SCHEMA EVOLUTION FOR STARS AND SNOWFLAKES

Christian E. Kaas Torben Bach Pedersen Bjørn D. Rasmussen
Aalborg University
Fr. Bajers Vej 7E, DK-9220 Aalborg Ø, Denmark

Keywords: Star schemas, snowflake schemas, schema evolution

Abstract: The most common implementation platform for multidimensional data warehouses is RDBMSs storing data in relational *star* and *snowflake* schemas. DW schemas evolve over time, which may invalidate existing analysis queries used for reporting purposes. However, the evolution properties of star and snowflake schemas have not previously been investigated systematically. This paper systematically investigates the evolution properties of star and snowflake schemas. Eight evolution operations are considered, covering insertion and deletion of dimensions, levels, dimension attributes, and measure attributes. For each operation, the formal semantics of the changes for star and snowflake schemas are given, and instance adaption and impact on existing queries are described. Finally, we compare the evolution properties of star and snowflake schemas, concluding that the star schema is considerably more robust towards schema changes than the snowflake schema.

1 INTRODUCTION

Data warehouses (DWs) based on a multidimensional (MD) data model are used in almost all businesses (Kimball, 1996; Pedersen et al., 2001b). Multidimensional models view data as multidimensional *cubes* where data is captured as *facts* with associated numeric *measures*. The facts are characterized by a number of *dimensions* organized in a hierarchy of *levels*. DW queries fall in two categories, *browse* queries that inspect the dimension hierarchies, and *aggregate* queries that aggregate measure values up to a given combination of dimension levels (Kimball, 1996).

Multidimensional DWs can be implemented using specialized multidimensional OLAP (MOLAP) DBMSs (Pedersen et al., 2001b), but the most common implementation platform is relational OLAP (ROLAP), i.e., RDBMSs storing data in relational *star* and *snowflake* schemas (Kimball, 1996). Star schemas have one denormalized table per dimension, while snowflake schemas normalize the dimension tables with one table per dimension level. DW schemas almost always evolve over time (Kimball, 1996; Kimball et al., 1998), which may invalidate existing SQL analysis queries. ROLAP data warehouses are usually queried by two different sets of tools. On-Line Analytical Processing (OLAP) tools are used for on-line, ad-hoc analyses by knowledge workers. Here, the SQL queries are generated dynamically based on a conceptual multidimensional schema stored in the tool. Thus, schema changes in the DW can be handled "easily" in the sense that queries will return the same results. However, materialized aggregates will have to be partly recomputed before the performance is the same. In contrast, relational *reporting tools* are used for off-line, periodic reporting and analysis. Here, the SQL queries are typically fixed and embedded in reporting programs. Thus, schema evolution in the DW means that the programs must be modified manually, a very expensive, cumbersome, and error-prone process. The main motivation for this paper is thus to provide details on when and how DW schema evolution affects existing SQL queries in reporting tools.

A survey of data warehouse projects and research issues (Vassiliadis, 2000) confirms that DW schema evolution is important. However, to our knowledge, the evolution properties of star and snowflake schemas and the impact on existing DW queries have not previously been systematically investigated.

This paper remedies the situation by systematically investigating the evolution properties of star and snowflake schemas, and, by implication, of fact constellation schemas. We start by formally defining star and snowflake schemas, and browse and aggregate queries over them. We then consider eight evolution operations, covering insertion and deletion of dimensions, levels, dimension attributes, and measure attributes. For each operation, the formal semantics of the changes for star and snowflake schemas are given, and instance adaption and impact on existing browse and aggregate queries are described. As previous work has already dealt with evolution in the *data instances*, we do not cover this aspect. Finally, we compare the evolution properties of star and snowflake schemas, looking both at which changes cause existing browse or aggregate queries to fail, as well as the ease of implementation and instance adaption. In all cases, the star schema is either superior to the snowflake schema or has the same problems. We thus conclude that the star schema is considerably more robust towards schema changes than the snowflake schema.

A lot of previous work on general schema evolution has been performed (Roddick, 1992). However, this work does not take the special characteristics of multidimensional schemas and queries, e.g., dimensions with hierarchies, into account. Several multidimensional models based on, e.g., UML (Lujan-Mora et al., 2002) and E/R (Sapia et al., 1998) have been proposed for conceptual and logical DW design Some models (Pedersen et al., 2001a; Body et al., 2002; Eder et al., 2001; Letz et al., 2002; Morzy et al., 2003) also consider temporal aspects of the *data instances* such as multiple version of dimensions and facts, but not schema evolution in the DW. However, none of these models consider schema evolution. The paper (Levene et al., 2003) investigates other properties of star and snowflake schemas such as redundancy, but does not consider schema evolution. Several papers (Bouzheghoub et al., 2000; Vassiliadis et al, 2000) consider DW schema evolution at the coarse-grained level of materialized views, either focusing on DW design (Bouzheghoub et al., 2000) or DW quality (Vassiliadis et al, 2000), but do not consider the special characteristics of multidimensional schemas and queries. Another paper (Mendelzohn et al., 2000) considers querying across schema evolution, but in a "pure" multidimensional model, and thus do not investigate evolution in star and snowflake schemas. The same is true for the FIESTA framework (Blaschka et al., 1999; Blaschka, 2000). Another paper (Hurtado et al., 1999) considers both schema evolution in a "pure" multidimensional model, and the mapping to relational schemas, but considers mainly instance rather than schema updates (which are only treated briefly), and considers only dimension updates. In comparison, the present paper provides a detailed treatment of schema evolution, including a formal semantics of the changes, provides details on the impact on existing DW queries, and considers both dimensions and fact/measures. Kimball (Kimball, 1996; Kimball et al., 1998) informally mentions that schema changes should be added "gracefully" to allow existing queries to work, but neither investigates this systematically and formally, nor compares the evolution properties of star and snowflake schemas.

We believe this paper to be the first to systematically investigate the evolution properties of star and snowflake schemas, including giving a formal semantics for evolution operations on these schemas, and investigating the impact on existing DW queries.

The rest of the paper is structured as follows. Section 2 describes the multidimensional schemas and queries considered in the paper. Section 3 examines effects of the eight evolution operations, while Section 4 summarizes the evolution properties for star and snowflake schemas. Finally, Section 5 concludes and points to future work.

2 MULTIDIMENSIONAL SCHEMAS

We now formally define star and snowflake schemas. In the following, we refer to the intuitive "MD-schema" that corresponds to our *conceptual* understanding of the cube that the star or snowflake schema implements, i.e., a schema with notions like cube, measures, dimensions, levels, and attributes.

Definition 1 (Star schema) A star schema is a 6-tuple: <D, fact, A, *attrib*, PKD, PKF> where D is a set of dimension tables, fact is a fact table, A is a set of attributes each with a name and a type, *attrib*: $A \rightarrow D \cup \{\text{fact}\}$ is a function mapping an attribute to either a fact table or a dimension table, PKD $\subset$ A is the set of dimension table primary keys where the primary key of the dimension table $d_i \in$ D is denoted PKd_i: PK$d_i \in$ PKD, and PKF $\subset$ A is the (composite) primary key of the fact table. Each $K \in$ PKF is a foreign key to exactly one dimension table $d_i \in$ D. □

The star schema uses denormalized dimensions which means that the dimension levels in the MD schema do not appear in the star schema. Instead a dimension table d_i in the star schema definition consists of one of the base dimension levels defined in the MD schema and all the dimension levels above it in the MD schema. The attributes a_j, j=1..n of d_i corresponds to the attributes of the base dimension level and all its preceding dimension levels just described.

Definition 2 (Snowflake schema) A snowflake schema is an 8-tuple: <L, B, fact, A, *attrib*, PKL, FKL, PKF>, where L is a set of dimension level tables, B $\subset$ L is a set of base dimension level tables, fact is a fact table, A is a set of attributes, *attrib*: A $\rightarrow$ L $\cup$ {fact} is a function mapping an attribute to either a fact table or a dimension level table, PKL $\subset$ A is a set of primary keys where the primary key of the dimension level table l$_i$ $\in$ L is denoted PKl$_i$: PKl$_i$ $\in$ PKL, FKL $\subset$ A is a set of foreign keys where the set of foreign keys for the dimension level table l$_i$ $\in$ L is denoted FKl$_i$: FKl$_i$ $\subset$ FKL $\subset$ A, PKF $\subset$ A is the (composite) primary key of the fact table. Each K $\in$ PKF is a foreign key to exactly one base level dimension table d$_i$ $\in$ B. The snowflake schema can be viewed as a directed acyclic graph where the tables are the nodes and the foreign key dependencies represent the edges. The root of the graph will be the fact table and the root's directly connected nodes are the base dimension level tables. The edges emanating from the root connects to the base dimension level tables and each base dimension level table is the root of a subgraph. No two subgraphs can be connected.$\square$

The base dimension levels B of the snowflake schema are the dimension levels that are associated with facts in the MD-schema. However, this relation is represented as a foreign key dependency in the snowflake schema. The dimension level hierarchy of the MD-schema is also represented as foreign key dependencies where the table representing an underlying dimension level references the primary key of its upper dimension levels.

DW Queries Generally, only two query types are posed to data warehouses, namely 80% *browse* and 20% *aggregate* queries (Kimball, 1996). Browse queries only concern a single dimension and are used to investigate the dimension hierarchy. Aggregate queries aggregate the information in the fact table to a level specified using one or more dimensions. Prototypical star (top) and snowflake (bottom) browse queries are shown below.

SELECT DISTINCT($d_i.a_m$) FROM d_i WHERE P;
SELECT DISTINCT($l_j.a_m$) FROM $l_i, .., l_j$ WHERE P;

The star schema browse query returns all distinct values of the attribute a_m in the dimension table d_i for rows that match the predicate P. The snowflake schema browse query returns all distinct values of attribute a_m in dimension level table l_j for rows that match the predicate P. The FROM-clause of the browse query includes several dimension level tables. In addition the predicate P also describes how the dimension level tables $l_i, .., l_j$ shall be joined. Prototypical star (top) and snowflake (bottom) aggregate queries are shown below.

SELECT $d_i.a_{i1}, .., d_i.a_{ik}, .., d_j.a_{j1}, .., d_j.a_{jl}$
$AGG(fact.m_u), .., AGG(fact.m_v)$
FROM $d_i, .., d_j, fact$ WHERE P
GROUP BY $d_i.a_{i1}, .., d_i.a_{ik}, .., d_j.a_{j1}, .., d_j.a_{jl}$;

SELECT $l_i.a_{i1}, .., l_i.a_{ik}, .., l_j.a_{j1}, .., l_j.a_{jl}$,
$AGG(fact.m_u), .., AGG(fact.m_v)$
FROM $l_i, .., l_j, fact$ WHERE P
GROUP BY $l_i.a_{i1}, .., l_i.a_{ik}, .., l_j.a_{j1}, .., l_j.a_{jl}$;

The select list of the star schema aggregate query includes the dimension attributes $a_1, .., a_k$ of dimension table d_i and $a_1, .., a_l$ of dimension table d_j along with measures $m_u, .., m_v$ from the fact table fact. AGG can be one of the aggregate functions MIN, MAX, AVG, COUNT and SUM. Constraints that apply for rows being included is defined in the predicate P along with rules for joining dimension tables with the fact table. The snowflake query is similar, but includes several tables per dimension. Aggregate queries can use the aggregate functions MIN, MAX, COUNT, AVG and SUM. Of these, MIN, MAX, COUNT and AVG require the *granularity* of the fact table (Kimball, 1996) to remain constant in order to return correct results. This is not the case for SUM (SUM is *split-resistant*, see Section 3.7).

3 EVOLUTION OPERATIONS

We now go through each of the eight evolution operations and show their impact on star and snowflake schemas, and existing DW queries. We give a formal semantics for the change on star and snowflake schemas, in terms of the input (In), output (Out), preconditions (PCs), and schema changes (SCs). For star schema changes, we give as input a star schema: St = <D, fact, A, $attrib$, PKD, PKF> and for snowflake schemas the input: Sn = <L, B, fact, A, $attrib$, PKL, FKL, PKF>.

3.1 Insert attribute into level

Star changes Inserting an attribute a_{new} into a dimension level in an MD-schema corresponds to inserting an attribute a_{new} into a dimension table d_i in a star schema St = <D, fact, A, $attrib$, PKD, PKF>. **Adaption:** All rows in the dimension table being altered in the star schema must be updated such that all rows will either be assigned a default value for the new attribute or they will have to be updated separately. **Impact:** Inserting a new attribute into a dimension table cannot cause existing queries browse and aggregate queries to fail since no queries use this new attribute yet.

Semantics	
In	St, a$_{new}$, d$_i$
PCs	a$_{new}$ $\notin$ A, d$_i$ $\in$ D
SCs	St' = <D, fact, A$'$, $attrib'$,PKD,PKF> A$'$ = A $\cup$ {a$_{new}$} $attrib'$(a$_{new}$) = d$_i$
Out	St'

Snowflake changes Inserting an attribute a_{new} into a dimension level in an MD-schema corresponds to inserting an attribute a_{new} into a dimension level table l_i in a snowflake schema Sn = <L, B, fact, A, $attrib$, PKL, FKL, PKF>. **Adaption:** Much in the same way as the star schema the values for the new attribute is either set a default value or updated separately. If the table being altered is higher in the dimension level hierarchy (with coarser granularity) the updating of the rows will be faster than for the star schema since fewer rows needs to be updated. E.g. there are fewer rows in the “Category” table than in the Item table of the snowflake schema. **Impact:** Inserting a new attribute into a dimension level table cannot affect existing browse and aggregate queries using the schema since none of these includes the new attribute.

Semantics	
In	$Sn, \mathrm{a}_{new}, \mathrm{l}_i$
PCs	$\mathrm{a}_{new} \notin \mathrm{A}, \mathrm{l}_i \in \mathrm{L}$
SCs	Sn' = <L, B, fact, A$'$, $attrib'$,PKL,FKL,PKF>
	$\mathrm{A}' = \mathrm{A} \cup \{\mathrm{a}_{new}\}$
	$attrib'(\mathrm{a}_{new}) = \mathrm{l}_i$
Out	St'

3.2 Delete attribute from level

Star changes Deleting an attribute a_{del} from a dimension level in an MD-schema corresponds to deleting an attribute a_{del} from a dimension table d_i in a star schema St = <D, fact, A, $attrib$, PKD, PKF>. Before deleting the attribute the precondition has to be strengthened such that it is not the last attribute (besides the primary key) that is being deleted. **Adaption:** There is no need to update any of the rows when removing an attribute from a dimension table. **Impact:** Existing browse and aggregate queries which include this attribute will fail after performing this evolution operation on the star schema.

Semantics	
In	$St, \mathrm{a}_{del}, \mathrm{d}_i$
PCs	$\mathrm{a}_{del} \in \mathrm{A}$, $attrib(\mathrm{a}_{del}) = \mathrm{d}_i \in \mathrm{D}$
	$Q = attrib^{-1}(d_i) \wedge \lvert Q \rvert > 2$
SCs	St' = <D, fact, A$'$, $attrib'$,PKD,PKF>
	$attrib'(\mathrm{a}_{del}) = \emptyset$
	$\mathrm{A}' = \mathrm{A} \setminus \{\mathrm{a}_{del}\}$
Out	St'

Snowflake changes Deleting an attribute a_{del} from a dimension level in an MD-schema corresponds to deleting an attribute a_{del} from a dimension level table l_i in a snowflake schema Sn = <L, B, fact, $attrib$, PKL, FKL, PKF>. The precondition ensures that no last non-key attribute of dimension level table l_i can be deleted. **Adaption:** Dropping an attribute from a dimension level table does not require any rows to be updated. **Impact:** The outcome of deleting an attribute from a dimension level table will be that existing browse and aggregate queries which include this attribute will fail.

Semantics	
In	$Sn, \mathrm{a}_{del}, \mathrm{l}_i$
PCs	$\mathrm{a}_{del} \in \mathrm{A}$, $attrib(\mathrm{a}_{del}) = \mathrm{l}_i \in \mathrm{L}$
	$Q = attrib^{-1}(l_i) \wedge \lvert \mathrm{Q} \rvert > \lvert \mathrm{FKl}_i \rvert + 2$
SCs	Sn' = <L, B, fact, A$'$, $attrib'$,PKL,FKL,PKF>
	$attrib'(\mathrm{a}_{del}) = \emptyset$
	$\mathrm{A}' = \mathrm{A} \setminus \{\mathrm{a}_{del}\}$
Out	Sn'

3.3 Insert dimension level

There are three ways a new dimension level can be added to an MD-schema. Either a) as a new base dimension level, b) as a new top dimension level (not being directly connected to the fact) or c) between two existing dimension level.

Star changes Inserting a dimension level with a set of attributes attached in an MD-schema corresponds to inserting a set of attributes Q in a dimension level table d_i in a star schema St = <D, fact, A, $attrib$, PKD, PKF>. Additionally, in an MD-schema a classification is inserted but since the star schema is constrained in this matter no further than inserting a set of attributes is done. **Adaption:** Due to the denormalized dimensions of the star schema it makes no difference if the new dimension level is inserted as a new top dimension level or between two dimension levels. If, however, the new classification is inserted as a new base dimension level and causes a change in granularity then the fact table will have to be updated along with the dimension table. This is only possible if the facts can be “split” in a meaningful way, and the measure data allocated at the new, finer granularity. **Impact:** The new attributes which form the new classification will not cause existing browse queries to fail. Even if the new dimension level is inserted as a new base dimension level in the MD-schema and causes a change in granularity and thereby creating more rows in the dimension table it still will not affect browse queries because they include the DISTINCT keyword when querying a dimension table. Aggregate queries, on the other hand, can be affected by this operation when a new base dimension level is inserted causing change in granularity of the fact table. It is, however, only aggregate queries using MIN, MAX, COUNT and AVG that will fail while SUM will not because it is split-resistant. When a new base dimension level is inserted the impact is similar to inserting a new dimension into fact, see Section 3.7.

Semantics	
In	St, Q, d_i
PCs	$\mathrm{d}_i \in \mathrm{D}, \mathrm{Q} \cap \mathrm{A} = \emptyset, \mathrm{Q} \neq \emptyset$
SCs	St' = <D, fact, A$'$, $attrib'$,PKD,PKF>
	$\mathrm{A}' = \mathrm{A} \cup \mathrm{Q}$
	$\forall \mathrm{q} \in \mathrm{Q}$: $attrib'(\mathrm{q}) = \mathrm{d}_i$
Out	St'

Snowflake changes The semantics for inserting a new dimension level table depends on where it is being inserted. Inserting a new dimension level table l_{newtop} on top of an existing top dimension level table l$_i$ with a set of attributes Q in a snowflake schema Sn = <L, B, fact, A, $attrib$, PKL, FKL, PKF> has the following semantics.

Semantics	
In	Sn, l$_{newtop}$, l$_i$, Q
PCs	l$_{newtop}$ $\notin$ L, l$_i$ $\in$ L, Q $\cap$ A $= \emptyset$, {PKl$_{newtop}$} $\in$ Q, Q $\neq \emptyset$
SCs	Sn' = <L$'$, B, fact, A$'$, $attrib'$,PKL$'$,FKL$''$,PKF> L$'$ = L $\cup$ {l$_{newtop}$} PKL$'$ = PKL $\cup$ {PKl$_{newtop}$} FKL$'$ = FKL $\setminus$ FK$_i$ FK_{i_i}' = FKl$_i$ $\cup$ {PKl$_{newtop}$} FKL$''$ = FKL$'$ $\cup$ FKl$'_i$ A$'$ = A $\cup$ Q $\forall$q $\in$ Q: $attrib'$(q) = l$_{newtop}$
Out	Sn'

Inserting a new dimension level table l_{new} with a set of attributes Q between existing dimension level tables l_i and the set of tables M where l_i is classified according to the dimension level tables M in a snowflake schema Sn = <L, B, fact, A, $attrib$, PKL, FKL, PKF>:

Semantics	
In	Sn, l$_{new}$, l$_i$, M, Q
PCs	l$_{new}$ $\notin$ L, M = {m$_1$, .., m$_n$}, {l$_i$} $\cup$ M $\subseteq$ L Q = {PKl$_{new}$} $\cup$ {att$_1$, .., att$_n$}, Q $\cap$ A $= \emptyset$, Q $\neq \emptyset$
SCs	Sn' = <L$'$, B, fact, A$'$, $attrib'$, PKL$'$, FKL$''$, PKF> L$'$ = L $\cup$ {l$_{new}$} PKL$'$ = PKL $\cup$ {PKl$_{new}$} FKl$'_{new}$ = FKl$_{new}$ $\cup$ FKl$_i$ FKL$'$ = FKL $\setminus$ FKl$_i$ FKl$'_i$ = FKl$_i$ $\setminus$ {PKm$_1$, .., PKm$_n$} FKl$''_i$ = FKl$'_i$ $\cup$ {PKl$_{new}$} FKL$''$ = FKL$'$ $\cup$ FKl$_{new}$ $\cup$ FKl$''_i$ A$'$ = A $\cup$ Q $\forall$q $\in$ Q: $attrib'$(q) = l$_{new}$
Out	Sn'

Inserting a new dimension level $l_{newbase}$ with a set of attributes Q as new base dimension level replacing an existing base dimension level $l_{oldbase}$ in a snowflake schema Sn = <L, B, fact, A, $attrib$, PKL, FKL, PKF>:

Semantics	
In	Sn, l$_{newbase}$, l$_{oldbase}$, Q
PCs	l$_{newbase}$ $\notin$ L, l$_{oldbase}$ $\in$ L Q = {PKl$_{newbase}$} $\cup$ {att$_1$, .., att$_n$}, Q $\cap$ A $= \emptyset$, Q $\neq \emptyset$

SCs	Sn' = <L$'$, B$'$, fact, A$'$, $attrib'$, PKL$'$, FKL$'$, PKF$''$>
	L$'$ = L $\cup$ {l$_{newbase}$}
	FKl$'_{newbase}$ = FKl$_{newbase}$ $\cup$ {PKl$_{oldbase}$}
	FKL$'$ = FKL $\cup$ FKl$'_{newbase}$
	PKF$'$ = PKF $\setminus$ {PKl$_{oldbase}$}
	PKF$''$ = PKF$'$ $\cup$ {PKl$_{newbase}$}
	A$'$ = A $\cup$ Q
	$\forall$q $\in$ Q: $attrib'$(q) = l$_{newbase}$

Out	Sn'

Adaption: If the new dimension level table becomes a new top dimension level table all the rows in the former top dimension level table will need to have their new foreign key updated such that every row is classified according to its new top level. If instead the new dimension level table is inserted between two existing dimension level tables the lower table (with finer granularity) will have to be classified according to the new dimension level table and the new dimension level table will have to be classified according to the upper table (with coarser granularity). Lastly, if the new dimension level table becomes the new base table the fact table will have to be updated such that the existing fact references the new base table. Again, this is only possible if the facts can be "split" in a meaningful way. **Impact:** If the new dimension level table becomes a new top level table existing browse and aggregate queries will not fail since no existing queries will include this table. If, however, the new dimension level table is inserted between two dimension level tables both existing browse and aggregate queries will fail if they require that the two existing tables are able to join. A new base dimension level table will not cause existing browse queries to fail but all existing aggregate queries will fail because they try to join the former base level table with the fact table.

3.4 Delete dimension level

Just as for "Insert dimension level and insert into classification" there are three ways of deleting an existing dimension level and classification from an MD-schema. The dimension level being deleted along with its classification relationships can either be a) a top dimension level (not being directly connected to the fact), b) between dimension levels, or c) a base dimension level.

Star changes Deleting a dimension level l_{MD} in an MD-schema simply corresponds to deleting a set of attributes Q in a dimension table d_i in a star schema St. The attributes in Q must correspond to the attributes attached to l_{MD} and l_{MD} must not be a base level with no additional dimension levels in the dimension, if this is the case, the whole dimension should be deleted from the fact instead. **Adaption:** If the dimension level being deleted in the MD-schema is a top dimension level or is between two existing dimension levels there is no need for updating any instances in the dimension table because the denormalized dimension table in these cases has the same granularity as the base dimension level from the MD-schema. Therefore updating the instances of the dimension table containing the dimension level being deleted will only be required when it is the base dimension level in the MD-schema that is the one being deleted. When this is the case the fact table has to be updated to adapt to the new granularity. This is done by aggregating the fact table, see Section 3.7. **Impact:** Browse queries that use deleted dimension table attributes will of course no longer be able to execute. The same goes for aggregate queries but in addition if it was the base dimension level that was deleted from the MD-schema and has caused a change in granularity then these aggregate queries using MIN, MAX, COUNT and AVG will return incorrect results while SUM-queries continue to operate. Deleting the base dimension level in an MD-schema is similar to deleting a dimension level from a fact, see Section 3.8.

Semantics	
In	St, Q, d_i
PCs	$d_i \in$ D, Q $\subset$ A, $\lvert Q \rvert \neq 0$
SCs	St' = <D, fact, A$'$, $attrib'$, PKD, PKF> $\forall$q $\in$ Q: $attrib'$(q) = $\emptyset$
	A$'$ = A $\setminus$ Q
Out	St'

Snowflake changes Deleting a top dimension level l_{del} replacing it with l_i where l_i is classified to l_{del} in a snowflake schema Sn:

Semantics	
In	Sn, l_{del}, l_i
PCs	$\{l_{del}, l_i\} \in$ L $Q = attrib^{-1}(l_{del})$
SCs	Sn' = <L$'$, B, fact, A$'$, $attrib'$, PKL$'$, FKL$''$, PKF> FKL$'$ = FKL $\setminus$ FKl_i FKl'_i = FKl_i $\setminus$ $\{$PK$l_{del}\}$ FKL$''$ = FKL$'$ $\cup$ FK$l_i{}'$ $\forall$q $\in$ Q: $attrib'$(q) = $\emptyset$ A$'$ = A $\setminus$ Q L$'$ = L $\setminus$ $\{l_{del}\}$
Out	Sn'

Deleting a dimension level table l_{del} between existing level tables l_i and a set of dimension level tables M where l_i is classified according to the set of dimension levels M in a snowflake schema Sn:

Semantics	
In	Sn, l_{del}, l_i, M
PCs	M $\subset$ L, $\{l_{del}, l_i\} \in$ L $Q = attrib^{-1}(l_{del})$
SCs	Sn' = <L$'$, B, fact, A$'$, $attrib'$, PKL$'$, FKL$''$, PKF> FKL$'$ = FKL $\setminus$ FKl_i FK$l_i{}'$ = FKl_i $\setminus$ $\{$PK$l_{del}\}$ FK$l_i{}''$ = FK$l_i{}'$ $\cup$ FKl_del FKL$''$ = FKL$'$ $\cup$ FK$l_i{}''$ $\forall$q $\in$ Q: $attrib'$(q) = $\emptyset$ A$'$ = A $\setminus$ Q L$'$ = L $\setminus$ $\{l_{del}\}$
Out	Sn'

Deleting a base dimension level l_{del} and replacing it with a dimension level l_i where l_i is classified according to l_{del} in a snowflake schema Sn:

Semantics	
In	Sn, l_{del}, l_i
PCs	$\{l_{del}, l_i\} \in$ L, $Q = attrib^{-1}(l_{del})$
SCs	Sn' = <L$'$, B, fact, A$'$, $attrib'$, PKL$'$, FKL$'$, PKF$'$> PKF$'$ = PKF $\setminus$ PKl_{del} PKF$''$ = PKF$'$ $\cup$ PKl_i A$'$ = A $\setminus$ Q $\forall$q $\in$ Q: $attrib'$(q) = $\emptyset$ L$'$ = L $\setminus$ $\{l_{del}\}$
Out	Sn'

Notice the constraint of the operation above requiring a dimension level l_i to exist to replace a deleted dimension level l_{del}. The constraint can also be expressed as |FKldel| = 0. The reason for this constraint is due to the existence of the operation "Delete dimension from fact".

Adaption If the top dimension level table is dropped, no updates are necessary. Deleting a dimension level table between two other dimension level tables will require that the lower table (with finer granularity) is updated according to the deleted table's upper dimension level table. Deleting the base dimension level table requires that the fact table is updated such that the fact correspond to the new granularity set by what becomes the new base dimension level table. **Impact:** If the top dimension level table is deleted existing browse and aggregate queries that include this table will fail. Deleting a dimension level table between two tables will make queries that are joining over the deleted tables fail. Deleting a base dimension level table will invalidate browse and aggregate queries that use the deleted table.

3.5 Insert measure attribute

This changes the MD-schema by connecting a new measure to the fact.

Star changes For a star schema, the new attribute a_{new} must be added to the fact table in star schema St. **Adaption:** The existing instances of the fact table are either set a default value or updated from an outside source like an OLTP system. **Impact:** Adding a new measure attribute will, of course, not affect browse queries since browse queries only include a single dimension table. Nor will aggregate queries be affected since the new attribute is not included in any existing queries.

Semantics	
In	St, a_{new}
PCs	$a_{new} \notin$ A
SCs	St' = <D, fact, A$'$, $attrib'$, PKD, PKF> A$'$ = A $\cup$ $\{a_{new}\}$ $attrib'(a_{new})$ = fact
Out	St'

Snowflake changes Adding a new measure attribute into the fact table of a snowflake schema is quite the same semantics as for the star schema and hence we choose not to show the operation semantics.

3.6 Delete measure attribute

Star changes Deleting a measure attribute from the MD-schema will change the star schema St with a measure attribute a_{del} as seen below. **Adaption:** When an attribute is deleted from the fact table, no updating is required. When the attribute being deleted is the last measure attribute the fact table becomes a "factless fact table" (Kimball, 1996). **Impact:** Browse queries only include a single dimension table and are therefore not affected by this operation. Aggregate queries which include the deleted attribute will, however, fail after this operation has been performed.

Semantics	
In	St, a_{del}
PCs	$a_{del} \in$ A, $attrib(a_{del})$ = fact
SCs	St' = <D, fact, A$'$, $attrib'$, PKD, PKF> $attrib'(a_{del})$ = $\emptyset$ A$'$ = A $\setminus$ $\{a_{del}\}$
Out	St'

Snowflake changes Deleting a measure attribute from the fact table of a snowflake schema is the same operation as for star schemas.

3.7 Insert dimension into fact

Star changes The star schema St will create a new dimension table d_{new} with the set of attributes Q as seen below. **Adaption:** In most cases adding a new dimension to a fact will change the granularity of the fact table and all rows in the fact table will have to be deleted and reinserted. **Impact:** If the new dimension changes the granularity of the fact table it can cause existing aggregate queries to fail. More specifically, when the granularity is made finer, one fact tuple is spread across several tuples, which will cause existing aggregate queries using MIN, MAX, AVG and COUNT to fail. Aggregate queries using SUM, however, will continue to return correct results.

Semantics	
In	St, d_{new}, Q
PCs	$\mathrm{d}_{new} \notin$ D, Q ∩ A = ∅, $\{\mathrm{PKd}_{new}\} \in$ Q
SCs	St' = <D′, fact, A′, $attrib'$, PKD′, PKF′> D′ = D ∪ $\{\mathrm{d}_{new}\}$ PKF′ = PKF ∪ $\{\mathrm{PKd}_{new}\}$ A′ = A ∪ Q ∀q ∈ Q: $attrib'$(q) = $\{\mathrm{d}_{new}\}$
Out	St'

Snowflake changes The snowflake schema Sn is extended with a new dimension consisting of only one dimension level table l_{new} and attributes Q, see below. l_{new} is therefore a base dimension level table. Associating the fact table of a snowflake schema with a new base dimension level table has the same consequences for browse and aggregate queries as the star schema.

Semantics	
In	Sn, l_{new}, Q
PCs	$\mathrm{l}_{new} \notin$ L, Q ∩ A = ∅, Q ≠ ∅, $\mathrm{PKl}_{new} \in$ Q
SCs	Sn' = <L′, B′, fact, A′, attrib′, PKL, FKL, PKF′> L′ = L ∪ $\{\mathrm{l}_{new}\}$ PKF′ = PKF ∪ $\{\mathrm{PKl}_{new}\}$ A′ = A ∪ Q ∀q ∈ Q: $attrib'$(q) = l_{new}
Out	Sn'

3.8 Delete dimension from fact

Star changes The star schema does not have corresponding tables for every dimension level in the MD-schema, as in a snowflake schema. Thus, if the base dimension level is deleted from the fact it means that all the other dimension levels that are in the same dimension table in the star schema are also deleted. The star schema St will remove the dimension table d_{del} as seen below. **Adaption:** If deleting the dimension causes a change in granularity, all existing rows in the fact table will have to be recalculated by aggregating over all of the deleted dimension. **Impact:** Existing browse queries that include the deleted dimension table will fail. Aggregate queries, in the same way as for browse queries, will also fail if they include the deleted dimension table. Additionally if deleting the dimension table from the fact table causes a change in granularity existing aggregate queries using MIN, MAX, AVG and COUNT will fail, while SUM queries not referring the deleted level will continue to return correct results.

Semantics	
In	St, d_{del}
PCs	$\mathrm{d}_{del} \in$ D, $Q = attrib^{-1}(d_{del})$, $\{\mathrm{PK}_{del}\} \in$ Q
SCs	St' = <D′, fact, A′, $attrib'$, PKD′, PKF′> PKF′ = PKF \ $\{\mathrm{PKd}_{del}\}$ ∀q ∈ Q: $attrib'$(q) = ∅ A′ = A \ Q D′ = D \ $\{\mathrm{d}_{del}\}$
Out	St'

Snowflake changes When the base dimension level table l_{del} is removed in the snowflake schema Sn, the fact table drops its foreign key reference to l_{del}. Deleting a base dimension level table affects existing browse and aggregate queries in the same way as for the star schema.

Semantics	
In	Sn, l_{del}
PCs	$\mathrm{l}_{new} \in$ L, $\lvert\mathrm{FKl}_{del}\rvert$ = 0, $Q = attrib^{-1}(l_{del}$, $\{\mathrm{PKl}_{del}\} \in$ Q
SCs	Sn' = <L′, B′, fact, A′, $attrib'$, PKL′, FKL, PKF′> PKF′ = PKF \ PKl_{del} ∀q ∈ Q: $attrib'$(q) = ∅ A′ = A \ Q L′ = L \ l_{del}
Out	Sn'

4 COMPARISON

In the following, the results of Section 3 are summarized. For each evolution operation, we indicate the “ease of implementation” and “ease of updating” of the instances of the schemas. If the adaption task can be accomplished with a few SQL statements, it is “simple,” otherwise it is “hard.” We also summarize which evolution operations can result in invalidating browse and aggregate queries. The summarization is seen in the tables below.

Ease of implementation	**Star schema**	**Snowflake schema**
Insert attribute into level	Simple	Simple
Delete attribute from level	Simple	Simple
Insert dimension level	Simple	Hard
Delete dimension level	Simple	Hard
Insert measure attribute	Simple	Simple
Delete measure attribute	Simple	Simple
Insert dimension into fact	Hard	Hard
Delete dimension from fact	Hard	Hard

We see that in *all cases* the star schema is either simpler or just as hard to update as its snowflake counterpart. Especially the operations “Insert dimension level” and “Delete dimension level” are far simpler

Ease of updating	**Star schema**	**Snowflake schema**
Insert attribute into level	Simple	Simple
Delete attribute from level	Simple	Simple
Insert dimension level	Simple	Hard
Delete dimension level	Simple	Hard
Insert measure attribute into fact	Simple	Simple
Delete measure attribute from fact	Simple	Simple
Insert dimension into fact	Hard	Hard
Delete dimension from fact	Hard	Hard

Invalidates browse queries	**Star schema**	**Snowflake schema**
Insert attribute into level	No	No
Delete attribute from level	Yes	Yes
Insert dimension level		
- New top dimension level	No	No
- Between dimension levels	No	Yes
- New base dimension level	No	No
Delete dimension level		
- Delete top dimension level	Yes	Yes
- Between dimension levels	Yes	Yes
- Delete base dimension level	Yes	Yes
Insert measure attribute	No	No
Delete measure attribute	No	No
Insert dimension into fact	No	No
Delete dimension from fact	Yes	Yes

Invalidates MIN, MAX, COUNT and AVG aggregate queries	**Star schema**	**Snowflake schema**
Insert attribute into level	No	No
Delete attribute from level	Yes	Yes
Insert dimension level		
- New top dimension level	No	No
- Between dimension levels	No	Yes
- New base dimension level	Yes	Yes
Delete dimension level		
- Delete top dimension level	Yes	Yes
- Between dimension levels	Yes	Yes
- Delete base dimension level	Yes	Yes
Insert measure attribute	No	No
Delete measure attribute	Yes	Yes
Insert dimension into fact	Yes	Yes
Delete dimension from fact	Yes	Yes

Invalidates SUM aggregate queries	**Star schema**	**Snowflake schema**
Insert attribute into level	No	No
Delete attribute from level	Yes	Yes
Insert dimension level		
- New top dimension level	No	No
- Between dimension levels	No	Yes
- New base dimension level	No	Yes
Delete dimension level		
- Delete top dimension level	Yes	Yes
- Between dimension levels	Yes	Yes
- Delete base dimension level	Yes	Yes
Insert measure attribute	No	No
Delete measure attribute	Yes	Yes
Insert dimension level into fact	No	No
Delete dimension level from fact	Yes	Yes

to implement and update in the star schema due to its denormalized dimensions. When all attributes of a dimension are in the same table there are no primary and foreign key dependencies that have to be updated as in snowflake schemas. If we compare the two schema types' ability to let existing browse and aggregate queries continue to operate, the star schema again proves to be superior to the snowflake schema. Again it is the two operations mentioned above that cause the star schema to prevail. When performing the operation "Insert dimension level" aggregate queries using MIN, MAX, AVG and COUNT will *only* fail on the star schema if the new dimension level is inserted as a new *base* dimension and *also* causing a change in granularity. The snowflake schema, on the other hand, will fail both for new base dimension levels and new dimension levels between other dimension levels because the tables above and below in the dimension hierarchy (also including the fact table) will no longer be possible to join (their join-path is broken). When aggregate queries only use the SUM aggregate function, the star schema is much more robust than the snowflake schema as it will not matter if the granularity changes when inserting a new base dimension level since SUM is split-resistant.

5 CONCLUSION AND FUTURE WORK

Motivated by the popularity of star and snowflake schemas for DW implementation, the lack of detailed information on schema evolution for such schemas, and the problems with schema evolution in DW reporting environments, this paper systematically investigated the evolution properties of star and snowflake schemas. We formally defined star and snowflake schemas, and considered eight evolution operations, covering insertion and deletion of dimensions, levels, dimension attributes, and measure attributes. For each operation, the formal semantics of the changes for star and snowflake schemas were given, and instance adaption and impact on existing browse and aggregate queries was described. Finally, the evolution properties of star and snowflake schemas was compared, looking at the ease of implementation and instance adaption and which changes cause existing browse or aggregate queries to fail. In all cases, the star schema was either superior to the snowflake schema or had the same problems. Thus, the star schema is considerably more robust towards schema changes than the snowflake schema. This challenges the traditional relational assumption that normalized schemas are more flexible. Interesting aspects of future work include building a tool for supporting implementation and instance adaption for schema evolution in relational DWs, as well as a framework/tool for rewriting queries against old version of the schema to the new schema, thus easing the problems of DW evolution.

REFERENCES

M. Blaschka, C. Sapia and G. Höfling. On Schema Evolution in Multidimensional Databases. In *Proc. of DaWaK*, pp. 153–164, 1999.

M. Blaschka. *FIESTA, A Framework for Schema Evolution in Multidimensional Databases*. Ph. D. Thesis, TU Mnchen, 2000.

M. Body, M. Miquel., Y. Bdard., and A. Tchounikine. A Multidimensional and Multiversion Structure for OLAP Applications. In *Proc. of DOLAP*, 2002.

M. Bouzheghoub and Z. Kedad. A Logical Model for Data Warehouse Design and Evolution. In *Proc. of DaWaK*, pp. 178–188, 2000

J. Eder and C. Koncilia. Changes of Dimension Data in Temporal Data Warehouses. In *Proc. of DaWaK*, 2001.

C. A. Hurtado, A. O. Mendelzon, and A. A. Vaisman. Updating OLAP Dimensions. In *Proc. of DOLAP*, 1999.

R. Kimball. *The Data Warehouse Toolkit*. Wiley, 1996.

R. Kimball et al. The Data Warehouse Lifecycle Toolkit. Wiley, 1998.

C. Letz, E. Henn, and G. Vossen. Consistency in Data Warehouse Dimensions. In *Proc. of IDEAS*, 2002.

M. Levene and G. Loizou. Why is the Snowflake Schema a Good Data Warehouse Design?. *Information Systems*, 28(3):225–240, 2003.

S. Lujan-Mora, J. Trujillo, and I-Y. Song. Multidimensional Modeling With UML Package Diagrams. In *Proc. of ER*, pp. 199-213, 2002.

A. Mendelzohn and A. A. Vaisman. Temporal Queries in OLAP. In *Proc. of* VLDB, pp. 242–253, 2000

T. Morzy and R. Wrembel. Modeling a Multiversion Data Warehouse: a Formal Approach. In *Proc. of ICEIS*, 2003.

T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A Foundation for Capturing and Querying Complex Multidimensional Data. *Information Systems* 26(5):383–423, 2001.

T. B. Pedersen and C. S. Jensen. Multidimensional Database Technology. *IEEE Computer*34(12):40–46, 2001.

J. Roddick. Schema Evolution in Database Systems - An Annotated Bibliography. *SIGMOD Record* 21(4):35–40, 1992.

C. Sapia, M. Blaschka, G. Höfling and B. Dinter. Extending the E/R model for the Multidimensional Paradigm. In *Proc. of ER Workshops*, pp. 105–116, 1998.

P. Vassiliadis. Gulliver in the land of data warehousing: practical experiences and observations of a researcher. In *Proc. of DMDW*, pp. 12-1–12-16, 2000.

P. Vassiliadis, M. Bouzheghoub, and C. Quix. Towards Quality-Oriented Data Warehouse Usage and Evolution. *Information Systems* 25(2):89–115, 2000.