III

SQL Standardization and Beyond

23 Evaluating the Completeness of TSQL2
24 Notions of Upward Compatibility of Temporal Query Languages
25 Transitioning Temporal Support in TSQL2 to SQL3
26 Adding Valid Time to SQL/Temporal
27 Adding Transaction Time to SQL/Temporal
28 Temporal Statement Modifiers

The process that led to the TSQL2 language involved eighteen database specialists and lasted little more that one year. TSQL2 arguably remains the most comprehensively documented and most feature-rich temporal query language yet to be seen. Following its completion, it was time to subject the language to closer scrutiny than the hectic time schedule of the design process had permitted. Chapter 23 identifies several deficiencies in TSQL2: The language fails to satisfy two notions of completeness, and aspects of the language related to duplicates and nested queries are unsatisfactory.

Chapter 24 delves further into the properties that should be satisfied by a temporally extended query language and proposes the property of temporal upward compatibility that, together with conventional upward compatibility, aims to ensure that the temporal query language is legacy software friendly. Since most existing database applications manage temporal data and are prime candidates for benefitting from built-in temporal support in the database management system, these are important requirements. The chapter also reveals that no existing temporally extended SQL fully supports upward compatibility and temporal upward compatibility.

These insights set the stage for a consolidated temporal query language, and indeed, the remaining four chapters describe not one, but two such extended query

languages. The two languages started out as one and took their outset in these insights and in the realization that introducing so-called statement modifiers into TSQL2, while also carefully designing the defaults in the language, might yield a language that would satisfy all the major properties identified. The two languages grew to gradually become quite different. Substantial parts of both languages have been implemented.

The first language is covered in Chapters 25–27 and proposes the addition of built-in temporal support to the part of the SQL3 standard termed SQL/Temporal. This language represents an effort to increase the impact on practice of temporal query languages research. Chapter 25 provides an overview of the language, and Chapters 26 and 27 are the actual expert contributions, also termed change proposals, submitted to the ANSI and ISO standards bodies; they propose the inclusion of built-in valid-time support and transaction-time support into SQL/Temporal, respectively. The proposals were unanimously accepted by ANSI and were then forwarded to ISO, where it is unlikely that they will be voted on in time to be included into SQL3.

The second language, ATSQL, is described in Chapter 28. This language extends SQL–92 rather than SQL3 and is unaffected by the compromises that naturally accompany a standardization process. Rather, ATSQL aims to demonstrate concisely and precisely how the notion of temporal statement modifiers may be employed to define a temporally extended query language. As a result, the focus is on core functionality, and many of the advanced features of TSQL2, including aspects such as indeterminacy, schema versioning, and vacuuming, are not addressed. In addition, a set-based framework is assumed. The use of temporal statement modifiers, leads to a categorization of query language statements into three types, so-called temporally upward compatible, non-sequenced, and sequenced statements. It is attractive to be able to formulate as many statements as possible as sequenced statements because these statements offer the most built-in support for formulating statements; ATSQL offers a wider range of sequenced statements than does its sibling.