medium.com

Predicate Logic — The Magic Roots of Relational Database Management Systems

John Cofield

6–8 minutes

Procedural queries were too complex!

Queries in early database management systems (DBMS) required programmers to write complex programs to retrieve data. They needed to have intimate knowledge of database architecture and hardware implementation. Since processing power was limited and memory was an expensive resource, database specialists focused on optimizing performance within the resource constraints. Usability was an afterthought, if considered at all.

System R started with ALPHA

An IBM computer scientist named Edgar F. Codd authored a series of papers that changed the DBMS game in the early 1970s. Codd introduced a data model intended to isolate data from programs. At the time, database queries were procedural, dependent on the programming language of the host computer. This required the programmer to know details about the hardware. Codd's solution was to introduce a layer of abstraction separating the hardware from the application with a "sublanguage" tailored to database query.

Creating data independence eliminated hardware dependencies for both the data and application logic. Hardware could be changed without affecting the application logic. In addition, the sublanguage could be simplified because the user did not need know hardware details.

Codd implemented data sublanguage ALPHA (DSL ALPHA) in the System R project[1], an experimental relational database management system (RDBMS) at the IBM Research Laboratory in San Jose, CA.

Relational calculus

After studying several DBMS approaches, including procedural and algebraic, Codd concluded that relational calculus was the best approach for meeting his goals. In relational calculus[2], Codd applied predicate calculus with tuple variables, subscripts and the concept of range separability. Predicate calculus applies predicate logic to relationships between objects and/or properties. This approach used logic symbols (Example 1) to express these relationships.

∀: Universal quantifier (meaning "for all")
ꓱ: Existential quantifier (meaning "for each")
∧ : AND
∨ : OR
¬: NOT
→: ASSIGN TO

Example 1. Logic symbol examples

Predicate logic applies propositional logic to programming and software engineering. It is the main mathematical tool in programming and software engineering. In logic, a predicate represents properties or relations among objects, and is an expression that resolves to TRUE or FALSE. e.g. Bob is older than Tom. If Bob and Tom were born on the same day, in the same year, then that proposition is true. Codd used mathematical symbols to concisely express predicates in the ALPHA sublanguage to query the System R database.

Obstacles to adoption

While relational calculus resulted in a language that expressed queries very concisely, it did require users with mathematical expertise to understand the symbols. A protege of Codd, Donald Chamberlin, saw the symbol complexity as an obstacle to broad adoption. To solve that problem, Chamberlin and colleague Raymond Boyce developed a natural language query language that preserved the relational concepts from ALPHA — predicate logic and JOIN features — but replaced the symbols with English language keywords. They intended to make database queries more accessible to non-mathematical users. In addition, many of the mathematical symbols are not available on most keyboards.

Chamberlin and Boyce developed **S**tructured **E**nglish **QUE**ry **L**anguage (SEQUEL) which replaced ALPHA in System R. Above and beyond the query features, SEQUEL added database update and administrative tasks.[3] Examples 2 and 3 show a comparison between equivalent queries for a relational calculus query and a SEQUEL query. The source data is from table name `employee` with attributes (columns) `name`, `salary`, and `manager`.

RANGE employee e;
RANGE employee m;
GET w (e.name):∃m((e.manager = m.name)^(e.salary > m.salary))

Example 2. Relational Calculus version[3]

SELECT e.name
FROM employee e, employee m
WHERE e.manager = m.name AND e.salary > m.salary

Example 3. SEQUEL version[3]

In the examples, there are two predicates connected by the logic `AND` operator:

e.manager = m.name

e.salary > m.salary

Note that Example 3 uses the familiar SQL language form that is commonly used in standard SQL relational databases today. The field is specified in the `SELECT` statement, a `JOIN` is performed with the `FROM` clause, and the logical expression with two predicates is in the `WHERE` clause.

Fast forwarding to today, some common types of predicate functions and operators currently used in SQL databases include:

comparison (=, <>, <, >, <=, >=, !=)
BETWEEN
LIKE
EXITS | NOT EXISTS
IN | NOT IN
NULL

SQL still dominates

In the end, Chamberlin's and Boyce's instincts proved to be right about using English language statements instead of symbols. By the late 80's, the English language approach gained popular a great deal of popularity, ultimately being officially adopted as a standard by the ANSI and ISO standard groups. We now refer to Codd's sublanguage concept as a query language.

While relational databases still dominate 78% of the DBMS market as of 2022[4], there are SQL alternatives — sometimes called NoSQL — that are gaining traction for special purposes. A number of leading DBMS companies now offer NoSQL products either as stand-alone products or as mixed mode features in their existing SQL products. The bottom line is that SQL will be around for a long time.

References

1. Chamberlin, Donald D., "System R: Relational Approach to Database

Management”, https://dl.acm.org/doi/pdf/10.1145/320455.320457

2. E.F. Codd, “A Data Base Sublanguage Founded on the Relational Calculus”, Proc. ACM SIGFIDET Workshop on Data Description, Access, and Control, ACM Press, 1971, https://dl.acm.org/doi/10.1145/1734714.1734718

3. Chamberlin, Donald D., “Early History of SQL”, https://ieeexplore.ieee.org/document/6359709

4. Gartner, “Market Share Analysis: Database Management Systems, Worldwide, 2022”, June 2023, https://www.gartner.com/en/documents/4432699

5. CODD, E.F. “A relational model of data for large shared data banks”. Comm. ACM 13, 6 (June 1970), https://dl.acm.org/doi/pdf/10.1145/362384.362685

6. CODD, E.F., “RELATIONAL COMPLETENESS OF DATA BASE SUBLANGUAGES”, 1976, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=6a048dc38250ffce49c5e6a5040b4c91ca05e83d