Relational data model

The relational model, introduced by E. F. Codd in 1970, is based on predicate logic and set theory. **Predicate logic**, used extensively in mathematics, provides a framework in which an assertion (statement of fact) can be verified as either true or false. For example, suppose that a student with a student ID of 12345678 is named Wanyama Wafula. This assertion can easily be demonstrated to be true or false.
Set theory is a mathematical science that deals with sets, or groups of things, and is used as the basis for data manipulation in the relational model. For example, assume that set A contains three numbers: 16, 24, and 77. This set is represented as A(16, 24, 77). Furthermore, set B contains four numbers: 44, 77, 90, and 11, and so is represented as B(44, 77, 90, 11). Given this information, you can conclude that the intersection of A and B yields a result set with a single number, 77. This result can be expressed as A _ B = 77. In other words, A and B share a common value, 77.

Based on these concepts, the relational model has three well-defined components:
1. A logical data structure represented by relations.
2. A set of integrity rules to enforce that the data are and remain consistent over time.
3. A set of operations that defines how data are manipulated .

A LOGICAL VIEW OF DATA

The relational data model changed theway of viewing data in database by allowing the designer to focus on the logical representation of the data and its relationships, rather than on the physical storage details.
To use an automotive analogy, the relational database uses an automatic transmission to relieve you of the need to manipulate clutch pedals and gearshifts.
In short, the relational model enables you to view data *logically* rather than *physically*.
The practical significance of taking the logical view is that it serves as a reminder of the simple file concept of data storage. Although the use of a table, quite unlike that of a file, has the advantages of structural and data independence, a table does resemble a file from a conceptual point of view. Because you can think of related records as being stored in independent tables, the relational database model is much easier to understand than the hierarchical and network models.
Logical simplicity tends to yield simple and effective database design methodologies. Because the table plays such a prominent role in the relational model, it deserves a closer look. Therefore, our discussion begins with an exploration of the details of table structure and contents.

Table and its characteristics

The logical view of the relational database is facilitated by the creation of data relationships based on a logical construct known as a relation. Because a relation is a mathematical construct, end users find it much easier to think of a relation as a table. A table is perceived as a two-dimensional structure composed of rows and columns.

A table contains a group of related entity occurrences, that is, an entity set. For example, a STUDENT table contains a collection of entity occurrences, each representing a student. For that reason, the terms *entity set* and *table* are often used interchangeably.

Characteristics of a Relational Table

- A table is perceived as a two-dimensional structure composed of rows and columns.
- Each table row (**tuple**) represents a single entity occurrence within the entity set.
- Each table column represents an attribute, and each column has a distinct name.
- Each row/column intersection represents a single data value.
- All values in a column must conform to the same data format.
- Each column has a specific range of values known as the **attribute domain**.
- The order of the rows and columns is immaterial to the DBMS.
- Each table must have an attribute or a combination of attributes that uniquely identifies each row.

Keys

In the relational model, keys are important because

- They are used to ensure that each row in a table is uniquely identifiable.
- They are also used to establish relationships among tables and to ensure the integrity of the data.

A **key** consists of one or more attributes that determine other attributes. For example, an invoice number identifies all of the invoice attributes, such as the invoice date and the customer name.

.

The key's role is based on a concept known as **determination**.

In the context of a database table, the statement "A determines B" indicates that if you know the value of attribute A, you can look up (determine) the value of attribute B. For example, knowing the STU_NUM in the STUDENT table means that you are able to look up (determine) that student's last name, grade, phone number, and so on. The shorthand notation for "A determines B" is A⟶ B. If A determines B, C, and D, you write A ⟶ B, C, D. Therefore, using the attributes of the STUDENT table , you can represent the statement "STU_NUM determines STU_LNAME" by writing:

STU_NUM ⟶ STU_LNAME

The principle of determination is very important because it is used in the definition of a central relational database concept known as functional dependence. The term **functional dependence** can be defined most easily this way: the attribute B is functionally dependent on A if A determines B. More precisely:

The attribute B is functionally dependent on the attribute A if each value in column A determines *one and only one* value in column B.

The functional dependence definition can be generalized to cover the case in which the determining attribute values occur more than once in a table. Functional dependence can then be defined this way:

Attribute A determines attribute B (that is, B is functionally dependent on A) if all of the rows in the
table that agree in value for attribute A also agree in value for attribute B.

Keep in mind that it might take more than a single attribute to define functional dependence; that is, a key may be composed of more than one attribute. Such a multiattribute key is known as a **composite key**.

Any attribute that is part of a key is known as a **key attribute**. For instance, in the STUDENT table, the student's last name would not be sufficient to serve as a key. On the other hand, the combination of last name, first name, initial,and phone is very likely to produce unique matches for the remaining attributes. For example, you can write:

STU_LNAME, STU_FNAME, STU_INIT, STU_PHON E ⟶ STU_HRS, STU_CLASS
or
STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE ⟶ STU_HRS, STU_CLASS, STU_GPA
or
STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE ⟶ STU_HRS, STU_CLASS, STU_GPA, STU_DOB

Given the possible existence of a composite key, the notion of functional dependence can be further refined by specifying **full functional dependence**:

If the attribute (B) is functionally dependent on a composite key (A) but not on any subset of that composite key, the attribute (B) is fully functionally dependent on (A).

Within the broad key classification, several specialized keys can be defined. For example, a **superkey** is any key that uniquely identifies each row. In short, the superkey functionally determines all of a row's attributes. In the STUDENT table, the superkey could be any of the following:
STU_NUM
STU_NUM, STU_LNAME
STU_NUM, STU_LNAME, STU_INIT
In fact, STU_NUM, with or without additional attributes, can be a superkey even when the additional attributes are redundant.
A **candidate key** can be described as a superkey without unnecessary attributes, that is, a minimal superkey. Using this distinction, note that the composite key
foreign key (**FK**) this is the primary key of one table appears as the *foreign key* in a related table. It is an attribute whose values match the primary key values in the related table.

INTEGRITY RULES
There are two integrity rules that a relation database model should exhibit namely:

a) Integrity rule : it stataes that All primary key entries are unique, and no part of a primary key may be null. The purpose of this rule is each row will have a unique identity, and foreign key values can properly reference primary key

values. For instance no invoice can have a duplicate number, nor can it be null. In short, all invoices are uniquely identified by their invoice number.

b) Referential integrity: states that a foreign key may have either a null entry, as long as it is not a part of its table's primary key, or an entry that matches the primary key value in a table to which it is related. (Every non-null foreign key value *must* reference an *existing* primary key value.) This means that is possible for an attribute NOT to have a corresponding value, but it will be impossible to have an invalid entry. The enforcement of the referential integrity rule makes it impossible to delete a row in one table whose primary key has mandatory matching foreign key values in another table. For instance a customer might not yet have an assigned sales representative (number), but it will be impossible to have an invalid sales representative (number).

RELATIONAL SET OPERATORS

The data in relational tables are of limited value unless the data can be manipulated to generate useful information. **Relational algebra** defines the theoretical way of manipulating table contents using the eight relational operators: SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT, and DIVIDE.
The relational operators have the property of **closure**; that is, the use of relational algebra operators on existing relations (tables) produces new relations. Their *use* can easily be illustrated as follows:

1. SELECT, also known as RESTRICT, yields values for all rows found in a table that satisfy a given condition. SELECT can be used to list all of the row values, or it can yield only those row values that match a specified criterion. In other words, SELECT yields a horizontal subset of a table.
2. PROJECT yields all values for selected attributes. In other words, PROJECT yields a vertical subset of a table.
3. UNION combines all rows from two tables, excluding duplicate rows. The tables must have the same attribute characteristics (the columns and domains must be compatible) to be used in the UNION. When two or more tables share the same number of columns, and when their corresponding columns share the same (or compatible) domains, they are said to be **union-compatible**.
4. INTERSECT yields only the rows that appear in both tables. As was true in the case of UNION, the tables must be union-compatible to yield valid results. For example, you cannot use INTERSECT if one of the attributes is numeric and one is character-based.
5. DIFFERENCE yields all rows in one table that are not found in the other table; that is, it subtracts one table from the other. As was true in the case of UNION, the tables must be union-compatible to yield valid results.
6. PRODUCT yields all possible pairs of rows from two tables—also known as the Cartesian product. Therefore if one table has six rows and the other table has three rows, the PRODUCT yields a list composed of $6 \times 3 = 18$ rows.
7. JOIN allows information to be combined from two or more tables. JOIN is the real power behind the relational database, allowing the use of independent tables linked by common attributes.

A **natural join** links tables by selecting only the rows with common values in their common attribute(s). A natural join is the result of a three-stage process:

- First, a PRODUCT of the tables is created
- Second, a SELECT is performed on the output of Step a to yield only the rows for which the AGENT_CODE values are equal.
- A PROJECT is performed on the results of Step b to yield a single copy of each attribute, thereby eliminating duplicate columns.

8. The DIVIDE operation uses one single-column table (e.g., column "a") as the divisor and one 2-column table (i.e., columns "a" and "b") as the dividend. The tables must have a common column (e.g., column "a"). The output of the DIVIDE operation is a single column with the values of column "a" from the dividend table rows where the value of the common column (i.e., column "a") in both tables matches.

THE DATA DICTIONARY AND THE SYSTEM CATALOG

The **data dictionary** provides a detailed description of all tables found within the user/designer-created database. Thus, the data dictionary contains at least all of the attribute names and characteristics for each table in the system. In short, the data dictionary contains metadata—data about data. The data dictionary is sometimes described as "the database designer's database" because it records the design decisions about tables and their structures.

Like the data dictionary, the **system catalog** contains metadata. The system catalog can be described as a detailed system data dictionary that describes all objects within the database, including data about table names, the table's creator and creation date, the number of columns in each table, the data type corresponding to each column, index filenames, index creators, authorized users, and access privileges. Because the system catalog contains all required data dictionary information, the terms *system catalog* and *data dictionary* are often used interchangeably. In fact, current relational database software generally provides only a system catalog, from which the designer's data dictionary information may be derived. The system catalog is actually a system-created database whose tables store the user/designer-created database characteristics and contents. Therefore, the system catalog tables can be queried just like any user/designer-created table.

RELATIONSHIPS WITHIN THE RELATIONAL DATABASE

The following types of reltionship are exhibited in relational model:

a) One to many - The 1:M relationship is the relational modeling ideal. Therefore, this relationship type should be the norm in any relational database design. There is only one row in one table for any given row in in another table table, but there may be many rows in the other table for any given row in one table. The one-to-many (1:M) relationship is easily implemented in the relational model by putting the *primary key of the 1 side in the table of the many side as a foreign key.*

b) One to one - The 1:1 relationship should be rare in any relational database design. in this relationship, one entity can be related to only one other entity, and vice versa. For example, one department chair—a professor—can chair

only one department, and one department can have only one department chair. The entities PROFESSOR and DEPARTMENT thus exhibit a 1:1 relationship

c) Many to many : M:N relationships cannot be implemented as such in the relational model. However, M:N relationships can be implemented by creating a new entity in 1:M relationships with the original entities.

INDEXES

An **index** is an orderly arrangement used to logically access rows in a table. An index is used to locate a needed item quickly. From a conceptual point of view, an index is composed of an index key and a set of pointers. The **index key** is, in effect, the index's reference point. More formally, an index is an ordered arrangement of keys and pointers. Each key points to the location of the data identified by the key.