Relational model

The **relational model** (**RM**) for database management is an approach to managing data using a structure and language consistent with first-order predicate logic, first described in 1969 by Edgar F. Codd,[1][2] where all data is represented in terms of tuples, grouped into relations. A database organized in terms of the relational model is a relational database.

The purpose of the relational model is to provide a declarative method for specifying data and queries: users directly state what information the database contains and what information they want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for answering queries.

Most relational databases use the SQL data definition and query language; these systems implement what can be regarded as an engineering approximation to the relational model. A *table* in an SQL database schema corresponds to a predicate variable; the contents of a table to a relation; key constraints, other constraints, and SQL queries correspond to predicates. However, SQL databases deviate from the relational model in many details, and Codd fiercely argued against deviations that compromise the original principles.[4]

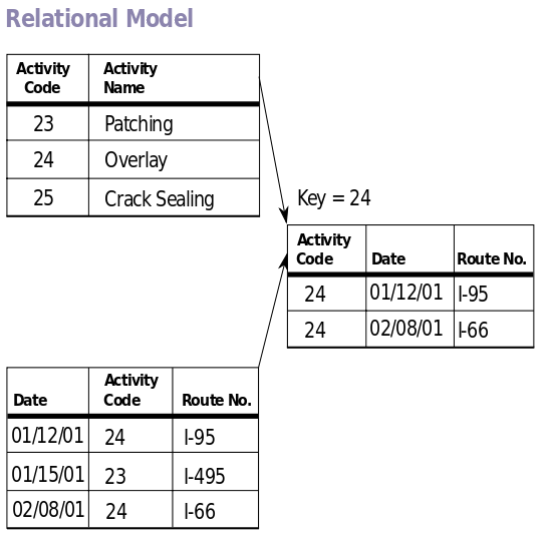
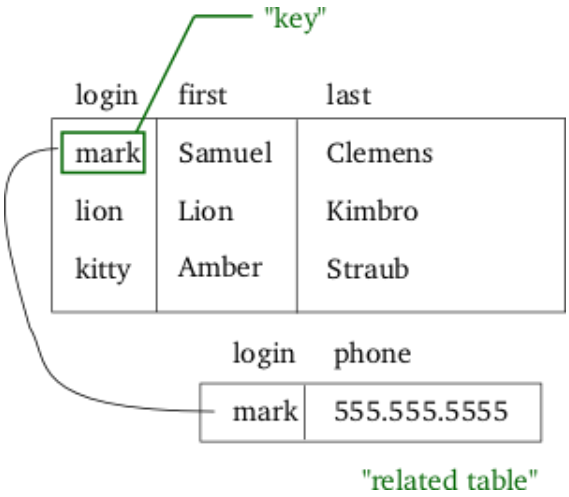


Diagram of an example database according to the relational model[3]

Contents



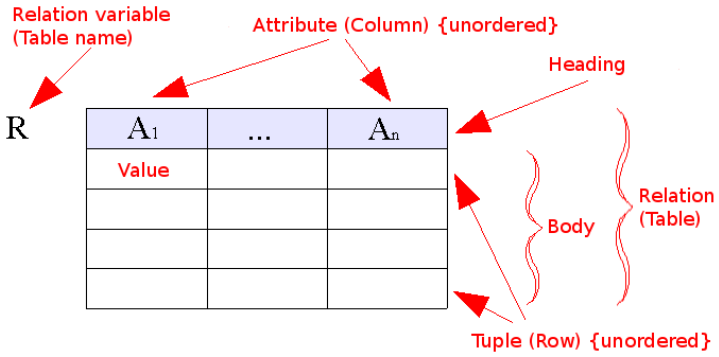
In the relational model, related records are linked together with a "key".

Overview

The relational model's central idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values. The content of the database at any given time is a finite (logical) model of the database, i.e. a set of relations, one per predicate variable, such that all predicates are satisfied. A request for information from the database (a database query) is also a predicate.

Alternatives

Other models are the hierarchical model and network model. Some systems using these older architectures are still in use today in data centers with high data volume needs, or where existing systems are so complex and abstract it would be cost-prohibitive to migrate to systems employing the relational model. Also of note are newer object-oriented databases.



Relational model concepts.

Implementation

There have been several attempts to produce a true implementation of the relational database model as originally defined by Codd and explained by Date, Darwen and others, but none have been popular successes so far. As of October 2015 Rel is one of the more recent attempts to do this.

The relational model was the first database model to be described in formal mathematical terms. Hierarchical and network databases existed before relational databases, but their specifications were relatively informal. After the relational model was defined, there were many attempts to compare and contrast the different models, and this led to the emergence of more rigorous descriptions of the earlier models; though the procedural nature of the data manipulation interfaces for hierarchical and network databases limited the scope for formalization.

History

The relational model was invented by Edgar Codd as a general model of data, and subsequently promoted by Chris Date and Hugh Darwen among others. In The Third Manifesto (first published in 1995) Date and Darwen attempt to show how the relational model can allegedly accommodate certain "desired" object-oriented features.

Controversies

Codd himself, some years after publication of his 1970 model, proposed a three-valued logic (True, False, Missing/NULL) version of it to deal with missing information, and in his *The Relational Model for Database Management Version 2* (1990) he went a step further with a four-valued logic (True, False, Missing but Applicable, Missing but Inapplicable) version.[5] But these have never been implemented, presumably because of attending complexity. SQL's NULL construct was intended to be part of a three-valued logic system, but fell short of that due to logical errors in the standard and in its implementations.[6]

Topics

The fundamental assumption of the relational model is that all data is represented as mathematical *n*-ary **relations**, an *n*-ary relation being a subset of the Cartesian product of *n* domains. In the mathematical model, reasoning about such data is done in two-valued predicate logic, meaning there are two possible evaluations for each proposition: either *true* or *false* (and in particular no third value such as *unknown*, or *not applicable*, either of which are often associated with the concept of NULL). Data are operated upon by means of a relational calculus or relational algebra, these being equivalent in expressive power.

The relational model of data permits the database designer to create a consistent, logical representation of information. Consistency is achieved by including declared ***constraints*** in the database design, which is usually referred to as the logical schema. The theory includes a process of database normalization whereby a design with certain desirable properties can be selected from a set of logically equivalent alternatives. The access plans and other implementation and operation details are handled by the DBMS engine, and are not reflected in the logical model. This contrasts with common practice for SQL DBMSs in which performance tuning often requires changes to the logical model.

The basic relational building block is the domain or data type, usually abbreviated nowadays to ***type***. A *tuple* is an ordered set of ***attribute values***. An attribute is an ordered pair of ***attribute name*** and ***type name***. An attribute value is a specific valid value for the type of the attribute. This can be either a scalar value or a more complex type.

A relation consists of a ***heading*** and a ***body***. A heading is a set of attributes. A body (of an *n*-ary relation) is a set of *n*-tuples. The heading of the relation is also the heading of each of its tuples.

A relation is defined as a set of *n*-tuples. In both mathematics and the relational database model, a set is an *unordered* collection of unique, non-duplicated items, although some DBMSs impose an order to their data. In mathematics, a tuple has an order, and allows for duplication. E. F. Codd originally defined tuples using this mathematical definition.[2] Later, it was one of E. F. Codd's great insights that using attribute names instead of an ordering would be so much more convenient (in general) in a computer language based on relations. This insight is still being used today. Though the concept has changed, the name "tuple" has not. An immediate and important consequence of this distinguishing feature is that in the relational model the Cartesian product becomes commutative.

A table is an accepted visual representation of a relation; a tuple is similar to the concept of a *row*.

A *relvar* is a named variable of some specific relation type, to which at all times some relation of that type is assigned, though the relation may contain zero tuples.

The basic principle of the relational model is the Information Principle: all information is represented by data values in relations. In accordance with this Principle, a relational database is a set of relvars and the result of every query is presented as a relation.

The consistency of a relational database is enforced, not by rules built into the applications that use it, but rather by *constraints*, declared as part of the logical schema and enforced by the DBMS for all applications. In general, constraints are expressed using relational comparison operators, of which just one, "is subset of" (⊆), is theoretically sufficient. In practice, several useful shorthands are expected to be available, of which the most important are candidate key (really, superkey) and foreign key constraints.

Interpretation

To fully appreciate the relational model of data it is essential to understand the intended *interpretation* of a relation.

The body of a relation is sometimes called its extension. This is because it is to be interpreted as a representation of the extension of some predicate, this being the set of true propositions that can be formed by replacing each free variable in that predicate by a name (a term that designates something).

There is a one-to-one correspondence between the free variables of the predicate and the attribute names of the relation heading. Each tuple of the relation body provides attribute values to instantiate the predicate by substituting each of its free variables. The result is a proposition that is deemed, on account of the appearance of the tuple in the relation body, to be true. Contrariwise, every tuple whose heading conforms to that of the relation, but which does not appear in the body is deemed to be false. This assumption is known as the closed world assumption: it is often violated in practical databases, where the absence of a tuple might mean that the truth of the corresponding proposition is unknown. For example, the absence of the tuple ('John', 'Spanish') from a table of language skills cannot necessarily be taken as evidence that John does not speak Spanish.

For a formal exposition of these ideas, see the section Set-theoretic Formulation, below.

Application to databases

A **data type** as used in a typical relational database might be the set of integers, the set of character strings, the set of dates, or the two boolean values *true* and *false*, and so on. The corresponding **type names** for these types might be the strings "int", "char", "date", "boolean", etc. It is important to understand, though, that relational theory does not dictate what types are to be supported; indeed, nowadays provisions are expected to be available for *user-defined* types in addition to the *built-in* ones provided by the system.

Attribute is the term used in the theory for what is commonly referred to as a **column**. Similarly, **table** is commonly used in place of the theoretical term **relation** (though in SQL the term is by no means synonymous with relation). A table data structure is specified as a list of column definitions, each of which specifies a unique column name and the type of the values that are permitted for that column. An **attribute *value*** is the entry in a specific column and row, such as "John Doe" or "35".

A **tuple** is basically the same thing as a **row**, except in an SQL DBMS, where the column values in a row are ordered. (Tuples are not ordered; instead, each attribute value is identified solely by the **attribute name** and never by its ordinal position within the tuple.) An attribute name might be "name" or "age".

A **relation** is a **table** structure definition (a set of column definitions) along with the data appearing in that structure. The structure definition is the **heading** and the data appearing in it is the **body**, a set of rows. A database **relvar** (relation variable) is commonly known as a **base table**. The heading of its assigned value at any time is as specified in the table declaration and its body is that most recently assigned to it by invoking some **update operator** (typically, INSERT, UPDATE, or DELETE). The heading and body of the table resulting from evaluation of some query are determined by the definitions of the operators used in the expression of that query. (Note that in SQL the heading is not always a set of column definitions as described above, because it is possible for a column to have no name and also for two or more columns to have the same name. Also, the body is not always a set of rows because in SQL it is possible for the same row to appear more than once in the same body.)

SQL and the relational model

SQL, initially pushed as the standard language for relational databases, deviates from the relational model in several places. The current ISO SQL standard doesn't mention the relational model or use relational terms or concepts. However, it is possible to create a database conforming to the relational model using SQL if one does not use certain SQL features.

The following deviations from the relational model have been noted in SQL. Note that few database servers implement the entire SQL standard and in particular do not allow some of these deviations. Whereas NULL is ubiquitous, for example, allowing duplicate column names within a table or anonymous columns is uncommon.

Duplicate rows
: The same row can appear more than once in an SQL table. The same tuple cannot appear more than once in a relation.

Anonymous columns
: A column in an SQL table can be unnamed and thus unable to be referenced in expressions. The relational model requires every attribute to be named and referenceable.

Duplicate column names
: Two or more columns of the same SQL table can have the same name and therefore cannot be referenced, on account of the obvious ambiguity. The relational model requires every attribute to be referenceable.

Column order significance
: The order of columns in an SQL table is defined and significant, one consequence being that SQL's implementations of Cartesian product and union are both noncommutative. The relational model requires there to be no significance to any ordering of the attributes of a relation.

Views without CHECK OPTION
: Updates to a view defined without CHECK OPTION can be accepted but the resulting update to the database does not necessarily have the expressed effect on its target. For

example, an invocation of INSERT can be accepted but the inserted rows might not all appear in the view, or an invocation of UPDATE can result in rows disappearing from the view. The relational model requires updates to a view to have the same effect as if the view were a base relvar.

Columnless tables unrecognized

SQL requires every table to have at least one column, but there are two relations of degree zero (of cardinality one and zero) and they are needed to represent extensions of predicates that contain no free variables.

NULL

This special mark can appear instead of a value wherever a value can appear in SQL, in particular in place of a column value in some row. The deviation from the relational model arises from the fact that the implementation of this *ad hoc* concept in SQL involves the use of three-valued logic, under which the comparison of NULL with itself does not yield *true* but instead yields the third truth value, *unknown*; similarly the comparison NULL with something other than itself does not yield *false* but instead yields *unknown*. It is because of this behaviour in comparisons that NULL is described as a mark rather than a value. The relational model depends on the law of excluded middle under which anything that is not true is false and anything that is not false is true; it also requires every tuple in a relation body to have a value for every attribute of that relation. This particular deviation is disputed by some if only because E. F. Codd himself eventually advocated the use of special marks and a 4-valued logic, but this was based on his observation that there are two distinct reasons why one might want to use a special mark in place of a value, which led opponents of the use of such logics to discover more distinct reasons and at least as many as 19 have been noted, which would require a 21-valued logic. SQL itself uses NULL for several purposes other than to represent "value unknown". For example, the sum of the empty set is NULL, meaning zero, the average of the empty set is NULL, meaning undefined, and NULL appearing in the result of a LEFT JOIN can mean "no value because there is no matching row in the right-hand operand". There are ways to design tables to avoid the need for NULL, typically what may be considered or resemble high degrees of database normalization, but many find such impractical. It can be a hotly debated topic.

Relational operations

Users (or programs) request data from a relational database by sending it a query that is written in a special language, usually a dialect of SQL. Although SQL was originally intended for end-users, it is much more common for SQL queries to be embedded into software that provides an easier user interface. Many Web sites, such as Wikipedia, perform SQL queries when generating pages.

In response to a query, the database returns a result set, which is just a list of rows containing the answers. The simplest query is just to return all the rows from a table, but more often, the rows are filtered in some way to return just the answer wanted.

Often, data from multiple tables are combined into one, by doing a join. Conceptually, this is done by taking all possible combinations of rows (the Cartesian product), and then filtering out everything except the answer. In practice, relational database management systems rewrite ("optimize") queries to perform faster, using a variety of techniques.

There are a number of relational operations in addition to join. These include project (the process of eliminating some of the columns), restrict (the process of eliminating some of the rows), union (a way of combining two tables with similar structures), difference (that lists the rows in one table that are not found in the other), intersect (that lists the rows found in both tables), and product (mentioned above, which combines each row of one table with each row of the other). Depending on which other sources you consult, there are a number of other operators – many of which can be defined in terms of those listed above. These include semi-join, outer operators such as outer join and outer union, and various forms of division. Then there are operators to rename columns, and summarizing or aggregating operators, and if you permit relation values as attributes (RVA – relation-valued attribute), then operators such as group and ungroup. The SELECT statement in SQL serves to handle all of these except for the group and ungroup operators.

The flexibility of relational databases allows programmers to write queries that were not anticipated by the database designers. As a result, relational databases can be used by multiple applications in ways the original designers did not foresee, which is especially important for databases that might be used for a long time (perhaps several decades). This has made the idea and implementation of relational databases very popular with businesses.

Database normalization

Relations are classified based upon the types of anomalies to which they're vulnerable. A database that's in the first normal form is vulnerable to all types of anomalies, while a database that's in the domain/key normal form has no modification anomalies. Normal forms are hierarchical in nature. That is, the lowest level is the first normal form, and the database cannot meet the requirements for higher level normal forms without first having met all the requirements of the lesser normal forms.[7]

Examples

Database

An idealized, very simple example of a description of some relvars (relation variables) and their attributes:

- Customer (**Customer ID**, Tax ID, Name, Address, City, State, Zip, Phone, Email,Sex)
- Order (**Order No**, Customer ID, Invoice No, Date Placed, Date Promised, Terms, Status)
- Order Line (**Order No**, **Order Line No**, Product Code, Qty)
- Invoice (**Invoice No**, Customer ID, Order No, Date, Status)
- Invoice Line (**Invoice No**, **Invoice Line No**, Product Code, Qty Shipped)
- Product (**Product Code**, Product Description)

In this design we have six relvars: Customer, Order, Order Line, Invoice, Invoice Line and Product. The bold, underlined attributes are *candidate keys*. The non-bold, underlined attributes are *foreign keys*.

Usually one candidate key is chosen to be called the primary key and used in preference over the other candidate keys, which are then called alternate keys.

A *candidate key* is a unique identifier enforcing that no tuple will be duplicated; this would make the relation into something else, namely a bag, by violating the basic definition of a set. Both foreign keys and superkeys (that includes candidate keys) can be composite, that is, can be composed of several attributes. Below is a tabular depiction of a relation of our example Customer relvar; a relation can be thought of as a value that can be attributed to a relvar.

Customer relation

Customer ID	Tax ID	Name	Address	[More fields…]
1234567890	555-5512222	Munmun	323 Broadway	…
2223344556	555-5523232	Wile E.	1200 Main Street	…
3334445563	555-5533323	Ekta	871 1st Street	…
4232342432	555-5325523	E. F. Codd	123 It Way	…

If we attempted to *insert* a new customer with the ID *1234567890*, this would violate the design of the relvar since **Customer ID** is a *primary key* and we already have a customer *1234567890*. The DBMS must reject a transaction such as this that would render the database inconsistent by a violation of an integrity constraint.

Foreign keys are integrity constraints enforcing that the value of the attribute set is drawn from a *candidate key* in another relation. For example, in the Order relation the attribute **Customer ID** is a foreign key. A *join* is the operation that draws on information from several relations at once. By joining relvars from the example above we could *query* the database for all of the Customers,

Orders, and Invoices. If we only wanted the tuples for a specific customer, we would specify this using a restriction condition.

If we wanted to retrieve all of the Orders for Customer *1234567890*, we could query the database to return every row in the Order table with **Customer ID** *1234567890* and join the Order table to the Order Line table based on **Order No**.

There is a flaw in our database design above. The Invoice relvar contains an Order No attribute. So, each tuple in the Invoice relvar will have one Order No, which implies that there is precisely one Order for each Invoice. But in reality an invoice can be created against many orders, or indeed for no particular order. Additionally the Order relvar contains an Invoice No attribute, implying that each Order has a corresponding Invoice. But again this is not always true in the real world. An order is sometimes paid through several invoices, and sometimes paid without an invoice. In other words, there can be many Invoices per Order and many Orders per Invoice. This is a **many-to-many** relationship between Order and Invoice (also called a *non-specific relationship*). To represent this relationship in the database a new relvar should be introduced whose role is to specify the correspondence between Orders and Invoices:

```
OrderInvoice (Order No, Invoice No)
```

Now, the Order relvar has a *one-to-many relationship* to the OrderInvoice table, as does the Invoice relvar. If we want to retrieve every Invoice for a particular Order, we can query for all orders where **Order No** in the Order relation equals the **Order No** in OrderInvoice, and where **Invoice No** in OrderInvoice equals the **Invoice No** in Invoice.

Set-theoretic formulation

Basic notions in the relational model are *relation names* and *attribute names*. We will represent these as strings such as "Person" and "name" and we will usually use the variables $r, s, t, \ldots$ and a, b, c to range over them. Another basic notion is the set of *atomic values* that contains values such as numbers and strings.

Our first definition concerns the notion of *tuple*, which formalizes the notion of row or record in a table:

Tuple
: A tuple is a partial function from attribute names to atomic values.

Header
: A header is a finite set of attribute names.

Projection
: The projection of a tuple t on a finite set of attributes A is $t[A] = \{(a, v) : (a, v) \in t, a \in A\}$.

The next definition defines *relation* that formalizes the contents of a table as it is defined in the relational model.

Relation
: A relation is a tuple (H, B) with H, the header, and B, the body, a set of tuples that all have the domain H.

Such a relation closely corresponds to what is usually called the extension of a predicate in first-order logic except that here we identify the places in the predicate with attribute names. Usually in the relational model a database schema is said to consist of a set of relation names, the headers that are associated with these names and the constraints that should hold for every instance of the database schema.

Relation universe
: A relation universe U over a header H is a non-empty set of relations with header H.

Relation schema
: A relation schema (H, C) consists of a header H and a predicate $C(R)$ that is defined for all relations R with header H. A relation satisfies a relation schema (H, C) if it has header H and satisfies C.

Key constraints and functional dependencies

One of the simplest and most important types of relation constraints is the *key constraint*. It tells us that in every instance of a certain relational schema the tuples can be identified by their values for certain attributes.

Superkey

A superkey is written as a finite set of attribute names.
A superkey K holds in a relation (H, B) if:

- $K \subseteq H$ and
- there exist no two distinct tuples $t_1, t_2 \in B$ such that $t_1[K] = t_2[K]$.

A superkey holds in a relation universe U if it holds in all relations in U.
Theorem: A superkey K holds in a relation universe U over H if and only if $K \subseteq H$ and $K \to H$ holds in U.

Candidate key

A superkey K holds as a candidate key for a relation universe U if it holds as a superkey for U and there is no proper subset of K that also holds as a superkey for U.

Functional dependency

A functional dependency (FD for short) is written as $X \to Y$ for X, Y finite sets of attribute names.
A functional dependency $X \to Y$ holds in a relation (H, B) if:

- $X, Y \subseteq H$ and
- $\forall$ tuples $t_1, t_2 \in B$, $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$

A functional dependency $X \to Y$ holds in a relation universe U if it holds in all relations in U.

Trivial functional dependency

A functional dependency is trivial under a header H if it holds in all relation universes over H.
Theorem: An FD $X \to Y$ is trivial under a header H if and only if $Y \subseteq X \subseteq H$.

Closure

Armstrong's axioms: The closure of a set of FDs S under a header H, written as S^+, is the smallest superset of S such that:

- $Y \subseteq X \subseteq H \Rightarrow X \to Y \in S^+$ (reflexivity)
- $X \to Y \in S^+ \wedge Y \to Z \in S^+ \Rightarrow X \to Z \in S^+$ (transitivity) and
- $X \to Y \in S^+ \wedge Z \subseteq H \Rightarrow (X \cup Z) \to (Y \cup Z) \in S^+$ (augmentation)

Theorem: Armstrong's axioms are sound and complete; given a header H and a set S of FDs that only contain subsets of H, $X \to Y \in S^+$ if and only if $X \to Y$ holds in all relation universes over H in which all FDs in S hold.

Completion

The completion of a finite set of attributes X under a finite set of FDs S, written as X^+, is the smallest superset of X such that:

- $Y \to Z \in S \wedge Y \subseteq X^+ \Rightarrow Z \subseteq X^+$

The completion of an attribute set can be used to compute if a certain dependency is in the closure of a set of FDs.
Theorem: Given a set S of FDs, $X \to Y \in S^+$ if and only if $Y \subseteq X^+$.

Irreducible cover

An irreducible cover of a set S of FDs is a set T of FDs such that:

- $S^+ = T^+$
- there exists no $U \subset T$ such that $S^+ = U^+$
- $X \to Y \in T \Rightarrow Y$ is a singleton set and

- $X \to Y \in T \land Z \subset X \Rightarrow Z \to Y \notin S^+$.

Algorithm to derive candidate keys from functional dependencies

```
INPUT: a set S of FDs that contain only subsets of a header H
OUTPUT: the set C of superkeys that hold as candidate keys in
        all relation universes over H in which all FDs in S hold
begin
  C := ∅;          // found candidate keys
  Q := { H };      // superkeys that contain candidate keys
  while Q <> ∅ do
    let K be some element from Q;
    Q := Q − { K };
    minimal := true;
    for each X->Y in S do
      K' := (K − Y) ∪ X;   // derive new superkey
      if K' ⊂ K then
        minimal := false;
        Q := Q ∪ { K' };
      end if
    end for
    if minimal and there is not a subset of K in C then
      remove all supersets of K from C;
      C := C ∪ { K };
    end if
  end while
end
```

See also

- Domain relational calculus
- List of relational database management systems
- Query language
  - Database query language
  - Information retrieval query language
- Relation
- Relational database
- Relational database management system
- The Third Manifesto (TTM)
- Tuple-versioning

References

1. Codd, E.F (1969), *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks*, Research Report, IBM.
2. Codd, E.F (1970). "A Relational Model of Data for Large Shared Data Banks" (http://www.acm.org/classics/nov95/toc.html). *Communications of the ACM*. Classics. **13** (6): 377–87. doi:10.1145/362384.362685 (https://doi.org/10.1145%2F362384.362685).
3. *Data Integration Glossary* (http://knowledge.fhwa.dot.gov/tam/aashto.nsf/All+Documents/4825476B2B5C687285256B1F00544258/$FILE/DIGloss.pdf) (PDF), US: Department of Transportation, August 2001.
4. Codd, E. F (1990), *The Relational Model for Database Management*, Addison-Wesley, pp. 371–388, ISBN 0-201-14192-2.
5. Date, C.J (2006). "18. Why Three- and Four-Valued Logic Don't Work". *Date on Database: Writings 2000–2006*. Apress. pp. 329–41. ISBN 978-1-59059-746-0.
6. Date, C.J (2004). *An Introduction to Database Systems* (8 ed.). Addison Wesley. pp. 592–97. ISBN 0-321-19784-4.
7. David M. Kroenke, *Database Processing: Fundamentals, Design, and Implementation* (1997), Prentice-Hall, Inc., pages 130–144

Further reading

- Date, C. J.; Darwen, Hugh (2000). *Foundation for future database systems: the third manifesto; a detailed study of the impact of type theory on the relational model of data, including a comprehensive model of type inheritance*

(2 ed.). Reading, MA: Addison-Wesley. ISBN 0-201-70928-7.
- ——— (2007). *An Introduction to Database Systems* (8 ed.). Boston: Pearson Education. ISBN 0-321-19784-4.

External links

- Childs (1968), *Feasibility of a set-theoretic data structure: a general structure based on a reconstituted definition of relation* (http://hdl.handle.net/2027.42/4164) (research), Handle cited in Codd's 1970 paper.
- Darwen, Hugh, *The Third Manifesto (TTM)* (http://www.thethirdmanifesto.com/).
- Relational Databases (https://curlie.org/Computers/Software/Databases/Relational) at Curlie (based on DMOZ)
- "Relational Model" (http://c2.com/cgi/wiki?RelationalModel), *C2*.
- *Binary relations and tuples compared with respect to the semantic web* (http://blogs.sun.com/bblfish/entry/why_binary_relations_beat_tuples) (World Wide Web log), Sun.