The Relational Database Development

An Honors Thesis (ID 499)

by

Steven E. Purtee

Thesis Director

Ball State University
Muncie, Indiana
November, 1985

Summer/1986
Database technology is one of the fastest growing areas in the computer and information sciences. With this growth, there also has been an accompanying growth of confusion. Managers and administrators refer to the mass of data from which budgets and decisions are drawn as a database. This description is somewhat correct; but, it is even more limiting. To the database technician, a database is a collection or grouping of physical records that are of similar definition and serve a single general application purpose. But a true database is more than this "grouping." A database, in the most general sense, is a repository of stored information. The information that is stored is both integrated and shared. It is a union of otherwise distinct files which may be shared among users for use in any of a number of ways; this is the database philosophy.

There are several ideas that are maintained in the database philosophy of data processing environments. Some of them according to Ross (1978) are:

- Independence of data between applications;
- Programs and file storage;
- Reduction of data redundancy;
- Improvement of physical access to data;
- Simplification of application programming;
- Consolidation of update procedures;
- Reduction of the need for program maintenance;
Standardization of data definition and documentation.

The result of these ideas, occurring in many database systems on the market, is an improved productivity.

The first published paper on the relational model of databases was written by E.F. Codd in 1970. His paper, "A Relational Model of Data for Large Shared Data Banks," gave Codd credit for the development of the relational model. This paper gave as offspring two of the most widely used families of Relational languages, Relational Calculus and Relational Algebra.

The underlying concept of all relational systems is the table. The table, with its row/column construction, is an excellent mechanism for understanding relational systems. The following is an example of a "courses-taken" relation from a school data model (Ross, 1978).

<table>
<thead>
<tr>
<th>Courses-Taken</th>
<th>Student Number</th>
<th>Student Name</th>
<th>Courses-Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1016</td>
<td>Frank Mann</td>
<td>History</td>
</tr>
<tr>
<td></td>
<td>1099</td>
<td>Sam Dunn</td>
<td>English, Social Science</td>
</tr>
<tr>
<td></td>
<td>1861</td>
<td>Bo Russ</td>
<td>Algebra, History</td>
</tr>
<tr>
<td></td>
<td>1902</td>
<td>Seth Johnson</td>
<td>Social Science</td>
</tr>
<tr>
<td></td>
<td>2006</td>
<td>John Kitt</td>
<td>Geology</td>
</tr>
</tbody>
</table>

This table has typical structure. There are three columns each with a heading. The elements in each column of the table are referred to as attributes. The heading of each column describes the meaning of the attribute to the table. Each row in the table describes something about which information
is kept. Every row in the table contains similar information, but no two rows contain the same information. Each row in the table is referred to as a tuple. In mathematics, any structure like a table can be treated as a relation. If certain conditions for the table hold true, then mathematical relational theory can be used in relational processing. The following are some basic rules for relational tables that will provide a significant theoretical foundation (Mayne, 1983):

- within a relational system the table must contain only one type of tuple. Each tuple has a fixed number of attributes, all of which are explicitly named;

- within a table the attributes are distinct, and repeating groups are not allowed;

- each tuple within a table is unique; there are no duplicate tuples;

- the order of the tuples is insignificant and need not be maintained;

- the attributes within any column take their values from a domain of possible attribute values;

- new tables may be produced from matching attribute values from the same domain in two existing tables. A domain is a group of values from which the actual attribute values in a column are taken.

The basic rules as given are sufficient, but further mathematical definition is needed as given by Date (1981):

**Definition:** Given a collection of sets $D_1, D_2, \ldots, D_n$, not necessarily distinct, $R$ is a relation on those $n$ sets if it is a set of ordered $n$-tuples $\langle d_1, d_2, \ldots, d_n \rangle$ such that $d_1$ belongs to $D_1$, $d_2$ belongs to $D_2$, \ldots, and $d_n$ belongs to $D_n$. Sets $D_1, D_2, \ldots, D_n$ are the domains of relation $R$. The value $n$ is the degree of $R$. 

Our "courses-taken" example has a degree of three. The three domains are sets of values representing Student Numbers, Student Names, and Courses-Taken. There may exist particular values of the domains that do not appear in the table at this time.

It is a standard convention that a relation be physically ordered by a candidate key. A candidate key is one or more data item types, columns, whose values uniquely identify tuples of the relation. In the "courses-taken" example, Student Number is the candidate key. A relation may also have more than one candidate key. One is usually referred to as the primary key, while others may be referred to as secondary or tertiary keys. A foreign key is one or more data types, columns, whose values correspond to those of a primary key, or any candidate key, within another relation. Foreign keys are essential in the fact that they are the basic building blocks for effecting data interrelationships in the relational model. When working with or creating foreign keys there are three important points to remember:

- a foreign key need not be a candidate key of its own relation;

- the column headings for the foreign key and the corresponding candidate key do not need to be the same;

- neither candidate keys nor foreign keys imply the existence of any physical structure for storing relations. The relational database may be supported by a number of physical devices which may remain unnoticed by the user.
Relational Languages

There are three broad groups of relational languages that support relational systems; they are relational algebras and relational calculus as defined by Codd (1970, 1971), and also query and display languages.

Relational Algebra

Relational algebra is a collection of operations on relations, with each operation taking on one or more relation sets as its operand and producing another operand as its result. Codd originally defined a set of these operations in 1972, and showed that those operations were "relationally complete."
The classical relational algebra operations are those of union, intersection, difference, and Cartesian product. Along with these relational algebra operations there are also three fundamental relational operators, which are:

Selection - which creates a subset of all tuples in a table;

Projection - which creates a subset of all attributes in a table;

Join - which combines two tables forming one.

For all traditional set operations except the Cartesian product, the two operand relations must be "union-compatible;" that is, they must be of the same degree n and the jth attributes of the two relations, j in the range 1 to n, must be drawn from the same domain, though they need not have the same name. The UNION of two relations A and B, A UNION B, is the set of all tuples belonging to either relation A or relation B, or
both relations, represented by t. The INTERSECTION of two relations A and B, A INTERSECT B, is the set of all tuples t belonging to both relation A and relation B. The difference between two relations A and B, A MINUS B, is the set of all tuples t belonging to relation A and not to relation B. In this particular operation the order of the operands, relation A and relation B, is extremely important. Just as in arithmetic A MINUS B is not the same as B MINUS A, just as 6 - 8 is not the same as 8 - 6. In the union, intersection, and difference relational algebra operations, the resulting relation has the same attribute headings as the first operand.

The Cartesian product of two relations A and B, A TIMES B, is the set of all tuples t such that t is the concatenation of a tuple a belonging to relation A and a tuple b belonging to relation B. The concatenation of a tuple a=(a1,a2,...,am) and a tuple b=(bm+1,...,bm+n), in that order, is the tuple t=(a1,...,am,bm+1,...,bm+n). In the Cartesian product relational algebra operation, attributes of a derived relation can have non-unique unqualified names, but their qualified names must be unique. To form the Cartesian product of a relation R with itself, the expression R TIMES R is illegal, because it would violate the unique naming rule. Therefore, we must introduce an alias, R1:

R1 ALIASES R.

With R1 named as an alias, R1 TIMES R, or R TIMES R1, will generate the required product without violating the unique
naming rule (Date, 1981).

As for the relational operators, selection is the simplest operation. Selection simply selects certain tuples from the original table producing a new table, using some selection criteria to pick the tuples. If a specific attribute value is used as the criteria for selection, then the new table produced could have all, some, or possibly none of the tuples from the original table. The resulting table depends upon the attribute value involved in the selection criteria. Selection acts on the rows of the original table, while projection acts on the columns of the table. In the resulting table that is projected, there might be some duplicate tuples, because two tuples from the original table might have differed only by columns that have been eliminated in the projection. Since duplicate tuples are not allowed in a relational table, only one copy of such tuples is retained. The result of a projection usually contains fewer columns or attributes than the original table, since projection on all columns would simply copy the table. It will also usually contain fewer tuples than the original table, because duplicate tuples will normally be eliminated. Some implementations allow tuples, while others make it optional. Most database systems allow the selection and projection operations to be combined into a single operation. The combination of the two operations allows one to select from a table both the tuples and the attributes.
Since one often needs to combine information from different tables, the relational join operator is introduced. The join operator acts to create a new table from two already existing tables. There are several kinds of join, of which the natural join is the most common. The join is done by matching the attribute values occurring in a pair of attributes, or if you wish columns, one from each table. The two columns usually represent some similar attribute such as "employee number." Whatever columns are chosen, the values for the matching pair of attributes will be drawn from the same domain. The join operation produces a new table containing records formed by concatenating records from the two original tables, with duplicates removed. The two tables that participate in the join may have different numbers of tuples and different numbers of attributes. As previously stated, the two tables are usually joined on a similar pair of attributes, whose attribute values are drawn from a common domain. Although the attribute values are drawn from the same domain, the join does not need to have the values of the equality be equal. There are other forms of the join operation depending upon the implementation. The join operation works in theory as follows (Mayne, 1983):

- the first row of the first table is selected and the value is extracted from the column being used for the join;

- the other table is then examined on the matching column, taking records one by one until a match is found between the values in the two columns;
- when a match is found a new record is created for the new table. This record is formed by joining together the rows from the two original tables;

- the process continues until the end of the second table is reached;

- then the process begins again, taking the next row from the first table scanning all the rows of the second table, until the first table has been completed.

The new table might contain identical rows initially, but according to our duplicate rule all but one of the identical rows are eliminated.

In order to keep clarity in the meaning of the relational table, the two columns used for the join operation should contain comparable field values. The method of the join operation previously described would be very time consuming if it were implemented in that fashion. Most modern database systems use techniques for all operations, that optimize the actual time taken to process such queries.

Relational Calculus

The relational calculus is simply a notation for expressing the definition of a new relation in terms of existing tables. Relational calculus expresses the result of a query in terms of relations that are already formed. Queries in the relational calculus languages are of the form:

\[
\langle \text{target} \rangle \text{ where } \langle \text{predicate} \rangle
\]

in which \( \langle \text{target} \rangle \) specifies the relations and attributes required and \( \langle \text{predicate} \rangle \) gives the properties or conditions that must be satisfied for a proper query.
As an example of relational calculus notation, the expression to find the names and reference numbers of customers in Cincinnati is as follows:

\{\text{Customer.Name, Customer.Reference}: \text{Customer.City} = 'Cincinnati'\}

The braces indicate that the expression is a set definition for a new table or relation. The two attributes of the relation are Customer.Name and Customer.Reference, which occur in the expression inside parentheses. The colon stands for 'where.' The property or condition to be satisfied, the predicate, is that Customer.City should be equal to 'Cincinnati'.

The relational calculus languages use the idea of a variable which replaces, as a place-holder, the table or relation name it will range over. These place-holders are called tuple-variables. Date (1981) offers a good elemental definition of tuple-variables:

Tuple-variables $T, U, V, \ldots \ldots \text{Each tuple-variable is constrained to range over some named relation or over a union of relations( In the union case the relations concerned must not only be union-compatible but must also have identical attribute names). If tuple-variable } T \text{ represents tuple } t, \text{ then the expression } T.A \text{ represents the } A\text{-component of } t, \text{ at that time, where } A \text{ is an attribute of the relation over which } T \text{ ranges.}$

In the customer relation we can do selection of names of customers in Cincinnati by the expression:

\{\text{Customer.Name}:\text{Customer.City} = 'Cincinnati'\}

Using the tuple-variable $X$ which ranges over Customer to represent the Customer relation changes the previous expression to:
\{(X.Name): X.City = 'Cincinnati'}

An important reason for using tuple-variables is that in the relational calculus languages a single query may require the use of one table in more than one way. The distinct multiple uses of the relation must be distinguished in the calculus notation.

An example taken from Mayne(1983) expertly displays the use of multiple tuple-variables ranging over the same relation. For instance, "Get the employee number of every employee who earns more than his manager" could be incorporated into a table showing employee number, employee's manager, and salary. There are several attributes needed to satisfy the query; they are the employee themself, and from the same table the employee's manager in order to find the manager's salary. The employee relation is defined as:

\text{STAFF(EMP#:&E#, ENAME:Name, MGR:&E# ...)}

representing the following table:

<table>
<thead>
<tr>
<th>STAFF</th>
<th>EMP#</th>
<th>ENAME</th>
<th>MGR</th>
</tr>
</thead>
<tbody>
<tr>
<td>EO1</td>
<td>Jones</td>
<td>EO2</td>
<td></td>
</tr>
<tr>
<td>EO2</td>
<td>Smith</td>
<td>E1C</td>
<td></td>
</tr>
<tr>
<td>EO3</td>
<td>Brown</td>
<td>EO4</td>
<td></td>
</tr>
<tr>
<td>EO4</td>
<td>Ellis</td>
<td>E19</td>
<td></td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Range of X is STAFF. This means that X is a tuple-variable that ranges over STAFF. The expression \text{X.EMP#} will refer to the employee serial number of some record or tuple in the STAFF table. For the "salary query" we need another element in the table for the employee salary. With this
attribute included, the STAFF relation has changed to the following:

\[
\text{STAFF}(\text{EMP#}, \text{ENAME}, \text{MGR}, \text{SAL})
\]

\(X\) and \(XM\) will be the two tuple-variables for the tuples of the desired employees and their managers. The symbol \(\exists\) is read "there exists."

\(X\) range \(\text{STAFF}\)
\(XM\) range \(\text{STAFF}\)

The expression for the required set is:

\[\{X.\text{EMP#}\} \text{ where } (XM.\text{EMP#} = X.\text{MGR} \text{ and } XM.\text{SAL} < X.\text{SAL})\]

Another approach to relational calculus, besides tuple-oriented relational calculus, is domain-oriented relational calculus. A domain-variable ranges over one of the database domains in the same way a tuple-variable ranges over a complete tuple. Such a variable is a place holder for a value from a database domain:

Range of \(X\) is \(E\#\)

means that \(X\) is a domain-variable whose values are employee serial numbers.

The difference between domain calculus and tuple calculus lies basically in how the user perceives the database. For the STAFF database, the tuple calculus encourages the user to think in terms of four entity types:

\[(\text{EMP#}, \text{ENAME}, \text{MGR}, \text{SAL})\]

each having various properties. Yet domain calculus encourages the user to think in terms of many more entity types:
(E#, Name, Salary, Position, Dept, ...)
and to see the relations as representing various associations
among those entity types. Ullman(1979) proved that any
relational algebra expression can be converted into an ex-
pression of the tuple calculus form. He also showed that
any tuple calculus expression can be converted to an equivalent
domain calculus expression, and that any domain calculus
expression can be converted to an equivalent relational algebra
expression. The three languages are therefore all equivalent,
but not equal, to each other in their selective power.

Query By Example

Query by example was originally developed by M.M. Zloof
at the IBM Yorktown Heights Research Laboratory in 1975.
Query by example, or QBE, is used to refer to both the system
itself and to the user language the system supports. QBE is
designed for use with visual display terminals, with its
primary design being geared toward ease of use rather than
mathematical elegance. The name 'Query By Example' stems
from the fact that examples are used in most operations.
The basic idea is that the user creates the formula of the
query by entering an example of the answer of a possible
query in the correct place in the table. One or more tables
are used in the query by multiple inputs by the user and the
system. The user inputs parts for the query that will direct
the search for the proper answers and define the answers that
are wanted. The normal build-up procedure is described as
follows, according to Date(1981).

The user begins by creating a skeleton table. This is usually done with the use of a certain function key on the terminal, which will display the blank table. Within the skeleton table the user must identify the table to be selected for the query. The user enters the table name (X) followed by "P." next to the row where column names are to be placed. QBE responds to the "P." by filling in the blanks appropriately. The "P." stands for "Print." The "P." indicates the target of the query. The target of the query is the values that are to appear in the result.

<table>
<thead>
<tr>
<th>X</th>
<th>EMP#</th>
<th>ENAME</th>
<th>MGR</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.E1</td>
<td></td>
<td></td>
<td>E02</td>
</tr>
</tbody>
</table>

"E1" is an example element, which are indicated by underlining and are examples of possible answers to the query. "E02" is a constant element. The E02 indicates that the answer must be "E02." The query would read, "Print all EMP# values where the corresponding MGR.E# is E02." Choice of an example element is completely arbitrary; it is merely a domain variable, and may actually be omitted completely. To obtain information about a particular attribute, "P." must be placed in the appropriate column. To obtain information about combinations of attributes one needs to place "P." under the table name column. In more complex queries, other than those previously
explained, example elements are used to establish connections between rows. QBE has a number of important features. If a column is too narrow, columns need to be added, rows need to be added, columns need to be deleted, or rows need to be deleted, and so on, additional function keys are provided. Although the language was developed at IBM laboratories, certain features are not supported in the IBM implementation. For further readings on QBE I suggest Date(1981) or for more extensive readings Sandbury(1981).

SQL

The most widely used relational language is probably SQL, Standard Query Language. Unlike QBE, SQL does not automatically eliminate redundant duplicates from the result of a query. The basic form of the SQL query construct is the SELECT - FROM - WHERE command. If one wanted to know the name of employee number 11 from a relational table named employees. The SQL command would be:

```sql
SELECT ENAME
FROM EMPLOYEES
WHERE E# = 11
```

The SELECT clause specifies the names of the attributes from which selection is to be made. The FROM clause specifies the table from which the attributes are to be selected. And the WHERE clause specifies the condition to be met to yield the correct result. Although explicit attribute selection can be made, an asterisk can be used to imply that all the attributes
in detail are needed. Since no compound selections from
different tables are available with the previous construct,
more specification is needed. Compound specification is done
in the WHERE clause. There are basically two ways to obtain
compound table queries.

The first way is by using the nested SELECT-FROM-WHERE
command. The construct format for this is as follows:

```
SELECT attributes of A
FROM table A
WHERE table A attribute =
  (SELECT table B attribute
   FROM table B
   WHERE table B condition)
```

The previous command actually selects the attributes in table
B that meet the second condition specified. From this, the
result is then matched with a corresponding table A attribute.
The new result is selected where the match occurs. The SELECTed
final result is the attributes of A specified within the
previous result. In order for the SQL language to operate
correctly, the attributes chosen for matching need to be on
the same domain.

The second way to produce compound table queries is the
joining method. In this method, the WHERE clause creates a
link between the two tables by specifying the columns that
are to be matched.

```
SELECT columns (from table A and table B)
FROM table A, table B
WHERE column from A = column from B
```
For further readings on SQL I suggest Date (1981) and Chamberlin (1980).

Now that we have a good background on database systems, I will discuss some of the systems that are in implementation today.

**QBE**

Query By Example is an IBM produced database management system which runs under the VM/370 operating system. QBE, the language, is based on a graphical representation of tables of data as presented previously. QBE supports the following built-in functions for counting and performing arithmetic functions on numeric data:

- CNT
- SUM
- AVG
- MAX
- MIN

The meaning of these functions is clear from their names.

A useful concept, QBE makes use of a Command Box and a Condition Box. The Command Box is used to direct output to either a printer or a specified disk file. It also allows for a query to be stored and for a stored query to be executed. The Condition Box allows complex retrieval conditions to be specified for multi-column conditions as well as logical expressions. These two "boxes" make the language severely more powerful than without and were not mentioned before in order to avoid any unnecessary confusion. One major advantage of QBE is that it allows the user to construct the query in
a way which seems natural to the user. The order of rows created within a query table is entirely arbitrary. Take the previous QBE example for instance, "Print all EMP# values where the corresponding MGR.E# is EO2." The user may think about the query as follows:

Pick out all the manager E#'s equal to EO2, then pick out all the employee NAME's working under these managers, then pick out the corresponding employee E#'s.

If this pseudo-query was used, then the user would probably complete the query tables in the order: MGR, ENAME, EMP#.

Even if the user used a different pseudo-query it would make no difference, the output is the same. QBE is a highly non-procedural language that is capable of supporting several different ways of thinking; not forcing each user to look at and think about a query in the same way.

ORACLE

ORACLE is a relational database management system which is produced by Relational Software Inc. The software will run on DEC's, PDP, and VAX computers under VMS, RSX, IAS, UNIX, and RSTS operating systems. The main language interface supported by ORACLE uses SQL, a language developed by IBM for their "System R" and published in 1976.

ORACLE is designed to support operations on tree-structured tables. This means that there is a recursive relationship in the table. An ORACLE database may be accessed by using COBOL, FORTRAN, PL/1, C, and any other language which supports
a Call facility. All of the functions of SQL are available to the programmer when using one of these languages.

With ORACLE's UFI, User Friendly Interface, allowing user terminal command execution, IAF, Interactive Application Facility, allowing interactive applications to be defined and processed, and RPT, a report writer used in connection with RPF, a report formatter, allowing specifications of headings, control breaks, and a definition of the required database, ORACLE is a database system with multiple uses and areas of application.

Conclusion

There has been a considerable amount of work on database systems in the last ten years. Originally, the systems developed were far from what they are today. The latest objective of database research is to have each attribute and relation retain an innate meaning during processing. With the groundwork set for a relational database approach, the following are the objectives of Codd's (1974) relational approach which should be considered whenever reviewing or designing a database system:

1. Provide a high degree of data independence;

2. Provide a community view of data, so that a wide variety of users may interact with a common view, as well as an individualized user's views;

3. Simplify the job of database administrator;

4. Introduce a theoretical foundation for database management;
5. Merge the statistical and managerial fields for later commercial use;

6. Raise database application programming to a level in which relations are treated as operands.
Literature Cited


Codd, E. F. 1971. A Data Base Sublanguage Founded on the Relational Calculus. ACM SIGFIDET Workshop on Data Description, Access, and Control.


Codd, E. F. 1974. Understanding Relations, Installment no. 4. SIGMOD bulletin FDT 6, no. 4.


