Prologue

HISTORICAL OVERVIEW OF DATABASE TECHNOLOGY

This prologue provides a brief historical perspective of developments in database technology, and then reviews and contrasts three current approaches to elevate the initial design of database systems to a conceptual level.

Beginning in the late 1970s, the old network and hierarchic database management systems (DBMSs) began to be replaced by relational DBMSs, and by the late 1980s relational systems performed sufficiently well that the recognized benefits of their simple bag-oriented data structure and query language (SQL) made relational DBMSs the obvious choice for new database applications. In particular, the simplicity of Codd’s relational model of data where all facts are stored in relations (sets of ordered n-tuples) facilitated data access and optimization for a wide range of application domains (Codd, 1970). Although Codd’s data model was purely set-oriented, industrial relational DBMSs and SQL itself are bag-oriented, since SQL allows keyless tables, and SQL queries queries may return multisets (Melton & Simon, 2002).

Unlike relational databases, network and hierarchic databases store facts in not only record types but also navigation paths between record types. For example, in a hierarchic database the fact that employee 101 works for the Sales department would be stored as a parent-child link from a department record (an instance of the Department record type where the deptName attribute has the value ‘Sales’) to an employee record (an instance of the Employee record type where the empNr attribute has the value 101).

Although relational systems do support foreign key “relationships” between relations, these relationships are not navigation paths; instead they simply encode constraints (e.g. each deptName in an Employee table must also occur in the primary key of the Department table) rather than ground facts. For example, the ground fact that employee 101 works for the Sales department is stored by entering the values 101, ‘Sales’ in the empNr and deptName columns on the same row of the Employee table.

In 1989, a group of researchers published “The Object-Oriented Database System Manifesto” in which they argued that object-oriented databases should replace relational databases (Atkinson et al. 1989). Influenced by object-oriented programming languages, they felt that databases should support not only core databases features such as persistence, concurrency, recovery, and an ad hoc query facility, but also object-oriented features such as complex objects, object identity, encapsulation of behavior with data, types or classes, inheritance (subtyping), overriding and late binding, computational completeness, and extensibility. Databases conforming to this approach are called object-oriented databases (OODBs) or simply object databases (ODBs).

Partly in response to the OODB manifesto, one year later a group of academic and industrial researchers proposed an alternative “3rd generation DBMS manifesto” (Stonebraker et al., 1990). They considered network and hierarchic databases to be first generation, and relational databases to be second generation, and argued that third generation databases should retain the capabilities of relational systems while extending them with object-oriented features. Databases conforming to this approach are called object-relational databases (ORDBs).
While other kinds of databases (e.g. deductive, temporal, and spatial) were also developed to address specific needs, none of these has gained a wide following in industry. Deductive databases typically provide a declarative query language such as a logic programming language (e.g. Prolog), giving them powerful rule enforcement mechanisms with built-in backtracking and strong support for recursive rules (e.g. computing the transitive closure of an ancestor relation).

Spatial databases provide efficient management of spatial data, such as maps (e.g. for geographical applications), 2-D visualizations (e.g. for circuit designs), and 3-D visualizations (e.g. for medical imaging). Built-in support for spatial data types (e.g. points, lines, polygons) and spatial operators (e.g. intersect, overlap, contains) facilitates queries of a spatial nature (e.g. how many residences lie within 3 miles of the proposed shopping center?).

Temporal databases provide built-in support for temporal data types (e.g. instant, duration, period) and temporal operators (e.g. before, after, during, contains, overlaps, precedes, starts, minus), facilitating queries of a temporal nature (e.g. which conferences overlap in time?).

A more recent proposal for database technology employs XML (eXtensible Markup Language). XML databases store data in XML (eXtensible Markup Language), with their structure conforming either to the old DTD (Document Type Definition) or the newer XSD (XML Schema Definition) format. Like the old hierarchic databases, XML is hierarchic in nature. However XML is presented as readable text, using tags to provide the structure. For example, the facts that employees 101 and 102 work for the Sales department could be stored (along with their names and birth dates) in XML as follows.

```xml
<department name = "Sales">
  <employee empNr = "101">
    <name>Fred Smith</name>
    <birthdate>1946-02-15</birthdate>
  </employee>
  <employee empNr = "102">
    <name>Sue Jones</name>
    <birthdate>1980-06-30</birthdate>
  </employee>
</department>
```

Just as SQL is used for querying and manipulating relational data, the XQuery language is now the standard language for querying and manipulating XML data, (Melton & Buxton, 2006).

One very recent proposal for a new kind of database technology is the so-called “ontology database”, which is proposed to help achieve the vision of the semantic web (Berners-Lee et al., 2001). The basic idea is that documents spread over the Internet may include tags to embed enough semantic detail to enable understanding of their content by automated agents. Built on Unicode text, URIrefs (Uniform Resource Identifiers) to identify resources, XML and XSD datatypes, facts are encoded in RDF (Resource Description Framework) triples (subject, predicate, object) representing binary relationships from a node (resource or literal) to another node. RDF Schema (RDFS) builds on RDF by providing inbuilt support for classes and subclassing. The Web Ontology Language (OWL) builds on these underlying layers to provide what is now the most popular language for developing ontologies (schemas and their database instances) for the semantic web.

OWL includes three versions. OWL Lite provides a decidable, efficient mechanism for simple ontologies composed mainly of classification hierarchies and relationships with simple constraints. OWL DL (the “DL” refers to Description Logic) is based on a stronger SHOIN(D) description logic that is still decidable. OWL Full is more expressive but is undecidable, and even goes beyond even first order logic.
All of the above database technologies are still in use, to varying degrees. While some legacy systems still use the old network and hierarchic DBMSs, new database applications are not built on these obsolete technologies. Object databases, deductive databases, and temporal databases provide advantages for niche markets. However the industrial database world is still dominated by relational and object-relational DBMSs. In practice, ORDBs have become the dominant DBMS, since virtually all the major industrial relational DBMSs (e.g. Oracle, IBM DB2, and Microsoft SQL Server) extended their systems with object-oriented features, and also expanded their support for data types including XML. The SQL standard now includes support for collection types (e.g. arrays, row types and multisets, recursive queries and XML). Some ORDBMSs (e.g. Oracle) include support for RDF. While SQL is still often used for data exchange, XML is being increasingly used for exchanging data between applications.

In practice, most applications use an object model for transient (in-memory) storage, while using an RDB or ORDB for persistent storage. This has led to extensive efforts to facilitate transformation between these differently structured data stores (known as Object-Relational mapping). One interesting initiative in this regard is Microsoft’s Language Integrated Query (LINQ) technology, which allows users to interact with relational data by using an SQL-like syntax in their object-oriented program code.

Recently there has been a growing recognition that the best way to develop database systems is by transformation from a high level, conceptual schema that specifies the structure of the data in a way that can be easily understood and hence validated by the (often nontechnical) subject matter experts, who are the only ones who can reliably determine whether the proposed models accurately reflect their business domains.

While this notion of model driven development was forcefully and clearly proposed over a quarter century ago in an ISO standard (van Griethuysen, 1982), only in the last decade has it begun to be widely accepted by major commercial interests. Though called differently by different bodies (e.g. the Object management Group calls it “Model Driven Architecture” and Microsoft promotes model driven development based on Domain Specific Languages) the basic idea is to clearly specify the business domain model at a conceptual level, and then transform it as automatically as possible to application code, thereby minimizing the need for human programming. In the next section we review and contrast three of the most popular approaches to specifying high level data models for subsequent transformation into database schemas.

CONCEPTUAL DATABASE MODELING APPROACHES

In industry, most database designers either use a variant of Entity Relationship (ER) modeling or simply design directly at the relational level. The basic ER approach was first proposed by Chen (1976), and structures facts in terms of entities (e.g. Person, Car) that have attributes (e.g. gender, birthdate) and participate in relationships (e.g. Person drives Car). The most popular industrial versions of ER are the Barker ER notation (Barker, 1990), Information Engineering (IE) (Finkelstein, 1998), and IDEF1X (IEEE, 1999). IDEF1X is actually a hybrid of ER and relational, explicitly using relational concepts such as foreign keys. Barker ER is currently the best and most expressive of the industrial ER notations, so we focus our ER discussion on it.

The Unified Modeling Language (UML) was adopted by the Object Management Group (OMG) in 1997 as a language for object-oriented (OO) analysis and design. After several minor revisions, a major overhaul resulted in UML version 2.0 (OMG, 2003), and the language is still being refined. Although suitable for object-oriented code design, UML is less suitable for information analysis (e.g. even UML 2 does not include a graphic way to declare that an attribute is unique), and its textual Object Constraint
Language (OCL) is too technical for most business people to understand (Warmer & Kleppe, 2003). For such reasons, although UML is widely used for documenting object-oriented programming applications, it is far less popular than ER for database design.

Despite their strengths, both ER and UML are fairly weak at capturing the kinds of business rules found in data-intensive applications, and their graphical language does not lend itself readily to verbalization and multiple instantiation for validating data models with domain experts.

These problems can be remedied by using a fact-oriented approach for information analysis, where communication takes place in simple sentences, each sentence type can easily be populated with multiple instances, attributes are avoided in the base model, and far more business rules can be captured graphically. At design time, a fact-oriented model can be used to derive an ER model, a UML class model, or a logical database model.

Object Role Modeling (ORM), the main exemplar of the fact-oriented approach, originated in Europe in the mid-1970s (Falkenberg, 1976), and been extensively revised and extended since, along with commercial tool support (e.g. Halpin, Evans, Hallock & MacLean, 2003). Recently, a major upgrade to the methodology resulted in ORM 2, a second generation ORM (Halpin 2005; Halpin & Morgan 2008). Neumont ORM Architect (NORMA), an open source tool accessible online at www.ORMFoundation.org, is under development to provide deep support for ORM 2 (Curland & Halpin, 2007).

ORM pictures the world simply in terms of objects (entities or values) that play roles (parts in relationships). For example, you are now playing the role of reading, and this prologue is playing the role of being read. Wherever ER or UML uses an attribute, ORM uses a relationship. For example, the Person.birthdate attribute is modeled in ORM as the fact type Person was born on Date, where the role played by date in this relationship may be given the rolename “birthdate”.

ORM is less popular than either ER or UML, and its diagrams typically consume more space because of their attribute-free nature. However, ORM arguably offers many advantages for conceptual analysis, as illustrated by the following example, which presents the same data model using the three different notations.

In terms of expressibility for data modeling, ORM supports relationships of any arity (unary, binary, ternary or longer), identification schemes of arbitrary complexity, asserted, derived, and semiderived facts and types, objectified associations, mandatory and uniqueness constraints that go well beyond ER and UML in dealing with n-ary relationships, inclusive-or constraints, set comparison (subset, equality, exclusion) constraints of arbitrary complexity, join path constraints, frequency constraints, object and role cardinality constraints, value and value comparison constraints, subtyping (asserted, derived and semiderived), ring constraints (e.g. asymmetry, acyclicity), and two rule modalities (alethic and deontic (Halpin, 2007a)). For some comparisons between ORM 1 and ER and UML see Halpin (2002, 2004).

As well as its rich notation, ORM includes detailed procedures for constructing ORM models and transforming them to other kinds of models (ER, UML, Relational, XSD etc.) on the way to implementation. For a general discussion of such procedures, see Halpin & Morgan (2008). For a detailed discussion of using ORM to develop the data model example discussed below, see Halpin (2007b).

Figure 1 shows an ORM schema for a fragment of a book publisher application. Entity types appear as named, soft rectangles, with simple identification schemes parenthesized (e.g. Books are identified by their ISBN). Value types (e.g. character strings) appear as named, dashed, soft rectangles (e.g. BookTitle). Predicates are depicted as a sequence of one or more role boxes, with at least one predicate reading. By default, predicates are ready left-right or top-down. Arrow tips indicate other predicate reading directions. An asterisk after a predicate reading indicates the fact type is derived (e.g. best sellers are derived using the derivation rule shown). Role names may be displayed in square brackets next to the role (e.g. totalCopiesSold).
A bar over a sequence of one or more roles depicts a uniqueness constraint (e.g. each book has at most one booktitle, but a book may be authored by many persons and vice versa). The external uniqueness constraint (circled bar) reflects the publisher’s policy of publishing at most one book of any given title in any given year. A dot on a role connector indicates that role is mandatory (e.g. each book has a booktitle).

Subtyping is depicted by an arrow from subtype to supertype. In this case, the PublishedBook subtype is derived (indicated by an asterisk), so a derivation rule for it is supplied. Value constraints are placed in braces (e.g. the possible codes for Gender are ‘M’ and ‘F’).

The ring constraint on the book translation fact type indicates that relationship is acyclic. The exclusion constraint (circled X) ensures that no person may review a book that he or she authors. The frequency constraint (≥ 2) ensures that any book assigned for review has at least two reviewers. The subset constraint (circled ⊆) ensures that if a person has a title that is restricted to a specific gender (e.g. ‘Mrs’ is restricted to females), then that person must be of that gender—an example of a constraint on a conceptual join path. The textual declarations provide a subtype definition and two derivation rules, one in attribute style (using role names) and one in relational style. ORM schemas can also be automatically verbalized in natural languages sentences, enabling validation by domain experts without requiring them to understand the notation (Curland & Halpin, 2007).

Figure 2 depicts the same model in Barker ER notation, supplemented by textual rules (6 numbered constraints, plus 3 derivations) that cannot be captured in this notation.

Barker ER depicts entity types as named, soft rectangles. Mandatory attributes are preceded by an asterisk and optional attributes by “o”. An attribute that is part of the primary identifier is preceded by “#”, and a role that is part of an identifier has a stroke “|” through it.

All relationships must be binary, with each half of a relationship line depicting a role. A crowsfoot indicates a maximum cardinality of many. A line end with no crowsfoot indicates a maximum cardinality of one. A solid line end indicates the role is mandatory, and a dashed line end indicates the role is optional. Subtyping is depicted by Euler diagrams with the subtype inside the supertype. Unlike ORM and UML, Barker ER supports only single inheritance, and requires that the subtyping always forms a partition.
Figure 3 shows the same model as a class diagram in UML, supplemented by several textual rules captured either as informal notes (e.g. acyclic) or as formal constraints in OCL (e.g. yearPublished -> notEmpty()) or as nonstandard notations in braces (e.g., the {P} for preferred identifier and {Un} for uniqueness are not standard UML). Derived attributes are preceded by a slash. Attribute multiplicities are assumed to be 1 (i.e. exactly one) unless otherwise specified (e.g. restrictedGender has a multiplicity of [0..1], i.e. at most one). A “*” for maximum multiplicity indicates “many”.
Part of the problem with the UML and ER models is that in these approaches personTitle and gender would normally be treated as attributes, but for this application we need to talk about them to capture a relevant business rule. The ORM model arguably provides a more natural representation of the business domain, while also formally capturing much more semantics with its built-in constructs, facilitating transformation to executable code. This result is typical for industrial business domains.

Figure 4 shows the relational database schema obtained by mapping these data schemas via ORM’s Rmap algorithm (Halpin & Morgan, 2008), using absorption as the default mapping for subtyping. Here square brackets indicate optional, dotted arrows indicate subset constraints, and a circled “X” depicts an exclusion constraint. Additional constraints are depicted as numbered textual rules in a high level relational notation. For implementation, these rules are transformed further into SQL code (e.g. check clauses, triggers, stored procedures, views).

CONCLUSION

While many kinds of database technology exist, RDBs and ORDBs currently dominate the market, with XML being increasingly used for data exchange. While ER is still the main conceptual modeling approach for designing databases, UML is gaining a following for this task, and is already widely used for object oriented code design. Though less popular than ER or UML, the fact-oriented approach exemplified by ORM has many advantages for conceptual data analysis, providing richer coverage of business rules, easier validation by business domain experts, and semantic stability (ORM models and queries are un-impacted by changes that require one to talk about an attribute). Because ORM models may be used to generate ER and UML models, it may also be used in conjunction with these if desired.
With a view to providing better support at the conceptual level, the OMG recently adopted the Semantics of Business Vocabulary and Business Rules (SBVR) specification (OMG, 2007). Like ORM, the SBVR approach is fact oriented instead of attribute-based, and includes deontic as well as alethic rules. Many companies are now looking to model-driven development as a way to dramatically increase the productivity, reliability, and adaptability of software engineering approaches. It seems likely that both object-oriented and fact-oriented approaches will be increasingly utilized in the future to increase the proportion of application code that can be generated from higher level models.

REFERENCES


