Conjunctive Query Equivalence of Keyed Relational Schemas

(Extended Abstract)

Joseph Albert*
Computer Science Dept.
Portland State University
P.O. Box 751
Portland, OR 97210
jalbert@acm.org

Yannis Ioannidis*
Computer Sciences Dept.
University of Wisconsin
1210 W. Dayton St.
Madison, WI 53706
yannis@cs.wisc.edu

Raghu Ramakrishnan*
Computer Sciences Dept.
University of Wisconsin
1210 W. Dayton St.
Madison, WI 53706
raghu@cs.wisc.edu

Abstract

The notion of when two schemas are equivalent is fundamental to database design, schema integration, and data model translation. An important notion of schema equivalence, *query equivalence* was introduced in [3], and used to evaluate the correctness of schema transformations. The logically equivalent notion of *calculous equivalence*, as well as three progressively weaker notions of schema equivalence were introduced in 1984 by Hull [9, 10], who showed that two schemas with no dependencies are equivalent (under all four notions of equivalence) if and only if they are identical (up to renaming and re-ordering of attributes and relations). Hull also conjectured that the same result holds for schemas with primary keys. In this work, we resolve the conjecture in the affirmative for the case of query equivalence based on mappings using conjunctive relational queries with equality selections.

1 Introduction

A fundamental concept in database theory is that of *schema equivalence*. Informally, two schemas are equivalent if either one can simulate the other in terms of their capacities to store database instances and support queries. An understanding of schema equivalence is important for schema integration in heterogeneous multidatabase systems, [4, 16], where two schemas with dependencies describing the semantics of the data are given, and one would like to integrate the schemas. Because the schemas to be integrated may have semantic incompatibilities, it may be necessary to transform one or both of the schemas to equivalent schemas in preparation for integration.

For example, suppose one wants to integrate the following two relational schemas with key dependencies and referential integrity constraints. Key attributes are marked with an asterisk, and referential integrity constraints are shown using standard inclusion dependency notation.

employee(ss*, eName, salary, depId)
department(deptId*, deptName, mgr)
salespeople(ss*, yearsExp)

employee[depId] ⊆ department[deptId]
salespeople[ss] ⊆ employee[ss]
employee[ss] ⊆ salespeople[ss]

Schema 1

empl(ssn*, ename, sal, dep, yrsExp)
dept(departId*, dName, manager)

empl[dep] ⊆ dept[departId]

Schema 2

Assume it is desirable to integrate the two schemas by integrating the employee relation in the first schema with the empl relation in the second schema to form a unified employee relation, and to integrate the department relation from the first schema with the dept relation in the second schema to form a unified department relation. In this case, there is a semantic incompatibility in that the yearsExp attribute of an employee is stored in a separate relation, salespeople, so that a fully general integration of the employee relation and empl relation is not posssible.

However, it can be shown that the first schema could be transformed into an equivalent schema in which the incompatibility is removed. Such a schema, Schema 1′, is shown here with Schema 2.

employee(ss*, eName, salary, depId, yearsExp)
department(deptId*, deptName, mgr)
salespeople(ss*)

employee[depId] ⊆ department[deptId]
salespeople[ss] ⊆ employee[ss]
employee[ss] ⊆ salespeople[ss]

Schema 1′

empl(ssn*, ename, sal, dep, yrsExp)
dept(departId*, dName, manager)

empl[dep] ⊆ dept[departId]

Schema 2

*This is a Massive Digital Data Systems (MDDS) project sponsored by the Advanced Research and Development Committee of the Community Management Staff.

PODS '97 Tucson Arizona USA

Note that in the absence of the inclusion dependencies specified, Schema 1 and Schema 1′ would not be equivalent. With the dependencies that hold on Schema 1, however, the transformation is equivalence preserving, so that Schema 1 and Schema 1′ are equivalent, and the incompability has been removed. The employee and empl relations now can be integrated into a unified relation. Syntactic characterizations of equivalence of relational schemas with various families of dependencies, such as primary keys, referential integrity constraints, functional dependencies, are needed. In particular, one would like to have a set of transformations for which all schemas equivalent to a given schema can be generated by applying some sequence of transformations from the set.

The notion of schema equivalence also is important in database design [5, 8, 11, 17] where, given a schema proposed for some application, one may want to choose an equivalent schema that satisfies one or more desirable normal forms. Indeed, schema equivalence was first proposed in this context by Codd [8], wherein two schemas are considered equivalent if they can support the same queries. Subsequently, a notion of schema equivalence was proposed in which two schemas, both of which are decompositions of the same universal relation, are equivalent if the set of instances of the universal relation for which the decomposition is lossless is the same for both schemas [6, 12]. That is, either schema can *represent* the same set of universal instances. This notion of equivalence was used for database design, but has the limitation that it only applies to pairs of schemas both of which are projections of the same universal relation scheme. In general, such an assumption is not possible in multidatabase schema integration. Moreover, closed-form characterizations of this form of equivalence are not available, although an algorithm to test for such equivalence is given in [6]. Similar notions of equivalence were defined in [2, 14].

Another notion of equivalence that has been proposed considers two schemas to be equivalent if there is a bijection between the set of database instances of one schema and the set of instances of the other [13, 15]. However, this simply means that the set of instances of one schema has the same cardinality as the set of instances of the other schema. Moreover, if the domain of values available to store in a database is infinite, then all schemas are equivalent.

The limitations of these notions of equivalence are overcome by the notion of query equivalence that was introduced in [3], and studied by Hull, who also introduced three progressively less restrictive notions of equivalence, Z-generic equivalence, Z-internal equivalence, and absolute equivalence, and provided a rich foundation of theoretical results concerning schema equivalence [10]. Hull also showed that for relational schemas with no dependencies, all four notions of schema equivalence are logically equivalent, and that two relational schemas with no dependencies are equivalent if and only if they are identical, up to renaming and re-ordering of attributes and relations. Thus a characterization of schema equivalence for relational schemas with no dependencies is available.

Hull conjectured that this should generalize to relational schemas with primary keys, that is, that they are equivalent if and only if they are identical, up to renaming and re-ordering of attributes and relations. We resolve this conjecture in the affirmative for *conjunctive query equivalence*, where instance mappings are conjunctive relational algebra queries with equality selections. (Query equivalence in [10] uses the full relational algebra for such mappings.)

Such a result demonstrates that two keyed schemas support the same conjunctive queries if and only if there are identical, up to renaming and re-ordering of attributes and relations. This is a negative result about the existence of equivalence-preserving transformations for schemas having only primary keys, which suggests that other dependencies are important for transforming schemas in meaningful ways. Indeed, the example above shows that for schemas having both primary key dependencies and referential integrity constraints, there are non-trivial schema transformations that preserve equivalence.

A characterization of equivalence for schemas with only primary keys is critical to obtaining similar characterizations for schemas with other dependencies. For instance, when schemas with primary keys and referential integrity constraints are considered, a schema with only primary keys is a degenerate case where it happens that no referential integrity constraints have been specified. Thus, it would be impossible to characterize equivalence of schemas with primary keys and referential integrity constraints without characterizing equivalence of schemas with only primary keys.

The present report is an extended abstract of a full paper that includes complete proofs as well as additional motivational material for the results stated here [1]. The rest of this abstract is organized in the following manner. Section 2 presents precise formal definitions of standard concepts pertaining to the relational model of data as well as definitions of other concepts used or introduced in this work. Section 3 states various results concerning conjunctive queries in general as well as the central results of this work that pertain to conjunctive query dominance and equivalence of relational schemas. Concluding remarks are given in Section 4.

2 Preliminaries

In this section, we formalize what is meant by a schema and define various concepts and notation. We assume that the reader is familiar with the relational model of data [7]. A *domain* is a countably infinite set of atomic values. A collection of *attribute types* over some domain D is a finite collection of disjoint subsets of D. Attribute types are also (countably) infinite. An *attribute* is a pair consisting of a name (called the name of the attribute) and an attribute type (called the type of the attribute). A *relation scheme* consists of a name (name of the relation) and an ordered list of attributes, generally written $R[A_1, A_2, ..., A_k]$. R is the name of the relation. If for each i, N_i is the type of attribute A_i, then an instance r of relation R is just some subset of the cross-product $N_1 \times N_2 \times ... \times N_k$. The tuple $\langle N_1, N_2, ..., N_k \rangle$ is called the *type* of the relation R.

The set of all instances of R, written $i(R)$, is defined as the power set of the cross-product of attribute types in the scheme. A *relational database schema* is a tuple of relation schemes. A database instance of the schema is a tuple of instances of each relation scheme in the schema. We write $i(S)$ for the set of all instances of schema S.

A *superkey dependency* on a relation is a declared subset of attributes of the relation. The subset of attributes is called a superkey. A superkey dependency on some relation is satisfied by an instance of the relation if any pair of distinct tuples in the instance have different values for at least one of the attributes in the superkey. If no proper subset of some superkey of a relation is also a superkey, then the superkey is called a *key*, and the associated dependency is called a key dependency. A *keyed schema* is one where a single key is specified for each relation in the schema, and no other dependencies are specified to hold in the schema.

A schema for which no dependencies are specified is called an *unkeyed* schema.

A *functional dependency* on a schema is a pair of attribute sets. If X and Y are the two sets of attributes, the dependency is usually written $X \rightarrow Y$. If all of the attributes in both X and Y belong to the same relation, then an instance of that relation is said to satisfy dependency if every pair of tuples of the relation which differ on some attribute in Y also differ on some attribute in X. Otherwise, the dependency fails for the given relation instance. An instance of the schema satisfies some functional dependency $X \rightarrow Y$ if all of the attributes in both X and Y belong to the same relation, and the instance of this relation in the database instance satisfies the dependency. If all of the attributes in X and Y do not belong to the same relation, then the functional dependency fails for any instance of the schema. Note that allowing functional dependencies to be expressed in this way differs from the usual formalization where a functional dependency is only defined using attributes from a single relation, but this trivial extension allows for a concise statement of some of the results below.

A *view* over a schema S is a pair (V, q), where V is a relation scheme and $q : i(S) \rightarrow i(V)$ maps each instance of S to an instance of V. The mapping q is called a *query*. If $q(d) = a$ for some database d, and a is an instance of V, then a is called the *answer* to the query q for database d. The type of the view, as well as the type of the query, is just the type of V. A *query language* consists of a set of queries together with a syntax capable of specifying any query in the set.

Definition: Given some schema S on which two queries $q : i(S) \rightarrow i(V)$ and $q' : i(S) \rightarrow i(V)$ having the same type are defined, we say that q is *contained in* q', written $q \sqsubseteq q'$, if for every instance $d \in i(S)$, $q(d) \subseteq q'(d)$. ■

Definition: We say that q is *equivalent* to q', written $q \cong q'$, if $q \sqsubseteq q'$ and $q' \sqsubseteq q$. ■

Definition: Given schemas $S_1 = \langle R_1^1, R_2^1, ..., R_n^1 \rangle$ and $S_2 = \langle R_1^2, R_2^2, ..., R_m^2 \rangle$, and a query language, L, then $\alpha = \langle v_1, v_2, ..., v_m \rangle$ is a *query mapping* from S_1 to S_2 if each v_k is a view over S_1 defined using queries in L, and the type of v_k is the same as the type of R_k^2 for each k. ■

For each instance of S_1, the query mapping defines an instance of S_2, since each v_k defines an instance of R_k^2. We write $\alpha : i(S_1) \rightarrow i(S_2)$. If α is a query mapping from the keyed schema S_1 to the keyed schema S_2, then we say that α is *valid* if it maps each instance of S_1 satisfying the key dependencies for S_1 to an instance of S_2 satisfying the key dependencies for S_2. Query mappings between unkeyed schemas are always valid.

Definition: Let S_1 and S_2 be two keyed schemas, and let L be a query language. Then we say that S_2 *L-dominates* S_1, written $S_1 \preceq_L S_2$, if there are valid query maps $\alpha : i(S_1) \rightarrow i(S_2)$ and $\beta : i(S_2) \rightarrow i(S_1)$ such that $\beta \circ \alpha$ is the identity map on $i(S_1)$. To indicate the query mappings that establish the dominance we sometimes write $S_1 \preceq_L S_2$ by (α, β). ■

Definition: If $S_1 \preceq_L S_2$ and $S_2 \preceq_L S_1$, then we say that the two schemas are *L-equivalent*, written $S_1 \equiv_L S_2$. ■

The notions of L-dominance and L-equivalence were introduced in [3]. We sometimes write $S_1 \preceq S_2$, or $S_1 \equiv S_2$, when the particular language L is clear from the context. The following result is proved in [10].

Theorem (Hull 1986) *If L is the relational algebra, S_1 and S_2 are schemas with no dependencies, then $S_1 \equiv_L S_2$, if and only if S_1 and S_2 are identical up to renaming and reordering of attributes and relations.* ■

Hull also conjectured that this result holds for keyed schemas, but this conjecture remains open.

Definition: A *conjunctive query* is a relational algebra query that can be expressed using only the operations of select, project, join, and cartesian product. ■

A conjunctive query view (V, q) is specified using a syntactic style borrowed from Datalog. However, the syntax used here is more restrictive than Datalog, allowing only distinct variables as placeholders in columns of relations, with all selection and join conditions occurring in a separate list of equality predicates included in the conjunct:

$$V(A_1, A_2, ..., A_n) \ :- \ R_1(X_1^1, ..., X_{i_1}^1), ..., R_k(X_1^k, ..., X_{i_k}^k), \ equality - list.$$

Each R_i is a relation, and each X_i^j is a distinct variable serving as a placeholder. The A_i's are (not necessarily distinct) variables that occur among the X_i^j variables to signify this variable is in the result of the query. Other variables might be dummy placeholders to signify attributes in the R_i that are projected out of the result, or variables participating in joins or selections whose columns do not appear in the final result.

As in Datalog, a comma between two relations signifies a join or cross-product. The equality list is a list of equality predicates with form either $X = Y$ or $X = a$. In the first case, the two variables X and Y are being equated. If both X and Y are used as placeholders in the same relation, then this is a column selection. If the two variables occur in different relations, then this corresponds to a join condition. For the equality predicate $X = a$, the column of the relation containing the variable X as a placeholder in the query has a selection condition, selecting tuples with value for that attribute equal to the constant a. Constants may occur explicitly among the A_i. All variables occurring in equality predicates in the equality list must also occur as a placeholder for some attribute in some relation occurring in the body of the query. Note that all conjunctive relational algebra queries with equality selections can be expressed with the syntax just described. For the remainder of this paper, "conjunctive query" means "conjunctive query with equality selections".

Definition: For any attribute A assigned from a column in the result of a conjunctive query, we say that A *receives* attribute B from relation R if in the representation of the query, A is assigned from a variable that occurs at or is equated to a variable at the location of attribute B in R. If an attribute A is assigned by a constant symbol, then we say that attribute A receives the constant. ■

Thus, in the query:

$$R(X, Y, Z) \ :- \ P(X, Y), Q(T, Z), Y = T.$$

the second attribute of relation R receives from P the second attribute listed in the scheme of P, and it also receives from

Q the first attribute listed in the scheme of Q. On the other hand, in the query:

$$R(a, Y, X) \ :- \ P(X, Y).$$

the first attribute of relation R receives the constant a. An attribute can receive multiple, distinct attributes, as shown in the first example.

Definition: An instance d of some schema is *attribute-specific* if, for any two distinct attributes A and B in the schema, $\pi_A(d) \cap \pi_B(d) = \emptyset$. ■

Definition: In a conjunctive query, a join is an *identity join* if all of the relations participating in the join are the same relation, and every join condition equates an attribute in one occurrence of the relation in the query body to the same attribute in another occurrence of the same relation in the query body. ■

For example, in the query:

$$Q(X, Y, Z) : -R(X, Z), R(Y, T), Z = T.$$

the join condition is an identity join. This is because the join is of a relation with itself, and the only join condition equates the second attribute of relation R to itself. On the other hand, in the query:

$$Q(X, Y, Z) : -R(X, Y, Z), R(T, U, V), Y = T, Z = V.$$

there is a self-join that is *not* an identity join. In this case, the join condition $Y = T$ equates two different attributes of relation R. A cross-product of a relation with itself (some number of times) is a degenerate identity join.

Definition: A relation R occurring in the body of a conjunctive query is *ij-saturated* if no occurrence of R in the query participates in a selection condition, all join conditions involving R are identity joins, and all possible identity join conditions for R can be inferred from the equality conditions specified. ■

Thus, R is ij-saturated in the following query:

$$Q(X, Y) \ :- \ R(X, Y), R(A, B), R(C, D), X = A, \\ X = C, Y = B, Y = D.$$

The join condition $A = C$ is inferred by transitivity from $X = A$ and $X = C$. But R is not ij-saturated in the query:

$$Q(X, Y) : -R(X, Y), R(A, B), R(C, D), X = A, \\ X = C, A = C, Y = B.$$

This is because neither $Y = D$ nor $B = D$ can be inferred from the list of join conditions.

Definition: A conjunctive query is ij-saturated if every relation occurring in its body is ij-saturated. ■

Note that given any conjunctive query q that has no selection conditions and no join conditions other than identity joins, we can construct an ij-saturated query $\hat{q}$ that has the same number of occurrences of relations among its literals as the original query q, but with the extra identity join conditions added so each relation is ij-saturated. For example, given the query:

$$Q(X, Y) \ :- \ R(X, Y), R(A, B), R(C, D), X = A, \\ X = C, A = C, Y = B.$$

we can construct the ij-saturated query:

$$Q'(X, Y) \ :- \ R(X, Y), R(A, B), R(C, D), X = A, X = C, \\ A = C, Y = B, Y = D, B = D.$$

Note that $\hat{q} \sqsubseteq q$ always holds because $\hat{q}$ is just q with extra join conditions added.

Definition: A conjunctive query is a *product query* if there are no selection or join conditions, and every relation occurring in the body of the query occurs only once. That is, a product query can consist of only a single relation, or a cross-product of distinct relations. ■

3 Results

In this section, we show that Hull's result stated in the previous section can be generalized to keyed schemas for query equivalence by conjunctive relational algebra with equality selections. For ease of presentation, we write $S_1 \preceq S_2$ by (α, β) to mean that S_2 dominates S_1 by the conjunctive query mappings α and β, throughout this section.

The following two lemmas demonstrate some basic properties of conjunctive queries, and their proofs are straightforward.

Lemma 1 *Every ij-saturated query is equivalent to a product query having the same relations in its body as the ij-saturated query.*

Lemma 2 *Given a conjunctive query q defined over some schema S such that q has no selection conditions nor any join conditions that are not identity joins, there exists a product query $\hat{q}$ satisfying the following conditions:*
a) $\hat{q} \sqsubseteq q$;
b) for every $d \in i(S)$, any functional dependency that holds on $q(d)$ also holds on $\hat{q}(d)$;
c) for every $d \in i(S)$, if $q(d)$ is non-empty, then $\hat{q}(d)$ is non-empty;
d) all of the relations occurring in the body of q also occur in the body of $\hat{q}$.

The next three lemmas present some properties of conjunctive query maps that establish dominance.

Lemma 3 *If $S_1 \preceq S_2$ by (α, β) then for every attribute A occurring in S_1 there is some attribute B in S_2 such that A is received by B under α, and B is received by A under β.*

Lemma 4 *If $S_1 \preceq S_2$ by (α, β) and B is an attribute in S_2, then if B is received by some attribute A in S_1 under β, then A must be received by attribute B under α.*

Lemma 5 *Let $S_1 \preceq S_2$ by (α, β) and let B be an attribute in S_2 that receives some attribute A under α. If B is received by some attribute in S_1 under β, then B must be received by A under β.*

The following theorem shows when functional dependencies in one schema can be inferred from functional dependencies that hold in a schema that dominates the first schema, and can be used to infer key dependencies in a schema that is dominated by another schema.

Theorem 6 *Let S_1 and S_2 be keyed schemas such that $S_1 \preceq S_2$ by (α, β) for conjunctive query mappings α and β. Suppose that $Y \rightarrow B$ holds in some relation R in schema S_2 for attribute B and attribute set Y. Suppose B is received by*

some attribute A under β, and every attribute in Y is received by an attribute in some set X of attributes in S_1 under β. Then it follows that the functional dependency $X \rightarrow A$ must hold in schema S_1.

The following lemma shows that when $S_1 \preceq S_2$, all of the data values for the key attributes in S_1 are encoded by α in key attributes in S_2, although they also may be mapped extraneously to other non-key attributes.

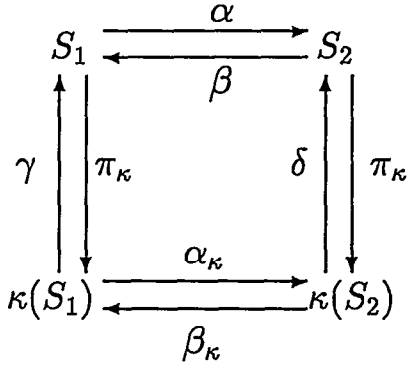
Lemma 7 *If S_1 and S_2 are keyed schemas, and $S_1 \preceq S_2$ by (α, β), then if some non-key attribute B in some relation in S_2 receives some key attribute K in some relation in S_1 under α, and either B is received by K under β, or B is involved in a join or selection condition in the body of some query in β, then:*
a) K is received by some key attribute K' in S_2 under α with K' in the same relation as B; and
b) for any database instance in the range of α, K' and B have the same value in each tuple of the relation containing them.

Definition: If S is a keyed schema, $\kappa(S)$ is the unkeyed schema that can be obtained by deleting all non-key attributes from each relation scheme, and dropping the key dependencies. Thus, for each relation scheme R in S, there is a relation scheme R' in $\kappa(S)$ whose scheme consists only of the key attributes of R. ■

Definition: If S is a keyed schema, and d is a database instance of S, then $\pi_\kappa(d)$ is the database instance of $\kappa(S)$ that corresponds to projecting all of the non-key attributes out of the database instance d. ■

Let S_1 and S_2 be keyed schemas with $S_1 \preceq S_2$ by (α, β) for conjunctive query maps α and β. We would like to construct query maps α_κ and β_κ such that $\kappa(S_1) \preceq \kappa(S_2)$ by $(\alpha_\kappa, \beta_\kappa)$. If A is the collection of attribute types and D the domain of values for the schema S_1, then let $f : A \rightarrow D$ be some fixed, arbitrary map such that $f(T) \in T$ for each $T \in A$. That is, the mapping f is a choice function that associates each attribute type with a constant value belonging to that attribute type.

We will define mappings γ and δ so that α_κ is given by $\pi_\kappa \circ \alpha \circ \gamma$, and β_κ is given by $\pi_\kappa \circ \beta \circ \delta$. The mapping relationships are shown in the following figure.



First we define the mapping $\gamma : i(\kappa(S_1)) \rightarrow i(S_1)$ as follows. The mapping γ is a conjunctive query mapping such that for any relation R in S_1 having n key attributes and m non-key attributes, the query in γ to define R from $\kappa(S_1)$ is given by:

$$R(K_1, K_2, ..., K_n, c_1, c_2, ..., c_m) : -R'(K_1, K_2, ..., K_n),$$

where R' is the relation in $\kappa(S_1)$ corresponding to R but with non-key attributes projected out. We are assuming without loss of generality that the key attributes of relation R are ordered so as to correspond to the leftmost n variables of R, and that the attributes of R' obey the same order. A similar assumption will be made in the definition of δ below. Each c_i is a constant symbol, and $c_i = f(T)$ where T is the type of the attribute corresponding to the position of c_i in R. Note that $\pi_\kappa(\gamma(d_\kappa)) = d_\kappa$, for any database instance d_κ of $\kappa(S_1)$.

Given an arbitrary database instance d of $\kappa(S_1)$, define $\alpha_\kappa(d) = \pi_\kappa(\alpha(\gamma(d)))$. Note that α_κ is a conjunctive query mapping from $\kappa(S_1)$ to $\kappa(S_2)$, since the conjunctive rules for α_κ can be constructed by simple query substitution of each query in γ for the relations appearing in the body of α.

To define the mapping β_κ, we first define the mapping $\delta : i(\kappa(S_2)) \rightarrow i(S_2)$ as follows. The mapping δ is a conjunctive query mapping, such that for any relation R in S_2 having n key attributes and m non-key attributes, the query in δ to define R from $\kappa(S_2)$ is given by:

$$R(K_1, K_2, ..., K_n, t_1, t_2, ..., t_m) : -R'(K_1, K_2, ..., K_n),$$

where R' is the relation in $\kappa(S_2)$ corresponding to R but with non-key attributes projected out. Each t_i either is a constant symbol or variable, and is defined as follows. Let attribute B have type T and let it be the attribute of relation R whose placeholder in the conjunctive query is t_i.
1. If attribute B receives some constant b under α, then t_i is just the constant b.
2. If attribute B receives a non-key attribute N from S_1 under α, then t_i is just the constant $f(T)$.
3. If attribute B receives a key attribute K from S_1 under α, and either B is received by K under β, or B is involved in a join or selection condition in the body of some query in β, then t_i is just K_j, where K_j is the variable in the position of the key attribute K' in R that is guaranteed by Lemma 7 to receive attribute K and have the same value as B in every tuple in R.
4. Otherwise, t_i is just the constant $f(T)$.

Given any database instance e of $\kappa(S_2)$, the mapping β_κ is defined by $\beta_\kappa(e) = \pi_\kappa(\beta(\delta(e)))$. Clearly β_κ is a conjunctive query map since each conjunctive query in δ can be substituted for the appropriate relations in the body of each query in β.

The mapping β_κ must map a database instance in the range of α_κ back to its pre-image under α_κ (recoverying the original database instance). Recall that α_κ creates values for the non-key attributes that are projected out of some instance of $\kappa(S_1)$, and applies the map α to the resulting instance of S_1. This results in an instance of S_2 for which the non-key attributes are then projected out to form the result of the map α_κ.

The map δ above re-creates these non-key attribute values that were projected out. While there typically is not sufficient information in the key attributes to re-create the missing non-key attribute values precisely, the next lemma shows that δ re-creates the values accurately for any non-key attributes that can affect the result of applying the map β to the resulting instance with the values re-created.

Lemma 8 *Let S_1 and S_2 be keyed schemas. If e is an instance of S_2 such that there is some instance d_κ of $\kappa(S_1)$ satisfying $e = \alpha(\gamma(d_\kappa))$, then $\beta(\delta(\pi_\kappa(e)) = \beta(e)$.*

The following theorem is central to establishing the main result of the paper, but also is of independent interest, since it establishes an important property of conjunctive query dominance. In particular, it shows that for one schema to be dominated by a second schema, its key set must be dominated by the key set of the second schema. This result is important because given some keyed schema S, $\kappa(S)$ is an unkeyed schema, so this result allows results concerning unkeyed schemas to be used in reasoning about keyed schemas. For instance, if one wanted to show that some schema S_1 were *not* dominated by some other schema S_2, it would suffice to show that $\kappa(S_1)$ was not dominated by $\kappa(S_2)$, which in turn might be demonstrated using techniques concerning unkeyed schemas.

Theorem 9 *If S_1 and S_2 are keyed schemas, and $S_1 \preceq S_2$, then $\kappa(S_1) \preceq \kappa(S_2)$.*

The following three lemmas demonstrate some additional properties of conjunctive query maps that establish schema dominance, and can be proved in a straightforward manner.

Lemma 10 *Let S_1 and S_2 be keyed schemas, and suppose that $S_1 \preceq S_2$ by (α, β). Then there cannot be two distinct attributes in S_1 that receive the same attribute in S_2 under β.*

Lemma 11 *Let S_1 and S_2 be keyed schemas, and suppose that $S_1 \preceq S_2$ by (α, β). If, for every attribute type T, the number of attributes in S_1 of type T is the same as the number of attributes in S_2 of type T, then every attribute in S_2 is received by some attribute in S_1 under β.*

Lemma 12 *Let S_1 and S_2 be keyed schemas, and suppose that $S_1 \preceq S_2$ by (α, β). If, for every attribute type T, the number of attributes in S_1 of type T is the same as the number of attributes in S_2 of type T, then there cannot be two distinct attributes in S_2 that are received by the same attribute in S_1 under β.*

We now are ready to state the central result of the paper, namely that keyed schemas can be conjunctive query equivalent if and only if they are the same, up to renamings and re-orderings.

Theorem 13 *If S_1 and S_2 are keyed schemas, then $S_1 \equiv S_2$ if and only if S_1 and S_2 are identical up to renaming and re-ordering of relations or attributes.*

4 Conclusions

Schema equivalence is a fundamental property of relational database schemas, and is of critical importance for such problems as database design, data model translation, and multidatabase schema integration. Yet, while the notion of schema equivalence has been known for many years, there are surprisingly few results to characterize the equivalence of relational schemas.

For example, a thorough understanding of database design or multidatabase schema integration would require, at a minimum, characterizations of schema equivalence for various classes of dependencies, such as primary keys, primary keys plus referential integrity constraints, as well as other families of dependencies of interest. However, these problems remain open.

In this work, we have provided a number of results concerning conjuctive query equivalence of relational schemas with primary keys, including having shown that two relational schemas with primary keys are conjunctive query equivalent if and only if they are identical (up to renaming and re-ordering of attributes and relations).

These results make substantial progress toward a characterization of the equivalence of schemas with primary keys (where the full relational algebra is available for schema mappings), and develop techniques that may be applicable to the solution of other problems concerning schema equivalence.

Acknowledgements

The authors would like to thank Rick Hull for the insightful commentary that he provided on an earlier draft of this paper.

References

[1] J. Albert, Y. Ioannidis, and R. Ramakrishnan. Conjunctive query equivalence of keyed relational schemas. Technical Report Dept. of Computer Science, TR-1341, University of Wisconsin-Madison, 1997.

[2] A.K. Arora and C.R. Carlson. The information preserving properties of relational database transformations. In *Proc. of the Int'l VLDB Conf.*, 1979.

[3] P. Atzeni, G. Aussiello, C. Batini, and M. Moscarini. Inclusion and equivalence between relational database schemes. *Theoretical Computer Science*, 19:267–285, 1982.

[4] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[5] C. Beeri, P. A. Bernstein, and N. Goodman. A sophisticate's introduction to database normalization theory. In *Proc. of the Int'l VLDB Conf.*, pages 113–124, 1978.

[6] C. Beeri, A.O. Mendelzon, Y. Sagiv, and J.D. Ullman. Equivalence of relational database schemes. *SIAM Journal on Computing*, 10(2):352–370, May 1981.

[7] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 1970.

[8] E.F. Codd. Further normalization of the data base relational model. In R. Rustin, editor, *Data Base Systems*, pages 33–64. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[9] R. Hull. Relative information capacity of simple relational database schemata. In *Proc. of the ACM Conf. on Principles of Database Systems*, pages 97–109, Waterloo, April 1984.

[10] R. Hull. Relative information capacity of simple relational database schemata. *SIAM Journal on Computing*, 15(3):846–886, August 1986.

[11] T.-W. Ling, F.W. Tompa, and T. Kameda. An improved third normal form for relational databases. *ACM TODS*, 6(2), 1981.

[12] D. Maier, A.O. Mendelzon, F. Sadri, and J.D. Ullman. Adequacy of decompositions of relational databases. *J. of Computer and System Sciences*, 21(3):368–379, December 1980.

[13] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In *Proc. of the Int'l VLDB Conf.*, Dublin, September 1993.

[14] J. Rissanen. On equivalences of database schemes. In *Proc. of the ACM Conf. on Principles of Database Systems*, Los Angeles, CA, March 1982.

[15] A. Rosenthal and D. Reiner. Theoretically sound transformations for practical database design. In Salvatore T. March, editor, *Proc. of the Int'l Conf. on the Entity-Relationship Approach*, pages 115–131, NYC, NY, 1987. Elsevier Science Publishers B. V. (North-Holland).

[16] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[17] C. Zaniolo. A new normal form for the design of relational database schemata. *ACM TODS*, 7(3):489–499, 1982.