Why is the Snowflake Schema a Good Data Warehouse Design?

Mark Levene and George Loizou
School of Computer Science and Information Systems
Birkbeck College, University of London
Malet Street, London WC1E 7HX, U.K.
Email: {mark,george}@dcs.bbk.ac.uk

Abstract

Database design for data warehouses is based on the notion of the snowflake schema and its important special case, the star schema. The snowflake schema represents a dimensional model which is composed of a central fact table and a set of constituent dimension tables which can be further broken up into subdimension tables. We formalise the concept of a snowflake schema in terms of an acyclic database schema whose join tree satisfies certain structural properties. We then define a normal form for snowflake schemas which captures its intuitive meaning with respect to a set of functional and inclusion dependencies. We show that snowflake schemas in this normal form are independent as well as separable when the relation schemas are pairwise incomparable. This implies that relations in the data warehouse can be updated independently of each other as long as referential integrity is maintained. In addition, we show that a data warehouse in snowflake normal form can be queried by joining the relation over the fact table with the relations over its dimension and subdimension tables. We also examine an information-theoretic interpretation of the snowflake schema and show that the redundancy of the primary key of the fact table is zero.

Key words. Data warehouse design, star and snowflake schema, independent and separable database schema, acyclic database schema.

1 Introduction

A data warehouse is an integrated and time-varying database primarily used for the support of management decision making [Inm96, CD97, KRRT98]. A data warehouse often integrates heterogeneous data from multiple and distributed information sources and contains historical and aggregated data. As an example, a sales data warehouse may contain information on the products sold, the time of sale, the place of sale and the sales person. Typically such a data warehouse will be of orders of magnitude larger than an operational database which does not contain specific sales data, but rather contains the company details, including details about product range, outlet locations and personnel.

In terms of data modelling it is beneficial to view a data warehouse in terms of a *dimensional model* which is composed of a central *fact table* and a set of surrounding *dimension tables* each corresponding to one of the components or *dimensions* of the fact table. In the above example the fact table models the actual sales data and each dimension, such as: the

product detail, the time of sale, the outlet in which the product was sold and the sales personnel, is modelled by a separate dimension table. In relational database terms the fact table contains all the necessary foreign key attributes referencing the primary keys of the constituent dimension tables. Conceptually this leads to a star-like data structure, which is called a *star schema*. According to [KRRT98] dimensional modelling actually predates the entity-relationship modelling approach, which is the conventional way in which to design a relational database coupled with normalisation theory [MR92, LL99a]. Star schemas can be refined into *snowflake schemas* providing support for attribute hierarchies by allowing the dimension tables to have subdimension tables. For example, the dimension table storing the outlet in which the product was sold may have a subdimension table containing demographic information of the area of sale. There is debate on the benefits of having such subdimension tables, since it will, in general, slow down query processing, but in some cases it provides a necessary logical separation of data such as in the case of the demographic information subdimension [KRRT98].

As a running example, we exhibit in Figure 1 a snowflake schema for a data warehouse for keeping track of students' attendance in a college. The fact table of this schema is ATTENDANCE, and its dimension tables are DATE, ROOM, LECTURER, STUDENT and COURSE; HOLIDAY is a subdimension of DATE, and DEPARTMENT is a subdimension of LECTURER, STUDENT and COURSE. For clarity of the figure we have omitted the labels on edges, which for each edge between two tables is the intersection of their attributes. The snowflake schema is structured in such a way that the labels of edges represent a foreign to primary key relationship between the parent table and its child. The meaning of the tables is self-evident from their attributes, observing that DEPARTMENT represents the department in which the course is given; we note that this is not necessarily the department to which the student belongs. Thus lecturer_no, student_no and course_no are only unique in the context of a particular department.

Although snowflake schemas enjoy a relatively simple structure, they are widely used in practice [KRRT98] and recommended in all the data warehouse design methodologies we are aware of to date. The reason for their success is that they are: intuitive and easy to understand, amenable to query optimisation since arbitrary n-way joins with the fact table can be evaluated by a single pass through the fact table, can accommodate for aggregate data, and are easily extensible by adding new attributes to the fact table or to one or more of the dimension tables and new dimension tables to the schema without interfering with existing database programs.

There has been some recent research on formalising data warehouses using a graph-theoretic approach to model snowflake schemas. In [GMR98] it is shown how a snowflake schema can be derived from an entity-relationship diagram and then modified accordingly to remove uninteresting attributes. In [LAW98] several normal forms were defined for data warehouses within a general multidimensional model where functional dependencies induce attribute hierarchies. More recently, a methodology for deriving a snowflake schema from an operational database schema has been presented in [HLV00]. Finally, it is worth referring to [KM00] where the concept of a *data webhouse* is introduced. A data webhouse is essentially a data warehouse intended to capture clickstream log data for ecommerce decision making.

Despite the wide use of snowflake schemas, to our knowledge, no theoretical underpinning has been given to date which may point to its tangible benefits. Moreover, traditional

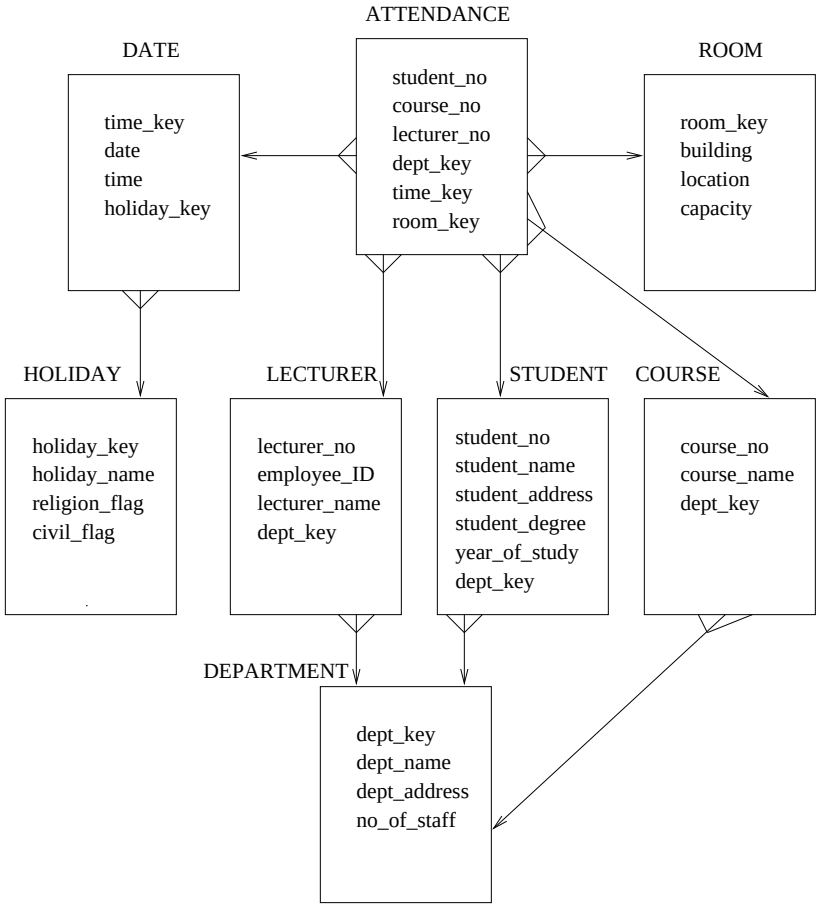

Figure 1: A snowflake schema for a student attendance data warehouse

relational database design theory is a rich area with many theoretical and practical results [MR92, LL99a], which may benefit data warehouse design. To remedy this situation we define a normal form for data warehouses, called *Snowflake Schema Normal Form* (SSNF), which captures the intuition behind the snowflake schema by building on relevant concepts from relational database design theory. (See also *Extended Snowflake Schema Normal Form* (ESSNF).)

The said definition of SSNF imposes a natural syntactic restriction on the database schema $\mathbf{R}$ that it be acyclic, thus inducing a join tree structure on $\mathbf{R}$ [BFMY83, Fag83]. (The precise subclass of join trees we are interested in is given in Definition 3.1; we note that a star schema is a special case of a snowflake schema having a join tree whose height is one.) In addition, we require that the intersection of any two relation schemas in $\mathbf{R}$, which labels an edge in the join tree, be a foreign key of one relation schema referencing the primary key of the other. Further conditions on a database schema in SSNF are that the hierarchical structure of the join tree induces its integrity constraints in terms of *Functional Dependencies* (FDs) and a restricted class of *Inclusion Dependencies* (INDs). Finally, the root of the join tree, whose relation schema corresponds to the fact table, must be in BCNF. (The precise formulation of SSNF is given in Definition 3.3.)

Our definition of SSNF captures the intuition behind data warehouse design and our

results provide semantic justification for the definition of SSNF. To motivate these results we briefly describe the concepts of *independent* [Sag83, AC91, Sag91] and *separable* [CM87] database schemas $\mathbf{R}$ with respect to a set of integrity constraints Σ consisting of FDs and INDs over $\mathbf{R}$. Given a database d over $\mathbf{R}$, independence implies that maintaining local consistency of the relations in d, i.e. ensuring that d satisfies the INDs in Σ and that the relations in d satisfy the FDs in Σ, is sufficient to ensure global consistency, i.e. the existence of a *representative instance* over the set of all attributes $\mathcal{U}$ in $\mathbf{R}$ that satisfies the set F of FDs (see Definition 2.14). The reason the representative instance is an important concept is that it provides us with a means of testing the satisfaction of interrelational constraints, which may hold in the join of several relations in d. Separability is an extension of independence, which, in addition, implies that updates to relations in d are independent in the sense that we cannot deduce additional tuples in any relation in d via a join of several relations in the database d. We show that when $\mathbf{R}$ is in SSNF with respect to Σ then it is independent, and if, in addition, the relation schemas in $\mathbf{R}$ are incomparable then it is also separable. Thus for database schemas in SSNF integrity maintenance in the presence of updates is easily enforced. We also show that tuples in the representative instance over $\mathcal{U}$, i.e. tuples over the fact table extended with the relevant information in the constituent dimension tables, can be computed via the snowflake join (cf. star join [OG95]). This implies that the results of queries over a data warehouse which is in SSNF maintain their consistency, i.e. satisfy the induced set of FDs over their schema.

We also examine an information-theoretic interpretation of the snowflake schema following the work of [Mal86, CP87, Lee87, Mal88], which allows us to accommodate probabilistic information in the data warehouse. This is especially important, since decision making often involves probabilistic reasoning [Lin85]. We show that the redundancy in the snowflake join of the primary key of the fact table is zero, i.e. it is minimal. Against this measure of redundancy, which is the standard evaluation criterion for assessing relational database schemas, SSNF is optimal.

In summary the paper establishes a theoretical underpinning for data warehouse design by building upon an acyclic structure and utilising the notions of independence and separability. Since, to our knowledge, there is no formal definition of the concept of a snowflake schema to be found in the data warehousing literature it is not possible to give a formal proof of the equivalence of SSNF and the intuitive concept of a snowflake schema. The best we can offer is a "concept of proof" in the sense that all examples of snowflake schemas we have encountered in the literature satisfy SSNF.

The layout of the rest of the paper is as follows. In Section 2 we present the background in relational database theory necessary for the rest of the paper. In Section 3 we formalise the notion of a snowflake schema, define SSNF and present our results concerning the beneficial properties of snowflake schemas. In Section 4 we consider an extension of the formalisation presented in Section 3 in order to include a broader class of snowflake schemas. In Section 5 we analyse the snowflake schema from an information-theoretic point of view and show that its entropy has a particularly simple form. Finally, in Section 6 we give our concluding remarks.

2 Relations and Data Dependencies

We now present the background material necessary for the development of the ideas in the paper; see [LL99a] for additional details on relational database theory.

We use the notation |S| to denote the cardinality of a set S. If S is a subset of T we write S $\subseteq$ T and if S is a proper subset of T we write S $\subset$ T. Furthermore, S and T are *incomparable* if S $\not\subseteq$ T and T $\not\subseteq$ S. We often denote the singleton {A} simply by A, and the union of two sets S, T, i.e. S $\cup$ T, simply by ST.

Definition 2.1 (Database schema and database) Let $\mathcal{U}$ be a finite set of attributes. A *relation schema* R is a finite sequence of distinct attributes from $\mathcal{U}$.

A *database schema* over $\mathcal{U}$ is a finite set $\mathbf{R} = \{R_0, R_1, \ldots, R_n\}$, such that each $R_i \in \mathbf{R}$ is a relation schema and $\bigcup_i R_i = \mathcal{U}$.

We assume a countably infinite domain of values, $\mathcal{D}$, partially ordered; without loss of generality we assume that $\mathcal{D}$ is an antichain, i.e. $\forall c_1, c_2 \in \mathcal{D}, c_1 \leq c_2$ if and only if $c_1 = c_2$. The domain of R, denoted as DOM(R), is defined as the Cartesian product $\mathcal{D} \times \ldots \times \mathcal{D}$ (|R| times). An R-tuple (or simply a tuple whenever R is understood from context) is a member of DOM(R).

A *relation* r over R is a finite (possibly empty) set of R-tuples. A *database* d over $\mathbf{R}$ is a family of $n+1$ relations $\{r_0, r_1, \ldots, r_n\}$ such that each $r_i \in d$ is over $R_i \in \mathbf{R}$.

From now on we let $\mathbf{R} = \{R_0, R_1, \ldots, R_n\}$ be a database schema over $\mathcal{U}$ and $d = \{r_0, r_1, \ldots, r_n\}$ be a database over $\mathbf{R}$. Furthermore, we let $r \in d$ be a relation over a relation schema R $\in \mathbf{R}$. As usual uppercase letters (which may be subscripted) from the end of the alphabet such as X, Y, Z will be used to denote sets of attributes, while those from the beginning of the alphabet such as A, B, C will be used to denote single attributes or singleton sets of attributes.

Definition 2.2 (Projection) The *projection* of an R-tuple t onto a set of attributes Y $\subseteq$ R, denoted by t[Y], is the restriction of t to Y. The projection of a relation r over R onto Y $\subseteq$ R, denoted as $\pi_{\mathrm{Y}}(r)$, is defined by $\pi_{\mathrm{Y}}(r) = \{t[\mathrm{Y}] \mid t \in r\}$. The projection of a relation s over $\mathcal{U}$ onto $\mathbf{R}$, denoted as $\pi_{\mathbf{R}}(s)$, is the database $\{\pi_{R_0}(s), \pi_{R_1}(s), \ldots, \pi_{R_n}(s)\}$.

Definition 2.3 (Natural join) The *natural join* (or simply the join), $\bowtie$, of two relations r_1 over R_1 and r_2 over R_2 is a relation r over R $= R_1 \cup R_2$ defined by

$$r_1 \bowtie r_2 = \{t \mid \exists t_1 \in r_1 \text{ and } \exists t_2 \in r_2 \text{ such that } t[R_1] = t_1 \text{ and } t[R_2] = t_2\}.$$

Definition 2.4 (Join dependency) A *Join Dependency* (or simply a JD) for a database schema $\mathbf{R}$ is a statement of the form $\bowtie[\mathbf{R}]$. A JD $\bowtie[\mathbf{R}]$ is said to be *trivial* if one of its components is $\mathcal{U}$.

A JD $\bowtie[\mathbf{R}]$ is satisfied in a relation r over $\mathcal{U}$, (alternatively, $\mathbf{R}$ is a *lossless join decomposition* of $\mathcal{U}$), denoted by $r \models \bowtie[\mathbf{R}]$, if

$$r = \pi_{R_0}(r) \bowtie \pi_{R_1}(r) \bowtie \ldots \bowtie \pi_{R_n}(r).$$

Definition 2.5 (Functional dependency) A *Functional Dependency* over $\mathcal{U}$ (or simply an FD) is a statement of the form $\mathrm{X} \to \mathrm{Y}$, where $\mathrm{X}, \mathrm{Y} \subseteq \mathcal{U}$ are sets of attributes. An FD of the form $\mathrm{X} \to \mathrm{Y}$ is said to be *trivial* if $\mathrm{Y} \subseteq \mathrm{X}$.

An FD $\mathrm{X} \to \mathrm{Y}$ is satisfied in a relation $r \in d$ over R, denoted by $r \models \mathrm{X} \to \mathrm{Y}$, whenever $\mathrm{XY} \subseteq \mathrm{R}$ and $\forall t_1, t_2 \in r$, if $t_1[\mathrm{X}] = t_2[\mathrm{X}]$ then $t_1[\mathrm{Y}] = t_2[\mathrm{Y}]$. A relation $r \in d$ satisfies a set F of FDs over $\mathcal{U}$, if for all $\alpha \in \mathrm{F}$, $r \models \alpha$.

From now on we let F be a set of FDs over $\mathcal{U}$.

Definition 2.6 (Closure) The set of FDs that are logically implied by F is denoted by F^+. A set of FDs G over $\mathcal{U}$ is a *cover* of F if $G^+ = F^+$.

The *closure* of a set of attributes $\mathrm{X} \subseteq \mathcal{U}$ with respect to F, denoted by $C_{\mathrm{F}}(\mathrm{X})$, is the set of attributes $\{\mathrm{A} \mid \mathrm{X} \to \mathrm{A} \in F^+\}$.

Definition 2.7 (Embedded FDs and dependency preservation) The *projection* of F onto a relation schema R, denoted by $F^+|\mathrm{R}$, is the set of FDs $\mathrm{X} \to \mathrm{Y} \in F^+$ such that $\mathrm{XY} \subseteq \mathrm{R}$. An FD $\mathrm{X} \to \mathrm{Y}$ is *embedded* in R if $\mathrm{XY} \subseteq \mathrm{R}$.

A database schema $\mathbf{R}$ is *dependency preserving* with respect to F (alternatively $\mathbf{R}$ preserves F) if there is a cover G of F such that each FD $\mathrm{X} \to \mathrm{Y} \in \mathrm{G}$ is embedded in some $R_i \in \mathbf{R}$.

From now on we assume that $\mathbf{R}$ is a dependency preserving database schema with respect to F and let F_i be a cover of $F^+|R_i$ which is embedded in R_i; we assume without loss of generality that F_i is a *canonical* cover, i.e. for all $\mathrm{X} \to \mathrm{Y} \in F_i$, $\mathrm{Y} = \{\mathrm{A}\}$ and $\mathrm{A} \notin \mathrm{X}$. (See [LL99a] for more details on the formal properties of FDs.)

Definition 2.8 (Inclusion dependency) An *Inclusion Dependency* (or simply an IND) over $\mathbf{R}$ is a statement of the form $R_i[\mathrm{X}] \subseteq R_j[\mathrm{Y}]$, where $R_i, R_j \in \mathbf{R}$ and $\mathrm{X} \subseteq R_i$, $\mathrm{Y} \subseteq R_j$ are sequences of distinct attributes such that $|\mathrm{X}| = |\mathrm{Y}|$. An IND is said to be *trivial* if it is of the form $\mathrm{R}[\mathrm{X}] \subseteq \mathrm{R}[\mathrm{X}]$. An IND is said to be *typed* if it is of the form $\mathrm{R}[\mathrm{X}] \subseteq \mathrm{S}[\mathrm{X}]$.

An IND $R_i[\mathrm{X}] \subseteq R_j[\mathrm{Y}]$ over $\mathbf{R}$ is satisfied in d, denoted by $d \models R_i[\mathrm{X}] \subseteq R_j[\mathrm{Y}]$, whenever $\pi_{\mathrm{X}}(r_i) \subseteq \pi_{\mathrm{Y}}(r_j)$, where $r_i, r_j \in d$ are the relations over R_i and R_j, respectively. A database d over $\mathbf{R}$ satisfies a set I of INDs over $\mathbf{R}$, if for all $\alpha \in \mathrm{I}$, $d \models \alpha$.

From now on we let I be a set of INDs over $\mathbf{R}$ and let $\Sigma = \mathrm{F} \cup \mathrm{I}$. A database d over $\mathbf{R}$ satisfies Σ, denoted by $d \models \Sigma$, whenever each $r_i \in d$ satisfies all the FDs in F_i and d satisfies all the INDs in I. Also, we denote by Σ^+ the set of FDs and INDs that are logically implied by Σ.

Definition 2.9 (Graph representation of INDs) The graph representation of a set of INDs I over $\mathbf{R}$ is a directed graph $G_I = (\mathrm{N}, \mathrm{E})$, which is constructed as follows. Each relation schema R in $\mathbf{R}$ has a separate node in N labelled by R, i.e. we do not distinguish between nodes and their labels. There is an edge $(\mathrm{R}, \mathrm{S}) \in \mathrm{E}$ if and only if there is a nontrivial IND $\mathrm{R}[\mathrm{X}] \subseteq \mathrm{S}[\mathrm{Y}] \in \mathrm{I}$.

Definition 2.10 (Dag-like INDs) A set I of INDs over $\mathbf{R}$ is *dag-like* if

1) I is a set of typed INDs, and

2) G_I is a rooted directed acyclic graph.

Definition 2.11 (Keys and key-based INDs) A set of attributes $\mathrm{X} \subseteq R_i$ is a *superkey* for R_i with respect to F_i (or simply X is a superkey for R_i if F_i is understood from context) if $C_{F_i}(\mathrm{X}) = R_i$ holds; X is a *key* for R_i with respect to F_i if it is a superkey for R_i with respect to F_i and for no proper subset $\mathrm{Y} \subset \mathrm{X}$ is Y a superkey for R_i with respect to F_i. The *primary key* of R_i is one of the keys for R_i with respect to F_i, which is designated by the database designer.

A database schema $\mathbf{R}$ is in *Boyce-Codd Normal Form* (or simply BCNF) with respect to F if for all $R_i \in \mathbf{R}$, for all nontrivial FDs $\mathrm{X} \rightarrow \mathrm{Y} \in F_i$, X is a superkey for R_i with respect to F_i.

An IND $R_i[\mathrm{X}] \subseteq R_j[\mathrm{Y}]$ is *key-based* if Y is a key for R_j with respect to F_j. When Y is the primary key of R_j then X is called a *foreign key* of R_i.

Definition 2.12 (No interaction between FDs and INDs) A set $\Sigma = \mathrm{F} \cup \mathrm{I}$ of FDs and INDs do *not interact* when $\mathrm{X} \rightarrow \mathrm{Y} \in \Sigma^+$ if and only if $\mathrm{X} \rightarrow \mathrm{Y} \in F^+$ and $\mathrm{R}[\mathrm{X}] \subseteq \mathrm{S}[\mathrm{Y}] \in \Sigma^+$ if and only if $\mathrm{R}[\mathrm{X}] \subseteq \mathrm{S}[\mathrm{Y}] \in I^+$.

In Section 3 we will show that for the subclass of dag-like INDs we are interested in, F and I do not interact.

The chase procedure provides us with a very useful algorithm which forces a database to satisfy a set of integrity constraints.

Definition 2.13 (Chase procedure for FDs) We assume a countably infinite domain of *marked nulls*, $\mathcal{V}$, which is disjoint from $\mathcal{D}$; without loss of generality we assume that $\mathcal{V}$ is linearly ordered and that $\forall c \in \mathcal{D}, \forall v \in \mathcal{V}, c < v$.

Given a tuple $t \in r_i$, where $r_i \in d$ is over $R_i \in \mathbf{R}$, $\mathrm{pad}(t)$ is a tuple over $\mathcal{U}$ such that $\mathrm{pad}(t)[R_i] = t$, and for all $\mathrm{A} \in \mathcal{U} - R_i$, $\mathrm{pad}(t)[\mathrm{A}] \in \mathcal{V}$, i.e. it is a marked null. Given a relation $r_i \in d$ over $R_i \in \mathbf{R}$ we define $\mathrm{pad}(r_i) = \bigcup_{t \in r_i} \mathrm{pad}(t)$, and $\mathrm{pad}(d) = \bigcup_{r_i \in d} \mathrm{pad}(r_i)$, such that for all $t_1, t_2 \in \mathrm{pad}(d)$, for all A, $\mathrm{B} \in \mathcal{U}$, if $t_1[\mathrm{A}]$, $t_2[\mathrm{B}] \in \mathcal{V}$, and either $t_1 \neq t_2$ or $\mathrm{A} \neq \mathrm{B}$, then $t_1[\mathrm{A}] \neq t_2[\mathrm{B}]$, i.e. all marked nulls in distinct positions of $\mathrm{pad}(d)$ are distinct.

The chase of d with respect to a set F of FDs over $\mathcal{U}$, denoted by CHASE(d, F), is the result of applying the FD rule, as defined below, to the *current state*, say $\hat{r}$, of CHASE(d, F) until it cannot be further applied to $\hat{r}$ or until a *contradiction* is detected. The state of CHASE(d, F) prior to the first application of the FD rule is $\mathrm{pad}(d)$.

FD rule: If $\mathrm{X} \rightarrow \mathrm{Y} \in \mathrm{F}$ and $\exists t_1, t_2 \in \hat{r}$ such that $t_1[\mathrm{X}] = t_2[\mathrm{X}]$ but $t_1[\mathrm{Y}] \neq t_2[\mathrm{Y}]$, then

1) if for some $\mathrm{A} \in \mathrm{Y}$ such that $t_1[\mathrm{A}] \neq t_2[\mathrm{A}]$ and $t_1[\mathrm{A}], t_2[\mathrm{A}] \in \mathcal{D}$, then a contradiction is detected, otherwise

2) for all $\mathrm{A} \in \mathrm{Y}$, change all the occurrences in $\hat{r}$ of the larger of the values of $t_1[\mathrm{A}]$ and $t_2[\mathrm{A}]$ to the smaller of the values of $t_1[\mathrm{A}]$ and $t_2[\mathrm{A}]$.

Often we refer to an application of the FD rule during the computation of the chase as a *chase step*. When the FD rule cannot be further applied to the current state, $\hat{r}$, of CHASE(d, F) or a *contradiction* is detected, then we remove *subsumed* tuples from the result to obtain CHASE(d, F). A tuple t_2 is subsumed by a tuple t_1, if $t_1 \neq t_2$ and for all A $\in \mathcal{U}$, t_1[A] $\leq$ t_2[A].

We note that all operations that we have defined on relations, including dependency satisfaction, equally apply to relations which have marked nulls in them. We further note that, on treating the marked nulls in CHASE(d, F) as distinct domain values, CHASE(d, F) $\models$ F if and only if no contradictions are detected during the computation of the chase [Hon82].

Definition 2.14 (Representative instance and consistency) The *representative instance* [Sag83] of d with respect to F, denoted as $\mathrm{RI}_{\mathrm{F}}(d)$ (or simply RI(d) whenever F is understood from context), is constructed as follows. First we invoke the chase procedure on pad(d) with respect to F. If CHASE(d, F) $\models$ F, then we set RI(d) to CHASE(d, F), otherwise we set RI(d) to pad(d). In either case we replace all the marked nulls in the resulting relation with the minimum value of all these marked nulls.

If there are no contradictions in CHASE(d, F) then we say that RI(d) satisfies F, denoted by RI(d) $\models$ F. A database d is said to be *consistent* with respect to F if RI(d) $\models$ F, otherwise it is said to be *inconsistent* with respect to F.

Definition 2.15 (Independent and separable database schemas) A database schema $\mathbf{R}$ is said to be *independent* [Sag83, AC91, Sag91] with respect to Σ if for all databases d over $\mathbf{R}$, $d \models \Sigma$ implies that RI(d) $\models$ F, i.e. d satisfies Σ implies that d is consistent with respect to F.

For a relation s over $\mathcal{U}$, which may contain marked nulls, we define $\pi^{\downarrow}_{\mathbf{R}}(s)$ to be the database obtained from $\pi_{\mathbf{R}}(s)$ after removing from each resulting relation over R_i all tuples that have at least one marked null over any attribute in R_i.

A database schema $\mathbf{R}$ is said to be *separable* [CM87] with respect to Σ if $\mathbf{R}$ is independent with respect to Σ and for all d over $\mathbf{R}$ such that $d \models \Sigma$, $d = \pi^{\downarrow}_{\mathbf{R}}\mathrm{RI}(d)$.

We next define the notion of an acyclic database schema [Fag83] by using the concept of a join tree [BFMY83]. (The original definition of acyclicity does not require that the intersection of R and S be nonempty.)

Definition 2.16 (Join tree and acyclic database schema) A *join tree* for $\mathbf{R}$ is a tree whose node set is $\mathbf{R}$ such that

1) each of its edges (R, S) is labelled by the nonempty set of attributes R $\cap$ S; and

2) for every distinct pair R, S $\in \mathbf{R}$ and for every attribute A $\in$ R $\cap$ S, each edge along the unique path between R and S includes A in its label.

A database schema $\mathbf{R}$ is *acyclic* if it has a join tree, which we denote by JT($\mathbf{R}$).

Definition 2.17 (Rooted tree) When we distinguish a node in a tree, T, as its *root*, we can view T as a directed graph with unique directed paths from the root of T to all its leaves [BH90].

From now on we will assume that trees are rooted and consider them as special cases of directed acyclic graphs.

We define the *height* of a node n in a tree T recursively as follows: if n is the root node of T then its height is zero, otherwise the height of n is one plus the height of its parent node. The height of T is defined to be the maximal height among all the leaf nodes of T.

A *topological sort* of a directed acyclic graph G is a process of assigning a linear ordering to the nodes of G so that if there is an edge in G from node n_i to node n_j, then n_i precedes n_j in the linear ordering.

We say that a node n_j is a *descendant* of a node n_i in G if there is a directed path from n_i to n_j; we take n_i to be a descendant of itself. (We take *parent* to be irreflexive.)

3 Snowflake and Star Schemas

We formalise the notion of a snowflake schema and its important special case the star schema as an acyclic database schema having a join tree which satisfies two structural properties. We then define a normal form for snowflake schemas with respect to a set of FDs and INDs and show that it has several fundamental desirable properties. In particular, our results imply that the relations over the fact and dimension tables can be updated independently of each other as long as referential integrity is maintained. Moreover, we show that when in SSNF the data warehouse can be queried via the snowflake join which is the join of the relation over the fact table with the relations over its dimension and subdimension tables.

We next present the notion of a snowflake schema as a special kind of an acyclic database schema. The definition captures the strucutral properties of snowflake schemas regardless of the specific integrity constraints that need to be maintained; the data dependencies of snowflake schemas are introduced in Definition 3.3.

Definition 3.1 (Snowflake and star schema) A database schema $\mathbf{R}$ is a *snowflake schema* if it is acyclic and its join tree has a designated root, which we take to be R_0, satisfying the following structural conditions:

1) no two distinct edges have identical labels; and

2) if there is a label X_j in JT($\mathbf{R}$) such that $\{X_{i1}, X_{i2}, \ldots, X_{im}\}$ is the set of all labels in JT($\mathbf{R}$), satisfying $X_j \subset X_{ik}$, for $k = 1, 2, \ldots, m$, and for no X_{ik} in this set does there exist a label X' in JT($\mathbf{R}$) satisfying $X_j \subset X' \subset X_{ik}$, then for some $k \in \{1, 2, \ldots, m\}$, R_{ik} is the parent node of R_j in JT($\mathbf{R}$), where $X_j = R_{ik} \cap R_j$ and $X_{ik} = R_i \cap R_{ik}$ is the label of the edge (R_i, R_{ik}), R_i being the parent of R_{ik} in JT($\mathbf{R}$).

If $\mathbf{R}$ is a snowflake schema and the height of JT($\mathbf{R}$) is one, then $\mathbf{R}$ is called a *star schema*.

When $\mathbf{R}$ is a snowflake schema, R_0 is called the *fact table* while the R_i's of height one are called the *dimension tables* and the remaining relation schemas of height greater than one are called the *subdimension tables*.

As will be clarified below there is a semantic correspondence between our notion of a snowflake schema and the data dependencies that emanate from it.

We observe that condition (1) above is a restriction on the class of acyclic database schemas while condition (2), which does not restrict the class of acyclic database schemas, restricts the class of join trees. The motivation for condition (1) stems from the fact that labels of the form X_i represent primary keys and thus there is no justification for having two separate relation schemas having identical primary keys, since they can be coalesced into a single relation schema. Regarding condition (2) it enforces a hierarchical dependence of the primary keys of relation schemas in terms of their set inclusion relationship, thereby modelling many-to-one relationships.

We exhibit the significance of condition (2) with a simple example. Let $\mathbf{R} = \{R_0, R_1, R_2\}$, with R_0 = (A,B,C), R_1 = (A,B) and R_2 = (A). The join tree having R_0 as the root with two children R_1 and R_2 violates condition (2), while the join tree having R_0 as the root with a single child R_1, and R_1 having a single child R_2, satisfies condition (2).

Referring back to the running example of Figure 1, we observe that its underlying database schema is acyclic and therefore it has a join tree. Suppose we remove any two of the incoming edges to DEPARTMENT; for concreteness assume we remove the edges from LECTURER to DEPARTMENT and from STUDENT to DEPARTMENT. Then it can be seen that conditions (1) and (2) are satisfied for this schema and thus it is a snowflake schema. Regarding condition (1) the labels on the two edges from ATTENDANCE to LECTURER and STUDENT represent the primary keys of the two latter dimension tables, and regarding condition (2) {dept_key} is the largest proper subset of {course_no, dept_key} that is also the label of the edge from COURSE to DEPARTMENT.

The next definition of the join graph of a snowflake schema ensures that the hierarchical dependence induced by condition (2) is maintained between all its dimensions and subdimensions. It also allows us to infer a set of typed INDs for the snowflake schema; see Definition 3.3.

Definition 3.2 (The join graph of a snowflake schema) We convert the join tree, JT($\mathbf{R}$), of a snowflake schema $\mathbf{R}$, into a directed acyclic graph, called the *join graph* of $\mathbf{R}$ and denoted by JG($\mathbf{R}$), as follows. For every relation schema R_i, whose parent is R_i' and such that $X_i = R_i' \cap R_i$, add an edge from R_i to R_j whose parent is R_j' with $X_j = R_j' \cap R_j$, if $X_j \subset X_i$ and there is no other label X' in JT($\mathbf{R}$) satisfying $X_j \subset X' \subset X_i$. The label of the new edge (R_i, R_j) is X_j.

We note that in JG($\mathbf{R}$) if a node has more than one parent, then the labels of its incoming edges are identical; this is a byproduct of $\mathbf{R}$ being acyclic. We further note that JG($\mathbf{R}$) is acyclic, since the new edges are oriented away from the root node.

Referring back to Figure 1, consider the join tree resulting from removing the edges from LECTURER and STUDENT to DEPARTMENT. The resulting join tree satisfies Definition 3.1. On using Definition 3.2 we convert the said join tree to the corresponding join graph by adding the edges from LECTURER and STUDENT to DEPARTMENT.

Let $\mathbf{R}$ be a snowflake schema. *From now on* we assume that whenever R_i and R_j are distinct relation schemas in $\mathbf{R}$, then $i < j$ if and only if R_i precedes R_j in a topological sort of JG($\mathbf{R}$). Moreover, we let

1) $X_i = R_j \cap R_i$, for $i = 1, \ldots, n$, where R_j is the parent node of R_i in JT($\mathbf{R}$), be the label of the edge from R_j to R_i;

2) $X_0 = \bigcup_{i=1}^{k} X_i$, where $\{R_1, R_2, \ldots, R_k\}$ is the set of children of R_0 in JT($\mathbf{R}$); and

3) $Y_i = R_i - X_i$, for $i = 0, 1, \ldots, n$.

We observe that for all $i, j \in \{0, 1, \ldots, n\}$, with $i \neq j$, $Y_i \cap Y_j = \emptyset$.

We next define a normal form for snowflake schemas which imposes further restrictions on the labels of edges in the join tree of $\mathbf{R}$ taking into account the functional and inclusion dependencies. The main motivation behind the definition is that it encapsulates the intuitions behind the snowflake schema, which have been advocated by database practitioners, by capturing the hierarchical dependence between dimensions and subdimensions via the join tree and data dependency information via primary key to foreign key relationships.

Definition 3.3 (Snowflake schema normal form) A database schema $\mathbf{R}$ is in *Snowflake Schema Normal Form* (SSNF) with respect to a set $\Sigma = \mathrm{F} \cup \mathrm{I}$ of FDs and INDs over $\mathbf{R}$ (or simply in SSNF if Σ is understood from context), if

1) $\mathbf{R}$ is a snowflake schema;

2) R_0 is in BCNF with respect to F_0;

3) for $i = 0, 1, \ldots, n$, X_i is the primary key of R_i with respect to F_i;

4) for $i = 0, 1, \ldots, n$, $C_{\mathrm{F}}(R_i) = \bigcup_{j \geq i} R_j$ such that R_j is a descendant of R_i in JG($\mathbf{R}$); and

5) we have $\mathrm{I} = \{R_i[X_j] \subseteq R_j[X_j] \mid i, j \in \{0, 1, \ldots, n\}$ and R_i is a parent of R_j in JG($\mathbf{R}$)$\}$.

Condition (1) is a structural restriction that was motivated after Definition 3.1. Condition (2) minimises the redundancy of the relation over the fact table. Condition (3) is the standard requirement of a snowflake schema [KRRT98]. It implies that $\mathbf{R}$ is a *lossless join decomposition* of $\mathcal{U}$ [LL99a]. Condition (4) maximises the efficiency of queries involving relations over dimension/subdimension tables, since it specifies that the sequence of joins that is needed to query information emanating from any such table involves only itself and its subdimension tables. Condition (5) guarantees referential integrity, noting that I is dag-like by (1) and key-based by (3). We observe that due to (1) and (3) it is always true that $\bigcup_{j \geq i} R_j \subseteq C_{\mathrm{F}}(R_i)$.

We believe that SSNF captures most situations that arise in practice, and we have not found a natural example of a snowflake schema which is not in SSNF. Although there may be good reason to further restrict dimension tables to satisfy higher normal forms than first normal form, this is not necessary to obtain our main results.

The following lemma is a corollary of the main result in [LL01] due to the special structure of snowflake schemas.

Lemma 3.1 If Σ is in SSNF then F and I do not interact. $\square$

We note that as a result of Lemma 3.1 we obtain the desirable property that the implication problem for FDs or INDs in Σ is polynomial-time decidable [BB79, CV83]. (In general, when there is interaction between the FDs and INDs, the implication problem for FDs and typed INDs, with acyclic G_I, is NP-hard [CK86].)

The next lemma shows that in the special case when only primary and foreign keys are specified then $\mathbf{R}$ is in SSNF; the lemma does *not* cover any situation where F has additional FDs. This special case of SSNF is important, since it is directly induced by the structure of a snowflake schema and reflects practitioners' intuition.

Lemma 3.2 Let $\mathbf{R}$ be a snowflake schema and $\mathrm{F} = \{X_0 \rightarrow Y_0, X_1 \rightarrow Y_1, \ldots, X_n \rightarrow Y_n\}$ be a set of FDs over $\mathcal{U}$, where for $i = 1, 2, \ldots, n$, $F^+|X_i$ contains only trivial FDs. Also, let $\mathrm{I} = \{R_i[X_j] \subseteq R_j[X_j] \mid i, j \in \{0, 1, \ldots, n\}$ and R_i is a parent of R_j in $\mathrm{JG}(\mathbf{R})\}$ be a set of INDs over $\mathbf{R}$. Then $\mathbf{R}$ is in SSNF with respect to Σ, where $\Sigma = \mathrm{F} \cup \mathrm{I}$.

Proof. Conditions (1) and (5) of SSNF are evident from the statement of the lemma, so it remains to show that conditions (2), (3) and (4) hold.

To prove (2) it is sufficient to show that $\{X_0 \rightarrow Y_0\}$ is a cover of F_0. By Definition 3.1 and the notation introduced after Definition 3.2 we have that for all R_i distinct from R_0 we have $Y_i \cap R_0 = \emptyset$. Moreover, $Y_0 \cap X_i = \emptyset$, so the result follows. (In fact, with the assumptions of the lemma it is not hard to show the tighter result that $\mathbf{R}$ is in BCNF with respect to F.)

Condition (3) follows from the assumption that, for $i = 1, 2, \ldots, n$, $F^+|X_i$ contains only trivial FDs and the fact that R_0 is in BCNF with respect to F_0.

It remains to establish (4). By the observation after Definition 3.3 we have $\bigcup_{j \geq i} R_j \subseteq C_{\mathrm{F}}(R_i)$, so we need to show that $C_{\mathrm{F}}(R_i) \subseteq \bigcup_{j \geq i} R_j$. Suppose that this is not the case. Then, for some relation schema R_j which is not a descendant of R_i we have a nontrivial FD $X_j \rightarrow \mathrm{A} \in F^+$, where $X_j \subseteq C_{\mathrm{F}}(R_i)$ and $\mathrm{A} \notin R_i$. By the structure of F and Definition 2.16 it follows that $X_j \subseteq X_i$. By part (1) of Definition 3.1 $X_j \neq X_i$ implying that $X_j \subset X_i$, and by Definitions 3.1 and 3.2 R_j must be a descendant of R_i. This concludes the result. □

Referring back to Figure 1 let $\mathbf{R}$ be the snowflake schema comprising the tables in the figure and let F and I be the set of FDs and INDs over $\mathbf{R}$, which are induced by the structure of its join tree. That is, F contains the FDs:

time_key → {date, time, holiday_key},
holiday_key →{holiday_name, religion_flag, civil_flag},
{lecturer_no, dept_key} → {employee_ID, lecturer_name},
{student_no, dept_key} → {student_name, student_address, student_degree, year_of_study},
{course_no, dept_key} → course_name,
room_key → {building, location, capacity} and
dept_key → {dept_name, dept_address, no_of_staff}

and I contains the INDs:

ATTENDANCE[time_key] ⊆ DATE[time_key],
DATE[holiday_key] ⊆ HOLIDAY[holiday_key],
ATTENDANCE[lecturer_no, dept_key] ⊆ LECTURER[lecturer_no, dept_key],

LECTURER[dept_key] $\subseteq$ DEPARTMENT[dept_key],
ATTENDANCE[student_no, dept_key] $\subseteq$ STUDENT[student_no, dept_key],
STUDENT[dept_key] $\subseteq$ DEPARTMENT[dept_key],
ATTENDANCE[course_no, dept_key] $\subseteq$ COURSE[course_no, dept_key],
COURSE[dept_key] $\subseteq$ DEPARTMENT[dept_key] and
ATTENDANCE[room_key] $\subseteq$ ROOM[room_key].

By Lemma 3.2 it follows that $\mathbf{R}$ is in SSNF with respect to $\Sigma = \mathrm{F} \cup \mathrm{I}$. We note that if we add an attribute to DEPARTMENT such as post_code and the additional FD, dept_address $\rightarrow$ post_code, then the conditions of Lemma 3.2 do not hold, since DEPARTMENT would not be in BCNF. Despite this $\mathbf{R}$ would still be in SSNF.

The next result shows that SSNF implies independence, and if, in addition, the relation schemas in $\mathbf{R}$ are pairwise incomparable then SSNF implies separability.

Theorem 3.3 The following statements are true:

1) If $\mathbf{R}$ is in SSNF with respect to Σ then it is independent with respect to Σ.

2) If $\mathbf{R}$ is in SSNF with respect to Σ and for all $i, j \in \{0, 1, \ldots, n\}$, with $i \neq j$, R_i and R_j are incomparable, then it is separable with respect to Σ.

Proof. Let $t \in \mathrm{pad}(d)$ originate from R_i and let $t' \in r'$, where r' is the current state of CHASE(d, F) and t' is the current state of t. Moreover let $\rho(t')$ be the set of all *attributes* $\mathrm{A} \in \mathcal{U}$ such that either $t'[\mathrm{A}]$ is nonnull or is a marked null which was equated to some other marked null in obtaining r' from pad(d). Thus $\rho(t) = R_i$ and $\rho(t) \subseteq \rho(t')$. Moreover, $\rho(t') \subseteq C_{\mathrm{F}}(R_i)$ by a straightforward induction on the number of chase steps which were applied to pad(d) in order to obtain r'.

For part (1) we assume that $\mathbf{R}$ is in SSNF with respect to Σ but is not independent with respect to Σ. Thus there is a database d over $\mathbf{R}$ such that $d \models \Sigma$ but RI(d) is inconsistent with respect to F. Consider the first chase step, during the computation of CHASE(d, F), which detects a contradiction when applying an FD, say $\mathrm{X} \rightarrow \mathrm{A}$, to the two tuples, say t'_i and t'_j, in the current state of the chase. Moreover, assume that t'_i and t'_j are the current states of $t_i, t_j \in \mathrm{pad}(d)$ originating from R_i and R_j, respectively. It follows that $\mathrm{XA} \subseteq C_{\mathrm{F}}(R_i), C_{\mathrm{F}}(R_j)$.

We claim that i and j are such that either R_i is a descendant of R_j in JG($\mathbf{R}$) or vice versa. To prove the claim assume that neither R_i nor R_j is a descendant of the other in JG($\mathbf{R}$). Now, by condition (4) of SSNF XA is a subset of the union of the descendant relation schemas of both R_i and R_j. Therefore, by condition (1) of SSNF, it must be the case that $\mathrm{XA} \subseteq R_i, R_j$, since $\mathbf{R}$ being acyclic implies that each edge along the unique path in JT($\mathbf{R}$) between R_i and R_j includes XA in its label. However, XA cannot be a subset of any key for any relation schema with respect to its set of FDs; this contradicts condition (3) of SSNF. Hence the said claim follows.

There are two further steps to consider. Firstly, consider the case when $i = j$. Let Y be the maximal set of attributes $\mathrm{Y} \subseteq R_i$ such that $\mathrm{Y} \subseteq \mathrm{X}$ and $\mathrm{XA} \subseteq C_{\mathrm{F}}(\mathrm{Y})$. Now, if $\mathrm{A} \in R_i$ then, since both t'_i and t'_m are the current states of two distinct tuples, say t_i and t_m, which originate from R_i, it must be that $t'_i[\mathrm{YA}] = t_i[\mathrm{YA}]$ and $t'_m[\mathrm{YA}] = t_m[\mathrm{YA}]$. Thus $r_i \not\models \mathrm{X} \rightarrow \mathrm{A}$, which contradicts our assumption that $d \models \mathrm{F}$ implying that RI(d) must be consistent. On the other hand, if $\mathrm{A} \notin R_i$, then due to condition (4) of SSNF $\mathrm{A} \in R_{ik}$

for some descendant R_{ik} of R_i. Thus by condition (5) of SSNF we have a chain of INDs, $R_i[X_{i1}] \subseteq R_{i1}[X_{i1}], R_{i1}[X_{i2}] \subseteq R_{i2}[X_{i2}], \ldots, R_{i(k-1)}[X_{ik}] \subseteq R_{ik}[X_{ik}]$ such that by condition (3) of SSNF X_{ij} is the primary key of R_{ij}, for $j = 1, 2, \ldots, k$, and $X_{i1} \subseteq$ Y. It follows that $r_{ik} \not\models X_{ik} \rightarrow$ A, which contradicts our assumption that $d \models$ F implying that RI(d) must be consistent. We note that A cannot be a member of two relation schemas on distinct paths leading out of R_i, since by the acyclicity of $\mathbf{R}$ this would imply that A $\in R_i$.

Secondly, consider the case when $i \neq j$; assume without loss of generality that R_j is a descendant of R_i. We consider the various possibilities for A to be included in R_i or R_j. If A $\in R_i$, then it is also the case that A $\in R_j$, since XA $\subseteq C_{\mathrm{F}}(R_i), C_{\mathrm{F}}(R_j)$, and thus XA $\in R_i \cap R_j$ contradicting the fact that X_j is a key for R_j. If A $\in R_j$ but A $\notin R_i$, then we can deduce that $r_j \not\models X' \rightarrow$ A, for some subset $X' \subseteq X_j$, which contradicts our assumption that $d \models$ F implying that RI(d) must be consistent. Finally, if $A \notin R_i, R_j$, then by an argument similar to the case when $i = j$ we can arrive at a contradiction of our assumption that $d \models$ F, which implies that RI(d) must be consistent.

For part (2) we need to show that for all d over $\mathbf{R}$ such that $d \models \Sigma$, $d = \pi^{\downarrow}_{\mathbf{R}}\mathrm{RI}(d)$. Let d be a database over $\mathbf{R}$ and assume that $d \models \Sigma$. By part (1) d is consistent, i.e. RI(d) $\models$ F. Let $d' = \pi^{\downarrow}_{\mathbf{R}}\mathrm{RI}(d)$.

Let r'_i be the relation in d' over $R_i \in \mathbf{R}$. We need to show that $r'_i = r_i$, where r_i is the relation in d over R_i. Let $t_j \in \mathrm{pad}(d)$ originate from R_j, with $j \neq i$, and let $t'_j \in \mathrm{RI}(d)$ be the final state of t_j. Assume that $t'_j[R_i]$ is nonnull and thus $C_{\mathrm{F}}(R_i) \subseteq C_{\mathrm{F}}(R_j)$. We claim that $t'_j[R_i] \in r_i$ proving the result.

There are three cases to consider. In the first case neither R_i nor R_j is a descendant of the other in JG($\mathbf{R}$). It follows that $R_i \subseteq R_j$ by conditions (1) and (4) of SSNF, since $\mathbf{R}$ is acyclic, which contradicts our assumption that R_i and R_j are incomparable. In the second case R_i is a descendant of R_j in JG($\mathbf{R}$). By condition (5) of SSNF $t_j[X_i] = t_i[X_i]$ for some $t_i \in r_i$, and by condition (3) of SSNF and the fact that d is consistent it follows that $t'_j[R_i] = t_i$ as required. In the third case R_j is a descendant of R_i in JG($\mathbf{R}$). By condition (4) of SSNF we have $C_{\mathrm{F}}(R_i) = C_{\mathrm{F}}(R_j)$. Moreover, by the incomparability of R_i and R_j there is an attribute, say A, in $R_i - R_j$ that does not appear in the label of any edge in JG($\mathbf{R}$) on the unique path between R_i and R_j. Thus by condition (4) of SSNF A $\notin C_{\mathrm{F}}(R_j)$ leading to a contradiction. This proves the result. □

In part (2) of Theorem 3.3 we cannot relax the restriction that the R_i in $\mathbf{R}$ be incomparable. As a counterexample consider a database schema $\mathbf{R} = \{R_0, R_1\}$, with $R_0 \subset R_1$. In this case $\mathbf{R}$ is not separable with respect to $\Sigma = \{R_0[R_0] \subseteq R_1[R_0], R_0 \rightarrow (R_1 - R_0)\}$, since, in general, $r_0 \subset \pi_{R_0}(r_1)$, where $d = \{r_0, r_1\}$ is a database over $\mathbf{R}$ satisfying Σ.

We next define the snowflake join of a database d over a snowflake schema $\mathbf{R}$.

Definition 3.4 (Snowflake join and star join) Let $\mathbf{R}$ be a snowflake schema and let R_{α_0}, R_{α_1}, …, R_{α_n} be a topological sort of JG($\mathbf{R}$). The snowflake join of a database d over $\mathbf{R}$, denoted as SNOW(d), is given by

$$\mathrm{SNOW}(d) \quad = \quad (\cdots (r_{\alpha_0} \bowtie r_{\alpha_1}) \bowtie \cdots \bowtie r_{\alpha_n}).$$

When $\mathbf{R}$ is a star schema then SNOW(d) is called the *star join* of d.

The next theorem shows that the snowflake join contains the tuples over the fact table extended with the relevant information in the constituent dimension and subdimension tables. This implies that the results of queries over a data warehouse, which is in SSNF, maintain their consistency, i.e. satisfy the projected set of FDs over their schema.

Theorem 3.4 Let $\mathbf{R}$ be in SSNF with respect to Σ. Then $\mathrm{RI}^{\downarrow}(d) = \mathrm{SNOW}(d)$, if $d \models \Sigma$, where $\mathrm{RI}^{\downarrow}(d)$ is the set of all tuples in $\mathrm{RI}(d)$ having no null values in them.

Proof. Suppose that $d \models \Sigma$ and thus by Theorem 3.3 d is consistent, i.e. $\mathrm{RI}(d) = \mathrm{CHASE}(d, \mathrm{F})$ and $\mathrm{RI}(d) \models \mathrm{F}$. We prove the result by induction on the height, say k, of $\mathrm{JT}(\mathbf{R})$.

(*Basis*): If $k = 0$, then the result is trivial, since $d = \{r_0\}$ and thus $\mathrm{RI}(d) = \mathrm{RI}^{\downarrow}(d) = r_0$, since by hypothesis $d \models \Sigma$.

(*Induction*): Assume the result holds when the height of $\mathrm{JT}(\mathbf{R})$ is k, with $k \geq 0$; we then need to prove that the result holds when the height of $\mathrm{JT}(\mathbf{R})$ is $k+1$.

Let $R_{\alpha_0}, R_{\alpha_1}, \ldots, R_{\alpha_m}$ be the set of all relation schemas in $\mathrm{JT}(\mathbf{R})$ whose height is less than or equal to k, and let $d_k = \{r_{\alpha_0}, r_{\alpha_1}, \ldots, r_{\alpha_m}\}$. Then by inductive hypothesis, noting condition (2) of Definition 3.1, we have

$$\mathrm{RI}^{\downarrow}(d_k) = \mathrm{SNOW}(d_k) = (\cdots(r_{\alpha_0} \bowtie r_{\alpha_1}) \bowtie \cdots \bowtie r_{\alpha_m}).$$

Now, let $R_{\alpha_{m+1}}, \ldots, R_{\alpha_n}$ be the remaining relation schemas in $\mathbf{R}$ whose height in $\mathrm{JT}(\mathbf{R})$ is $k+1$. It remains to show that

$$\mathrm{RI}^{\downarrow}(d) = (\cdots(\mathrm{RI}^{\downarrow}(d_k) \bowtie r_{\alpha_{m+1}}) \bowtie \cdots \bowtie r_{\alpha_n}) = \mathrm{SNOW}(d).$$

Now, $\mathrm{RI}^{\downarrow}(d) \subseteq \mathrm{SNOW}(d)$ by conditions (3) and (5) of SSNF, since $\mathrm{RI}(d) = \mathrm{CHASE}(d_k, \mathrm{F})$. So we need to establish that $\mathrm{SNOW}(d) \subseteq \mathrm{RI}^{\downarrow}(d)$. Now, since $d \models \mathrm{F}$, we have $\pi_{R_j}(\mathrm{SNOW}(d)) \models F_j$, where $j \in \{m+1, \ldots, n\}$. Therefore, on using the inductive hypothesis, $\mathrm{SNOW}(d) \models \mathrm{F}$, since by condition (4) of SSNF and the fact that R_j is a leaf node in $\mathrm{JT}(\mathbf{R})$ we have $C_F(R_j) = R_j$. The result that $\mathrm{SNOW}(d) \subseteq \mathrm{RI}^{\downarrow}(d)$ follows, on account of the fact that $\mathrm{SNOW}(d_k)$ joins with a relation r_j over R_j only through a foreign to primary key relationship, where referential integrity is maintained through the INDs in I. □

4 An Extension of Snowflake Schemas

We observe that Definition 2.16 of acyclicity is more restrictive, i.e. condition (1), than the classical one encountered in [BFMY83]. Hereafter we extend the concept of snowflake schema by relaxing this restriction so as to include a broader class of data warehouse designs. We demonstrate this extension via an example.

Consider part of a data warehouse dealing with a distributed supply chain, whose fact table is FACT and whose dimension tables are CUSTOMER, SUPPLIER and NATION. The attributes of the tables are given by

FACT = (cust_key, supp_key, amount),
CUSTOMER = (cust_key, cname, caddress, cust_nation_key),
SUPPLIER = (supp_key, sname, saddress, supp_nation_key) and
NATION = (nation_key, nname).

This database schema is acyclic according to the original definition of acyclicity given in [BFMY83] but it is *not* acyclic according to Definition 2.16, since NATION does not have any attributes in common with any of the other relation schemas. To alleviate this problem we relax Definition 2.16 of a join tree by modifying part (1) of the definition so that R ∩ S *may be empty*. Moreover, we relax part (1) in Definition 3.1 of a snowflake schema **R** so that the condition that no two distinct edges have identical labels applies only to edges having *nonempty* labels. We note that as a result of these modifications we may consider separately all relation schemas, say R, whose intersection with their parent schema in the join tree, say S, is empty. For the rest of this section we will let **S** be the database schema resulting from removing all such relation schemas R from a snowflake schema **R**. Regarding our example **R** = {FACT, CUSTOMER, SUPPLIER, NATION} and **S** = {FACT, CUSTOMER, SUPPLIER}.

Continuing the example, F contains the FDs:

{cust_key, supp_key} → amount,
cust_key → {cname, caddress, cust_nation_key},
supp_key → {sname, saddress, supp_nation_key} and
nation_key → nname

and I contains the INDs:

FACT[cust_key] ⊆ CUSTOMER[cust_key],
FACT[supp_key] ⊆ SUPPLIER[supp_key],
CUSTOMER[cust_nation_key] ⊆ NATION[nation_key] and
SUPPLIER[supp_nation_key] ⊆ NATION[nation_key].

We now modify Definition 3.3 of SSNF so that **R** is in *Extended Snowflake Schema Normal Form* (ESSNF) if

1) **S** (rather than **R**) is in SSNF;

2) I is a set of key-based INDs; and

3) the graph representation of I, G_I, is acyclic; here we may assume that G_I is connected.

For our example it can easily be verified that **R** is in ESSNF, since (1) **S** satisfies Definition 3.3 of SSNF, (2) I is key-based, due to the fact that nation_key is the primary key of NATION and (3) G_I is acyclic. We note that in this example F and I possess the desirable property of having no interaction, since all the relation schemas in **R** are in BCNF; see Theorem 10.21 in [MR92] and Corollary 3.6 in [LL99b].

We can extend the results of Section 3 for a database schema **R** in ESSNF via a natural transformation of **R** into a database schema **R′** that can be shown to be in SSNF. This provides justification for the definition of ESSNF.

We demonstrate the transformation on our example without giving the formal definition. We tranform **R** into **R′** = {FACT, CUSTOMER′, SUPPLIER′}, where

CUSTOMER′ = (cust_key, cname, caddress, cust_nation_key, cust_nname),
SUPPLIER′ = (supp_key, sname, saddress, supp_nation_key, supp_nname),

and the FDs cust_nation_key → cust_nname and supp_nation_key → supp_nname replace the FD nation_key → nname in F. On the database level, we replace the relation CUSTOMER by its natural join with the relation over NATION, after a suitable renaming of attributes, and correspondingly we replace the relation SUPPLIER by its natural join with the relation over NATION again after a suitable renaming of attributes. As a result the two INDs involving NATION are no longer relevant in the context of $\mathbf{R}'$. We note, however, that FACT is still in BCNF.

Although it is tempting to extend SSNF to cyclic database schemas, it is an open problem to what extent our results will generalise in such cases.

5 Information-Theoretic Interpretation

We now utilise the information-theoretic treatment of relational databases developed in [Mal86, CP87, Lee87, Mal88]. This approach is important since it allows us to accommodate for probabilistic information in the data warehouse, which is fundamental in decision making [Lin85]. Herein we show that the redundancy in the snowflake join of the primary key of the fact table is zero, i.e. it is minimal.

We interpret R_i as a sequence of distinct random variables, and assume that tuples in relations r_i over R_i are distributed according to a probability function p_i, where $p_i(t)$ is the probability of a tuple t over R_i for $t \in \mathrm{DOM}(R_i)$; in particular, $t \in r_i$ if and only if $p_i(t) > 0$. (In the absence of any further information we can assume a uniform distribution of the tuples in a relation.) The distribution of the projection $\pi_{\mathrm{X}}(r_i)$, with $\mathrm{X} \subseteq R_i$, is interpreted as the marginal distribution of X based on p_i. (See [Hil91] for a tutorial which links relational database concepts with probability concepts.)

Definition 5.1 (Entropy) The entropy of a set X of attributes of a relation schema R with respect to a relation r over R, denoted by $H_r(\mathrm{X})$ (or simply $H(\mathrm{X})$ whenever r over R is understood from context), is given by

$$H_r(\mathrm{X}) = \sum_{x \in \mathrm{DOM(X)}} p(x) \log p(x),$$

where p is the probability function for R, i.e. $p(t) > 0$ if and only if $t \in r$. We take the base of the logarithm to be 2 and use the standard convention that $0 \log 0 = 0$.

The next theorem was proved in [Mal86, Lee87].

Theorem 5.1 The following statements are true:

1) $H(\mathrm{X}) \leq H(\mathrm{XY})$.

2) A relation r over R satisfies the FD X → Y if and only if $H(\mathrm{X}) = H(\mathrm{XY})$.

3) $H(\mathrm{Y} \mid \mathrm{X}) = H(\mathrm{XY}) - H(\mathrm{X})$, where $H(\mathrm{Y} \mid \mathrm{X})$ is the entropy of Y conditional on X, i.e.

$$H(\mathrm{Y} \mid \mathrm{X}) = \sum_{xy \in \mathrm{DOM(XY)}} p(xy) \log p(y \mid x). \quad \square$$

Definition 5.2 (Redundancy) The *redundancy* in a relation r over R of a set of attributes $\mathrm{X} \subseteq \mathrm{R}$ is the conditional entropy $H(\mathrm{R}{-}\mathrm{X} \mid \mathrm{X})$.

We observe that by Theorem 5.1 $H(\mathrm{R}{-}\mathrm{X} \mid \mathrm{X}) \geq 0$, with equality if and only if $H(\mathrm{X}) = H(\mathrm{R})$, i.e. when X is a superkey for R. If X is not a superkey for R then redundancy will arise when some subset of r satisfies a nontrivial FD $\mathrm{X} \rightarrow \mathrm{Y}$, with $\mathrm{XY} \subset \mathrm{R}$. The most important case of redundancy is when $\mathrm{Y} \subseteq C_{\mathrm{F}}(\mathrm{X})$ implying that r satisfies the nontrivial FD $\mathrm{X} \rightarrow \mathrm{Y}$ but X is not a superkey for R. This provides a justification for BCNF, since $\mathbf{R}$ is in BCNF with respect F if and only if all the relations in d have no redundancy due to F. (For an in-depth treatment of redundancy in relational databases when the constraints are FDs and INDs, see [LV00].)

We now discuss a general justification of SSNF in terms of entropy, when $\mathbf{R}$ is a star schema, i.e. the height of $\mathrm{JT}(\mathbf{R})$ is one. Let us assume that the cardinality of the relation r_i over the ith dimension table is N_i, for $i = 1, 2, \ldots, n$. Then the cardinality N_0 of the relation r_0 over the fact table R_0 is of the order of $\prod_{i=1}^{n} N_i$. This product is a rough upper bound on the cardinality of r_0 and a better estimate of the cardinality of r_0 can be obtained if we know the fraction of each r_i which is recorded in r_0. It follows that the entropy of r_0 can be approximated as the sum of the entropies of the r_i's, for $i = 1, 2, \ldots, n$. On the basis of the observations made after Definition 5.2 this rough analysis justifies R_0 being in BCNF in terms of minimising redundancy in d. It is less important that R_i, for $i = 1, 2, \ldots, n$, be in BCNF since the redundancy $H(R_i{-}\mathrm{X} \mid \mathrm{X})$ in $r_i \in d$ over R_i is expected to be insignificant relative to the potential redundancy in r_0 were it not to be in BCNF. On the other hand, by not decomposing the dimension tables R_i query processing can be optimised via the *star join* [OG95].

The following result is a special case of Theorem 7 in [Mal88] and Theorem 4 in [Lee87].

Lemma 5.2 Let $\mathbf{R}$ be a snowflake schema and r be a relation over $\mathcal{U}$. Then $r \models \bowtie[\mathbf{R}]$ if and only if

$$H_r(\mathcal{U}) = \sum_{i=0}^{n} H(R_i) - \sum_{i=1}^{n} H(X_i),$$

where R_i and X_i are as introduced after Definition 3.2. $\square$

The next result follows from Theorem 4.2 in [LL99a], which implies that $\mathbf{R}$ is a lossless join decomposition of $\mathcal{U}$, Lemma 5.2, and Theorem 5.1.

Theorem 5.3 Let $\mathbf{R}$ be a snowflake schema, F^+ contain the set of FDs $\mathrm{G} = \{X_0 \rightarrow Y_0, X_1 \rightarrow Y_1, \ldots, X_n \rightarrow Y_n\}$ over $\mathcal{U}$, and let r be a relation over $\mathcal{U}$. Then $r \models \mathrm{G}$ if and only if

$$H_r(\mathcal{U}) = \sum_{i=0}^{n} H(R_i) - \sum_{i=1}^{n} H(X_i) = H(R_0) = H(X_0). \quad \square$$

Thus the conditional entropy $H_r(\mathcal{U}-X_0 \mid X_0)$ is minimal, i.e. zero, when r is the snowflake join SNOW(d). The next corollary states that when $\mathbf{R}$ is in SSNF this is indeed the case.

Corollary 5.4 Let d be a database over a database schema $\mathbf{R}$ which is in SSNF with respect to a set Σ of FDs and INDs. Then $d \models \Sigma$ implies that $H_{\mathrm{SNOW}(d)}(\mathcal{U} - X_0 \mid X_0) = 0$. □

6 Concluding Remarks

We have formalised the intuition behind the snowflake schema via the notion of an acyclic database schema and its join graph. We have defined SSNF which is a normal form for data warehouses and shown that it possesses several desirable properties. In particular, database schemas $\mathbf{R}$ which are in SSNF are independent as well as separable when the relation schemas in $\mathbf{R}$ are incomparable. That is, integrity maintenance and updates of functional dependencies can be carried out independently on a single relation basis. We have also shown that the snowflake join of the relations in a database over a database schema, which is in SSNF, comprises the nonnull tuples in the representative instance. To enlarge the class of database schemas covered by SSNF we have extended the notion of SSNF to ESSNF. Finally, by using an information-theoretic approach, we have shown that the redundancy in the snowflake join of the primary key of the fact table is zero.

In practice it may be the case that the simplicity of a star schema will suffice [KRRT98]. It is not clear that the overheads in further normalising the dimension tables of a star schema, in order to obtain a snowflake schema, outweigh the simplicity of the star schema, since as argued in Section 5 the size of the relation over the fact table dominates the size of the database. On the other hand, the generality of a snowflake schema allows us to model attribute hierarchies which may save significant space in some cases and provide logical separation between a dimension and its subdimensions.

A remaining open problem is to devise an efficient algorithm for constructing database schemas which obey SSNF. Such an algorithm would have to be integrated within a data warehouse design methodology. Moreover, an important aspect of data warehouses, which we have not considered in this work, is the incorporation of a cost model for queries over a snowflake schema which can be used to optimise the height of its join tree.

Acknowledgements. The authors would like to thank the referees for their constructive comments. We especially thank one of the referees who suggested the extension of snowflake schema detailed in Section 4.

References

[AC91] P. Atzeni and E.P.F. Chan. Independent database schemes under functional and inclusion dependencies. *Acta Informatica*, 28:777–799, 1991.

[BB79] C. Beeri and P.A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Transactions on Database Systems*, 4:30–59, 1979.

[BFMY83] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30:479–513, 1983.

[BH90] F. Buckley and F. Harary. *Distance in Graphs*. Addison-Wesley, Redwood City, Ca., 1990.

[CD97] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26:65–74, 1997.

[CK86] S.S. Cosmadakis and P.C. Kanellakis. Functional and inclusion dependencies: A graph theoretic approach. In P.C. Kanellakis and F. Preparata, editors, *Advances in Computing Research*, volume 3, pages 163–184. JAI Press, Greenwich, 1986.

[CM87] E.P.F. Chan and A.O. Mendelzon. Independent and separable database schemes. *SIAM Journal on Computing*, 16:841–851, 1987.

[CP87] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *Proceedings of International Conference on Very Large Data Bases*, pages 71–81, Brighton, 1987.

[CV83] M.A. Casanova and V.M.P. Vidal. Towards a sound view integration methodology. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 36–47, Atlanta, 1983.

[Fag83] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30:514–550, 1983.

[GMR98] M. Golfarelli, D. Maio, and S. Rizzi. Conceptual design of data warehouses from E/R schemes. In *Proceedings of the Hawaii International Conference on System Sciences*, pages 334–343, Hawaii, 1998.

[Hil91] J.R. Hill. Relational databases: A tutorial for statisticians. In *Proceedings of Symposium on the Interface between Computer Science and Statistics*, pages 86–93, Seattle, Wa., 1991.

[HLV00] B. Hüsemann, J. Lechtenbörger, and G. Vossen. Conceptual data warehouse design. In *Proceedings of International Workshop on Design and Management of Data Warehouses*, Stockholm, 2000.

[Hon82] P. Honeyman. Testing satisfaction of functional dependencies. *Journal of the ACM*, 29:668–677, 1982.

[Inm96] W.H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Chichester, second edition, 1996.

[KM00] R. Kimball and R. Mertz. *The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse*. John Wiley & Sons, Chichester, 2000.

[KRRT98] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses*. John Wiley & Sons, Chichester, 1998.

[LAW98] W. Lehner, J. Albrecht, and H. Wedekind. Normal forms for multidimensional databases. In *Proceedings of International Conference on Scientific and Statistical Data Management*, pages 63–72, Capri, 1998.

[Lee87] T.T. Lee. An information-theoretic analysis of relational databases - Part I: Data dependencies and information metric. *IEEE Transactions on Software Engineering*, 13:1049–1061, 1987.

[Lin85] D.V. Lindley. *Making Decisions*. John Wiley & Sons, London, 1985.

[LL99a] M. Levene and G. Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer-Verlag, London, 1999.

[LL99b] M. Levene and G. Loizou. How to prevent interaction of functional and inclusion dependencies. *Information Processing Letters*, 71:115–125, 1999.

[LL01] M. Levene and G. Loizou. Guaranteeing no interaction between functional dependencies and tree-like inclusion dependencies. *Theoretical Computer Science*, 254:683–690, 2001.

[LV00] M. Levene and M.W. Vincent. Justification for inclusion dependency normal form. *IEEE Transactions on Knowledge and Data Engineering*, 12:281–291, 2000.

[Mal86] F.M. Malvestuto. Statistical treatment of the information content of a database. *Information Systems*, 11:211–223, 1986.

[Mal88] F.M. Malvestuto. Existence of extensions and product extensions for discrete probability distributions. *Discrete Mathematics*, 69:61–77, 1988.

[MR92] H. Mannila and K.-J. Räihä. *The Design of Relational Databases*. Addison-Wesley, Reading, Ma., 1992.

[OG95] P.E. O'Neil and G. Graefe. Multi-table joins through bitmapped join indices. *ACM SIGMOD Record*, 24:8–11, 1995.

[Sag83] Y. Sagiv. A characterization of globally consistent databases and their access paths. *ACM Transactions on Database Systems*, 8:266–286, 1983.

[Sag91] Y. Sagiv. Evaluation of queries in independent database schemes. *Journal of the ACM*, 38:120–161, 1991.