Data Modelling and Databases - Week 1 (Book)

- Author: Ruben Schenk
- Date: 02.03.2021
- Contact: ruben.schenk@inf.ethz.ch

2. The Relational Model of Data

2.1 An Overview of Data Models

2.1.1 What is a Data Model?

A `data model` is a notation for describing data or information. The description generally consists of three parts:

1. *Structure of the data*: In the database world, data models are at somewhat higher level than data structures, and are sometimes referred to as *conceptual model* to emphasize the difference in level.
2. *Operations on the data*: We are generally allowed to perform a limited set of `queries` (operations that retrieve information) and `modifications` (operations that cahnge the database).
3. *Constraints on the data*: Database data models usually have a way to describe limitations on what the data can be.

2.1.3 The Relational Model in Brief

The relational model is based on tables. The structure portion of the relational model migh appear to resemble an array of structs in C, where the column headers are the field names, and each of the rows represent the values of one struct in the array.

2.2 Basics of the Relational Model

The relational model gives us a single way to represent data: as a two-dimensional table called a `relation`.

In this section, we shall introduce the most important terminology regarding relations, and illustrate them with the *Movies* relation:

`title`	`year`	`length`	`genre`
Gone With the Wind	1939	231	drama
Star Wars	1977	124	sciFi
Wayne's World	1992	95	comedy

Table 2.3: The realtion 'Movies'

2.2.1 Attributes

The columns of a relation are named by `attributes`, in Tab. 2.3 the attributes are *title, year, length* and *genre*.

2.2.2 Schemas

The name of a relation and the set of attributes for a relation is called the `schema` for that relation. Thus, the schema for relation *Movies* as above is

> Movies(title, year, length genre)

The set of schemas for the relations of a database is called a `relational database schema`, or just `database schema`.

2.2.3 Tuples

The rows of a relation, other than the header row containing the attribute names, are called `tuples`. A tuple has one compinent for each attribute of the relation.

For example,

> (Gone With the Wind, 1939, 231, drama)

is the first tuple of Tab. 2.3.

2.2.4 Domains

The relational mode requires that each component of each tuple to be *atomic*, that is, it must be of some elementary type such as integer or string.

It is further assumed that associated with each attribute of a relation is a `domain`, that is, a particular elementary type.

It is possible to include the domain, or data type, for each attribute in a relation schema. We shall do so by appending a colon and a type after attributes:

> Movies(title:string, year:integer, length:integer, genre:string)

2.2.5 Equivalent Representations of a Relation

One important remark is that relations are sets of tuples, not lists of tuples. Thus *the order in which the tuples of a relation are represented is immaterial.*

2.2.6 Relation Instances

A relation about movies is not static, rather, relations change over time. It is less common for the schema of a relation to change though. However, there are situations where we might want to add or delete attributes.

Schema changes, while possible in commercial database systems, can be very expensive, because each of perhaps millions of tuples need to be rewritten to add or delete components.

We shall call a set of tuples for a given relation an `instance` of that relation. For example, in 1990, the relation *Movies* did not contain the tuple for *Wayne's World*. However, a conventional database system maintains only one version of any relation: the set of tuples that are in the relation "now". This instance of the relation is called the `current relation`.

2.2.7 Keys of Relations

There are many constraints on relations that the relational model allows us to place on database schemas. However, one kind of constraint is so fundamental that we shall introduce if there: `key constraints`. A set of attributes forms a `key` for a relation if we do not allow two tuples in a relation instance to have the same values in all the attributes of the key.

We distinguish further between `candidate keys`, which is the minimal set of fields that identify each tuple uniquely, and `primary key`, which is one chosen candidate key, marked in the schema by underlining it.

We indicate the attribute or attributes that form a key for a relation ny underlining the key attribute(s). For example, the *Movies* relation could have its schema written as:

> Movies(title, year, length, genre)

While we might be sure that *title* and *year* can serve as a key for *Movies*, many real-world databses use artificial keys, doubting that it is safe to make any assumption about the values of attributes outside their control.

> Short summaries of some keywords:
>
> - Relation: 2D representation of data
> - Database schema: A set of relation schemas
> - Relation schema: Name and a set of attributes (or fields or columns)
> - Attribute: Name and domain (e.g., integer, string, etc.)
> - Tuples: A row of a relation containing the attribute names
> - Key: Set of attributes that uniquely identify each tuple

2.3 Defining a Relation Schema in SQL

SQl is the principal language used to describe and manipulate relational databases. There are two aspects to SQL:

1. The `Data-Definition` sublanguage for declaring database schemas and
2. The `Data-Manipulation` sublanguage for querying databases and for modifying the database.

2.3.1 Relations in SQL

SQL makes a distinction between three kinds of relations.

1. Stored relations, which are called `tables`. These are the kind of relations we deal with ordinarily - a relation that exists in the database.
2. `Views`, which are relations defined by computation. These relations are not stored, but are constructed when needed.
3. `Temporary tables`, which are constructed by the SQL language processor when it performs its job of executing queries and data modifications.

2.3.2 Data Types

Let us introduce the primitive data types that are supported by SQL systems:

- `CHAR(n)` denotes a fixed-length string of up to n characters. `VARCHAR(n)` also denotes a string of up to n characters, with the difference being, that `CHAR` implies that store strings are padded to make n characters.
- `BIT(n)` denotes a bit of strings of length n and `BIT VARYING(n)` denotes bit strings of length up to n.
- `BOOLEAN` denotes an attribute whose value is logical. It can either be `TRUE`, `FALSE`, or `UNKNOWN`.
- `INT` or `INTEGER` denotes typical integer values.
- `FLOAT` or `REAL` are used for typical floating-point numbers. A higher precision can be obtained with the type `DOUBLE PRECISION`. `DECIMAL(n, d)` allows values that consist of n decimal digits, with the decimal point assumed to be d positions from the right.
- `DATE` and `TIME` are used to represent dates and times respectively. Dates are a quoted string of the form `'2021-03-02'` and times is a quoted string of the form `'13:52:31.5'`, where we may continue with a decimal point and as many significant digits as we want.

2.3.3 Simple Table Declarations

The simplest form of declaration of a relational schema consists of the key-words `CREATE TABLE` as shown below:

```
/* Code 2.7: SQL declaration of the table 'Movies' */
CREATE TABLE Movies (
    title           CHAR(100),
    releaseYear     INT,
    lengthMin       INT,
    genre           CHAR(10),
    studioName      CHAR(30),
    producerNum     INT
);
```

2.3.4 Modifying Relation Schemas

We can delete a relation R by the SQL statement `DROP TABLE R;`.

More frequently than we would drop a relation, we may need to modify the schema of an existing relation. These modifications are done by a statement that begins with the keyword `ALTER TABLE` and the name of the relation. We then have several options, the most important of which are

- `ADD` followed by an attribute name and its data type
- `DROP` followed by an attribute name

For example we could modify the *Movie* relation by adding an attribute *rating* with:

```
1       ALTER TABLE Movies ADD rating INT;
```

As another example, the `ALTER TABLE` statement:

```
1       ALTER TABLE Movies DROP studioName;
```

deletes the *studioName* attribute.

2.3.5 Default Values

When we create or modify tuples, we sometimes do not have values for all components. In general, any palce we decalre an attribute and its data type, we may add the keyword `DEFAULT` and an appropriate value. The value is either `NULL` or a constant.

As an example, we might want to use *?* as a default for *studioName* and *NULL* as a default *producerNum*. We could replace the declarations of *studioName* and *producerNum* in Code 2.7 by:

```
1       studioName      CHAR(30) DEFAULT '?',
2       producerNum     INT DEFAULT 0
```

2.3.6 Declaring Keys

There are two way to declare an attribute or set of attributes to be a key in the `CREATE TABLE` statement that defines a stored relation:

1. We may declare one attribute to be a key when that attribute is listed in the relation schema.
2. We may add to the list of items declared in the schema an additional declaration that say a particular attribute or set of attributes forms the key.

Remark: If the key consists of more than one attribute, we have to use method (2).

There are two declarations that may be used to indicate keyness:

- `PRIMARY KEY`, or
- `UNIQUE`

If `PRIMARY KEY` is used, then attributes in S, where S is the set of keys, are not allowed to have *NULL* as a value. *NULL* is permitted if the set S is declared as `UNQIUE`, however.

```
1       /* Code 2.9: Making 'fullName' the key. */
2       CREATE TABLE MovieStar (
```

```
    fullName    CHAR(30) PRIMARY KEY,
    address     VARCHAR(255),
    gender      CHAR(1),
    birthdate   DATE
);

/* Code 2.11: Making 'title' and 'releaseYear' be the key of 'Movies'. */
CREATE TABLE Movies (
    title           CHAR(100),
    releaseYear     INT,
    lengthMin       INT,
    genre           CHAR(10),
    studioName      CHAR(30),
    producerNum     INT,
    PRIMARY KEY (title, releaseYear)
);
```

2.4 An Algebraic Query Language

To begin our study of operations on relations, we shall learn about a special algebra, called `relational algebra`, that consists of some simple but powerful ways to construct new relations from given relations.

Relational algebra is not used today as a query language in commercial DBMS's. Rather, the "real" query language, SQL, incorporates relational algebra at its center.

2.4.2 What is an Algebra?

An algebra consists of operators and atomic operands. For instance, in the algebra of arithmetic, the atomic operands are variables like x and constants like 15. The operators are the usual arithmetic ones: addition, subtraction, multiplication, and division. Any algebra allows us to build `expressions` by applying operators to atomic operands and/or other expressions of the algebra.

Relational algebra is another example of an algebra. Its atomic operands are:

1. Variables that stand for relations.
2. Constants, which are finite relations.

2.4.3 Overview of Relational Algebra

The operations of the traditional relational algebra fall into four broad classes:

- The usual *set operations* - union, intersection, and difference - applied to relations.
- Operations that *remove parts of a relation*: "selection" eliminates some rows, and "projection" eliminates some columns.
- Operations that *combine the tuples of two relations*, including "Cartesian product" and various kinds of "join".
- An operation called "*renaming*" that does not affect the tuples, but changes the relation schema.

We generally shall refer to expressions of relational algebra as `queries`.

2.4.4 Set Operations on Relations

The three operations `union`, `intersection`, and `difference` are defined as follows on arbitrary sets R and S:

- $R \cup S$, the `union`, is the set of elements that are in R or S or both.
- $R \cap S$, the `intersection`, is the set of elements that are in both R and S.
- $R - S$, the `difference`. is the set of elements that are in R but not in S.

When we apply these operations to relations, we need to put some conditions on R and S:

1. R and S musthave schemas with identical sets of attributes, and the domains for each attrtibute must be the same in R and S.
2. Before we compute a set-theoretic operation of sets of tuples, the columns of R and S must be ordered so that the order of attributes is the same for both relations.

2.4.5 Projection

The `projection` operator is used to produce from a relation R a new relation that has only some of R's columns. The value of expression $\pi_{A_1,\, A_2,\ldots,\, A_n}(R)$ is a relation that has only the columns for attributes $A_1,\, A_2, \ldots,\, A_n$ of R.

Example: Assume the following relation *Movies*:

title	year	length	genre	studioName	producerNum
Star Wars	1977	124	sciFi	Fox	12345
Galaxy Quest	1999	104	comedy	DreamWorks	67890
Wayne's World	1992	95	comedy	Paramount	99999

Figure 2.13: The realtion 'Movies'.

We can now project thisrelation onto the first three attributes with the following expression:

$$\pi_{\text{title},\, \text{year},\, \text{length}}(\text{Movies})$$

The resulting relation is

title	year	length
Star Wars	1977	124
Galaxy Quest	1999	104
Wayne's World	1992	95

The result of the expression $\pi_{\text{genre}}(\text{Movies})$ is:

genre
sciFi
comedy

Remark: Notice that there are only two tuples in the resulting relation, since in the relational algebra of sets, duplicate tuples are always eliminated.

2.4.6 Selection

The `selection` operator, applied to a relation R, produces a new relation with a subset of R's tuples. The tuples in the resulting relations are those that satisfy some condition C that involves the attributes of R. We denote this operation by $\sigma_C(R)$.

C is a conditional expression of the type with which we are familiar from conventional programming languages. The only difference is that the operands in condition C are either constants or attributes of R.

Example: Let the relation *Movies* be as in Fig. 2.13. Then the value of expression $\sigma_{\text{length} \geq 100}(\text{Movies})$ is

title	year	length	genre	studioName	producerNum
Star Wars	1977	124	sciFi	Fox	12345
Galaxy Quest	1999	104	comedy	DreamWorks	67890

Another example would be the expression

$\sigma_{\text{length} \geq 100 \text{ AND studioName} = \text{'FOX'}}(\text{Movies})$

which results in the following relation:

title	year	length	genre	studioName	producerNum
Star Wars	1977	124	sciFi	Fox	12345

2.4.7 Cartesian Product

The `Cartesian product` of two sets R and S is the set of pairs that can be formed by choosing the first element of the pair to be any element of R and the second any element of S.

To disambiguate an attribute A that is in the schemas of both R and S, we use $R.A$ ofr the attribute from R and $S.A$ for the attribute from S.

Example: Consider the following two relations:

A	B
1	2
3	4

Figure 2.14: (a) Relation R

B	C	D
2	5	6
4	7	8
9	10	11

Figure 2.14: (b) Relation S

Then the result of the expression $R \times S$ is given by:

A	R.B	S.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Figure 2.14: (c) Result of $R \times S$

2.4.8 Natural Joins

Often we find a need to `join` them by pairing onlythose tuples that match in some way. The simplest sort of match is the `natural join` of two relations R and S, denoted $R \bowtie S$, in which we pair only those tuples from R and S that agree in whatever attributes are common to the schemas of R and S.

More precisely, let $A_1, A_2, \ldots, A_n$ be all the attributes that are in both the schema R and the schema of S. Then a tuple r from R and a tuple s from S are successfully paired if and only if r and s agree on each of the attributes $A_1, A_2, \ldots, A_n$.

If two tuples r and s are successfully paired, then the resulting tuple is called the `joined tuple`. A tuple that fails to pair with any tuple of the other relation in a join is said to be a `dangling tuple`.

Example: The natural join of the relations R and S from Fig. 2.14(a) and (b) is:

A	B	C	D
1	2	5	6
3	4	7	8

2.4.9 Theta-Joins

It is sometimes desireable to pair tuples from two relations on some other basis. For that purpose, we have a related notation called the `theta-join`. Historically, the "theta" refers to an arbitrary condition, which we shall represent by C.

The notation for a theta-join of relations R and S based on condition C is $R \bowtie_C S$. The result of this operation is constructed as follows:

1. Take the product of R and S.
2. Select from the product only those tuples that satisfy the condition C.

Example: Let $C := R.B < S.B$, then the result of the expression $R \bowtie_{R.B<S.B} S$ with the relations R and S as in Fig. 2.14(a) and (b) is:

A	R.B	S.B	C	D
1	2	4	7	8
1	2	9	10	11
3	4	9	10	11

2.4.10 Combining Operations to Form Queries

Relational algebra, like all algebras, allows us to form expressions of arbitrary complexity by applying operations to the result of other operations. One can construct expressions of relational algebra by applying operators to subexpressions, using parantheses when necessary to indicate grouping of operands.

Example: Suppose we want to know, from our *Movies* relation, "What are the titles and years of movies made by Fox that are at least 100 minutes long?". One way to compute the answer to this query is:

1. Select those *Movies* tuples that have $\text{length} \geq 100$.
2. Select those *Movies* tuples that have $\text{studioName} = \text{'Fox'}$.
3. Compute the intersection of (1) and (2).
4. Project the relation from (3) onto the attributes *title* and *year*.

Alternatively, we could represent the same expression in a conventional, linear notation, with parantheses. The formula:

$$\pi_{\text{title, year}}\left(\sigma_{\text{length} \geq 100}(\text{Movies}) \cap \sigma_{\text{studioName} = \text{'Fox'}}(\text{Movies})\right)$$

represents the same expression.

2.4.11 Naming and Renaming

We shall use the operator $\rho_{S(A_1, A_2, \ldots, A_n)}(R)$ to rename a relation R. The resulting relation has exactly the same tuples as R, but the name of the relation is S. Moreover, the attributes of the result relation S are named $A_1, A_2, \ldots, A_n$.

If we only want to change the name of the realtion to S and leave the attributes as they are in R, we can just say $\rho_S(R)$.

2.4.12 Relationships Among Operations

Some of the operations that we described in Section 2.4 can be expressed in terms of other relational-algebra operations.

`Intersection` can be expressed in terms of set difference:

$$R \cap S = R - (R - S)$$

`Theta-join` can be expressed by product and selection:

$$R \bowtie_C S = \sigma_C(R \times S)$$

Let C be of the form $R.A_1 = S.A_1 \text{ AND } R.A_2 = S.A_2 \text{ AND } \ldots$, where $A_1, A_2, \ldots, A_n$ are all the attributes appearing in the schemas of both R and S. Let L bet the list of attributes in the schema of R followed by those attributes in the schema of S that are not also in the schema of R. Then we can represent `natural join` by:

$$R \bowtie S = \pi_L\Big(\sigma_C(R \times S)\Big)$$