Databases	Lecture 4
ENGR 3599	Feb 14, 2019

Relational Algebra and Querying

We can certainly query a Relational Model using low-level CRUD operations that give us access to each relation, getting a list of keys of a relation, and looking up tuples by key.

But the Relational Model supports a particularly nice way to query data, via an algebra of operations.

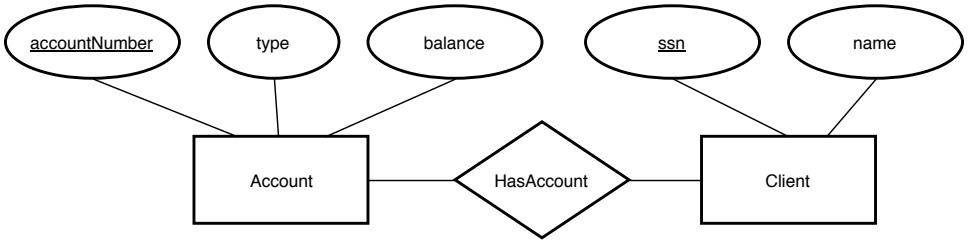
Algebra = operations + equality between sequence of operations.

Relational algebra defines operations that take schematized relations (relations with an associated schema) and return schematized relations. There are many variants of relational algebra, so we focus on core operations common to all of the variants.

Relational algebra operations are used to query the data, at least when we can express the result of a query as a schematized relation.

These operations are selection, projection, union, product, and renaming.

Examples will involve the following set of relations, capturing a simple database of bank accounts, with the following ER diagram:



This can be translated to the following relations in the relational model, populated with some sample data:

Account:

accountNumber	type	balance
991	checking	1000
992	saving	500
993	checking	50
994	checking	100

Client:

ssn	name
1111	Riccardo
2222	Taylor
3333	Alexandra

HasAccount:

accountNumber	ssn
991	1111
992	1111
993	2222
994	1111
994	3333

Selection: Selection is used to filter out tuples from a relation.

Operation σ_p where p is a tuple predicate takes a schematized relation and returns a schematized relation with the same schema, but whose tuples are restricted to only those tuples that satisfy the predicate.

We're remaining vague about what we take as possible predicates. Generally, this will be a test on the values of components of a tuple.

Example: $\sigma_{\text{name=Riccardo}}(\mathsf{Client})$ yields the following schematized relation:

ssn	name
1111	Riccardo

whereas $\sigma_{\text{type=Checking}}(\mathsf{Account})$ yields:

accountNumber	type	balance
991	checking	1000
993	checking	50
994	checking	100

whereas $\sigma_{\text{type=Checking}\wedge\text{balance}>90}(\mathsf{Account})$ yields:

accountNumber	type	balance
991	checking	1000
994	checking	100

Projection: Projection is used to disregard components (columns) from a relation.

Operation $\pi_{a_1,\ldots,a_k}$ where $a_1, \ldots, a_k$ are labels takes a schematized relation and returns a schematized relation in which only components corresponding to labels $a_1, \ldots, a_k$ are kept from each tuple in the relation.

Example: $\pi_{\text{name}}(\mathsf{Client})$ yields the following schematized relation:

name
Riccardo
Taylor
Alexandra

where $\pi_{\text{accountNumber,balance}}(\mathsf{Account})$ yields:

accountNumber	balance
991	1000
992	500
993	50
994	100

Note that since relations are sets of tuples, projection may force tuples to be identified. Thus, $\pi_{\text{type}}(\mathsf{Account})$ yields the schematized relation:

type
checking
saving

Another approach, the one actually used by modern databases is to take relations not to be sets of tuples, but multisets of tuples. This introduces complications of its own.[1]

[1] A multiset A in universe U (that is, where the elements of A are taken to be from U) is actually a function $\overline{A} \to \mathbb{N}$ where $\overline{A}$ is the set of elements in A. So working with multisets is really working with functions.

Union: Union is used to combine two relations.

Operation $\cup$ takes two schematized relations and returns a schematized relation with all the tuples from either relation. In order for that to even make sense, the two relations need to be compatible. There are a few ways of defining compatibility, some more stringent than others. The simplest to use is simply to require that the two relations have the same schema (the same columns in the same positions) and that the domains associated with the labels are the same in both relations.

Example: Suppose we have a relation Client2 with:

ssn	name
4444	Alice
5555	Bob
6666	Charlie

then Client $\cup$ Client2 yields the schematized relation:

ssn	name
1111	Riccardo
2222	Taylor
3333	Alexandra
4444	Alice
5555	Bob
6666	Charlie

Note that if Client2 held the tuple (2222,Darlene), the union Client $\cup$ Client2 would not be defined because component ssn is a primary key, and the union would require there being two distinct tuples with ssn value 2222, contradicting that ssn is a primary key.

Cartesian Product: Cartesian product is used to join two relations together.

Operations $\times$ takes two schematized relations R and S and creates a new schematized relation in which the tuples are all the possible concatenations of a tuple from R and a tuple of S, and the schema is a concatenation of the schema of R and the schema of S, with the indices of the labels suitably adjusted.

In order for this to make sense, we need to deal with the case where the schema of R and the schema of S share labels – a naive concatenation of the schemas would yield a schema where the label appears twice, which is nonsensical.

There are two solutions to the problem of shared labels: either we modify the labels automatically so that the label is tagged by the name of the relation it comes from (which requires every relation to have a name), or we simply forbid the product of two schematized relations with shared labels. We take the second approach here.

Example: The product Client $\times$ Account yields the schematized relation:

accountNumber	type	balance	ssn	name
991	checking	1000	1111	Riccardo
991	checking	1000	2222	Taylor
991	checking	1000	3333	Alexandra
992	saving	500	1111	Riccardo
992	saving	500	2222	Taylor
992	saving	500	3333	Alexandra
993	checking	50	1111	Riccardo
993	checking	50	2222	Taylor
993	checking	50	3333	Alexandra
994	checking	100	1111	Riccardo
994	checking	100	2222	Taylor
994	checking	100	3333	Alexandra

In general, the product of a relation with M tuples and a relation with N tuples will yield a relation with MN tuples.

Renaming: Renaming lets you change the labels of a schematized relation. This is useful to make schematized relations compatible for union or Cartesian product.

Operations $\rho_{a_1 \mapsto b_1, \ldots, a_k \mapsto b_k}$ takes a schematized relation and return a schematized relation with the same tuples but with a schema in which name a_i has been changed to b_i.

Example: we can rename the columns in HasAccount in order to join it with Client or Account using $\rho_{\text{accountNumber} \mapsto \text{HAaccount}, \text{ssn} \mapsto \text{HAssn}}(\text{HasAccount})$ to yield the schematized relation:

HAaccount	HAssn
991	1111
992	1111
993	2222
994	1111
994	3333

With all of these pieces in place, we can now write queries such as *what are the names of clients with at least one account with a balance of more than 200?*, such as

$$\pi_{\text{name}}(\sigma_{\text{balance}>200}(\sigma_{\text{accountNumber=HAaccount,ssn=HAssn}}(\mathsf{Client} \times \mathsf{Account} \times \overline{\mathsf{HasAccount}})))$$

where $\overline{\mathsf{HasAccount}}$ is the schematized relation

$$\rho_{\text{accountNumber}\mapsto\text{HAaccount,ssn}\mapsto\text{HAssn}}(\mathsf{HasAccount})$$

If you work the operations, you get the final schematized table

name
Riccardo

which is the result of our query.

Note that we could have implemented the same query with a different sequence of operations, namely, first restriction our attention to accounts with a balance > 200, and taking the product of those accounts with clients and their account information before projecting out the resulting names. You can check that the following expression

$$\pi_{\text{name}}(\sigma_{\text{accountNumber=HAaccount,ssn=HAssn}}(\mathsf{Client} \times \sigma_{\text{balance}>200}(\mathsf{Account}) \times \overline{\mathsf{HasAccount}}))$$

yields the same result

name
Riccardo

The fact that multiple sequence of operations yield the same result indicate that Relational Algebra is still a fairly operational approach to querying: it still requires us to determine what to do to a set of relations in order to extract the required information. Some sequence of operations may yield the result more efficiently — for instance, by building fewer intermediate relations. In the example above, the second way to compute the query should be faster, simply because the intermediate table produced by the three-way product is smaller. In general, it is not necessarily obvious which of the possibly many ways of sequencing operations to perform a query is better.

Existing database systems usually do not use Relational Algebra-like operations to describe queries, but rather a more declarative query language that does not prescribe the order of operations. The most common declarative query language is undoubtedly SQL (for Structure Query Language). A typical SQL query corresponding to the above query would look like:

```
SELECT Client.name
```

```
FROM Client, Account, HasAccount
WHERE Client.ssn = HasAccount.ssn
  AND Account.accountNumber = HasAccount.accountNumber
  AND Account.balance > 200
```

This does not specify an order of operations. Internally, most databases will take a SQL query such as this and go through a process called *query planning and optimization* to translate the query into a more procedural query using Relational Algebra-like operations. The task of the query planner is to figure out the most efficient sequence of operations (among the many possible) that will yield the correct result.

Formalization:

One advantage of relational algebra is that it can be made formal and be given an unambiguous semantics.

Recall that a relation R is a set of tuples, where if $t = (c_1, \ldots, c_k)$ is a tuple, then we write $t[i]$ for c_i.

A schematized relation is written $\langle R, \chi \rangle$, where R is an n-ary relation and $\chi : \mathcal{L} \to \{1, \ldots, n\}$ is a bijection where $\mathcal{L}$ is the set of labels of components of R. Intuitively, $\chi(a)$ gives the position of component named a in the tuple. If t is a tuple in a schematized relation $\langle R, \chi \rangle$, accessing a tuple component by name is written $t[a]$, and is a shorthand for $t[\chi(a)]$.

Selection:

$$\sigma_p \langle R, \chi \rangle = \langle \{t \mid t \in R, p(t) = \mathit{true}\}, \chi \rangle$$

Projection:

$$\pi_{a_1,\ldots,a_k} \langle R, \chi \rangle = \langle \{(t[a_1], \ldots, t[a_k]) \mid t \in R\}, \chi' \rangle$$

where $dom(\chi') = \{a_1, \ldots, a_k\}$ and $\chi'(a_i) = i$.

Union:

$$\langle R_1, \chi_1 \rangle \cup \langle R_2, \chi_2 \rangle = \langle R_1 \cup R_2, \chi_1 \rangle$$

This is only defined when $\chi_1 = \chi_2$.

Product:

$$\langle R_1, \chi_1 \rangle \times \langle R_2, \chi_2 \rangle = \langle \{t_1 + t_2 \mid t_1 \in R_1, t_2 \in R_2\}, \chi' \rangle$$

where $t_1 + t_2$ is tuple concatenation,

$$\chi'(a) = \begin{cases} \chi_1(a) & \text{if } a \in dom(\chi_1) \\ \chi_2(a) + n & \text{if } a \in dom(\chi_2) \end{cases}$$

and $|dom(\chi_1)| = n$. This is only defined when $dom(\chi_1) \cap dom(\chi_2) = \varnothing$.

Renaming:

$$\rho_{a_1 \mapsto b_1, \ldots, a_k \mapsto b_k} \langle R, \chi \rangle = \langle R, \chi' \rangle$$

where

$$\chi'(a) = \begin{cases} \chi(a_i) & \text{if } a = b_i \text{ for some } i \in \{1, \ldots, k\} \\ \chi(a) & \text{if } a \in dom(\chi) - \{b_1, \ldots, b_k\} \end{cases}$$