Conjunctive Query Equivalence of Keyed Relational Schemas

Joseph Albert
Yannis Ioannidis
Raghu Ramakrishnan

Technical Report #1341

May 1997

Conjunctive Query Equivalence of Keyed Relational Schemas

Joseph Albert, Yannis Ioannidis, Raghu Ramakrishnan

Computer Sciences Dept.

University of Wisconsin

1210 W. Dayton St.

Madison, WI 53706

jalbert@acm.org, {yannis,raghu}@cs.wisc.edu

An abbreviated version of this paper is to appear in PODS'97

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 [2], 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 [8, 9], 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, [3, 15], 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, consider the following two relational schemas with key dependencies and referential integrity constraints, and suppose one wants to integrate them. Key attributes are marked with an asterisk, and referential integrity constraints are shown using standard inclusion dependency notation.

employee(ss*, eName, salary, depId)	empl(ssn*, ename, sal, dep, yrsExp)
department(deptId*, deptName, mgr)	dept(departId*, dName, manager)
salespeople(ss*, yearsExp)	
employee[depId] $\subseteq$ department[deptId]	empl[dep] $\subseteq$ dept[departId]
salespeople[ss] $\subseteq$ employee[ss]	
employee[ss] $\subseteq$ salespeople[ss]	
Schema 1	**Schema 2**

Suppose 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 is straightforward to show 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)	empl(ssn*, ename, sal, dep, yrsExp)
department(deptId*, deptName, mgr)	dept(departId*, dName, manager)
salespeople(ss*)	
employee[depId] $\subseteq$ department[deptId]	empl[dep] $\subseteq$ dept[departId]
salespeople[ss] $\subseteq$ employee[ss]	
employee[ss] $\subseteq$ salespeople[ss]	
Schema 1′	**Schema 2**

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 [4, 7, 10, 16] 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 [7], 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 [5, 11]. 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 [5]. Similar notions of equivalence were defined in [1, 13].

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 [12, 14]. 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 [2], 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 on schema equivalence [9]. 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 [9] 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 non-trivial equivalence-preserving transformations for schemas with only primary keys, which suggests that other dependencies are important for transforming schemas in meaningful ways. Indeed, the example above shows that when both primary key dependencies and referential integrity constraints are available, there are non-trivial equivalence-preserving schema transformations.

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.

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 [6]. A *domain* is a countably infinite set of atomic values. A collection of *attribute types* over some domain $\mathcal{D}$ is a finite collection of disjoint subsets of $\mathcal{D}$. 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 two queries $q : i(S) \to i(V)$ and $q' : i(S) \to i(V)$ that have the same type, we say that q is *contained in* q', written $q \sqsubseteq q'$, if for every $d \in i(S)$, $q(d) \subseteq q'(d)$. ∎

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, $\mathcal{L}$, 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 $\mathcal{L}$, 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 $\alpha : i(S_1) \to i(S_2)$. 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 $\mathcal{L}$ be a query language. Then we say that S_2 *$\mathcal{L}$-dominates* S_1, written $S_1 \preceq_{\mathcal{L}} S_2$, if there are valid query maps $\alpha : i(S_1) \to i(S_2)$ and $\beta : i(S_2) \to i(S_1)$ 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 $S_1 \preceq_{\mathcal{L}} S_2$ by (α, β). ∎

Definition: If $S_1 \preceq_{\mathcal{L}} S_2$ and $S_2 \preceq_{\mathcal{L}} S_1$, then we say that the two schemas are *$\mathcal{L}$-equivalent*, written $S_1 \equiv_{\mathcal{L}} S_2$. ∎

The notions of L-dominance and L-equivalence were introduced in [2]. We sometimes write $S_1 \preceq S_2$, or $S_1 \equiv S_2$, when the particular language $\mathcal{L}$ is clear from the context. The following result is proved in [9].

Theorem (Hull 1986) *If $\mathcal{L}$ is the relational algebra, S_1 and S_2 are schemas with no dependencies, then $S_1 \equiv_{\mathcal{L}} S_2$, if and only if S_1 and S_2 are identical up to renaming and re-ordering 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".

Note that equality of variables in a conjunctive query induces a natural equivalence relation on the variables. In particular, V_1 is equivalent to V_2 either if $V_1 = V_2$ appears in the equality-list, or can be inferred by reflexivity, symmetry, or transitivity. We call the equivalence classes generated by this equivalence relation the *equality classes* of variables. In other words, V_1 and V_2 belong to the same equality class if $V_1 = V_2$ can be inferred from the equality predicates in the equality-list.

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

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.*

Proof: Given some ij-saturated query q, construct a query q' as follows:

1. eliminate all (identity) join conditions from the body of q;

2. eliminate all duplicate occurrences of any relation in the body of q.

3. replace any variable X in the head of the query that no longer occurs in the body with a variable Y that does still occur in the body, such that $X = Y$ is an identity join condition that can be inferred from the join conditions specified in the ij-saturated query. Such a Y always exists because q is ij-saturated.

Clearly the resulting query q' is a product query having the same relations in its body as q. We claim that $q \cong q'$. By construction, q and q' both have the same type, and are defined over the same schema. Let this schema be S and let d be an arbitrary instance of S. To show that $q \sqsubseteq q'$, let $\tau \in q(d)$. Then there must be tuples in each relation occurring in the body of q that are used in inferring τ. But then, since each relation in the product query q' occurs in the body of q, and q' has no selections of joins, τ must be a member of $q'(d)$, as required. To show that $q' \sqsubseteq q$, let $\tau' \in q'(d)$. Then there must be tuples in each relation occurring in the body of q' that are used to infer τ'. But since any tuple of a relation always satisifies an identity join, these tuples will satisfy the identity join conditions, and τ' will be a member of $q(d)$, as required. With this claim established, the lemma follows. ■

Lemma 2 *Given some conjunctive query q defined over 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 $\hat{q}$.

Proof: Let q be some conjunctive query as in the statement of the lemma. Then let q' be the ij-saturated query that results from adding to q all the valid identity join conditions missing from q. Clearly, $q' \sqsubseteq q$. If we take $\hat{q}$ to be the product query that Lemma 1 guarantees to be equivalent to q', then $\hat{q}$ is a product query satisfying condition (a) and (d) of the lemma. Condition (b) follows immediately from the fact that $\hat{q} \sqsubseteq q$, since if some functional dependency fails for some database state d, then there is a pair of tuples in $\hat{q}(d)$ that violate the dependency, and these tuples are in $q(d)$ also. For condition (c), let $\hat{q}$ be empty when evaluated over some database instance d. Then, since $\hat{q}$ is a cross product of the relations occurring in q (along with a projection to the attributes occurring in the head of q), it follows that one of these relations is empty in the database instance d. But then, the query q is empty when evaluated over d as well, establishing condition (c). ∎

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.*

Proof: Let A be an arbitrary attribute in schema S_1. Let d be some attribute-specific instance of S_1 in which there is some value a for A in the database instance such that a is not among any constants in any of the queries in the maps α or β. Since $\beta \circ \alpha$ is the identity map on $i(S_1)$, $\beta(\alpha(d)) = d$. Thus, the value a occurs as a value of attribute A in $\beta(\alpha(d))$, and since a does not occur among any of the constants in the queries in α or β, A must have received some attribute B of S_2 under β. Moreover, B must contain the value a in the instance $\alpha(d)$. Since d is an attribute-specific instance, and a doesn't occur in any query constants, B must receive A under α, establishing the lemma. ∎

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.*

Proof: Suppose not. Then there is some attribute A in S_1 that receives attribute B under β, but A is not received by B under α. Let d be an attribute-specific database instance having some value a for attribute A that is not among any of the query constants in α or β. Then, since A is not received by B under α, the value a will not be among the values found for attribute B in the database instance $\alpha(d)$. Thus, since A receives attribute B under β, the value a cannot be a value for attribute A in the database instance $\beta(\alpha(d))$, contradicting that $\beta \circ \alpha$ is the identity map on $i(S_1)$. ■

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.*

Proof: Let S_1, S_2, A, and B be as in the statement of the lemma, and suppose to the contrary that B is not received by A under β. Then B is received by some other attribute A' of S_1 under β. By Lemma 4, A' must be received by B under α. But since A is received by B under α, it must be that there is a join condition between A and A' in the query defining the relation of S_2 that contains B.

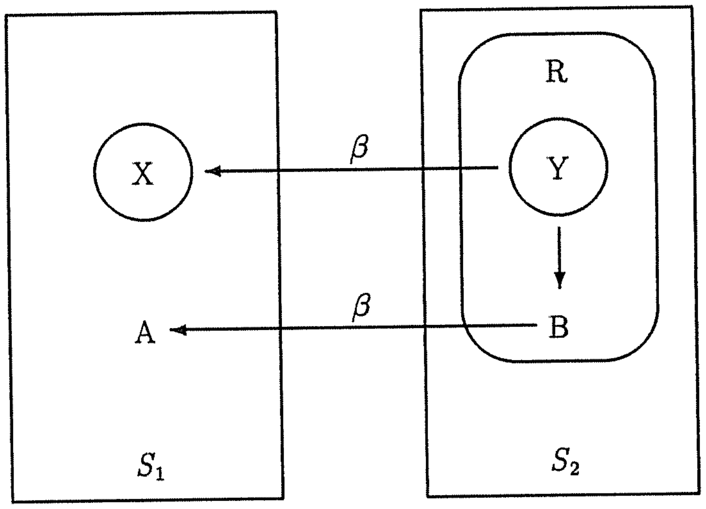
Let d be an attribute-specific database instance such that the relation(s) containing A and A' are non-empty, and there is some value a for attribute A that is not among any of the query constants in α or β. Then, since there are no values in common between A and A' in d, the join condition between A and A' must fail, whence the relation containing B must be empty in $\alpha(d)$. Since A received B under β, the relation containing A in $\beta(\alpha(d))$ must also be empty. But a is a value for attribute A in d, so the relation containing attribute A is not empty in d, contradicting that $\beta \circ \alpha$ is the identity map on $i(S_1)$. ■

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 $Y \to B$ 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 $X \to A$ must hold in schema S_1.

Proof: Suppose not. Then there exist keyed schemas S_1 and S_2 such that $S_1 \preceq S_2$ by (α, β) for conjunctive query mappings α and β, and a relation R in S_2 with Y a superkey of R, and B an attribute of R. Further, there is a set X of attributes such that every attribute in Y is received by some attribute in X under β, and an attribute A that receives B, but the dependency $X \to A$ fails in S_1. These relationships are shown in the following figure.



By Lemma 4, every attribute in the set Y receives some attribute in X under α, and B receives attribute A under α. Let d be an attribute-specific database instance of schema S_1 such that none of the values in the database d appear as constants in the queries in α or β, all the relations in d are non-empty, and the dependency $X \to A$ fails in d. Let q be the query in α that defines relation R under α. Note that the instance of R in $\alpha(d)$ must be non-empty, for otherwise, the relations in S_1 containing either attribute A or any attribute in X would be empty in $\beta(\alpha(d)) = d$, since each attribute in Y is received by an attribute in X under β, and B is received by A under β, a contradiction.

Since the dependency $X \to A$ fails, and all the attributes in X are received by an attribute in

Y, and B receives A, there must be some join or selection conditions that filter out the necessary tuples so that $Y \rightarrow B$ holds in R. If there is a selection of an attribute as a constant value, this selection will fail because no query constants are in the database instance d, whence R will be empty in $\alpha(d)$, a contradiction. There also can be no column selection (that is, a selection that selects tuples that agree on two columns of the same relation) since R would again be empty in $\alpha(d)$ on account of d being an attribute-specific instance. Similarly, any join condition that references different attributes (either different attributes of the same relation or different attributes of different relations) will fail because d is an attribute-specific instance, whence R would be empty in $\alpha(d)$, again a contradiction.

Thus, q has no selection conditions, and the only join conditions are identity joins. By Lemma 2, there is another query $\hat{q}$ such that all of the relations that occur in the body of q also occur in the body of $\hat{q}$, the dependency $Y \rightarrow B$ holds in the relation resulting from evaluating $\hat{q}$ over database d, and $\hat{q}$ has no selection or join conditions. Since, $\hat{q}$ can only contain cross-products and projection operations, this is a contradiction, since a functional dependency that does not hold over any relation in a set of relations (in this case, the relations occurring in the body of $\hat{q}$) cannot hold in their cross-product, and the theorem is established. ∎

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.

Proof: Let $S_1 \preceq S_2$ by (α, β) and let K and B be as in the statement of the lemma. Let R_1 be the relation in S_1 containing K, and let R_2 be the relation in S_2 containing B. Let P be a relation in S_1 that is defined by a query in β in which B is used either in a join or selection condition, or B is received by K (P will be the same relation as R_1 if B is received by K under β). Let $\{k_1, k_2\}$ be a set of two unique values belonging to the attribute type of attribute K such that neither k_1 nor k_2 occur as constants in the queries in α or β. Also define a function g on the domain of schema S_1 so that $g(k_1) = k_2$ and $g(k_2) = k_1$, but $g(x) = x$ when $x \neq k_i$ for $i = 1,2$.

Now let d be an attribute specific instance of S_1 such that all relations of the schema are non-empty, none of the values in d occur as constants in any of the queries in α or β, each attribute other than K has only a single value stored in d, but there are exactly two values, k_1 and k_2 stored for attribute K. Thus, the instance of relation R_1 in d has two tuples, and all other relation instances in d have a single tuple. Let r_2 be the instance of R_2 in $\alpha(d)$.

First we claim that r_2 is non-empty. In proof, suppose not. Then, since the relation P is defined by a query in β in which B either is received by an attribute in P or B is involved in a join or selection condition, it must be that the instance of P in $\beta(\alpha(d))$ is empty. This is a contradiction because all relations in d are non-empty and $d = \beta(\alpha(d))$, establishing the claim.

Next note that there can be no selection or join conditions in the query in α that defines R_2 except for selection or join conditions that equate an attribute to itself. This is because d is an attribute specific instance containing no query constants, so if there were a selection condition equating an attribute to a constant or a selection or join condition equating two different attributes in the query, r_2 would be empty, a contradiction.

Since r_2 is non-empty, there is some tuple $\tau \in r_2$. Because the instance r_2 is defined by a query in α, there is a derivation of the tuple τ. Let V be the variable in the position of attribute B in the head of the query, and suppose that the body of the query has n relations in it (the occurrences of relations in the body need not be distinct). Let these relations be called Q_i for $i = 1, 2, \dots, n$. Note that R_1 occurs one or more times among the Q_i. Then there exist n tuples, $\tau_1, \tau_2, \dots, \tau_n$ that are used to derive τ in r_2, and for each i, $\tau_i \in Q_i$. Now, define tuples τ_i' for $i = 1, 2, \dots, n$ as follows.

$\tau_i'.A = g(\tau_i.A)$ if Q_i is an occurrence of relation R_1, and has a variable in the same equality class as V in the position of attribute K

$\tau_i'.A = \tau_i.A$ otherwise.

Thus, each $\tau_i' \in Q_i$, and since the only join or selection conditions in the query are ones that equate an attribute with itself, there must be a derivation of some tuple $\tau' \in r_2$ using the tuples τ_i'.

By construction, $\tau'.B = g(\tau.B)$, so that $\tau.B \neq \tau'.B$. Since B is a non-key attribute, and two tuples which disagree on a non-key attribute must also disagree on some key attribute, there must be at least one key attribute of R_2 for which τ and τ' disagree. Let this key attribute be K'. Then it must be that $\tau'.K' = g(\tau.K')$, and by construction, the head of the query that defines R_2 has either variable V, or some other variable in the same equality class as V in the position of attribute K'. But this means that K' receives attribute K under α, establishing part (a) of the lemma, and since the variables in the head of the query in the positions for attributes K' and B are in the same equality class, K' and B must have the same value in every tuple in any instance of R_2 in the range of α, establishing part (b) of the lemma. ■

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 $\mathcal{A}$ is the collection of attribute types and $\mathcal{D}$ the domain of values for the schema S_1, then let $f : \mathcal{A} \rightarrow \mathcal{D}$ be some fixed, arbitrary map such that $f(T) \in T$ for each $T \in \mathcal{A}$. 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.

$$\begin{tikzcd}
S_1 \arrow[r, shift left, "\alpha"] \arrow[d, shift left, "\pi_\kappa"] & S_2 \arrow[l, shift left, "\beta"] \arrow[d, shift left, "\pi_\kappa"] \\
\kappa(S_1) \arrow[u, shift left, "\gamma"] \arrow[r, shift left, "\alpha_\kappa"] & \kappa(S_2) \arrow[l, shift left, "\beta_\kappa"] \arrow[u, shift left, "\delta"]
\end{tikzcd}$$

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 following 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 a database instance of S_2 such that there is some database instance d_κ of $\kappa(S_1)$ satisfying $e = \alpha(\gamma(d_\kappa))$, then $\beta(\delta(\pi_\kappa(e)) = \beta(e)$.*

Proof: First note that $\pi_\kappa(\delta(\pi_\kappa(e))) = \pi_\kappa(e)$, because δ creates values for the non-key attributes of e that are missing from $\pi_\kappa(e)$, and the outer application of π_κ just projects them back out again. Thus, $\delta(\pi_\kappa(e))$ and e have the same number of tuples in each relation, with identical key values. We now claim that if there are tuples τ in e and τ' in $\delta(\pi_\kappa(e))$ in the corresponding instances of the same relation that agree on key values, but disagree on non-key attributes, then those tuples disagree on non-key attributes that are neither received by any attributes in S_1 under β, nor participate in any selection or join conditions.

Suppose to the contrary, that a pair of tuples, τ in e and τ' in $\delta(\pi_\kappa(e))$ exist and agree on their key values, but disagree on some arbitrary non-key attribute B that either is received by some attribute in S_1 under β, or participates in a join or selection condition in a query in β. Let B have type T. By hypothesis there is some d_κ such that $e = \alpha(\gamma(d_\kappa))$. If attribute B receives some constant b under α, then attribute B of τ has value b from the query mapping, and attribute B of τ' has value b by construction of δ, a contradiction. If attribute B receives a non-key attribute under α, then attribute B of τ has value $f(T)$ by construction of γ, and attribute B of τ' has value $f(T)$ by construction of δ, again a contradiction. If attribute B receives a key attribute K under α, then by Lemma 7, there is a key attribute K' in R such that K' has the same value as attribute B in the tuple τ. By the construction of δ, attribute B also has the same value as K' in τ', again a contradiction, establishing the claim.

We have shown that $\delta(\pi_\kappa(e))$ and e have the same number of tuples in each relation, with identical key values, and either the two instances in fact are equal, or corresponding tuples that agree on key values may disagree only on non-key attributes that are not received by any attribute in S_1 under β and which do not participate in any join or selection conditions in any queries in β. But then, $\beta(\delta(\pi_\kappa(e))) = \beta(e)$, and the lemma follows. ■

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 $\kappa(S_1) \preceq \kappa(S_2)$.*

Proof: We show that $\kappa(S_1) \preceq \kappa(S_2)$ by $(\alpha_\kappa, \beta_\kappa)$. It suffices to show that $\beta_\kappa \circ \alpha_\kappa$ is the identity map on $i(\kappa(S_1))$. Suppose not. Then there is some database instance d_κ of $\kappa(S_1)$ such that $\beta_\kappa(\alpha_\kappa(d_\kappa)) \neq d_\kappa$. Noting that $d_\kappa = \pi_\kappa(\gamma(d_\kappa))$ by the definition of γ, and substituting the definitions of β_κ and α_κ, we have that $\pi_\kappa(\beta(\delta(\pi_\kappa(\alpha(\gamma(d_\kappa)))))) \neq \pi_\kappa(\gamma(d_\kappa))$. Since two database instances that disagree on their keys cannot be the same instance, we infer that $\beta(\delta(\pi_\kappa(\alpha(\gamma(d_\kappa))))) \neq \gamma(d_\kappa)$. Now, if we let $d = \gamma(d_\kappa)$, we have $\beta(\delta(\pi_\kappa(\alpha(d)))) \neq d$. But, letting $e = \alpha(d)$ Lemma 8 ensures that $\beta(\delta(\pi_\kappa(e))) = \beta(e)$, that is $\beta(\delta(\pi_\kappa(\alpha(d)))) = \beta(\alpha(d))$, whence, $\beta(\alpha(d)) \neq d$. This contradicts that $\beta \circ \alpha$ is the identity map on $i(S_1)$, and the theorem follows. ∎

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 such 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.*

Proof: Let S_1 and S_2 be as in the statement of the lemma, and suppose to the contrary that there is some attribute B in relation R in S_2 that is received under β by two distinct attributes A and A' in S_1. By Lemma 4, B receives both A and A' under α. Let d be an attribute-specific instance of S_1 such that the relation(s) containing A and A' is/are non-empty. Since A and A' are distinct attributes, they have no values in common in d. But B receives both A and A' under α, which means that there is a join condition between A and A' in the query defining R But then R must be empty, whence the relation containing A must be empty in the instance $\beta(\alpha(d))$, since A receives B under β. This is a contradiction, establishing the lemma. ■

Lemma 11 *Let S_1 and S_2 be keyed schemas such 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.*

Proof: Suppose not. Then there is some attribute C in S_2 which is not received by any attribute in S_1 under β. By Lemma 3, for every attribute A in S_1 there is some attribute B such that A is received by B under α and B is received by A under β. Since, in a query mapping, any attribute of some type T that is received by another attribute, is always received by an attribute with the same type T, and since there are the same numbers of attributes of each type in each schema, it follows that if T_C is the type of attribute C, there are fewer attributes of type T_C in S_2 that are received by attributes of type T_C in S_1 than there are attributes of type T_C in S_1. It follows that there must be two distinct attributes A and A' in S_1 such that the same attribute B is received by both A and A' under β, contradicting Lemma 10. ■

Lemma 12 *Let S_1 and S_2 be keyed schemas such 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.*

Proof: Let S_1 and S_2 be as in the statement of the lemma, and suppose to the contrary that some attribute A in S_1 receives two distinct attributes from S_2 under β. Then, since Lemma 11 guarantees that every attribute in S_2 is received by some attribute in S_1, and since the number of attributes of each type is the same in both schemas, it follows by simple counting that there must be some other attribute in S_2 which is received by two or more distinct attributes in S_1, contradicting Lemma 10. ∎

We now are ready to prove 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.*

Proof: One direction is trivial, that is, if two schemas are identical up to renaming and re-ordering of relations and attributes, then clearly they are equivalent. For the other direction, let S_1 and S_2 be keyed schemas with $S_1 \equiv S_2$. Then $S_1 \preceq S_2$ and $S_2 \preceq S_1$, and by Theorem 9, $\kappa(S_1) \preceq \kappa(S_2)$ and $\kappa(S_2) \preceq \kappa(S_1)$. Thus, $\kappa(S_1) \equiv \kappa(S_2)$. Note that all of the attributes of an unkeyed schema (schema with no depenencies) always implicitly form a key, so that $\kappa(S_1)$ and $\kappa(S_2)$ can be viewed as unkeyed schemas. The fact that the set of attributes of any relation in either schema forms a key is implicit. Thus, from the result of Hull [9] mentioned above, that states that equivalent unkeyed schemas are identical up to renaming of attributes and relations and re-ordering of attributes, we conclude that $\kappa(S_1)$ and $\kappa(S_2)$ must be identical, up to renaming of attributes and relations and re-ordering of attributes.

But this means that S_1 has the same number of relations as S_2, and, for every relation R_1 in S_1, if R_1 has key K, where K is a set of attributes, then there is a corresponding relation R_2 in S_2 with the same key, up to renaming and re-ordering of attributes. The same is true for S_2: corresponding to every relation in S_2 there is a corresponding relation in S_1 with the same key (up to renaming and re-ordering of attributes).

It remains to be shown that the two schemas also agree on non-key attributes. First, we claim that the number of occurrences of any attribute type among the non-key attributes in one of the schemas must be the same as the number of occurrences of the same attribute type among non-key attributes in the other schema. Without loss of generality, suppose to the contrary, that there is some attribute type A in S_1 that occurs a greater number of times among non-key attributes in S_1

than in S_2. Let (α, β) be the pair of conjunctive query maps establishing $S_1 \preceq S_2$.

By Lemma 3, each non-key attribute of type A in S_1 must be received by some attribute in S_2 under α. Because there are more non-key attributes of type A in S_1 than in S_2, it must be that there is some (key or non-key) attribute A_0 with type A in S_2 that receives more than one attribute with type A in S_1, under the map α. Call the relation in which A_0 occurs R_0, and let $A_1, ..., A_n$ be the attributes of type A in S_1 that are received by A_0 under α. Note that $A_1, ..., A_n$ must be bound by a join condition in the conjunctive query in α that defines R_0.

Let d be an attribute-specific instance of S_1 that includes some value $a \in \pi_{A_1}(d)$ that is not among the constants used in any of the queries in α or β. Then a is not found in the instance $\alpha(d)$, since d is an attribute-specific instance, whence the join condition which equates the attributes $A_1, ..., A_n$ must fail. However, β must map the instance $\alpha(d)$ back to d, since $\beta \circ \alpha$ is the identity map on $i(S_1)$. But this implies that A_1 must receive some attribute with type A in S_2 under β, and a must be a value of that attribute in the instance $\alpha(d)$. This is a contradiction, which establishes the claim.

We have established that the two schemas have the same number of relations, with corresponding key sets, and the same number of occurrences of each non-key attribute type. It remains only to be shown that the non-key attributes cannot be re-arranged so that the S_1 and S_2 are not identical.

Let (α, β) be the pair of conjunctive-query maps establishing $S_1 \preceq S_2$. Let $\{R_1, R_2, ..., R_n\}$ be the set of all relations in schema S_2. For each i, let K_i be the set of attributes comprising the key of R_i, and let N_i be the remaining attributes of R_i. The functional dependency, $K_i \rightarrow N_i$, which is just the key dependency for R_i, holds for each i.

Now, for each i, let $\hat{K}_i$ be the set of all attributes that receive some attribute in K_i under β, and let $\hat{N}_i$ be the set of all attributes that receive some attribute in N_i under β. The sets $(\hat{K}_i, \hat{N}_i)$ must be pairwise disjoint, since otherwise, there would an attribute in S_1 that receives two different attributes from S_2 under β, contradicting Lemma 10.

By Lemma 11 every attribute of each R_i is received by some attribute in S_1 under β. Thus, by Theorem 6, the functional dependencies $\hat{K}_i \rightarrow \hat{N}_i$ must hold in schema S_1 for each i. Since the only functional dependencies holding on S_1 are key dependencies, it follows that each $\hat{K}_i$ is a superkey of some relation, P_i in S_1, and that there are n such relations P_i.

For each i, and for every attribute type T, $\hat{N}_i$ must contain the same number attributes of type T as N_i. If not, for some i there either must be two attributes in N_i that are received by the same attribute in $\hat{N}_i$ under β, contradicting Lemma 12, or there must be an attribute in N_i that is received by two different attributes in $\hat{N}_i$, contradicting Lemma 10. The same argument can be used to show that $\hat{K}_i$ has the same number of occurrences of any attribute type among its attributes as K_i, for each i.

Thus, every non-key attribute in S_1 is an element of one of the $\hat{N}_i$, since for every attribute type T, there are the same number of non-key attributes of type T in both schemas. This also means that each $\hat{K}_i$ is in fact the key of P_i, since if some $\hat{K}_i$ were a proper superkey, there would be additional non-key attributes included in $\hat{K}_i$. Finally, because the only functional dependencies holding in S_1 are key dependencies, it follows that for each i, all of the attributes in $\hat{N}_i$ occur as non-key attributes in P_i, and the theorem follows. ∎

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

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

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

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

[14] 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).

[15] 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.

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