RDBMS and SQL

Roudranil Das

RDBM Sep-Nov 2022
Roll No: MDS202227
roudranil@cmi.ac.in

October 1, 2022

Contents

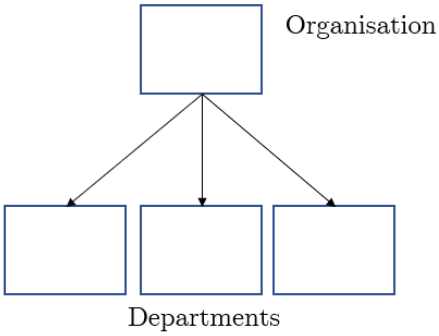
1 September 15, 2022

1.1 Introduction

- Need to organise and manipulate (large) volumes of data in a systematic way.
  - We could have customised formats for each requirement
  - Update and query
  - manipulation: requires knowledge of implementation. This brings up the argument of *imperative vs declarative* programming

What is the "best" way to declaratively work with data? The following methods were proposed:

IBM - Hierarchical model, like a tree (1960)



Codd - He came up with the idea that information should be stored in flat tables.
Rows are items
Columns are attributes

See the following
Parts:

ID-parts	...
...	...

Projects:

ID-projects	...
...	...

Parts in projects:

ID-parts	ID-projects	No.
...	...	...

However even though everybody agreed that Codd's method was better, nobody really decided on who would implement it. Fortunately IBM internally decided to do it. Out came the *System R project*. From there came `SQL` (structured query Language).

1.2 Relational Query Languages

- Procedural (imperative) vs Non procedural (declarative)
- Pure langugages:
  - Relational Algebra
  - Tuple relational calculus
  - Domain relational calculus
- The above 3 pure languages are identical in computing power.
- Why relational algebra?
  - Not turing-machine equivalent
  - consists of 6 basic operations

1.3 Relational Database

Mathematically, what we saw above is a relational database: there are sets S_1, S_2, and $S_1 \times S_2 \to$ Cartesian product. Say $S_1 =$ All possible names, $S_2 =$ All possible dates of birth.

We define a relation $R \subseteq S_1 \times S_2$. We can also do $D \subseteq S_1 \times S_2 \times \cdots \times S_n$.

A table is a relation.

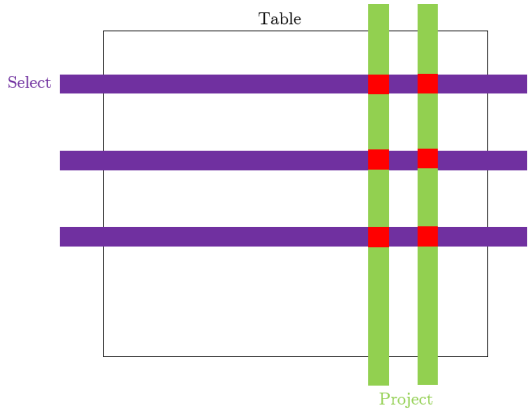
Type - Allowed values of Column j in S_j

Limitations:

- Relations are sets of tuples → there cannot be any duplicates.
- All columns must exist.
- Atomicity: every value must belong to the set, so much depends on what the set is. For example, in a library some books may have many authors. So is our set a set of all authors, or set of all subsets of all authors? That will decide the atomicity of the entries.

We need a mechanism to extract useful information from tables (relations). Operators to manipulate data in tables: - "Relational Algebra" (remember that outputs of σ and π are implicitly other tables)

1. Extract rows (items) based on some criterion: `Select` σ
   Notation: $\sigma_{\text{condition}}(\text{relation})$
   Example: $\sigma_{\text{supplier = "XYZ"}}$ (parts), $\sigma_{C_1 \wedge C_2}(\theta)$
2. Retain only relevant columns (attributes): `Project` π
   Notation: $\pi_{\text{column names}}(\text{relation})$
   Example: $\pi_{\text{company, salary}}(\text{Placement})$



3. `Cartesian product` $\times$
   Notation: $r_1 \times r_2$
4. `Renaming values` ρ
   Notation: $\rho_{f_1}(r_1) \times \rho_{f_2}(r_2)$
5. Output the union of tuples from two input relations: `Union` $\cup$
   Notation: $\pi_{\text{column}}(A) \cup \pi_{\text{column}}(B)$
6. Output the set difference: `Set difference` $-$
   Notation: $\pi_{\text{column}}(A) - \pi_{\text{column}}(B)$

Supppose we have courses data at CMI, say we want to have separate tables for each course. Say, we want to ask which students are registered in two courses. If we want to do this imperatively, then we can jsut run two loops across all courses and get our answer. What if we want to do it declaratively?

What we do is concatenating the two course tables, kind of like a cartesian product between their rows. For example if the first course had 2 students and the second one had 3, then our resulting table would have 6 rows. One small glitch would be that multiple columns might have the same name, which can be easily addressed by renaming them suitably. This should be done before we take the product. In this new table, we check for the rows with the two roll number columns having the same entry.

In practicse an easier solution is to use relation + column (rather than rename and then do stuff), this is provided the relations have different names. Then even if the columns have same names its okay. Otherwise we have to rename, either relations or columns.

2 September 22, 2022

2.1 Designing relations

Say we have the following database: information about course registrations:

- Student info: roll no, name, email
- Course info: code, title, instructor
- Registrations: which student is taking which course

One thing that we imagine that we can always do is create a single table large enough to store every bit of info that we want. How would that loook like?

Roll	Name	email	Code	title	instructor

This is a wasteful form of storing data, because there will not only be multiple occurances of something, this will be wastage of space and concurrent updates will be difficult. Another not obvious problem will be a student who is not taking any courses will not be here in this table. But we would obviously want to keep track of that student. What about courses that are not offered now? This is known as the Problem of expressiveness: Where to store records who do not fit the entire criteria?

Solution? Multiple tables - make a student table (with the roll, name and email), make a courses table (with code, title, instructor). Then the registrations table will have the necessary info, and say the grade as well. Suppose we want to know now the list of students taking the course RDBMS? Provided we know the course code of this course (from the courses table), we can find out the roll number of the students taking this course (from the registrations table). Then from the roll we can find the name and the email (from the students table). This is "Operational" or "Imperative" computation. To find the emails of the students:

$$\pi_{\text{email}}\left(\sigma_{\text{reg.code = rdbms.code}}\left(\text{reg} \times \left(\pi_{\text{code}}(\sigma_{\text{title = "RDBMS"}}(\text{courses}))\right)\right)\right)$$

2.2 Join

Now we have doing $\sigma_{\text{condition}}(r_1 \times r_2)$ very frequently. This is known as **Join**. This is also sometimes known as θ - Join.

The θ is more commonly $r_1 \cdot c = r_2 \cdot c$, which is generally known as **Natural Join**. Symbolically, $\theta - \text{join}(r_1 r_2)$ or $\sigma_\theta(r_1 \times r_2)$ or $r_1 \bowtie_\theta r_2$. If θ is equally shared across columns then $r_1 \bowtie r_2$.

2.3 Keys

Column / Attribute / Combination of columns which acts something like an identifier, it is unique to a row. For example in our student info table, the key was the roll number. Also the email, as the emails are unique. In the coures info table, the key was the course code. (title may not be unique). For registrations table, neither roll nor code are keys (same student may take multiple course, same course taken by many students). However the combination of roll and course code is a key as the same student cant register in the same course multiple times.

- Let $k \subseteq R$ (R being our relation)
- K is called a **Superkey** of R if values of K are sufficient to identify a unique tuple of each possible relation $r(R)$, for example, in the previous case for students, we have both roll and email, as well as (roll, email).
- Superkey K is a **Candidate key** is K is minimal (my guess is that it is only 1 column), for example roll and email but not (roll, email).
- One of the candidate keys is selected to be the **Primary key**.

- **Foreign key constraint**: Values in one relation must appear in another.
  - **Referencing relation**
  - **Referenced relation**

2.4 Index

- Semnatic constraint on values in a table.
- exploit to store data more efficiently

Student	Course	Regn
Roll	CCode	(Roll, CCode)

Moreover note that all roll, code in registrations table has to exist in the students table and courses table.

2.5 Attribute types

- The *set of values* for each attribute is called the Domain of the attribute.
- Attribute values are generally required to be Atomic, that is, indivisible.
- The special value **null** is a member of every domain, indicating that the value is unknown.
- The null value causes complications in the definitions of many operations.