1. The Relational Model

All relational databases, including the eight products dealt with in the book, are based on The Relational Model by E.F.Codd, an IBM researcher, published in 1970. The relational model defines the theoretical basis of relational databases on the basis of set theory, mathematics and predicate logic.

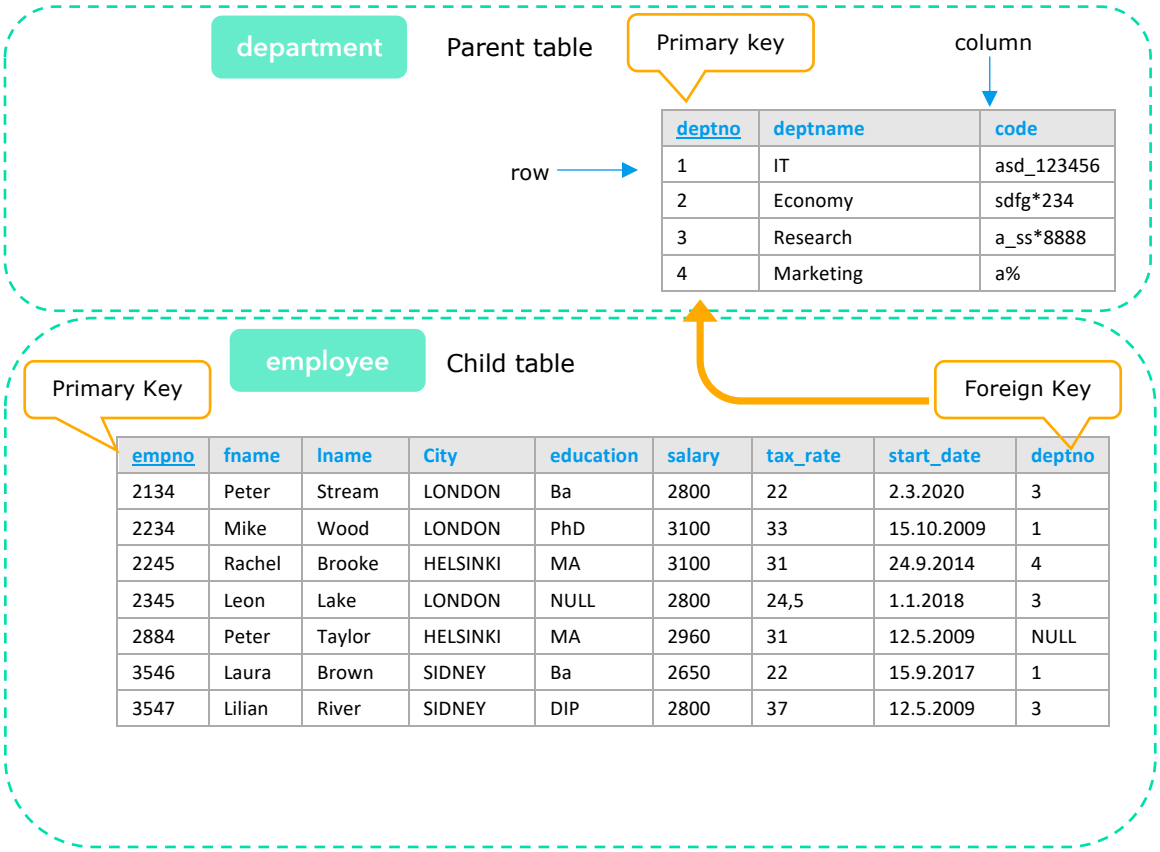
The relational model as defined by Codd has brought about a revolution in the database world as relational database products have replaced earlier hierarchical (e.g. DL/1) and networked (e.g. IDMS, MDBS) database types. The SQL language has become standardized as the database language of all database suppliers. The major part of all new information systems are built with the aid of relational database products.

The relational model can be divided into three parts: structure, processing, and integrity rules. I shall next review these parts. (This does not cover the whole relational model but it is a good start).

1.1 Structure

Table

The following figure contains a firm's departments and employees connected with the departments. One department may have many persons and one person always belongs to at most one department.



deptno	deptname	code
1	IT	asd_123456
2	Economy	sdfg*234
3	Research	a_ss*8888
4	Marketing	a%

empno	fname	lname	City	education	salary	tax_rate	start_date	deptno
2134	Peter	Stream	LONDON	Ba	2800	22	2.3.2020	3
2234	Mike	Wood	LONDON	PhD	3100	33	15.10.2009	1
2245	Rachel	Brooke	HELSINKI	MA	3100	31	24.9.2014	4
2345	Leon	Lake	LONDON	NULL	2800	24,5	1.1.2018	3
2884	Peter	Taylor	HELSINKI	MA	2960	31	12.5.2009	NULL
3546	Laura	Brown	SIDNEY	Ba	2650	22	15.9.2017	1
3547	Lilian	River	SIDNEY	DIP	2800	37	12.5.2009	3

Figure 1.1: Example of two tables: department and employee

Contextual data such as departments and persons are presented in a relational database as **tables** such as the tables "department" and "employee" above. A table has **columns** such as "deptno", "depname" and "code", and **rows**. Sometimes columns are called fields and rows records. A column has a defined data type such as numeric or character, and length. All data in a relational database are stored in tables and only in tables.
Tables are set up in the SQL language, its DDL (Data Definition Language) part. For more on this, see chapter on Table Definition and Modification.

Primary Key and Foreign Key

Each table is identified by a **primary key (PK)**, for example, in Figure 1.1., deptno in the Department table and empno in the Employee table (both appear underlined). The primary key must be **unique**, meaning that no two (or more) rows in that column may have the same value. For example, each row in the Employee table has its own unique identifier. A primary key may consist of more than one column. More on the definition of the primary key in the section 11 Table Definitions.

Employees belong a departments. A department may have many employees, but one employee may belong to one department only. This is called a **parent-child relationship** or one-to-many relationship. A parent may have many children but a child can only have one set of parents, see Figure 1.1. To establish the relationship, the child (Employee) table contains the link field or foreign key deptno that refers to the primary key deptno in the Department table. The referring table is called the child table, the referred table the parent table. Foreign keys are needed when tables are combined or joined. For more on the definition of foreign keys see the chapter on Referential Integrity.

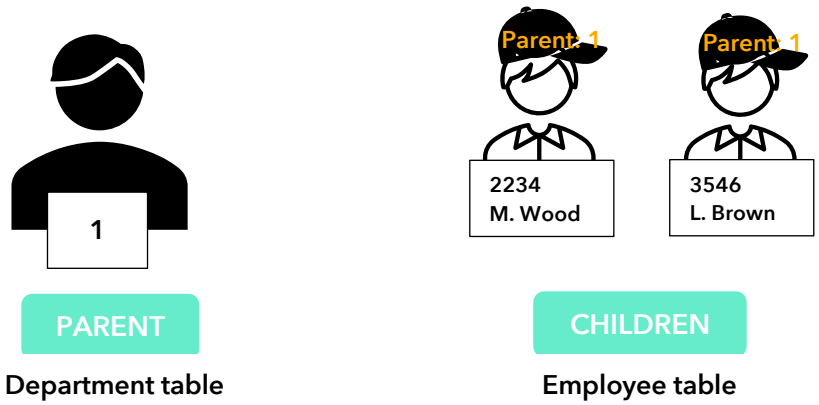


Figure 1.2 The parent's primary key is the children's foreign key

NULL

In Figure 1.1. Person 2345 has NULL in the column for education. If, at the input stage, there is no value for the column, the appropriate value will be marked NULL. It does not denote a blank, space or zero, but a value which is unknown. The inclusion of NULL implies that relational databases follow so-called ternary predicate logic. When it is asked whether a certain person has the degree M.A., the answer will be either true, untrue or maybe. Maybe is the result if the database entry is NULL. There are thus three values instead of the conventional two.

What are NULLs needed for? Let´s look at a table (see example below) with dates, temperatures and other measurements. Every morning new measurements are recorded in the table. On the fourth day, the thermometer is broken, and degrees are recorded as NULL, or an unknown temperature. SQL's mean function AVG computes average temperature bypassing NULLs, like it should. If NULL were not in use it would be difficult to determine what to record as temperature when the thermometer is broken. In many contexts NULL also presents problems. I shall deal with them later.

date	degrees	Other_measurem
1.12.2020	3	35
2.12.2020	0	27
3.12.2020	-1	27
4.12.2020	NULL	29
5.12.2020	1	31
...	...	...

The SQL language has operations for searching NULLs, see chapter on NULLs. In setting up a table one can determine whether NULL is permitted or not for each column.

1.2 Processing

Set Theory in the Background

It was Codd's brilliant innovation to base the processing of data on set theory. A table is made up of a set of rows. This set can be subjected to set operations such as "find all employees from LONDON (selection), their first names and last names (projection). Examples of set theoretical operations that can be done to a table are given in Figure 1.3.

Set theoretical operations may be used to process a table or several tables at once. They also apply to updating operations. A single update command, say in the middle of a Java program, may be used to update a large number of rows of a table, without loops.

Set theory is implemented by the SQL language, which is based completely on set theory. All the operations in Figure 1.3 can be performed with different forms of SQL's SELECT statement.

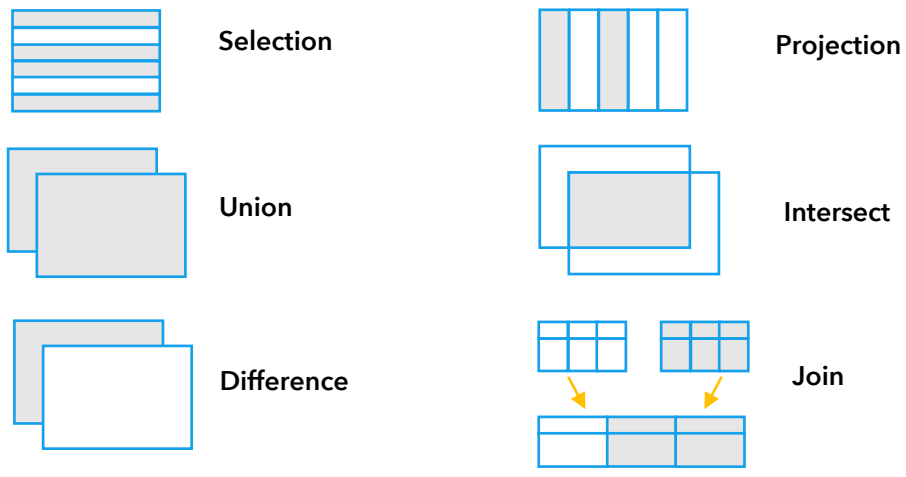


Figure 1.3: The main set-theoretical operations

All the columns of a table can be used as search arguments. There is no rule that a column should specifically be defined as a search key.

7. Joins

So far we have selected data from only one table at a time. However, in well designed realistic databases the data is spread across several (normalized) tables and it has to be collected and combined to get the desired final result. This is done by using joins. Joins form a central part of SQL-queries, that is why I will deal with themquite thoroughly in this chapter.

Tables can be joined using two different kinds of syntax. The newer and recommended one is the New Join-syntax which I will introduce first. Then I will describe the old syntax, which I refer to as the traditional syntax. Finally I will compare the two approaches.

7.1 New Join -syntax

Joining Two Tables

The following figure demonstrates the idea of joining

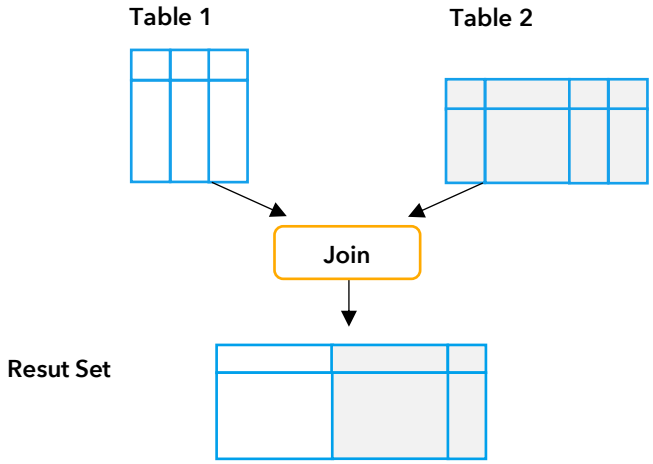


Figure 7.1: Joining combines columns from several tables into one result

Fetch all employees names and departments that are from LONDON. Order by department name

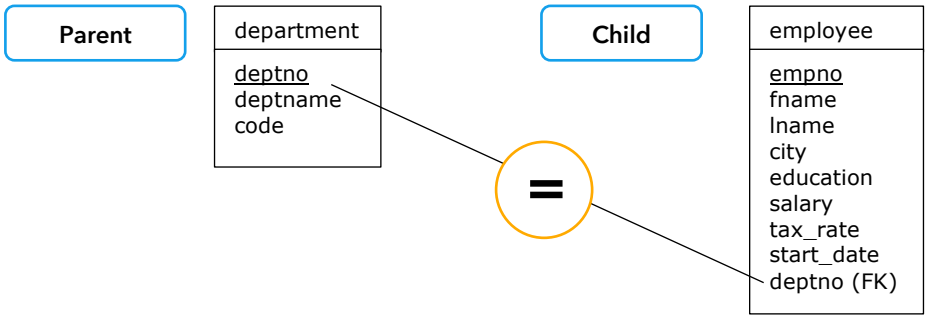


Figure 7.2: The table employee is the child table, whose foreign key (FK) deptno referres to the deptno column of the parent table (department). This connection (the "join condition") is given in the ON-clause: **`ON department.deptno = employee.deptno`**

```
SELECT department.deptno, deptname, lname, fname, city
FROM department
     JOIN employee
       ON (department.deptno = employee.deptno)    -- Join condition
WHERE city = 'LONDON'                              -- parenthesis only for clarity
ORDER BY deptname
```

```
   DEPTNO  DEPTNAME   LNAME    FNAME   CITY
--------- ----------  -------  ------  -------
        1 IT          Wood     Mike    LONDON
        3 Production  Stream   Peter   LONDON
        3 Production  Lake     Leon    LONDON
```

We needed two tables to obtain required data: department and employee. They are given in the FROM-part of the SELECT-statement. Now all columns from both tables can be referenced in the SELECT column list. It does not make any difference in which order you give the tables.

When joining rows from two tables it is important to understand how the rows are combined. This information is given after the ON keyword of the FROM clause by describing which column (or columns) in the parent row must have the same value as the column (or columns) in the child row. If the corresponding columns have the same value in both rows, these rows are combined into a new row in the result set. In nearly all cases (but not allways!) we combine the primary key of the parent table (here: department.deptno) to the foreign key of the child table. See figure below. On the left we have the employee table with employees from LONDON and on the right we have the department-table. The lines show how rows with the same deptno are combined.

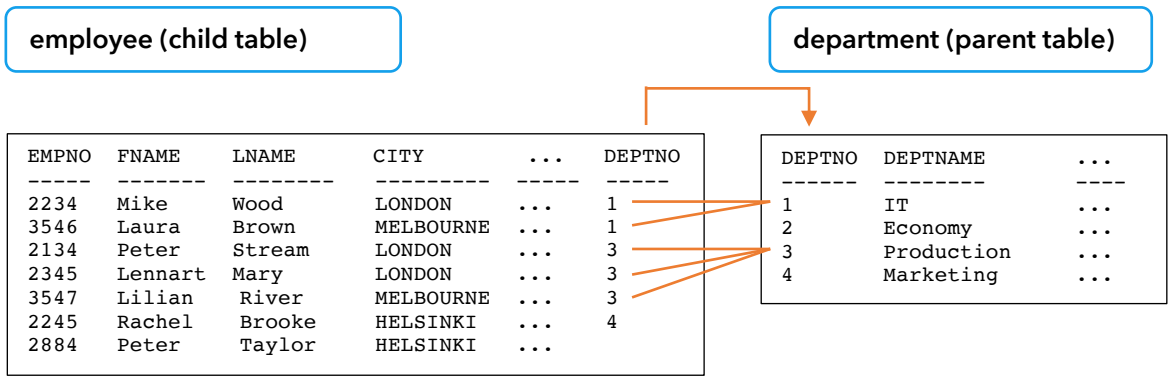


Figure 7.3: A concrete example of how rows are combined

Note that in the above SQL example the column name department.deptno referres to the column deptno in the department table. We say that we have *qualified* the deptno column when we prefix it with the name of the table it resides in. Qualifiying is necessary when a column of the same name can be found in two or more tables that take part in the join. In our example there is a column named deptno in both tables department and employee. Without qualifying, the system would not know which column we are referring to and would protest with a nasty error message, usually containing the word ”ambiguous”. The columns empno, lname, fname, city and deptname don’t have to be qualified, since columns with that name can only be found in one table.