The Relational Model (for Data Warehousing)

Jens Lechtenbörger

Winter Term 2021/2022

1 Motivation

Relational database technology is pervasive in our world. The *relational model*, defined in 1970 by Turing award winner Edgar F. Codd in the famous paper [Cod70], provides the foundations for that technology. On a small scale, web browsers make use of relational databases to store cookies and history (thus, our phones contain several of them!); on larger scales, organizations manage data for various applications such as ERP, CRM, or financial operations in such databases. For business intelligence purposes, organizations frequently integrate the data of these isolated, application-specific database systems in another relational database system, namely the data warehouse.

Since 1970, numerous commercial and free/libre and open source relational database management systems have been implemented, and new ones are still being developed continuously (e.g., to utilize technological advances).

Hence, a solid understanding of relational data management concepts is advantageous for the design and implementation of IT and information systems in general and for data warehouses in particular.

2 Learning Objectives

Learning objectives specify what I expect you to be able to do. In general, some learning objectives qualify as tasks in an exam.

- Prerequisites (sketched below as well)
  - Query relational databases with SQL
  - Design relational databases in third normal form (3NF)
    - Assert/identify FDs and keys
- Express queries in relational algebra
  - Create and read tree representations of query expressions (for query optimization)

3 Prerequisites

3.1 SQL

I suppose that you know SQL, both to define schemata (with `CREATE TABLE`) and to query data (with `SELECT`). While our first session will not make much use of SQL (except as points of reference when introducing the relational algebra below), future sessions will deal with (a) OLAP features of SQL that build upon standard aggregation queries and (b) query optimization. To refresh your querying skills, you could play the text adventure SQL Island. In that game, you are supposed to escape an island by executing appropriate SQL commands (to which you are introduced along the way). Students reported this to be fun!

Besides, in Learnweb you find a script to populate a PostgreSQL database with the TPC-H database schema, which will serve as sample database in upcoming sessions. More details are provided in a separate document with database instructions.

3.2 Vocabulary

In SQL, we define the *schema* of a table with `CREATE TABLE`, where a schema specifies the attributes/columns of the table with their data types and dependencies/constraints (e.g., primary and foreign keys, not-null and check constraints). With `INSERT`, `DELETE`, and `UPDATE` we *manipulate* the data stored as rows in tables. The collection of rows is also called *instance* of the schema. The *database management system* (DBMS) makes sure that instances conform to their schemata (by refusing manipulations that would violate data types or constraints).

"Tuple" is a formal term for "row", "relation" for the instance of a schema (a set of tuples).

Given a relation, a *key* is a minimum set of attributes to identify each tuple uniquely (i.e., there cannot exist two different tuples that share the same values for all key attributes). E.g., the set of attributes `FullName`, `FullAddress`, `DateOfBirth` might form a key in a relation about people; `Nationality` and `IDCardNo` might form another key in that same relation.

In SQL, we can declare one of the keys as *primary key* to the DBMS.

In practice, the designer frequently adds an artificial key without any application semantics as primary key to the schema (e.g., an ID column with automatically increasing integer values); such artificial keys are called *surrogate keys*.

Keys are a special form of *functional dependencies* (FDs). Hopefully, you have seen FDs in the context of database normalization to avoid redundancy and database anomalies.

In case you are not familiar with database normalization, the 6-page article [Ken83] is "A simple guide to five normal forms in relational database theory". It contains the following informal descriptions of second (2NF) and third normal form (3NF):

- "Second normal form is violated when a nonkey field is a fact about a subset of a key."

- "Third normal form is violated when a nonkey field is a fact about another nonkey field, . . . "
- "Under second and third normal forms, a nonkey field must provide a fact about the key, the whole key, and nothing but the key."

For class purposes, we focus on the third normal (3NF).

3.3 FDs and 3NF with Python

This introduction to functional dependencies and 3NF normalization with synthesis might be useful as crash course. Note that the introduction is generated from documentation of this Python module, which allows you to experiment. A Jupyter notebook for in-browser experiments is available as well.

In an upcoming session, you will see data warehouse schema design along the lines of 3NF normalization.

3.4 Self-study

You might use tasks in this section to check your understanding before our class meeting. Please use our discussion forum or shared pad in case of questions.

3.4.1 Keys

What is a key, what a primary key? Suggest more keys for the relation about people mentioned above.

3.4.2 Functional Dependencies

Consider the sample relation **Account** visualized in Table 1, where **Balance** indicates the balance of an account at the end of a given day, while **CustAge** indicates the age of a customer on that day. The Python module also contains FDs for that relation.

Table 1: Sample Account relation.

Account	AccID	Date	CustID	Type	Balance	CustAge
	1	2015-01-01	42	checking	1000	42
	2	2015-01-01	42	savings	2000	42
	3	2015-01-01	1	checking	-300	22
	1	2015-01-02	42	checking	960	43

1. Based on your domain knowledge, which FDs would you *assert* for the schema of that relation (i.e., which FDs are semantic integrity constraints that will hold in every instance)?
2. Verify that the relation satisfies the following (incomplete) list of FDs:
   - AccID Date → CustAge
   - AccID Date → AccID Date CustID Type Balance CustAge
   - AccID → CustID

- Balance → AccID
- Date → Date

3. Why are the following no valid FDs for that relation?

   - CustID → AccID
   - AccID → Balance
   - CustID Date → Balance

4. What can you say about the birthday of the customer with ID 42? What about the net worth of transactions on the account with ID 1 on 2015-01-02?

5. Why is the schema of Table 1 not in 3rd Normal Form (3NF)?

3.4.3 Solutions?

I *strongly* suggest that you think about the above questions on your own *now*. Answering them might take seconds (if you are familiar with the topics) or days (e.g., if this is your first encounter with the relational model). If you are not sure whether your answers are correct, first note that above I pointed to an introduction to functional dependencies and 3NF normalization with synthesis, which might be useful as crash course. Go there first.

Afterwards, you can find some anwers to the above questions here. Please keep in mind that we learn by doing, not by watching others.

4 Relational Algebra

SQL is a *declarative* query language as users declare *how* the result should *look* like (e.g., in terms of selection conditions in the `WHERE` clause and desired attributes after `SELECT`). It does *not* specify *what operations* to *execute* in what order (e.g., when joining multiple tables) or with what data structures and algorithms; instead, *procedural* languages are concerned with such more execution-specific questions.

Relational algebra (introduced by Codd in his seminal paper [Cod70]) is a procedural query language for relational data (see, e.g., [AHV95; GUW08; Vos08] for modern introductions), whose basic operations are the following six: selection, projection, natural join, union, difference, and renaming.

As part of query processing, the DBMS translates a SQL query into an equivalent algebraic representation, ultimately resulting in a *query evaluation plan* (QEP), which specifies what operations to execute in what order with what algorithms and data structures. We will revisit QEPs in the context of query processing and optimization for OLAP.

The following explanations cover selection, project, and natural join based on examples without any formalization. Please ask (or consult textbooks) if necessary. Also note that the tool RelaX allows you to practice your query skills, for which you find a pointer below.

Consider a relation r over a set X of attributes. Roughly, the *selection* of r according to some selection condition ϕ, denoted by $\sigma_\phi(r)$, produces a subset

Table 2: Result of $\sigma_{\text{CustAge}>42}(\text{Account})$.

AccID	Date	CustID	Type	Balance	CustAge
1	2015-01-02	42	checking	960	43

that contains those tuples of r that satisfy the condition ϕ. Thus, ϕ is what would occur in a `WHERE` clause in SQL. See Table 2 for an example.

Given a subset of $Z \subseteq X$, the *projection* of r on Z, denoted by $\pi_Z(r)$, produces a relation where the attributes of r that do not occur in Z are removed (i.e., that is restricted to the attributes in Z). Thus, Z would occur as attribute list after the `SELECT` keyword in SQL. See Table 3 for an example.

Table 3: Result of $\pi_{\text{AccID,Date,Balance}}(\text{Account})$.

AccID	Date	Balance
1	2015-01-01	1000
2	2015-01-01	2000
3	2015-01-01	-300
1	2015-01-02	960

The *natural join* of relations r_1 and r_2, denoted by $r_1 \bowtie r_2$, produces a relation over the union of attributes of r_1 and r_2, where tuples of r_1 and r_2 are combined in a "natural" way based on equality of shared attributes. E.g., if r_1 is the result shown in Table 3 and r_2 is a relation that stores for each AccID the city of the bank's branch running the account, the result of $r_1 \bowtie r_2$ might appear as in Table 4.

Table 4: Result of $r_1 \bowtie r_2$.

AccID	Date	Balance	BranchCity
1	2015-01-01	1000	Münster
2	2015-01-01	2000	Frankfurt
3	2015-01-01	-300	München
1	2015-01-02	960	Münster

In addition, for relations sharing the same attributes, their *union* and their *difference* are just their usual set operations.

Besides, *renaming* of attribute A into attribute B of r, denoted by $\rho_{\text{B} \leftarrow \text{A}}$, is a technical operation (that corresponds to `AS` in SQL), e.g., to make sure that attribute names are set up properly for join operations.

Building upon the above six basic operations, other operations can be defined, e.g., intersection, Cartesian product, or variants of join. In addition, *extensions* of the relational algebra have been defined that increase the expressive power, e.g., for queries involving aggregation.

Note that the relational algebra is an algebra in the sense that inputs to operations are relations, their outputs are relations, and operations can be nested. For example, the following query asks for names of customers with large account balances.

$$\pi_{\text{Name,Balance}}(\text{Customer} \bowtie \sigma_{\text{Balance}>1000000}(\text{Account}))$$

4.1 Self-study with RelaX

The tool RelaX allows you to execute queries expressed in the relational algebra. This hyperlink leads to a configuration of RelaX which is set up with a toy database for sample queries (based on that Gist). The following is an ASCII formulation of the previous query for use with RelaX:

```
pi name, balance (
    Customer
      join
    sigma balance > 1000000 (Account) )
```

Copy and paste the ASCII formulation into this RelaX configuration and press "execute query". Note how not only the result but also a tree representation (which is the basis for a query evaluation plan, to be discussed as part of OLAP optimization) and a representation using the symbols introduced above are shown.

How does the previous query work? What about the following variant?

```
sigma balance > 1000000 (
    pi name, balance ( Account join Customer ) )
```

5 Tentative Session Plan

Our session might follow this agenda, where different students/groups might work with different speeds on items 2–4. I reserve about 20 minutes for item 5.

1. Interactive review of self-study tasks
2. Normalize schema of Table 1
3. Express sample queries in the Relational Algebra
4. First look at Task 1 of Exercise Sheet 2
5. Introduction to next topic

Bibliography

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: https://wiki.epfl.ch/provenance2011/documents/foundations%20of%20databases-abiteboul-1995.pdf.

[Cod70] E. F. Codd. "A Relational Model of Data for Large Shared Data Banks". In: *Commun. ACM* 13.6 (1970), pp. 377–387. DOI: 10.1145/362384.362685. URL: https://doi.org/10.1145/362384.362685.

[GUW08] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. 2nd. Pearson, 2008.

[Ken83] William Kent. "A Simple Guide to Five Normal Forms in Relational Database Theory". In: *Commun. ACM* 26.2 (1983), pp. 120–125. DOI: 10.1145/358024.358054. URL: https://doi.org/10.1145/358024.358054.

[Vos08] G. Vossen. *Data Models, Database Languages and Database Management Systems*. 5th (in German). Oldenbourg, 2008.

License Information