Informatics 1B, 2008

School of Informatics, University of Edinburgh

Data and Analysis

Note 4

Tuple-relational Calculus

Alex Simpson

Part I — Structured Data

Data Representation:

Note 1 The entity-relationship (ER) data model

Note 2 The relational model

Data Manipulation:

Note 3 Relational algebra

Note 4 Tuple-relational calculus

Note 5 The SQL query language

Motivation

Tuple-relational calculus is another way of writing queries for relational data.

Its power lies in the fact that it is entirely *declarative*

That is, we specify the properties of the data we are interested in retrieving, but (in contrast to relational algebra) we do not describe a method by which the data can be retrieved

Basic format

Queries are based on *tuple variables*.

Each tuple variable has an associated schema. The variable ranges over all possible tuples of values matching the schema declaration.

A query has the form

$$\{T \mid p(T)\}$$

where T is a tuple variable and $p(T)$ is a (first-order logic) formula (in which the tuple variable T occurs free).

The result of this query is the set of all possible tuples t (consistent with the schema of T) for which the formula $p(T)$ evaluates to true with $T = t$

Simple example

Find all students at least 19 years old

$$\{S \mid S \in \texttt{Students} \wedge S.\texttt{age} > 18\}$$

In detail:

- Tuple variable S is introduced
- S instantiated over all tuples in the Students table
- Predicate $S.\texttt{age} > 18$ is evaluated on each individual tuple
- If and only if the predicate evaluates to true, the tuple is propagated to the output

Formal syntax of atomic formulae

An *atomic formula* is one of the following:

- $\boldsymbol{R} \in \mathit{Rel}$
- $\boldsymbol{R.a}\ \ \mathit{op}\ \ \boldsymbol{S.b}$
- $\boldsymbol{R.a}\ \ \mathit{op}\ \ \mathit{constant}$
- $\mathit{constant}\ \ \mathit{op}\ \ \boldsymbol{S.b}$

where: $\boldsymbol{R}, \boldsymbol{S}$ are tuple variables, *Rel* is a relation name, $\boldsymbol{a}, \boldsymbol{b}$ are attributes of $\boldsymbol{R}, \boldsymbol{S}$ respectively, and *op* is any operator in the set $\{>, <, =, \neq, \geq, \leq\}$

Formal syntax of (composite) formulae

A *formula* is (recursively defined) to be one of the following:

- any atomic formula
- $\neg \boldsymbol{p}$, $\boldsymbol{p} \wedge \boldsymbol{q}$, $\boldsymbol{p} \vee \boldsymbol{q}$, $\boldsymbol{p} \Rightarrow \boldsymbol{q}$
- $\exists \boldsymbol{R}.\ \boldsymbol{p}(\boldsymbol{R})$, $\forall \boldsymbol{R}.\ \boldsymbol{p}(\boldsymbol{R})$

where $\boldsymbol{p}(\boldsymbol{R})$ denotes a formula in which the variable $\boldsymbol{R}$ appears free.

N.B. First-order logic was introduced in more detail in Inf1A Computation & Logic. Here, we use different notation for the connectives: $\neg$ for *not*; $\wedge$ for *and*; $\vee$ for *or*; and $\Rightarrow$ for $\longrightarrow$. Our notation agrees with Ramakrishnan & Gehrke "Database Management Systems". The main difference from standard first-order logic is the use of variables ranging over tuples (rather than individuals), and the correspondingly specialised forms of atomic formula.

A subtle point

In ordinary first-order logic, one can, in theory, form quantifications $\exists \boldsymbol{R}.\ \boldsymbol{p}$ and $\forall \boldsymbol{R}.\ \boldsymbol{p}$ even when $\boldsymbol{R}$ does not occur in $\boldsymbol{p}$. (In practice, such quantifications are normally useless since they are vacuous.)

In tuple-relational calculus we only allow $\exists \boldsymbol{R}.\ \boldsymbol{p}$ and $\forall \boldsymbol{R}.\ \boldsymbol{p}$ when $\boldsymbol{R}$ occurs free in $\boldsymbol{p}$ for the following reason.

- Under this rule, every tuple variable $\boldsymbol{R}$ that appears in a formula is forced to appear in at least one atomic subformula. The atomic formulae in which $\boldsymbol{R}$ appears then determine the schema of $\boldsymbol{R}$. The schema is taken to be the smallest one containing all the fields that are declared as attributes of $\boldsymbol{R}$ within the formula itself.

Illustrative example

An example illustrating the previous point.

$$\{P \mid \exists S \in \texttt{Students}\,(S.\texttt{age} > 20 \;\wedge\; P.\texttt{name} = S.\texttt{name} \;\wedge\; P.\texttt{age} = S.\texttt{age})\}$$

- The schema of S is that of the **`Students`** table. This is declared by the atomic formula $S \in \texttt{Students}$.
- The schema of P has just two fields **`name`** and **`age`**, with the same types as the corresponding fields in **`Students`**.
- The query returns a table with two fields **`name`** and **`age`** containing the names and ages of all students aged 21 or over.

Note the use of $\exists S \in \texttt{Students}\,(p)$ for $\exists S\,(S \in \texttt{Students} \wedge p)$.
We make free use of such (standard) abbreviations.

Further examples (1)

Query: Find the names of students who are taking Informatics 1

Relational algebra:

$$\pi_{\texttt{Students.name}}(\texttt{Students} \bowtie_{\texttt{Students.mn}=\texttt{Takes.mn}} (\texttt{Takes} \bowtie_{\texttt{Takes.code}=\texttt{Courses.code}} (\sigma_{\texttt{name}=\text{'Informatics 1'}}(\texttt{Courses}))))$$

Tuple-relational calculus:

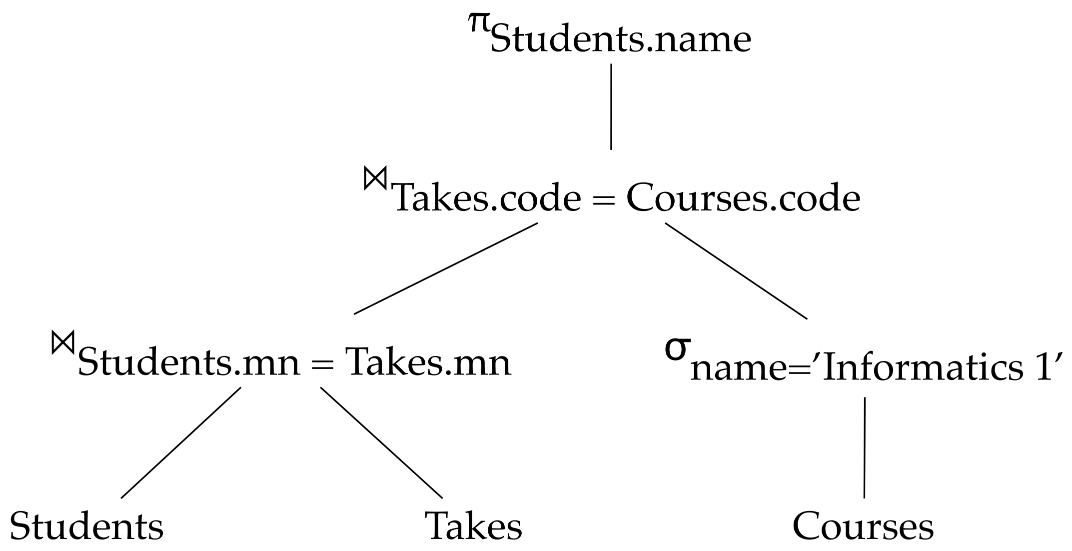
$$\{P \mid \exists S \in \texttt{Students}\ \exists T \in \texttt{Takes}\ \exists C \in \texttt{Courses}\ (C.\texttt{name} = \text{'Informatics 1'} \ \wedge\ C.\texttt{code} = T.\texttt{code} \ \wedge\ S.\texttt{mn} = T.\texttt{mn} \ \wedge\ P.\texttt{name} = S.\texttt{name})\}$$

Tree representation of algebraic expression (abstract syntax)

For the previous query, changing the bracketing does not change the query.

$$\pi_{\texttt{Students.name}}((\texttt{Students} \bowtie_{\texttt{Students.mn=Takes.mn}} \texttt{Takes}) \bowtie_{\texttt{Takes.code=Courses.code}} (\sigma_{\texttt{name=}\text{'Informatics 1'}}(\texttt{Courses})))$$

A tree representation can help one visualise a relational algebra query.



Further examples (2)

Query: Find the names of all courses taken by (everyone called) Joe

Relational algebra:

$$\pi_{\texttt{Courses.name}}((\sigma_{\texttt{name}=\text{'Joe'}}(\textbf{Students})) \bowtie_{\texttt{Students.mn}=\texttt{Takes.mn}} (\textbf{Takes} \bowtie_{\texttt{Takes.code}=\texttt{Courses.code}} \textbf{Courses}))$$

Tuple-relational calculus:

$$\{P \mid \exists S \in \textbf{Students}\ \exists T \in \textbf{Takes}\ \exists C \in \textbf{Courses}\ (S.\textbf{name} = \text{'Joe'} \ \wedge\ S.\textbf{mn} = T.\textbf{mn} \ \wedge\ C.\textbf{code} = T.\textbf{code} \ \wedge\ P.\textbf{name} = C.\textbf{name})\}$$

Further examples (3)

Query: Find the names of all students who are taking Informatics 1 or Geology 1

Relational algebra:

$$\pi_{\texttt{Students.name}}(\texttt{Students} \bowtie_{\texttt{Students.mn=Takes.mn}}$$
$$(\texttt{Takes} \bowtie_{\texttt{Takes.code=Courses.code}}$$
$$(\sigma_{\texttt{name='Informatics 1'} \vee \texttt{name=}\text{'Geology 1'}}(\texttt{Courses}))))$$

Tuple-relational calculus:

$$\{P \mid \exists S \in \texttt{Students}\ \exists T \in \texttt{Takes}\ \exists C \in \texttt{Courses}$$
$$((C.\texttt{name} = \text{'Informatics 1'} \ \vee \ C.\texttt{name} = \text{'Geology 1'}) \ \wedge$$
$$C.\texttt{code} = T.\texttt{code} \ \wedge S.\texttt{mn} = T.\texttt{mn} \ \wedge \ P.\texttt{name} = S.\texttt{name})\}$$

Further examples (4)

Query: Find the names of students who are taking both Informatics 1 and Geology 1

Relational algebra:

$$
\begin{aligned}
&\pi_{\text{Students.name}}(\\
&\quad(\texttt{Students} \bowtie_{\texttt{Students.mn}=\texttt{Takes.mn}}\\
&\qquad(\texttt{Takes} \bowtie_{\texttt{Takes.code}=\texttt{Courses.code}}\\
&\qquad\quad(\sigma_{\texttt{name}=\text{`Informatics 1'}}(\text{Courses}))))\\
&\quad\cap\\
&\quad(\texttt{Students} \bowtie_{\texttt{Students.mn}=\texttt{Takes.mn}}\\
&\qquad(\texttt{Takes} \bowtie_{\texttt{Takes.code}=\texttt{Courses.code}}\\
&\qquad\quad(\sigma_{\texttt{name}=\text{`Geology 1'}}(\text{Courses}))))\ )
\end{aligned}
$$

Further examples (4 continued)

Query: Find the names of students who are taking both Informatics 1 and Geology 1

Tuple-relational calculus:

$$\{P \mid \exists S \in \texttt{Students}\,(P.\texttt{name} = S.\texttt{name} \;\wedge$$
$$\forall C \in \texttt{Courses}$$
$$((C.\texttt{name} = \text{‘Informatics 1’} \;\vee\; C.\texttt{name} = \text{‘Geology 1’}) \Rightarrow$$
$$(\exists T \in \texttt{Takes}\,(T.\texttt{mn} = S.\texttt{mn} \;\wedge\; T.\texttt{code} = C.\texttt{code})))\,\}$$

Exercise. What does this query return in the case that there is no course in **`Courses`** called ‘Geology 1’? Find a way of rewriting the query so that it only returns an answer if both ‘Informatics 1’ and ‘Geology 1’ courses exist.

Further examples (5)

Query: Find the names of all students who are taking all courses

Tuple-relational calculus:

$$\{P \mid \exists S \in \texttt{Students}\,(P.\texttt{name} = S.\texttt{name} \wedge$$
$$\forall C \in \texttt{Courses}$$
$$(\exists T \in \texttt{Takes}\,(T.\texttt{mn} = S.\texttt{mn} \ \wedge \ T.\texttt{code} = C.\texttt{code})))\,\}$$

Exercise. Try to write this query in relational algebra.

Relational algebra and tuple-relational calculus compared

Relational algebra (RA) and tuple-relational calculus (TRC) have the *same* expressive power

That is, if a query can be expressed in RA, then it can be expressed in TRC, and vice-versa

Why is it useful to have both approaches?

Declarative versus procedural

Recall that TRC is *declarative* and RA is *procedural*.

This suggests the following methodology.

- *Specify* the data that needs to be retrieved using TRC.
- Translate this to an *equivalent query* in RA that gives an *efficient method* of retrieving the data.

This methodology underpins practical approaches to *query optimisation* in relational databases.

In practice, queries are written in a real-world query language such as SQL, rather than TRC.

Nevertheless, query optimisation is of enormous importance in applications.