Relational model [UNIT-II]

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 a 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.[3]

History

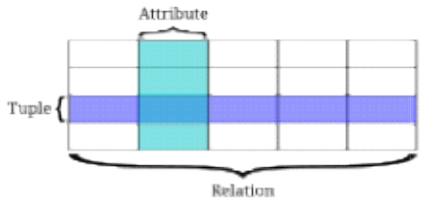
The relational model was developed by Edgar F. Codd as a general model of data, and subsequently promoted by Chris Date and Hugh Darwen among others. In their 1995 *The Third Manifesto*, Date and Darwen try to demonstrate how the relational model can accommodate certain "desired" object-oriented features.[4]

Extensions

Some years after publication of his 1970 model, Codd 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]

Conceptualization

Basic concepts



A relation with 5 attributes (its degree) and 4 tuples (its cardinality) can be visualized as a table with 5 columns and 4 rows. However, unlike rows and columns in a table, a relation's attributes and tuples are unordered.

A *relation* consists of a *heading* and a *body*. The heading defines a set of *attributes*, each with a *name* and *data type* (sometimes called a *domain*). The number of attributes in this set is the relation's *degree* or *arity*. The body is a set of *tuples*. A tuple is a collection of *n values*, where *n* is the relation's degree, and each value in the tuple corresponds to a unique attribute.[6] The number of tuples in this set is the relation's *cardinality*.[Z]:17–22

Relations are represented by *relational variables* or *relvars*, which can be reassigned.[Z]:22–24 A *database* is a collection of relvars.[Z]:112–113

In this model, databases follow the *Information Principle*: At any given time, all information in the database is represented solely by values within tuples, corresponding to attributes, in relations identified by relvars.[Z]:111

Constraints

A database may define arbitrary boolean expressions as constraints. If all constraints evaluate as *true*, the database is *consistent*; otherwise, it is *inconsistent*. If a change to a database's relvars would leave the database in an inconsistent state, that change is illegal and must not succeed.[Z]:91

In general, constraints are expressed using relational comparison operators, of which just one, "is subset of" (⊆), is theoretically sufficient[j]

Two special cases of constraints are expressed as *keys* and *foreign keys*:

Keys

A *candidate key*, or simply a *key*, is the smallest subset of attributes guaranteed to uniquely differentiate each tuple in a relation. Since each tuple in a relation must be unique, every relation necessarily has a key, which may be its complete set of attributes. A relation may have multiple keys, as there may be multiple ways to uniquely differentiate each tuple.[Z]:31–33

An attribute may be unique across tuples without being a key. For example, a relation describing a company's employees may have two attributes: ID and Name. Even if no employees currently share a name, if it is possible to eventually hire a new employee with the same name as a current employee, the attribute subset {Name} is not a key. Conversely, if the subset {ID} is a key, this means not only that no employees *currently* share an ID, but that no employees *will ever* share an ID.[Z]:31–33

Foreign keys

A *foreign key* is a subset of attributes *{A}* in a relation R_1 that corresponds with a key of another relation R_2, with the property that the projection of R_1 on *{A}* is a subset of the projection of R_2 on *{A}*. In other words, if a tuple in R_1 contains values for a foreign key, there must be a corresponding tuple in R_2 containing the same values for the corresponding key.[Z]:34

Relational operations

Users (or programs) request data from a relational database by sending it a query. In response to a query, the database returns a result set.

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.

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 (relation-valued attribute), then operators such as group and ungroup.

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

Database normalization

Relations are classified based upon the types of anomalies to which they're vulnerable. A database that is in the first normal form is vulnerable to all types of anomalies, while a database that is 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.[8]

Logical interpretation

The relational model is a formal system. A relation's attributes define a set of logical propositions. Each proposition can be expressed as a tuple. The body of a relation is a subset of these tuples, representing which propositions are true. Constraints represent additional propositions which must also be true. Relational algebra is a set of logical rules that can validly infer conclusions from these propositions.[7]:95–101

The definition of a *tuple* allows for a unique empty tuple with no values, corresponding to the empty set of attributes. If a relation has a degree of 0 (i.e. its heading contains no attributes), it may have either a cardinality of 0 (a body containing no tuples) or a cardinality of 1 (a body containing the single empty tuple). These relations represent Boolean truth values. The relation with degree 0 and cardinality 0 is *False*, while the relation with degree 0 and cardinality 1 is *True*.[7]:221–223

Example

If a relation of Employees contains the attributes *{Name, ID}*, then the tuple *{Alice, 1}* represents the proposition: "There exists an employee named *Alice* with ID *1*". This proposition may be true or false. If this tuple exists in the relation's body, the proposition

is true (there is such an employee). If this tuple is not in the relation's body, the proposition is false (there is no such employee).[7]:96–97

Furthermore, if *{ID}* is a key, then a relation containing the tuples *{Alice, 1}* and *{Bob, 1}* would represent the following contradiction:

1. There exists an employee with the name *Alice* and the ID *1*.
2. There exists an employee with the name *Bob* and the ID *1*.
3. There do not exist multiple employees with the same ID.

Under the principle of explosion, this contradiction would allow the system to prove that any arbitrary proposition is true. The database must enforce the key constraint to prevent this.[7]:104

Examples

Database

]

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

- Customer (**Customer ID**, Name)
- Order (**Order ID**, Customer ID, Invoice ID, Date)
- Invoice (**Invoice ID**, Customer ID, Order ID, Status)

In this design we have three relvars: Customer, Order, and Invoice. 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	Name
123	Alice

456 Bob

789 Carol

If we attempted to *insert* a new customer with the ID *123*, this would violate the design of the relvar since **Customer ID** is a *primary key* and we already have a customer *123*. The DBMS must reject a transaction such as this that would render the database inconsistent by a violation of an integrity constraint. However, it is possible to insert another customer named *Alice*, as long as this new customer has a unique ID, since the Name field is not part of the primary key.

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 *123*, we could query the database to return every row in the Order table with **Customer ID** *123* .

There is a flaw in our database design above. The Invoice relvar contains an Order ID attribute. So, each tuple in the Invoice relvar will have one Order ID, 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 ID 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 ID, Invoice ID)
```

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 ID** in the Order relation equals the **Order ID** in OrderInvoice, and where **Invoice ID** in OrderInvoice equals the **Invoice ID** in Invoice.

Application to relational databases

A **data type** in a relational database might be the set of integers, the set of character strings, the set of dates, etc. The relational model does not dictate what types are to be supported.

Attributes are commonly represented as **columns**, **tuples** as **rows**, and **relations** as **tables**. A table 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.

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 an **update operator** (typically, INSERT, UPDATE, or DELETE). The heading and body of the table resulting from evaluating a query are determined by the definitions of the operators used in that query.

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.[citation needed]

According to the relational model, a Relation's attributes and tuples are mathematical sets, meaning they are unordered and unique. In a SQL table, neither rows nor columns are proper sets. A table may contain both duplicate rows and duplicate columns, and a table's columns are explicitly ordered. SQL uses a Null value to indicate missing data, which has no analog in the relational model. Because a row can represent unknown information, SQL does not adhere to the relational model's *Information Principle*.[7]:153–155,162

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 and 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 on a finite set of attributes is .

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 with , the header, and , the body, a set of tuples that all have the domain .

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 over a header is a non-empty set of relations with header .

Relation schema

A relation schema consists of a header and a predicate that is defined for all relations with header . A relation satisfies a relation schema if it has header and satisfies .

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 a set of column headers for which the values of those columns concatenated are unique across all rows. Formally:

A superkey is written as a finite set of attribute names.

A superkey holds in a relation if:

- and
- there exist no two distinct tuples such that .

A superkey holds in a relation universe if it holds in all relations in .

Theorem: A superkey holds in a relation universe over if and only if and holds in .

Candidate key

A candidate key is a superkey that cannot be further subdivided to form another superkey.

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

Functional dependency

Functional dependency is the property that a value in a tuple may be derived from another value in that tuple.

A functional dependency (FD for short) is written as for finite sets of attribute names.

A functional dependency holds in a relation if:

- and
- tuples ,

A functional dependency holds in a relation universe if it holds in all relations in .

Trivial functional dependency

A functional dependency is trivial under a header if it holds in all relation universes over .

Theorem: An FD is trivial under a header if and only if .

Closure

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

- (reflexivity)
- (transitivity) and
- (augmentation)

Theorem: Armstrong's axioms are sound and complete; given a header and a set of FDs that only contain subsets of , if and only if holds in all relation universes over in which all FDs in hold.

Completion

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

- 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 of FDs, if and only if .

Irreducible cover

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

-