An Introduction to Relational Database Management System

History

The concept of relational databases was first described by Edgar Frank Codd (almost exclusively referenced as E. F. Codd in technical literature) in the IBM research report RJ599, dated August 19th, 1969. However, the article that is usually considered the cornerstone of this technology is "A Relational Model of Data for Large Shared Data Banks," published in Communications of the ACM(Vol. 13, No. 6, June 1970, pp. 377-87).

Additional articles by E. F. Codd throughout the 1970s and 80s are still considered gospel for relational database implementations. His famous "Twelve Rules for Relational Databases" were published in two Computerworld articles "Is Your DBMS Really Relational?" and "Does Your DBMS Run By the Rules?" on October 14, 1985, and October 21, 1985, respectively. He has since expanded on the 12 rules, and they now number 333, as published in his book "The Relational Model for Database Management, Version 2" (Addison -Wesley, 1990).

The language, SQL, was originally developed in the research division of IBM (initially at Yorktown Heights, N.Y., and later at San Jose, Calif., and Raymond Boyce and Donald Chamberlin were the original designers.) and has been adopted by all major relational database vendors. The name SQL originally stood for Structured Query Language. The first commercially available implementation of the language was named SEQUEL (for Sequential English QUEry Language) and was part of IBM's SEQUEL/DS product. The name was later changed for legal reasons. Thus, many long-time database developers use the pronunciation "see-quell."

SQL has been adopted as an ANSI/ISO standard. Although revised in 1999 (usually referenced as SQL99 or SQL3), most vendors are still not fully compliant with the 1992 version of the standard. The 1992 standard is smaller and simpler to reference for a user, and since only some of the 1999-specific requirements are typically implemented at this time, it may be a better starting point for learning the language.

Introduction

The database design phase is a very important step for all IT projects developing systems that rely on a database to adequately store, query, import & export data and support reporting. For such systems the operation of the database is critical hence its design and implementation must be long lasting, flawless and perfectly tailored to meet the requirements of the system.

The problem with data is that it changes. Not just its individual items' values change, but their structure and use, especially when kept over extended periods of time. Even for public records that may have been kept for hundreds of years, there are occasionally changes in what data elements are captured and recorded and how.

Therefore, a method to avoid problems due to duplication of data values and modification of structure and content has been developed. This method is called normalization.
You normalize a database in order to ensure data consistency and stability, to minimize data redundancy, and to ensure consistent updateability and maintainability of the data, and avoid update and delete anomalies that result in ambiguous data or inconsistent results.

**Some Key Concepts**

**Database**
A database is a collection of data that is organized in a systematic way so that its contents can easily be accessed, managed and updated. The most prevalent type of database is the relational database, a tabular database in which data is defined so that it can be reorganized and accessed in a number of different ways. A distributed database is one that can be dispersed or replicated among different points in a network. The software used to manage and query a database is known as a database management system (DBMS).

**Database Management System**
A Database Management System is a software environment that structures and manipulates data, and ensures data security, recovery, and integrity. The Data Platform relies on a database management system (RDBMS) to store and maintain all of its data as well as execute all the associated queries. There are two types of RDBMS: the first group consists of single software packages which support only a single database, with a single user access and are not scalable (i.e. cannot handle large amounts of data). Typical examples of this first group are MS Access and FileMaker.

The second group is formed by DBMS composed of one or more programs and their associated services which support one or many databases for one or many users in a scalable fashion. For example an enterprise database server can support the HR database, the accounting database and the stocks database all at the same time. Typical examples of this second group include MySQL, MS SQL Server, Oracle and DB2. The DBMS selected for the Data Platform is MS SQL Server from the second group.

**Table**
A table is set of data elements that has a horizontal dimension (rows) and a vertical dimension (columns) in a relational database system. A table has a specified number of columns but can have any number of rows. Rows stored in a table are structurally equivalent to records from flat files. Columns are often referred as attributes or fields. In a database managed by a DBMS the format of each attribute is a fixed datatype. For example the attribute date can only contain information in the date time format.

**Identifier**
An identifier is an attribute that is used either as a primary key or as a foreign key. The integer datatype is used for identifiers. In cases where the number of records exceed the allowed values by the integer datatype then a biginteger datatype is used.

**Primary key**
A column in a table whose values uniquely identify the rows in the table. A primary key value cannot be NULL.

**Foreign key**
A column in a table that does not uniquely identify rows in that table, but is used as a link to matching columns in other tables.
Relationship
A relationship is an association between two tables. For example the relationship between the table "hotel" and "customer" maps the customers to the hotels they have used.

Index
An index is a data structure which enables a query to run at a sublinear-time. Instead of having to go through all records one by one to identify those which match its criteria the query uses the index to filter out those which don't and focus on those who do.

View
A view is a virtual or logical table composed of the result set of a pre-compiled query. Unlike ordinary tables in a relational database, a view is not part of the physical schema: it is a dynamic, virtual table computed or collated from data in the database. Changing the data in a view alters the data stored in the database.

Query
A query is a request to retrieve data from a database with the SQL SELECT instruction or to manipulate data stored in tables.

SQL
Structured Query Language (SQL), pronounced "sequel", is a language that provides an interface to relational database systems. It was developed by IBM in the 1970s for use in System R. SQL is a de facto standard, as well as an ISO and ANSI standard.

Normalization
Normalization is the formalization of the design process of making a database compliant with the concept of a Normal Form. It addresses various ways in which we may look for repeating data values in a table. There are several levels of the Normal Form, and each level requires that the previous level be satisfied. I have used the wording (indicated in italicized text) for each normalization rule from the Handbook of Relational Database Design by Candace C. Fleming and Barbara von Halle.4

The normalization process is based on collecting an exhaustive list of all data items to be maintained in the database and starting the design with a few "superset" tables. Theoretically, it may be possible, although not very practical, to start by placing all the attributes in a single table. For best results, start with a reasonable breakdown.

First Normal Form
Reduce entities to first normal form (1NF) by removing repeating or multivalued attributes to another, child entity.

Basically, make sure that the data is represented as a (proper) table. While key to the relational principles, this is somewhat a motherhood statement. However, there are six properties of a relational table (the formal name for "table" is "relation"): 

- Property 1: Entries in columns are single-valued.
- Property 2: Entries in columns are of the same kind.
- Property 3: Each row is unique.
Property 4: Sequence of columns is insignificant.
Property 5: Sequence of rows is insignificant.
Property 6: Each column has a unique name.

The most common sins against the first normal form (1NF) are the lack of a Primary Key and the use of "repeating columns." This is where multiple values of the same type are stored in multiple columns. Take, for example, a database used by a company's order system. If the order items were implemented as multiple columns in the Orders table, the database would not be 1NF:

<table>
<thead>
<tr>
<th>OrderNo</th>
<th>Line1Item</th>
<th>Line1Qty</th>
<th>Line1Price</th>
<th>Line2Item</th>
<th>Line2Qty</th>
<th>Line2Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>245</td>
<td>PN768</td>
<td>1</td>
<td>Rs. 35</td>
<td>PN656</td>
<td>3</td>
<td>Rs. 15</td>
</tr>
</tbody>
</table>

To make this first normal form, we would have to create a child entity of Orders (Order Items) where we would store the information about the line items on the order. Each order could then have multiple Order Items related to it.

<table>
<thead>
<tr>
<th>OrderNo</th>
<th>Item</th>
<th>Qty</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>245</td>
<td>PN768</td>
<td>1</td>
<td>Rs. 35</td>
</tr>
<tr>
<td>245</td>
<td>PN656</td>
<td>3</td>
<td>Rs. 15</td>
</tr>
</tbody>
</table>

**Second Normal Form**

Reduce first normal form entities to second normal form (2NF) by removing attributes that are not dependent on the whole primary key.

The purpose here is to make sure that each column is defined in the correct table. Using the more formal names may make this a little clearer. Make sure each attribute is kept with the entity that it describes.

Consider the Order Items table that we established above. If we place Customer reference in the Order Items table (Order Number, Line Item Number, Item, Qty, Price, Customer) and assume that we use Order Number and Line Item Number as the Primary Key, it quickly becomes obvious that the Customer reference becomes repeated in the table because it is only dependent on a portion of the Primary Key - namely the Order Number. Therefore, it is defined as an attribute of the wrong entity. In such an obvious case, it should be immediately clear that the Customer reference should be in the Orders table, not the Order Items table.

So instead of:

<table>
<thead>
<tr>
<th>OrderNo</th>
<th>ItemNo</th>
<th>Customer</th>
<th>Item</th>
<th>Qty</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>245</td>
<td>1</td>
<td>SteelCo</td>
<td>PN768</td>
<td>1</td>
<td>Rs. 35</td>
</tr>
<tr>
<td>245</td>
<td>2</td>
<td>SteelCo</td>
<td>PN656</td>
<td>3</td>
<td>Rs. 15</td>
</tr>
<tr>
<td>246</td>
<td>1</td>
<td>Acme Corp</td>
<td>PN371</td>
<td>1</td>
<td>Rs. 2.99</td>
</tr>
<tr>
<td>246</td>
<td>2</td>
<td>Acme Corp</td>
<td>PN015</td>
<td>7</td>
<td>Rs. 5</td>
</tr>
</tbody>
</table>
We get:

<table>
<thead>
<tr>
<th>OrderNo</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>245</td>
<td>SteelCo</td>
</tr>
<tr>
<td>246</td>
<td>Acme Corp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OrderNo</th>
<th>ItemNo</th>
<th>Item</th>
<th>Qty</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>245</td>
<td>1</td>
<td>PN768</td>
<td>1</td>
<td>Rs. 35</td>
</tr>
<tr>
<td>245</td>
<td>2</td>
<td>PN656</td>
<td>3</td>
<td>Rs. 15</td>
</tr>
<tr>
<td>246</td>
<td>1</td>
<td>PN371</td>
<td>1</td>
<td>Rs. 2.99</td>
</tr>
<tr>
<td>246</td>
<td>2</td>
<td>PN015</td>
<td>7</td>
<td>Rs. 5</td>
</tr>
</tbody>
</table>

### Third Normal Form

Reduce second normal form entities to third normal form (3NF) by removing attributes that depend on other, nonkey attributes (other than alternative keys).

This basically means that we shouldn't store any data that can either be derived from other columns or belong in another table. Again, as an example of derived data, if our Order Items table includes Unit Price, Quantity, and Extended Price, the table would not be 3NF. So we would remove the Extended Price (= Qty * Unit Price), unless, of course, the value saved is a manually modified (rebate) price, but the Unit Price reflects the quoted list price for the items at the time of order.

Also, when we established that the Customer reference did not belong in the Order Items table, we said to move it to the Orders table. Now if we included customer information, such as company name, address, etc., in the Orders table, we would see that this information is dependent not so much on the Order per se, but on the Customer reference, which is a nonkey (not Primary Key) column in the Orders table. Therefore, we need to create another table (Customers) to hold information about the customer. Each Customer could then have multiple Orders related to it.

<table>
<thead>
<tr>
<th>OrderNo</th>
<th>Customer</th>
<th>Address</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>245</td>
<td>SteelCo</td>
<td>Delhi</td>
<td>Delhi</td>
</tr>
<tr>
<td>246</td>
<td>Acme Corp</td>
<td>Maharashtra</td>
<td>Bombay</td>
</tr>
<tr>
<td>247</td>
<td>SteelCo</td>
<td>Delhi</td>
<td>Delhi</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OrderNo</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>245</td>
<td>SteelCo</td>
</tr>
<tr>
<td>246</td>
<td>Acme Corp</td>
</tr>
<tr>
<td>247</td>
<td>SteelCo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer</th>
<th>Address</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>SteelCo</td>
<td>Delhi</td>
<td>Delhi</td>
</tr>
<tr>
<td>Acme Corp</td>
<td>Maharashtra</td>
<td>Bombay</td>
</tr>
</tbody>
</table>

Many database designers stop at 3NF, and those first three levels of normalization do provide the most bang for the buck. Indeed, these were the original normal forms described in E. F. Codd's first papers. However, there are currently four additional levels of normalization, so read on. Be aware of what you don't do, even if you stop with 3NF. In some cases, you may even need to de-normalize some for performance reasons.

To conclude, in the following section 10 tips has been presented which can help to ensure that databases are well designed and can be easily exported and manipulated with the minimum of difficulties.
To Develop a Prototype

Significant time can be saved by creating the structure in a simple desktop database (such as Microsoft Access) before finalising the design in one of the enterprise databases. The developer will be able to recognise simple faults and makes changes more rapidly than would be possible at a later date.

1. Split database structure into multiple tables
   Unlike paper-based structures, databases do not require the storage of all fields in a single table. For large databases it is useful to split essential information into multiple tables. Before creating a database, ensure that the data has been normalised to avoid duplication.

2. Use understandable field names
   The developer should avoid field names that are not instantly recognisable. Acronyms or internal references will confuse users and future developers who are not completely familiar with the database.

3. Avoid illegal file names
   It is considered good practice to avoid exotic characters in file or field names. Exotic characters would include ampersands, percentages, asterisks, brackets and quotation marks. You should also avoid spaces in field and table names.

4. Ensure Consistency
   Remain consistent with data entry. If including title (Mr, Miss, etc.) include it for all records. Similarly, if you have established that house number and address belong in different fields, always split them.

5. Avoid blank fields
   Blank fields can cause problems when interpreting the data at a later date. Does it mean that you have no information, or you have forgotten to enter the information? If information is unavailable it is better to provide a standard response (e.g. unknown).

6. Use standard descriptors for date and time
   Date and time can be easily confused when exporting database fields in a text file. A date that reads ‘12/04/2003’ can have two meanings, referring to April 12th or December 4th, 2003. To avoid ambiguity always enter and store dates with a four-digit century and times of day using the 24 hour clock. The ISO format (yyyy-mm-dd) is useful for absolute clarity, particularly when mixing databases at a later date.

7. Use currency fields if appropriate
   Currency data types are designed for modern decimal currencies and can cause problems when handling old style currency systems, such as Britain’s currency system prior to 1971 that divided currency into pounds, shillings and pence.

8. Avoid proprietary extensions
   Care should be taken when using proprietary extensions, as their use will tie your database to a particular software package. Examples of proprietary extensions include the user interface and application-specific commands.
9. Avoid the use of field dividers

Commas, quotation marks and semi-colons are all used as methods of separating fields when databases are exported to a plain text file and subsequently re-imported into another database. When entering data into a database you should choose an alternative character that represents these characters.

Relational Database Management System

E. F. Codd’s Twelve Rules for Relational Databases

Codd's twelve rules call for a language that can be used to define, manipulate, and query the data in the database, expressed as a string of characters. Some references to the twelve rules include a thirteenth rule - or rule zero:

1. Information Rule: All information in the database should be represented in one and only one way -- as values in a table.

2. Guaranteed Access Rule: Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.

3. Systematic Treatment of Null Values: Null values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way, independent of data type.

4. Dynamic Online Catalog Based on the Relational Model: The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data.

5. Comprehensive Data Sublanguage Rule: A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible:

   a. data definition
   b. view definition
   c. data manipulation (interactive and by program)
   d. integrity constraints
   e. authorization
   f. transaction boundaries (begin, commit, and rollback).

6. View Updating Rule: All views that are theoretically updateable are also updateable by the system.

7. High-Level Insert, Update, and Delete: The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data, but also to the insertion, update, and deletion of data.
8. **Physical Data Independence:** Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.

9. **Logical Data Independence:** Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

10. **Integrity Independence:** Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

11. **Distribution Independence:** The data manipulation sublanguage of a relational DBMS must enable application programs and terminal activities to remain logically unimpaired whether and whenever data are physically centralized or distributed.

12. **Nonsubversion Rule:** If a relational system has or supports a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules or constraints expressed in the higher-level (multiple-records-at-a-time) relational language.

In the early days of relational database products, these twelve rules were often used to evaluate RDBMSs. In the academic community, the discussions of full compliance of RDBMS versus the Relational Model continues, as does the discussion about whether the SQL language satisfies all the requirements. But we will stick to more practical matters.

For more information about what these twelve rules mean, see [http://www.frick-cpa.com/ss7/Theory_RelationalDB.asp#Codd](http://www.frick-cpa.com/ss7/Theory_RelationalDB.asp#Codd).  

**References**


Dr. E. F. Codd's 12 rules for defining a fully relational database (see [http://www.cis.ohio-state.edu/~sgomori/570/coddsrules.html](http://www.cis.ohio-state.edu/~sgomori/570/coddsrules.html)).

**Handbook of Relational Database Design** by Candace C. Fleming and Barbara von Halle (Addison Wesley, 1989).

[http://www.frick-cpa.com/ss7/Theory_RelationalDB.asp#Codd](http://www.frick-cpa.com/ss7/Theory_RelationalDB.asp#Codd)

Characteristics of a Relational Database by David R. Frick & Co., CPA.

SearchDatabase.com

C. J. Date, *Handbook of Relational Database Design* by Candace C. Fleming and Barbara von Halle (Addison Wesley, 1989).
