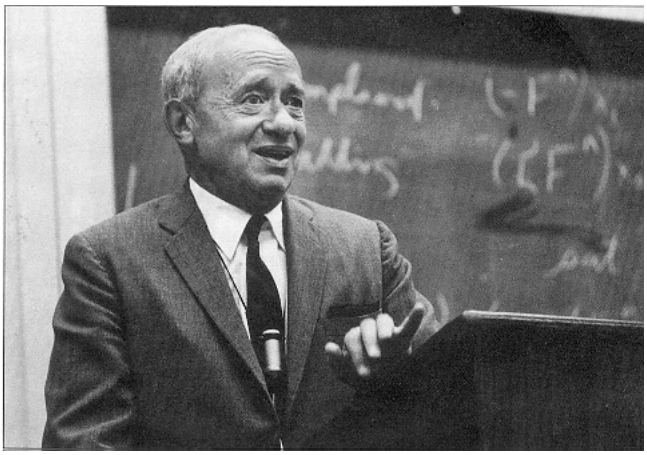
Applications of Alfred Tarski's Ideas in Database Theory*

Jan Van den Bussche

University of Limburg (LUC)
B-3590 Diepenbeek, Belgium

Abstract. Many ideas of Alfred Tarski—one of the founders of modern logic—find application in database theory. We survey some of them with no attempt at comprehensiveness. Topics discussed include the genericity of database queries; the relational algebra, the Tarskian definition of truth for the relational calculus, and cylindric algebras; relation algebras and computationally complete query languages; real polynomial constraint databases; and geometrical query languages.



Alfred Tarski, 1901–1983

To Dirk

1 Introduction

Alfred Tarski was one of the founders of modern logic, and a philosopher and mathematician of extraordinary breadth and depth. It is therefore not surprising that many of his ideas find application also in database theory, a field of

* I thank Janos Makowsky for having proposed me to write and present this paper. I owe a lot to Dirk Van Gucht, database theorist and Tarski fan, for having taught me so much during the past ten years, about database theory as well as about Tarski.

theoretical computer science where logic plays an important role. In this year of Tarski's hundredth anniversary, it seems desirable to survey some of these applications. We will not attempt to be comprehensive, however.

2 Relational database queries and logical notions

To begin our discussion, we fix some infinite universe $\mathbb{U}$ of atomic data elements. In a set-theoretic formalization they would play the role of "urelemente".

In the relational approach to database management, introduced by Codd [19], we define a *database schema* $\mathcal{S}$ as a finite set of relation names, each with an associated arity. A *relational database* $\mathbf{D}$ *with schema* $\mathcal{S}$ then assigns to each $R \in \mathcal{S}$ a *finite* n-ary relation $R^{\mathbf{D}} \subseteq \mathbb{U}^n$, where n is the arity of R.

We store information in a database so that we can retrieve it later. The answer of a *query* to a relational database is again a relation: this is very convenient as it allows us to compose queries, or to store answers of queries as additional relations in the database. The answers 'yes' or 'no' are represented by the nonempty and the empty relation of arity 0, respectively. For example, let $\mathcal{S} = \{R\}$ where the arity of R is 2. So, databases over $\mathcal{S}$ can be identified with finite binary relations on $\mathbb{U}$. Some examples of queries we might want to ask such databases are:

1. Is there an identical pair in R? (Answer: nullary relation.)
2. What are the elements occurring in the left column of R, but not in the right one? (Answer: unary relation.)
3. What are the 5-tuples $(x_1, x_2, x_3, x_4, x_5)$ such that (x_1, x_2), (x_2, x_3), (x_3, x_4), and (x_4, x_5) are all in R? (Answer: five-ary relation.)
4. What is the transitive closure of R? (Answer: binary relation.)
5. Which pairs of elements (x_1, x_2) are such that the sets $\{y \mid (x_1, y) \in R\}$ and $\{y \mid (x_2, y) \in R\}$ are nonempty and have the same cardinality? (Answer: binary relation.)
6. Is the cardinality of R a prime number? (Answer: nullary relation.)

At the most general level, we could formally define an *n-ary query on* $\mathcal{S}$ as a function q from databases $\mathbf{D}$ with schema $\mathcal{S}$ to finite relations $q(\mathbf{D}) \subseteq \mathbb{U}^n$. However, this definition is much too liberal. To illustrate this, let us take the same example schema $\mathcal{S}$ as above, and a, b and c three different elements of $\mathbb{U}$. Now consider the database $\mathbf{D}_0$ where $R^{\mathbf{D}_0} = \{(a, b), (a, c)\}$, and a unary query q_0 on $\mathcal{S}$ that maps $\mathbf{D}_0$ to the singleton $\{b\}$. This query does not seem "logical:" given the information provided by $\mathbf{D}_0$, there is no reason to favor b above c, as b and c are completely symmetric in $\mathbf{D}_0$. Note that none of the example queries given above has this "unlogical" nature: each of them can be answered purely on the basis of the information present in the database, and this is how it should be.

How can we formalize this intuitive notion of a "logical" query? Tarski has shown us how [60]. Consider the following cumulative hierarchy of universes $\mathbb{U}_0$, $\mathbb{U}_1$, $\mathbb{U}_2$, and so on, and their union $\mathbb{U}^*$:

$$\mathbb{U}_0 := \mathbb{U}, \quad \mathbb{U}_{n+1} := \mathbb{U} \cup \mathcal{P}(\mathbb{U}_n), \quad \mathbb{U}^* := \bigcup_n \mathbb{U}_n$$

Here $\mathcal{P}$ denotes the powerset operation. Most mathematical objects we want to construct on top of $\mathbb{U}$ can be formalized as elements of $\mathbb{U}^*$. For example, by the ordered pair construction $(x, y) := \{\{x\}, \{x, y\}\}$, ordered pairs of elements of $\mathbb{U}$ live in $\mathbb{U}_2$, and thus binary relations on $\mathbb{U}$ live in $\mathbb{U}_3$. Database queries also live in $\mathbb{U}^*$. For example, a unary query on binary relations, being itself a binary relation from binary relations to unary relations, lives in $\mathbb{U}_6$. More generally, any notion involving objects living in $\mathbb{U}^*$, such as a property of such objects, or a relation among such objects, can itself be formalized as an object living in $\mathbb{U}^*$.

Tarski now calls such a notion *logical* if it is left invariant by all possible permutations of $\mathbb{U}$. So, $P \in \mathbb{U}^*$ is logical if $f(P) = P$ for every permutation f of $\mathbb{U}$, where permutations of $\mathbb{U}$ are extended to $\mathbb{U}^*$ in the canonical manner. For example, no singleton $\{a\}$ with $a \in \mathbb{U}$ is logical: there is no purely logical reason to single out any particular atomic data element. The whole set $\mathbb{U}$ is logical, and so is the empty set. The identity relation $\{(x, x) \mid x \in \mathbb{U}\}$ is logical, and so is the diversity relation $\{(x, y) \mid x, y \in \mathbb{U},\ x \neq y\}$. The higher we go up in the cumulative hierarchy, the more complex logical notions we find. In particular, queries may or may not be logical. For example, the "unlogical" query q_0 from above is indeed not logical in the sense of Tarski. For if it were, it would have to be invariant under the transposition $t = (b\ c)$ and thus would have to contain not only the pair $(\mathbf{D}_0, \{b\})$ but also the pair $(t(\mathbf{D}_0), \{t(b)\}) = (\mathbf{D}_0, \{c\})$, which is impossible as q_0 is a function. On the other hand, all the earlier example queries 1–6 are readily seen to be logical.

Unaware of this,[1] Chandra and Harel [16], and independently Aho and Ullman [6], based on practical considerations, pointed out the following "universality property" (as A&U called it), or "consistency criterion" (as C&H called it) for database queries. It is now generally known as the *genericity* of database queries,[2] and says that for any query q, databases $\mathbf{D}_1$ and $\mathbf{D}_2$, and permutation f of $\mathbb{U}$, if $f(\mathbf{D}_1) = \mathbf{D}_2$, then also $f(q(\mathbf{D}_1)) = q(\mathbf{D}_2)$. Clearly, a query is generic in this sense if and only if it is logical. So, interestingly, Tarski's definition of logical notion somehow inevitably turned out to hold for database queries.

We note that Tarski saw his definition in the context of Klein's Erlanger Programm [66] in which different geometries are identified with the groups of transformations under which the fundamental notions of the geometry in question are left invariant. For example, topology could be defined as the geometry whose notions are left invariant by continuous transformations (homeomorphisms). According to Tarski then, logic is the "geometry" whose notions are left invariant by *all* transformations.

---

[1] The paper cited [60] was only published in 1986, but is based on a talk delivered by Tarski twenty years earlier.

[2] The specific term 'genericity' was first used for this purpose by Hull and Yap [36] and caught on.

3 The relational algebra and first-order queries

A fundamental insight of Codd was that many complex operations performed on data files can be expressed as combinations of five basic operators on relations:

1. *union* of two relations of the same arity;
2. *difference* between two relations of the same arity;
3. *cartesian product:* if r is of arity n and s is of arity m, then $r \times s$ equals $\{(x_1, \ldots, x_n, y_1, \ldots, y_m) \mid (x_1, \ldots, x_n) \in r \text{ and } (y_1, \ldots, y_m) \in s\}$;
4. *projection:* if $\bar{x} = (x_1, \ldots, x_n)$ is an n-tuple and $i_1, \ldots, i_p \in \{1, \ldots, n\}$, then $\pi_{i_1,\ldots,i_p}(\bar{x})$ equals $(x_{i_1}, \ldots, x_{i_p})$; if r is an n-ary relation then $\pi_{i_1,\ldots,i_p}(r)$ equals $\{\pi_{i_1,\ldots,i_p}(\bar{x}) \mid \bar{x} \in r\}$;
5. *selection:* if r is of arity n and $i, j \in \{1, \ldots, n\}$, then $\sigma_{i=j}(r)$ equals $\{(x_1, \ldots, x_n) \in r \mid x_i = x_j\}$.

A query on a schema $\mathcal{S}$ is said to be *expressible in the relational algebra* if it can be defined by an expression built from the relation names of $\mathcal{S}$ using the above five operators. For example, example query no. 2 from the previous section is easily expressed as $\pi_1(R) - \pi_2(R)$, and example query no. 3 as $\pi_{1,2,4,6,8}\sigma_{2=3}\sigma_{4=5}\sigma_{6=7}(R \times R \times R \times R)$.

A classical theorem of Codd [20] identifies the queries expressible in the relational algebra with the first-order queries. An n-ary query q on $\mathcal{S}$ is called *first-order* if there is a first-order formula $\varphi(x_1, \ldots, x_n)$ over the relational vocabulary $\mathcal{S}$, such that for every database $\mathbf{D}$,

$$q(\mathbf{D}) = \{(a_1, \ldots, a_n) \in |\mathbf{D}|^n \mid \mathbf{D} \models \varphi[a_1, \ldots, a_n]\}.$$

Here, $|\mathbf{D}|$ denotes the *active domain* of $\mathbf{D}$, consisting of all elements of $\mathbb{U}$ actually occurring in one of the relations of $\mathbf{D}$. In evaluating $\mathbf{D} \models \varphi[a_1, \ldots, a_n]$, we view $\mathbf{D}$ as an $\mathcal{S}$-structure (in the sense of model theory) with (finite) domain $|\mathbf{D}|$. Codd referred to first-order logic used to express queries in this way as the *relational calculus*.

Tarski being one of the founders of modern logic, it is not surprising that the first-order queries owe a lot to him. We mention just two things:

1. The now-standard definition of satisfaction of a formula in a structure was originally conceived by Tarski [55]. Thanks to Codd, every student of computer science gets in touch with the syntax and the Tarskian semantics of first-order logic, in the form of the relational calculus seen in the databases course.
2. In view of the previous section, we should also ask ourselves whether first-order queries actually satisfy the genericity criterion, or, equivalently, are they logical in the sense of Tarski? They sure are: in fact, already in 1936 Tarski and Lindenbaum [44] noted that not just first-order, but full typed higher-order logic can define only logical notions. Nowadays this sounds like a tautology, but back in the days when modern logic was still in the process of being defined, this was a fundamental observation.

In connection with Codd's theorem two more of Tarski's ideas find an application in database theory. We discuss them separately in the following subsections.

3.1 Relational completeness

Codd thought of his theorem as a completeness result for the relational algebra: the class of first-order queries was the reference level of expressiveness query languages should aim for. Codd called a query language *relationally complete* if it could express all first-order queries. Later, people started to realize that a lot of interesting queries are *not* first-order [6, 16, 17]. For example, of the list of example queries given in the previous section, queries no. 4, 5 and 6 are not first-order.

So, relational completeness is not everything. However, there still is a sense in which the relational algebra (or equivalently, first-order logic) can be considered a "complete" database query language, as was independently discovered by Bancilhon [8] and Paredaens [47]. They showed that for any database $\mathbf{D}$, and any relation $r \subseteq |\mathbf{D}|^n$ such that every automorphism of $\mathbf{D}$ is also an automorphism of r, there exists a first-order query q such that $q(\mathbf{D}) = r$. Here, an automorphism of $\mathbf{D}$ (r) is a permutation f of $\mathbb{U}$ such that $f(\mathbf{D}) = \mathbf{D}$ ($f(r) = r$). Note that the conditions of the theorem are necessary: for *any* generic n-ary query q, $q(\mathbf{D}) \subseteq |\mathbf{D}|^n$ and has at least the automorphisms of $\mathbf{D}$.

This "BP-completeness" of first-order logic, as it came to be called, actually follows from an early model-theoretic insight of Tarski, and another early model-theoretic theorem known as Beth's theorem. When he introduced the notion of elementary equivalence of structures [53, 54], Tarski noted that two *finite* structures are elementary equivalent only if they are isomorphic. Actually, given a finite structure $\mathbf{D}$ one can always write a single first-order sentence that is satisfied by any structure $\mathbf{D}'$ if and only if $\mathbf{D}'$ is isomorphic to $\mathbf{D}$.

Now let $\mathbf{D}$, over $\mathcal{S}$, and r be as in the BP-completeness theorem. Let $\mathcal{S}'$ be the expansion of $\mathcal{S}$ with an extra n-ary relation name R, and let $\mathbf{D}'$ be the expansion of $\mathbf{D}$ to $\mathcal{S}'$ by putting $R^{\mathbf{D}'} := r$. As $\mathbf{D}'$ is a finite structure, we can, by the above, write a first-order sentence φ such that any database $\mathbf{B}$ over $\mathcal{S}'$ satisfies φ iff $\mathbf{B}$ is isomorphic to $\mathbf{D}'$. Take any two $\mathbf{B}_1$ and $\mathbf{B}_2$ with $\mathbf{B}_1 \models \varphi$ and $\mathbf{B}_2 \models \varphi$; so there are permutations f_1 and f_2 of $\mathbb{U}$ so that $f_1(\mathbf{B}_1) = \mathbf{D}'$ and $f_2(\mathbf{B}_2) = \mathbf{D}'$. But then $f_2 f_1^{-1}$ is an automorphism of $\mathbf{D}$ and hence, by assumption, also of r. So, $f_2 f_1^{-1}(r) = r$, whence $R^{\mathbf{B}_1} = f_1^{-1}(r) = f_2^{-1}(r) = R^{\mathbf{B}_2}$.

We thus observe that φ implicitly defines R in terms of $\mathcal{S}$. By Beth's theorem, there is a first-order formula $\psi(\bar{x})$ over $\mathcal{S}$ such that in any model of φ the equivalence $\forall \bar{x}(R(\bar{x}) \leftrightarrow \psi)$ holds. This holds in particular in $\mathbf{D}'$ itself, by which we conclude that $\psi(\mathbf{D}) = r$.

We have thus easily derived the BP-completeness of first-order logic from some of the earliest model-theoretic results that were established. Still we recommend Paredaens's direct proof, which uses the relational algebra and is very elegant.[3]

[3] Recently, Cohen, Gyssens and Jeavons [21] showed that even the relational algebra without the union and difference operators, but with nonequality selections $\sigma_{i \neq j}$, is already BP-complete, on condition the active domain is directly available as one of the relations (or a projection of them), and has at least 3 elements.

3.2 Cylindric set algebras

Take a first-order formula φ, and let the different variables occurring in it, free or bound, be $x_1, \ldots, x_n$. When we follow the Tarskian semantics of φ on a structure A and determine inductively for every subformula ψ the set ψ^A of n-tuples $(a_1, \ldots, a_n) \in A^n$ under which ψ is true in A, we notice that at every inductive step we perform one of the following three operations on these n-ary relations:

1. *union*, to evaluate $\vee$;
2. *complementation* with respect to A^n, to evaluate $\neg$; and
3. *cylindrification* along dimension i, to evaluate $\exists x_i$.

By the cylindrification along dimension i of a relation $r \subseteq A^n$, with $i \in \{1, \ldots, n\}$, we mean the operation

$$\gamma_i(r) := \{(a_1, \ldots, a_n) \in A^n \mid \exists a \in A : (a_1, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_n) \in r\}.$$

These three operations, together with the constant relations $\delta_{ij} = \{(a_1, \ldots, a_n) \in A^n \mid a_i = a_j\}$, called the *diagonals* and needed to evaluate equality atoms $x_i = x_j$, constitute the *full n-dimensional cylindric set algebra with base A*. Cylindric set algebras are canonical examples of the general class of abstract *cylindric algebras,* which has an equational definition in the usual style abstract algebraic structures are defined. Cylindric algebras are the same to first-order logic as Boolean algebras are to propositional logic, and they were introduced by Tarski and his collaborators [31–34].

We thus see that a relational algebra in much the same spirit as Codd's was already considered by Tarski.[4] Concretely, let $\mathcal{S}$ be a schema, and take n strictly larger than the arity of every relation name in $\mathcal{S}$. We can build up *n-CSA expressions over $\mathcal{S}$* from the relation names in $\mathcal{S}$ and the constants δ_{ij} using the operators $e_1 \cup e_2$, $\neg e$, and $\gamma_i(e)$. When evaluating an n-CSA expression e on a database $\mathbf{D}$ with schema $\mathcal{S}$, the operators and constants are interpreted as in the full n-dimensional cylindric set algebra with base $|\mathbf{D}|$. Each relation name R is interpreted as the n-ary relation $R^{\mathbf{D}} \times |\mathbf{D}|^{n-m}$, if the arity of R is m. We then have:

Theorem 1. *An n-ary query is expressible in n-CSA in the sense just defined, if and only if it is expressible by a first-order formula using at most n different variables.*

Proof. Let the n variables be $x_1, \ldots, x_n$. The only thing we have to show is that atomic first-order formulas of the form $R(\ldots)$ can be expressed in n-CSA. Only for formulas of the form $R(x_1, \ldots, x_m)$ this is immediate by the expression R. The following example will suffice to explain the argument. Let $n = 3$, $m = 2$, and consider the formula $R(x_2, x_3)$. Note that the expression R expresses the

[4] Imielinski and Lipski [37] were the first to point out the connection between Codd's relational algebra and cylindric set algebras.

formula $R(x_1, x_2)$. We first copy the second column of R into its third column (which is "free:" recall that R stands for $R^{\mathbf{D}} \times |\mathbf{D}|$). Then we cylindrify the second column, after which we copy the first column to the now free second column. We finally cylindrify the first column. Formally, the following n-CSA expression expresses $R(x_2, x_3)$:

$$\gamma_1(\gamma_2(R \cap \delta_{2,3}) \cap \delta_{1,2})$$

In general, the existence of a "free column" guarantees us the room to perform the necessary transpositions on the columns to go from $R(x_1, \ldots, x_m)$ to $R(x_{\rho(1)}, \ldots, x_{\rho(m)})$ for an arbitrary permutation ρ of $\{1, \ldots, n\}$.

The trick used in the above proof is again an idea of Tarski [59]: he used it to give a substitution-free axiom system for first-order logic. Note how crucial it is in this respect that n is strictly larger than the arity of each relation name. Without this condition, the theorem does not seem to hold, although we have not proven this formally. The idea is that, with R binary for example, there are only a finite number of non-equivalent 2-CSA expressions over the schema $\{R\}$. However, there are infinitely many non-equivalent formulas with 2 variables over this schema.

The above theorem gives us a relational algebra for n-variable first-order logic. Bounded-variable fragments of first-order and infinitary logic were vigorously investigated in finite-model theory over the last decade [25, 46], for a large part motivated by database theory, in particular the seminal paper by Abiteboul and Vianu [5]. (Another major motivation is descriptive complexity [38].)

4 Relation algebras

In parallel with his work on cylindric algebras, Tarski also promoted *relation algebras* [18, 51, 61]. Like cylindric algebras, these are again a generalization of Boolean algebras, but in another direction. They have again an abstract equational definition, but we will only be concerned here with the operations of the *proper relation algebra with base* A, where A is a set of atomic data elements. These operations are defined on *binary* relations over A only and comprise the following:

1. *union*;
2. *complementation* with respect to A^2;
3. *composition:* $r \odot s := \{(x, y) \mid \exists z : (x, z) \in r \text{ and } (z, y) \in s\}$;
4. *conversion:* $\breve{r} := \{(x, y) \mid (y, x) \in r\}$.

For the remainder of this section, we fix a database schema $\mathcal{S}$ with all relation names binary. This is not a heavy restriction: an n-ary relation can easily and naturally be represented by n binary relations, and, as advocates of the *Decomposed Storage Model* argue quite convincingly [22, 41], it can even be beneficial to do so from a systems engineering point of view.

Again we can build expressions starting from the relation names in $\mathcal{S}$, and the constant Id, using the four operators above. We will call these *RA-expressions*. On a database $\mathbf{D}$ with schema $\mathcal{S}$, an RA-expression e can be evaluated by interpreting the relation names as given by $\mathbf{D}$, interpreting the constant Id as the identity relation on $|\mathbf{D}|$, and interpreting the operations relative to base set $|\mathbf{D}|$. The result is a binary relation $e(\mathbf{D})$. So, RA-expressions always express binary queries.

Queries expressible in RA are clearly first-order, and actually a substantial number of first-order queries are expressible in RA. For example:

1. Recalling example query no. 2 from Section 2,
$$(Id \cap R \odot (Id \cup \neg Id)) - (Id \cap (Id \cup \neg Id) \odot R)$$
expresses $\{(x,x) \mid \exists y\, R(x,y) \wedge \neg \exists z\, R(z,x)\}$.
2. Recalling example query no. 3, $R \odot R \odot R \odot R$ expresses $\{(x_1,x_5) \mid \exists x_2,x_3,x_4\ (R(x_1,x_2) \wedge R(x_2,x_3) \wedge R(x_3,x_4) \wedge R(x_4,x_5)\}$.
3. $R \odot (\neg Id \cap \breve{R} \odot R)$ expresses $\{(x,y) \mid R(x,y) \wedge \exists z \neq y : R(x,z)\}$.

Note that the above three example RA queries can actually already be expressed with a first-order formula using only 3 distinct variables. Indeed, in the first formula we could have reused the bound variable y instead of introducing a new bound variable z. The second query can be equivalently expressed as
$$\{(x,y) \mid \exists z \Big(\exists y \big(\exists z (R(x,z) \wedge R(z,y)) \wedge R(y,z) \big) \wedge R(z,y) \Big) \}$$
using just 3 variables. This is no coincidence; it is readily verified that any RA query is in FO^3 (first-order queries expressible using only 3 variables). Tarski and Givant [61] showed the converse: a binary query on binary relations is expressible in RA precisely when it is in FO^3.

4.1 From RA to FO by pairing

So, RA seems rather limited in expressive power. However, Tarski and Givant showed also that in the presence of a "pairing axiom," RA becomes equally powerful as full first-order logic. We give a nice concretization of this general idea, due to Gyssens, Saxton and Van Gucht [28].

Consider the following two "pairing" operations on a binary relation r on some base set A:

- *left tagging:* $r^{\triangleleft} := \{(x,(x,y)) \mid (x,y) \in r\}$;
- *right tagging:* $r^{\triangleright} := \{((x,y),y) \mid (x,y) \in r\}$.

Note that the resulting relations are not on A anymore, but on $A \cup A^2$. This suggests to build up a universe $\mathbb{U}^+$ on top of $\mathbb{U}$, similar to the universe $\mathbb{U}^*$ we considered in Section 2:
$$\mathbb{U}_0^+ := \mathbb{U}, \quad \mathbb{U}_{n+1}^+ := \mathbb{U}_n^+ \cup (\mathbb{U}_n^+)^2, \quad \mathbb{U}^+ := \bigcup_n \mathbb{U}_n^+.$$

Left tagging and right tagging can now be naturally viewed as operations on binary relations on $\mathbb{U}^+$.

Our objective, of course, is to add the pairing operations to RA. A problem with this, however, is that when we want to evaluate an expression containing these operations on a database $\mathbf{D}$, it no longer makes sense to perform the complementation operation with respect to $|\mathbf{D}|$, as we are really working in $|\mathbf{D}|^+$. We cannot complement with respect to the full $\mathbb{U}^+$ either, as this could yield infinite relations, and we have been working with finite databases from the outset. A simple way out is to redefine complementation relative to the active domain of a relation. So, for any binary relation r, we define

$$\neg r := \{(x, y) \in |r|^2 \mid (x, y) \notin r\},$$

where $|r| = \{x \mid \exists y : (x, y) \in r \text{ or } (y, x) \in r\}$. A second modification we make to RA is that we throw away Id, since we don't need it anymore: for example, $R \cap Id$ is expressible as $R^{\triangleleft} \odot (R^{\triangleleft})^{\smile} \cup (R^{\triangleright})^{\smile} \odot R^{\triangleright}$.

We denote RA, modified as just described, and enriched with the pairing operations, by RA^+. Evaluating an RA^+-expression e on a database $\mathbf{D}$ in general yields a binary relation $e(\mathbf{D})$ on $|\mathbf{D}|^+$. Such binary relations can represent n-ary relations on $|\mathbf{D}|$. For example, we can represent the ternary relation $\{(a, b, c), (d, e, f)\}$ as $\{(a, (b, c)), (d, (e, f))\}$. Using such representations, we leave it as an exercise to the reader to simulate Codd's relational algebra in RA^+. So RA^+ has the full power of the first-order queries.

We conclude that Tarski produced two alternatives for Codd's relational algebra: cylindric set algebra, and relation algebra with pairing. From a systems engineering perspective, Codd's algebra remains of course very attractive [48].

4.2 A computationally complete query language based on RA^+

Recall that on the most general level, given a schema $\mathcal{S}$, we defined a *generic n-ary query on* $\mathcal{S}$ to be any (possibly partial) function from databases with schema $\mathcal{S}$ to finite n-ary relations on $\mathbb{U}$ that is invariant under every permutation of $\mathbb{U}$. Genericity allows us to give a standard definition of when a query is "computable." Note that this is not immediate, because standard computability works with concrete data objects, like natural numbers, or strings over a finite alphabet, while our atomic data elements in $\mathbb{U}$ remain abstract.

The computability definition, given in a seminal paper by Chandra and Harel [16], goes as follows. Let $\mathbf{D}$ be a database and suppose the cardinality of $|\mathbf{D}|$ is m. Any bijection from $|\mathbf{D}| \to \{1, \ldots, m\}$ is called an *enumeration* of $\mathbf{D}$. The image of $\mathbf{D}$ under an enumeration yields a concrete object with which we can deal using the standard computational models. We now say that query q is *computable* if there is a computable function C in the standard sense, such that for every database $\mathbf{D}$, $q(\mathbf{D})$ is defined if and only if C is defined on every enumeration X of $\mathbf{D}$, and in this case $C(X)$ always equals an enumeration of $q(\mathbf{D})$.

We can capture the class of computable generic queries by making RA^+ into a programming language. It suffices to add variables holding finite binary relations

on $\mathbb{U}^+$, assignment statements of the form $X := e$, where X is a variable and e is an RA^+-expression over the relation names in $\mathcal{S}$ and the variables, and to build up programs from these statements using composition and while-loops of the form '**while** $e \neq \varnothing$ **do** P'. We give programs the obvious operational semantics. For example, the following program computes the transitive closure of R:

$X := R$;
while $(X \odot R) - X \neq \varnothing$ **do**
  $X := X \cup X \odot R$.

Every program expresses a query by designating one of the variables as the output variable. This query is evidently generic and computable.

Since the programming language just described is quite similar to the original query language 'QL' first proved to be computationally complete by Chandra and Harel [16], it does not harm to refer to it by the same name. Using as before a representation of n-ary relations on $\mathbb{U}$ by binary relations on $\mathbb{U}^+$, we then have:

Theorem 2 ([16], see also [3, 4, 1]). *Every computable generic query is expressible by a QL-program.*

We feel this result is a nice a-posteriori confirmation of Tarski's conviction that relation algebra (with pairing) is a formalism with all the expressive power one needs.

We conclude this section with three remarks. First, given that RA^+ expressions evaluate to relations on $\mathbb{U}^+$ rather than on $\mathbb{U}$, one could generalize the notion of query somewhat to yield relations on $\mathbb{U}^+$, rather than on $\mathbb{U}$, as output. Let us refer to this generalized notion of query as *+-query*. The notions of genericity and computable readily generalize to +-queries. Then under these generalizations, the language QL just defined is still computationally complete: every computable generic +-query on binary relations is expressible by a program in QL.

Second, in view of the universe $\mathbb{U}^*$ considered in Section 2, of which $\mathbb{U}^+$ is only a subuniverse, we can generalize +-queries further to *$*$-queries* which now yield relations on $\mathbb{U}^*$ as output. QL is then no longer complete [63], but can be easily reanimated by adding a third tagging operation:

– *set tagging:* $r^{\triangle} := \big\{\big(x, \{y \mid (x,y) \in r\}\big) \;\big|\; \exists y : (x,y) \in r\big\}$.

This is an operation on binary relations on $\mathbb{U}^*$ rather than on $\mathbb{U}^+$. We then again have that QL enriched with set tagging is computationally complete for the generic $*$-queries [23, 35].

Finally, note that although the tagging operations introduce pairs and sets of elements, these pairs and sets are still treated by the RA operations as atomic abstract elements. So it is natural to replace every element of $\mathbb{U}^* - \mathbb{U}$ occurring in the output of a $*$-query applied to a database $\mathbf{D}$ by a fresh new element in $\mathbb{U}$ not in $|\mathbf{D}|$. The class of abstract database transformations that can be obtained from computable generic $*$-queries in this way has a purely algebraic characterization [2, 64].

5 Constraint databases

Until now we have worked with relational databases over an unstructured universe $\mathbb{U}$ of atomic data elements. That the atomic data elements remain abstract and uninterpreted is one of the identifying features of classical database theory, and corresponds in practice to the generic bulk-processing nature of database operations. However, in reality $\mathbb{U}$ usually does have a structure of its own, in the form of predicates and functions defined on it, and there are applications where we want to take this structure into account. An important case, on which we will focus in this and the next section, is that of *spatial* databases containing information with a geometrical interpretation.

Suppose we want to store points in the plane in our database. In this case $\mathbb{U}$ is $\mathbb{R}$, the set of real numbers. We use a schema $\mathcal{S}$ with a binary relation name S. In a database $\mathbf{D}$ with schema $\mathcal{S}$, $S^{\mathbf{D}}$ then is a finite set of pairs of real numbers, i.e., a finite set of points in the plane. In the presence of an "interpreted" universe such as $\mathbb{R}$ we need to make a crucial but natural extension to the notion of first-order query: we make the interpreted predicates and functions of our universe available in our first-order formulas. Concretely, in the case of $\mathbb{R}$, we now use formulas over the vocabulary $\mathcal{S}$ expanded with the vocabulary $(<, +, \cdot, 0, 1)$ of ordered fields.

For example, suppose we want to ask whether all points in the database lie on a common circle around the origin. It is tempting to write the following formula for this purpose:

$$\exists r \forall x, y(S(x, y) \rightarrow x^2 + y^2 = r^2)$$

However, we should remember from Section 3 that we agreed to evaluate first-order formulas over the active domain of the database. In contrast, the above formula, and in particular the quantifier $\exists r$, is intended to be evaluated over the whole of $\mathbb{R}$. Since the radius of the circle does not need to be a coordinate of a point in the database, the formula is therefore incorrect as an expression of our query. A correct formula under the active-domain semantics is the following:

$$\exists x_1, y_1 \forall x, y(S(x, y) \rightarrow x^2 + y^2 = x_1^2 + y_1^2))$$

This example shows that the active-domain semantics for first-order formulas is not very natural in the case of spatial databases. It was fine in the case of an uninterpreted universe $\mathbb{U}$, because there, all elements of $\mathbb{U}$ not in the active domain of a database look alike with respect to that database [7]. In contrast, no two reals look alike in first-order logic over the reals with signature $(<, +, \cdot, 0, 1)$.

We thus have two ways of evaluating a first-order formula $\varphi(\bar{x})$ on a real database $\mathbf{D}$. We view $\mathbf{D}$ as an expansion of the structure $(\mathbb{R}, <, +, \cdot, 0, 1)$ with the relations of $\mathbf{D}$. When we then write $\mathbf{D} \models_{\text{adom}} \varphi[\bar{a}]$ for some tuple $\bar{a}$ of reals, we mean that $\varphi[\bar{a}]$ becomes true in $\mathbf{D}$ when we let each quantifier in φ range over $|\mathbf{D}|$ only. When we write $\mathbf{D} \models_{\text{natural}} \varphi[\bar{a}]$, we mean that $\varphi[\bar{a}]$ becomes true when we let each quantifier range over the whole of $\mathbb{R}$. The natural semantics is definitely more natural than the active-domain semantics, but is it really more powerful? The answer is no: Benedikt and Libkin [11] gave an algorithm that

turns any formula φ into another formula ψ such that for every real database $\mathbf{D}$ we have $\mathbf{D} \models_{\text{natural}} \varphi$ iff $\mathbf{D} \models_{\text{active}} \psi$. From now on we will stick to the natural semantics.

Given the natural semantics, the new issue arises that the result of a first-order query to a real database can easily be infinite, even though the database is finite. For example, the following first-order query returns all points in the convex closure of S:

$$\{(x,y) \mid \exists x_1, y_1, x_2, y_2, \lambda : S(x_1,y_1) \wedge S(x_2,y_2) \wedge 0 \leqslant \lambda \leqslant 1 \\ \wedge (x,y) = \lambda(x_1,y_1) + (1-\lambda)(x_2,y_2)\}$$

How do we represent these infinite sets?

A solution to this issue was proposed by Kanellakis, Kuper and Revesz in their novel framework of *constraint databases* [40]. Call the above formula φ, and take for example the simple database $\mathbf{D}_0$ with just two points in it: $S^{\mathbf{D}_0} = \{(0,0),(1,1)\}$. To evaluate φ on $\mathbf{D}_0$ we can try the following simple idea called *plug-in evaluation:* replace in φ every atomic subformula of the form $S(u,v)$ by a corresponding formula defining $S^{\mathbf{D}_0}$, i.e., the formula $(u = 0 \wedge v = 0) \vee (u = 1 \wedge v = 1)$. We get the following formula which is purely over the reals only; it no longer mentions any database relations:

$$\{(x,y) \mid \exists x_1, y_1, x_2, y_2, \lambda : \\ ((x_1,y_1) = (0,0) \vee (x_1,y_1) = (1,1)) \\ \wedge ((x_2,y_2) = (0,0) \vee (x_2,y_2) = (1,1)) \\ \wedge 0 \leqslant \lambda \leqslant 1 \wedge (x,y) = \lambda(x_1,y_1) + (1-\lambda)(x_2,y_2)\}$$

This formula defines the infinite set of points in $\mathbb{R}^2$ on the closed line segment between $(0,0)$ and $(1,1)$ and symbolically represents the answer of our query on our database.

We can always perform plug-in evaluation of a first-order query on a real database, provided the numbers occurring in the database are rational so that we can effectively write down the formula defining the answer. In general, the subsets of $\mathbb{R}^n$ that are definable (as n-ary relations) by first-order formulas over $\mathbb{R}$ are known as the *semi-algebraic sets* [10,14]. They have quite nice properties.

Is this representation of semi-algebraic sets by real formulas workable? Can we, e.g., effectively decide whether the set defined by a given real formula is nonempty? Thanks to Tarski's decision procedure for the first-order theory of the reals [52], the representation is indeed quite workable. The computational complexity of Tarski's original procedure is very high, but over the years there has been steady progress in algorithms for real algebraic geometry [9, 15, 30, 45]. A crucial parameter in the computational complexity of current algorithms is the number of quantifiers in the formula. In the case of plug-in evaluation this number depends only on the query formula and not on the database, which implies a polynomial time complexity.

Tarski actually gave a complete quantifier elimination procedure for real formulas, so we can define every semi-algebraic set already by a quantifier-free

formula. To arrive at the concept of constraint database, we make one final but logical step: we allow semi-algebraic sets not just as outputs of queries, but also as inputs. Specifically, we remove the restriction that each relation in the database must be finite, and require instead that each relation must be a semi-algebraic set. In a *constraint database* we thus no longer store any actual tuples, but we store a collection of quantifier-free formulas, one for each relation name in the schema of the database. Plug-in evaluation is still possible: given a first-order formula φ and a constraint database $\mathbf{D}$, we replace in φ every atomic subformula of the form $S(u, v)$, with S a relation name from the database schema, by the real formula $\gamma(u, v)$ defining $S^{\mathbf{D}}$. The result of all these replacements is a real formula defining $\varphi(\mathbf{D})$. Since thanks to Tarski we can work with quantifier-free formulas, the computational complexity of deciding nonemptiness and many other algorithms remains polynomial-time like we had with finite databases.

Constraint databases has been an active research area over the past decade [39, 43, 49, 62, 43], and the constraint database concept is not at all limited to the reals. In principle it works for any universe $\mathbb{U}$ with a certain structure (consisting of predicates and functions) that admits effective quantifier elimination (and for which truth of atomic sentences is decidable).[5] In this respect, recall that, as Tarski himself noted early on [57], his quantifier elimination for the reals implies that *every subset of $\mathbb{R}$ definable by a first-order formula with parameters over the reals is the union of a finite number of intervals.* This property alone is already responsible for a lot of the nice properties (referred to as "tame topology" [65]) of semi-algebraic sets. Any universe that has this property is called *o-minimal.* The "collapse" of the natural semantics for first-order logic on finite database to the active-domain semantics, which we pointed out earlier, does not just hold under the reals but under *any* universe that is o-minimal and admits quantifier elimination [11].

6 Geometric queries

We conclude our survey by making the circle complete and returning to the first topic we discussed: the notion of genericity for database queries, now reconsidered in the new setting of spatial databases.

We begin by noting that, when thinking about spatial data, the real numbers as "urelemente" are not the right level of abstraction. They show up merely as a convenient representation, via coordinates, for the real urelemente which are the points in our geometric space. Let us work as before in the real plane $\mathbb{R}^2$ (everything we will say in this section generalizes to arbitrary $\mathbb{R}^d$). This means in general that we only work with database schemas $\mathcal{S}$ where the arity of every relation name is a multiple of 2. In a *geometric database* with schema $\mathcal{S}$, each relation, of arity $2n$, say, is interpreted as an n-ary relation on $\mathbb{R}^2$. Following the constraint database approach outlined in the previous section, each relation is semi-algebraic. We call such databases *geometric.* An *n-ary geometric query*

[5] Interesting recent work even considers universes that are non-numeric, such as strings or terms [12, 13, 24].

over $\mathcal{S}$ then is a function mapping geometric databases with schema $\mathcal{S}$ to n-ary relations on $\mathbb{R}^2$.

Recalling our discussion in Section 2, we can now again call a geometric query *generic* if it is invariant under all permutations of $\mathbb{R}^2$ (note: $\mathbb{R}^2$, not $\mathbb{R}$). This does make sense: these are the queries that treat the points as uninterpreted abstract data elements. They are exactly the classical generic queries under the unstructured universe $\mathbb{U}$ where this $\mathbb{U}$ happens to be $\mathbb{R}^2$. Many interesting geometric queries are not so "absolutely" generic, however, and this is as it should be: as we mentioned at the end of Section 2, Tarski viewed logic as an "extreme" kind of geometry. Thus, by considering various other groups of transformations of $\mathbb{R}^2$, corresponding to various geometrical interpretations of our spatial data, we can reach the geometric queries that fit the particular interpretation.

Let us illustrate this using the group of affinities, i.e., the permutations of $\mathbb{R}^2$ that preserve betweenness. We call a geometric query *affine-generic* if it is invariant under all affinities of $\mathbb{R}^2$. For example, consider the following geometric queries over a set of points S (i.e., a binary relation S):

1. Is S nonempty? In first-order: $\exists x, y\, S(x, y)$
2. Is S convex? In first-order: $\forall x_1, y_1, x_2, y_2, \lambda : (S(x_1, y_1) \wedge S(x_2, y_2) \wedge 0 \leqslant \lambda \leqslant 1) \rightarrow S(\lambda(x_1, y_1) + (1 - \lambda)(x_2, y_2))$
3. Is S a circle? In first-order: $\exists r, x_0, y_0 \forall x, y : S(x, y) \leftrightarrow (x-x_0)^2+(y-y_0)^2 = r^2$

Query 1 is absolutely generic; query 2 is only affine-generic; and query 3 is not affine-generic (the notion of "circle" does not exist in affine geometry).

We now come to a natural question: how can we logically characterize the affine-generic first-order geometric queries? Tarski brings us inspiration. In his work on the axiomatization of elementary geometry [58, 50], Tarski considered first-order logical formalisms with variables ranging over points in the geometric space, and elementary predicates on these points as dictated by the geometry to be formalized. For example, elementary affine geometry in the real plane corresponds to doing first-order logic in the structure $(\mathbb{R}^2, \beta)$, where β denotes the ternary betweenness predicate on $\mathbb{R}^2$: $\beta(p, q, r)$ holds if point p lies on the closed line segment between points q and r.

Inspired by this, we can view a geometric database $\mathbf{D}$ with schema $\mathcal{S}$ as a constraint database over the interpreted universe $(\mathbb{R}^2, \beta)$ rather than over the universe $(\mathbb{R}, <, +, \cdot, 0, 1)$. Note that under this alternative view, the schema changes: the arity of each relation name S goes from $2n$ to n. We denote this "halved" schema by $\mathcal{S}'$. First-order queries under the alternative view now are expressed by first-order formulas over the vocabulary $(\mathcal{S}', \beta)$ rather than $(\mathcal{S}, <, +, \cdot, 0, 1)$. Let us refer to the original class of first-order queries as FO[$\mathbb{R}$] and to the new one as FO[β]. Example queries 1 and 2 from above are expressed in FO[β] as follows:

1. $\exists p\, S(p)$
2. $\forall p, q, r : (S(p) \wedge S(q) \wedge \beta(r, p, q)) \rightarrow S(r)$

Example query 3 is not in FO[β]: indeed, query 3 is not affine-generic, and FO[β] clearly contains only affine-generic queries. It is also clear that FO[β] is a subclass

of FO[$\mathbb{R}$]: we represent every point variable p by a pair of real variables (p_1, p_2), and $\beta(p, q, r)$ is easily expressible as $(q_1 - p_1) \cdot (p_2 - r_2) = (q_2 - p_2) \cdot (p_1 - r_1) \wedge 0 \leqslant (q_1 - p_1) \cdot (p_1 - r_1) \wedge 0 \leqslant (q_2 - p_2) \cdot (p_2 - r_2)$.

Interestingly, the converse holds as well:

Theorem 3 ([29]). *Every affine-generic geometric query in* FO[$\mathbb{R}$] *is in* FO[β].

The proof uses the original observation by Tarski that the geometrical constructions of $<$, $+$ and $\cdot$ of points on a line can be defined in first-order logic over β.

Other geometric interpretations can be captured in a similar way: for example, if we use instead of the betweenness predicate, the (4-ary) equal-distance predicate, we obtain euclidean rather than affine geometry. Capturing topological queries (invariant under all homeomorphisms) is much more difficult [42, 27].

7 Conclusion

We have surveyed some ideas and results from database theory related to Tarski's ideas. We have neither been comprehensive with respect to Tarski's work, nor with respect to database theory, and probably not even with respect to the applications of the former in the latter. A great source to learn more about Tarski are his Collected Papers [26]. A great source to learn more about database theory are the proceedings of the annual ACM Symposium on Principles of Database Systems published by ACM Press, and the biannual International Conference on Database Theory published in Springer's Lecture Notes in Computer Science.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
2. S. Abiteboul and P.C. Kanellakis. Object identity as a query language primitive. *Journal of the ACM*, 45(5):798–842, 1998.
3. S. Abiteboul and V. Vianu. Procedural languages for database queries and updates. *Journal of Computer and System Sciences*, 41(2):181–229, 1990.
4. S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43(1):62–124, 1991.
5. S. Abiteboul and V. Vianu. Computing with first-order logic. *Journal of Computer and System Sciences*, 50(2):309–335, 1995.
6. A.V. Aho and J.D. Ullman. Universality of data retrieval languages. In *Conference Record, 6th ACM Symposium on Principles of Programming Languages*, pages 110–120, 1979.
7. A.K. Aylamazyan, M.M. Gilula, A.P. Stolboushkin, and G.F. Schwartz. Reduction of the relational model with infinite domains to the case of finite domains. *Doklady Akademii Nauk SSSR*, 286(2):308–311, 1986. In Russian.

8. F. Bancilhon. On the completeness of query languages for relational data bases. In *Proceedings 7th Symposium on Mathematical Foundations of Computer Science*, volume 64 of *Lecture Notes in Computer Science*, pages 112–123. Springer-Verlag, 1978.
9. S. Basu. *Algorithms in Semi-Algebraic Geometry*. PhD thesis, New York University, 1996.
10. R. Benedetti and J.-J. Risler. *Real Algebraic and Semi-Algebraic Sets*. Hermann, 1990.
11. M. Benedikt and L. Libkin. Relational queries over interpreted structures. *Journal of the ACM*, 47(4):644–680, 2000.
12. M. Benedikt, L. Libkin, T. Schwentick, and L. Segoufin. String operations in query languages. In *Proceedings 20th ACM Symposium on Principles of Database Systems*, 2001.
13. A. Blumensath and E. Grädel. Automatic structures. In *Proceedings 15th IEEE Symposium on Logic in Computer Science*, pages 51–62, 2000.
14. J. Bochnak, M. Coste, and M.-F. Roy. *Real Algebraic Geometry*. Springer-Verlag, 1998.
15. B.F. Caviness and J.R. Johnson, editors. *Quantifier elimination and cylindrical algebraic decomposition*. Springer, 1998.
16. A. Chandra and D. Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2):156–178, 1980.
17. A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25:99–128, 1982.
18. L.H. Chin and A. Tarski. Distributive and modular laws in the arithmetic of relation algebras. *University of California Publications in Mathematics—New Series*, 1(9):341–384, 1951.
19. E. Codd. A relational model for large shared databanks. *Communications of the ACM*, 13(6):377–387, 1970.
20. E. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Data Base Systems*, pages 65–98. Prentice-Hall, 1972.
21. D. Cohen, M. Gyssens, and P. Jeavons. Derivation of constraints and database relations. In E.C. Freuder, editor, *Principles and Practice of Constraint Programming*, volume 1118 of *Lecture Notes in Computer Science*, pages 468–481, 1996.
22. G.P. Copeland and S. Khoshafian. A decomposition storage model. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, volume 14:4 of *SIGMOD Record*, pages 268–279. ACM Press, 1985.
23. E. Dahlhaus and J.A. Makowsky. Query languages for hierarchic databases. *Information and Computation*, 101(1):1–32, 1992.
24. E. Dantsin and A. Voronkov. Expressive power and data complexity of query languages for trees and lists. In *Proceedings 19th ACM Symposium on Principles of Database Systems*, pages 157–165, 2000.
25. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, second edition, 1999.
26. S.R. Givant and R.N. McKenzie, editors. *Alfred Tarski, Collected Papers*. Birkhäuser, 1986.
27. M. Grohe and L. Segoufin. On first-order topological queries. In *Proceedings 15th IEEE Symposium on Logic in Computer Science*, pages 349–360, 2000.
28. M. Gyssens, L.V. Saxton, and D. Van Gucht. Tagging as an alternative to object creation. In J.C. Freytag, D. Maier, and G. Vossen, editors, *Query Processing For Advanced Database Systems*, chapter 8. Morgan Kaufmann, 1994.

29. M. Gyssens, J. Van den Bussche, and D. Van Gucht. Complete geometric query languages. *Journal of Computer and System Sciences*, 58(3):483–511, 1999.
30. J. Heintz, T. Recio, and M.-F. Roy. Algorithms in real algebraic geometry and applications to computational geometry. In J. Goodman, R. Pollack, and W. Steiger, editors, *Discrete and Computational Geometry*, volume 6 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. AMS-ACM, 1991.
31. L. Henkin, J.D. Monk, and A. Tarski. *Cylindric Algebras. Part I.* North-Holland, 1971.
32. L. Henkin, J.D. Monk, and A. Tarski. *Cylindric Algebras. Part II.* North-Holland, 1985.
33. L. Henkin, J.D. Monk, A. Tarski, H. Andréka, and I. Németi. *Cylindric Set Algebras*, volume 883 of *Lecture Notes in Mathematics*. Springer-Verlag, 1981.
34. L. Henkin and A. Tarski. Cylindric algebras. In R.P. Dilworth, editor, *Lattice Theory*, volume 2 of *Proceedings of Symposia in Pure Mathematics*, pages 83–113. American Mathematical Society, 1961.
35. R. Hull and J. Su. Algebraic and calculus query languages for recursively typed complex objects. *Journal of Computer and System Sciences*, 47(1):121–156, 1993.
36. R. Hull and C.K. Yap. The format model, a theory of database organization. *Journal of the ACM*, 31(3):518–537, 1984.
37. T. Imielinski and W. Lipski. The relational model of data and cylindric algebras. *Journal of Computer and System Sciences*, 28:80–102, 1984.
38. N. Immerman. *Descriptive Complexity*. Springer, 1999.
39. P.C. Kanellakis. Constraint programming and database langauges: a tutorial. In *Proceedings 14th ACM Symposium on Principles of Database Systems*, pages 46–53, 1995.
40. P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, August 1995.
41. S. Khoshafian, G.P. Copeland, T. Jagodi, H. Boral, and P. Valduriez. A query processing strategy for the decomposed storage model. In *Proceedings of the Third International Conference on Data Engineering*, pages 636–643. IEEE Computer Society, 1987.
42. B. Kuijpers and V. Vianu. Topological queries. In Kuper et al. [43], chapter 10.
43. G. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases*. Springer, 2000.
44. A. Lindenbaum and A. Tarski. On the limitations of the means of expression of deductive theories. In *Logic, Semantics, Metamathematics. Papers from 1923–1938* [56], pages 384–392.
45. B. Mishra. Computational real algebraic geometry. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.
46. M. Otto. *Bounded variable logics and counting: a study in finite models*, volume 9 of *Lecture Notes in Logic*. Springer, 1997.
47. J. Paredaens. On the expressive power of the relational algebra. *Information Processing Letters*, 7(2):107–111, 1978.
48. R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, second edition, 2000.
49. P.Z. Revesz. Constraint databases: a survey. In L. Libkin and B. Thalheim, editors, *Semantics in Databases*, volume 1358 of *Lecture Notes in Computer Science*, pages 209–246. Springer, 1998.
50. W. Schwabhäuser, W. Szmielew, and A. Tarski. *Metamathematische Methoden in der Geometrie*. Springer-Verlag, 1983.

51. A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6:73–89, 1941.
52. A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, 1951.
53. A. Tarski. Some notions and methods on the borderline of algebra and metamathematics. In *Proceedings of the International Congress of Mathematicians, Cambridge, Mass, 1950*, volume 1, pages 705–720. American Mathematical Society, 1952.
54. A. Tarski. Contributions to the theory of models, I and II. *Indagationes Mathematicae*, 16:572–581 and 582–588, 1954. Volume III, which contains the list of references, is in volume 17 of the same journal.
55. A. Tarski. The concept of truth in formalized languages. In *Logic, Semantics, Metamathematics. Papers from 1923–1938* [56], pages 152–278.
56. A. Tarski. *Logic, Semantics, Metamathematics. Papers from 1923–1938.* Clarendon Press, Oxford, 1956.
57. A. Tarski. On definable sets of real numbers. In *Logic, Semantics, Metamathematics. Papers from 1923–1938* [56], pages 110–142.
58. A. Tarski. What is elementary geometry? In L. Henkin, P. Suppes, and A. Tarski, editors, *The Axiomatic Method, with Special Reference to Geometry and Physics*, pages 16–29. North-Holland, 1959.
59. A. Tarski. A simplified formalization of predicate logic with identity. *Archiv für Mathematische Logik und Grundlagenforschung*, 7:61–79, 1965.
60. A. Tarski. What are logical notions? *History and Philosophy of Logic*, 7:143–154, 1986. Edited by J. Corcoran.
61. A. Tarski and S. Givant. *A Formalization of Set Theory Without Variables*, volume 41 of *AMS Colloquium Publications.* American Mathematical Society, 1987.
62. J. Van den Bussche. Constraint databases: a tutorial introduction. *SIGMOD Record*, 29(3):44–51, 2000.
63. J. Van den Bussche and J. Paredaens. The expressive power of complex values in object-based data models. *Information and Computation*, 120:220–236, 1995.
64. J. Van den Bussche, D. Van Gucht, M. Andries, and M. Gyssens. On the completeness of object-creating database transformation languages. *Journal of the ACM*, 44(2):272–319, 1997.
65. L. van den Dries. *Tame Topology and O-Minimal Structures.* Cambridge University Press, 1998.
66. I.M. Yaglom. *Felix Klein and Sophus Lie: evolution of the idea of symmetry in the nineteenth century.* Birkhäuser, Boston, 1988.