer measures using breadth-first search. For each node visited do the following.

   (a) Let m' be the measure at current node. For each $u'_m \subseteq u_m$, pose the query $u'_m@m'$.

   (b) If the query is satisfied,

   - include the result in a set of partial results,
   - add the number of base units in u'_m to the size of the partial result,
   - and discontinue the search below this point.

2. Once the search has terminated,

   - the final partial result is the appropriate aggregate applied over the set of partial results, and
   - the completeness of the result is the percentage of the size of the partial result in relation to the size of u_m.

■

The cost of this algorithm is essentially the same as that for multiple query evaluations.

Example 4.6 [a partial result]
In the example sales data cube depicted in Figure 3, the query `(31 December 1995,United States)@(days,states)` can not be satisfied, however a partial answer can be given by aggregating all the values below 'time=31 December 1995, location=United States' in the multidimensional hierarchy. Only one value, for California, is available. However, if 90% of the stores are in California, then the answer is 90% complete.
■

5 Related Work

The ideas in this paper are synthesised from research in data warehouses, statistical databases, and temporal databases.

Efficient implementation of complete data cubes is a goal of the Stanford University Data Warehousing Project. Gupta, Harinarayan, and Quass describe a general SQL query rewriting technique to optimise aggregate queries [GHQ95]. The result of the rewrite is an efficient query execution plan that incorporates materialised aggregate views, i.e., points in a data cube, when those views are available. The focus of our paper, on an efficient technique to recognise which materialised views exist, is complementary and their techniques can be used to optimise the SQL queries generated by cubettes. Harinarayan, Rajaraman, and Ullman describe a near-optimal strategy to determine which parts of a data cube should be materialised to obtain the best performance at the lowest space overhead [HRU95]. When the strategy suggests that a materialised aggregate view is appropriate, a view at a particular measure is created. Their system is based on a lattice of measures only. Nevertheless, our paper is complementary insofar as once a large set of materialised views is created it is important to quickly identify whether that set satisfies a query.

Data cubes are a primitive statistical data model. In the tradition of the early, but influential SUBJECT data model [CS81], they support only two kinds of nodes: cluster nodes (for units) and cross product nodes (for combining domains). The greater sophistication in describing statistical data found in later statistical data models such as SAM* [Su83] and STORM [RS90], is absent from data cubes because data cubes do not have the same data modelling requirements. Statistical data models are designed to model complicated, nonstandardised, heterogeneous, real-world data sets whereas data cubes create their own simple, standardised, homogenised statistical data set, consequently, a simpler data model suffices.

Semantic, statistical data models suggest that, among other benefits, a meta-data or conceptual description of statistical data enables users to quickly find desired data [Sat91]. Cubette specifications are simple conceptual descriptions of the data in the incomplete data cube.

The statistical database aspect of data cubes is best understood as an extension of the work done by Malvestuto (and others) on *data integration* in statistical databases [Mal91]. Data integration is the creation of a unified view on a set of different, but homogeneous, statistical tables. In terms of the incomplete data cube model, Malvestuto uses a data model consisting only of units and shows how to determine whether queries are satisfiable or unsatisfiable given that the relationships between all the units are known. We extend Malvestuto's work by introducing measures. Measures are a useful conceptual tool, as evidenced by their independent development in other database subdisciplines [WJS93, WJL91]. More importantly, measures are the key to (relatively) fast query evaluation and computation of cubette extensions since measures quickly eliminate cubettes from consideration. Without measures, deciding if a cubette (partially) satisfies a query requires that all the units in the cubette be tested for overlap with the units in the query. In contrast, we have demonstrated that the cost of our evaluation algorithms is reasonable.

6 Conclusion

A data cube is a useful tool for exploring aggregate data because it enables users to view that data to any desired precision. An incomplete data cube is a data cube that supports all the precision control but which lacks some of the data to examine. Incomplete data cubes are useful in situations where the source data from which the cube is derived is no longer available or is too expensive to query frequently. Incomplete data cubes change the nature of retrieving information from a cube since the requested information may be missing.

The most important challenge in retrieving information from an incomplete data cube is to quickly determine whether or not the information exists. Towards this end, an effective representation of the aggregate data in the cube is essential. In this paper we developed a concise high-level specification of the information content of an incomplete data cube and showed how this specification could be used to efficiently retrieve information from the cube. We modeled an incomplete data cube as a collection of complete subcubes, which we called cubettes. Each cubette specification consists of a unit and a measure. The measure is the precision to which the aggregate data in the cubette is stored while the unit describes the extent of the available aggregate data. A request for data from the incomplete cube consults each cubette to determine if it can satisfy the request. The query evaluation mechanism does not combine data that is distributed over a number of cubettes, although the cubettes can be "extended" through the creation of virtual cubettes to synthesise data from a set of cubettes. If the requested data is not found in the cube we suggested several simple strategies to return some useful information to the user. One strategy is to suggest alternative nearby queries that can be satisfied. Another is to compute a partial answer along with a measure of the answer's completeness.

In future, we hope to extend the incomplete data cube design to permit more flexible management of the size of the cube. We plan to incorporate rules for 'vacuuming' historical data to reclaim space through automatic, timed roll-ups, an algorithm to find a minimal cubette specification, and a modified specification format to support partial specifications. The query power can also be extended by supporting temporal aggregates, moving window queries, and computing min-max bounds on partial results.

Acknowledgements

We would like to thank Richard T. Snodgrass for helpful suggestions on the paper and co-development of the units and measures, and also the comments made by the anonymous referees.

References

[CS81] P. Chan and A. Shoshani. SUBJECT: A directory driven system for organizing and accessing large statistical databases. In *Proceedings of the 7th Conference on Very Large Databases*, Los Altos, CA, September 1981.

[GHQ95] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *Proceedings of the 21st Conference on Very Large Databases*, Zurich, Switzerland, September 1995.

[HRU95] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. Submitted for publication., 1995.

[JCE+94] C. S. Jensen, J. Clifford, R. Elmasri, S. K. Gadia, P. Hayes, and S. Jajodia [eds]. A Glossary of Temporal Database Concepts. *ACM SIGMOD Record*, 23(1):52–64, March 1994.

[KEG93] Wolfgang Kainz, Max J. Egenhofer, and Ian Greasley. Modelling spatial relations and operations with partially ordered sets. *Int. J. Geographical Information Systems*, 7(3):215–229, 1993.

[Mal91] F. Malvestuto. Data Integration in Statistical Databases. In Z. Michalewicz, editor, *Statistical and Scientific Databases*, pages 201–232. Ellis Horwood Ltd., 1991.

[Mer74] G. H. Merriam, editor. *Webster's New Colligiate Dictionary*. Merriam-Webster, Springfield, Mass., 1974.

[MS93] J. Melton and A. R. Simon. *Understanding the New SQL: A Complete Guide*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.

[RS90] M. Rafanelli and A. Shoshani. STORM: a Statistical Object Representation Model. *IEEE CS Technical Committee on Database Engineering Bulletin*, 13(3), September 1990.

[Sat91] H. Sato. Statistical Data Models: from a Statistical Table to a Conceptual Approach. In Z. Michalewicz, editor, *Statistical and Scientific Databases*, pages 167–200. Ellis Horwood Ltd., 1991.

[Su83] S. Su. SAM*: A Semantic Association Model for Corporate and Scientific Databases. *Inf. Sciences*, 29:151, 1983.

[Wid95] J. Widom. Research problems in data warehousing. In *Proceedings of the 4th Int'l Conference on Information and Knowledge Management (CIKM)*, November 1995.

[WJL91] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with Granularity of Time in Temporal Databases. In *Proc. 3rd Nordic Conf. on Advanced Information Systems Engineering*, Trondheim, Norway, May 1991.

[WJS93] X. Wang, S. Jajodia, and V. Subrahmanian. Temporal Modules: An Approach Toward Temporal Databases. In *Proceedings of SIGMOD Conference*, pages 227–236, 1993.