ECS 165B: Database System Implementation
Lecture 5

UC Davis
April 8, 2011

Agenda

- Last time – file and buffer management; subversion and DavisDB logistics
- Today (something completely different) –

  A taste of database theory, Part 1
  relational algebra, relational calculus, and first-order logic

Announcements

Reminder: project Part 1 due in 9 days

Don't forget about `writeup.txt` (in particular, estimate time before you start coding)

Introduction to Database Theory, I: SQL, relational algebra, and relational calculus

SQL, relational algebra, and relational calculus

- In ECS165A, you already saw two different database query languages, SQL and relational algebra (RA):

```
select R.A, S.C
from R, S
where R.B = S.B
```

versus $\pi_{A,C}(R \bowtie S)$

- Today we'll look at a third query language, relational calculus:

$$\{(x, z) \mid \exists y \; R(x, y) \wedge S(y, z)\}$$

  Note, refers to attributes by position rather than by name; "unnamed" relational algebra does this too:

$$\pi_{1,4}(\sigma_{2=3}(R \times S))$$

Why talk about relational calculus (RC)?

- To show that SQL is not just some ad-hoc language that people cooked up in the 70s; rather, it is just first-order logic (FO) in disguise! Main result we'll see today:

  $$SQL = RA = RC = FO$$

  "Logic is the calculus of computer science" – Manna and Waldinger 1985

- Known results about first-order logic can be transferred to SQL

- Convenient formalism when considering query containment and query equivalence (we'll see these in another lecture)

Review: relational algebra (RA)

We'll use the "unnamed" version of the relational algebra:

- **predicate.** R
- **selection.** $\sigma_{i=j}(E)$ or $\sigma_{i=c}(E)$
- **projection.** $\pi_{i_1,\ldots,i_k}(E)$
- **cartesian product.** $E_1 \times E_2$
- **union.** $E_1 \cup E_2$
- **difference.** $E_1 - E_2$.

A "join" in the unnamed relational algebra is expressed using selection, projection, and cartesian product. No need for **renaming** (no names!), or **intersection** $E_1 \cap E_2$ (why?).

Introducing the relational calculus (RC)

- Database query language based on first-order logic
- **Syntax:** expressions of the form

  $$\{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\}$$

  where $\varphi(x_1, \ldots, x_n)$ is a first order formula with free variables $x_1, \ldots, x_n$.
- **Semantics:** return all tuples $(a_1, \ldots, a_n)$ such that $\varphi(a_1, \ldots, a_n)$ is true in the database.

What is a first-order formula?

- An expression built up using
  - **variables.** x, y, z, ...
  - **constants.** "Joe", 42, ...
  - **predicate symbols.** names of database relations
  - **logical connectives.** $\wedge, \vee, \neg, \rightarrow$
  - **equality** $=$
  - **quantifiers** $\forall, \exists$
- Examples of first-order formulae:
  - $\forall x \forall y \forall z\ R(x, y) \wedge R(y, z) \rightarrow R(x, z)$
  - $\forall x\ R(x, x)$
  - $\forall x \forall y\ R(x, y) \rightarrow R(y, x)$

  Q: what do three formulae above together say of R?
  - $S(x, x) \vee \exists y\ R(x, y)$ (x is a free variable)

Example: relational calculus queries

Database with three relations: **Class**(classId, className, roomNo); **Student**(studentId, studentName); and **Takes**(studentId, classId).

- "Find all students taking a class meeting in Wellman 1"

$$\{(x) \mid \exists s \exists c \exists n \ \textbf{Student}(s, x) \wedge \textbf{Takes}(s, c) \\ \wedge \textbf{Class}(c, n, \text{"Wellman 1"})\}$$

- "Find all pairs of students not taking a class together"

$$\{(x, y) \mid \exists s \exists s' \ \textbf{Student}(s, x) \wedge \textbf{Student}(s', y) \wedge \\ \neg \exists c (\textbf{Takes}(s, c) \wedge \textbf{Takes}(s', c))\}$$

Example: relational calculus queries (2)

Database with three relations: **Sailor**(sid, name, rating, age); **Boats**(bid, color); and **Reserves**(sid, bid, day).

- "Find all sailors with a rating above 7"
- "Find sailors rated above 7 who've reserved a red boat"
- "Find sailors who've reserved all boats"

Ruling out "bad" relational calculus queries

- It is possible to write relational calculus queries that return (a) **infinitely** many answers, or (b) answers that are finite but depend on things "**outside**" the database
  - case (a): $\{(x, n) \mid \neg \textbf{Student}(x, n)\}$
  - case (b), subtle!: $\{(n) \mid \forall x\ \textbf{Student}(x, n)\}$
- These "bad" queries are called **domain-dependent** queries: their answers depend on the underlying domain of the database, rather than what is actually in the database (its "active domain")
- Syntactic restriction to "safe" relational calculus queries ensures domain-independence

Domain versus active domain

- For any database, the tuples in the database are over some underlying **domain** of values (e.g., integers, strings, . . .).
- The **active domain** of the database is the set of all values that are actually found in the database.
- E.g., if the database has a single relation R with three tuples $(1, 2), (2, 3), (2, 4)$, then the active domain is $\{1, 2, 3, 4\}$. The domain might be, e.g., all the natural numbers.
- The user may know what the active domain is, but not the domain. (e.g., 32-bit integers versus 64-bit integers versus . . .)
- Can compute the active domain with a database query! e.g., $\pi_1(R) \cup \pi_2(R)$.

FO as a query language

If $\varphi(x_1, \ldots, x_n)$ is a first-order formula with free variables $x_1, \ldots, x_n$ then we can think of φ itself as a query, shorthand for the relational calculus query

$$\{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\}$$

In this sense, the relational calculus and first-order logic are really the same query language.

RA $\subseteq$ domain-independent FO

Theorem

Every relational algebra query can be rewritten as an equivalent domain-independent FO query.

Proof.

(Sketch.) If $E_1 \equiv \{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\}$ and $E_2 \equiv \{(y_1, \ldots, y_m) \mid \psi(y_1, \ldots, y_m)\}$, then, e.g.,

- $\pi_{1,3,2}(E_1) \equiv \{(x_1, x_3, x_2) \mid \exists x_4 \cdots \exists x_n \; \varphi(x_1, \ldots, x_n)\}$
- $\sigma_{2=3}(E_1) \equiv \{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n) \wedge x_2 = x_3\}$
- $E_1 \times E_2 \equiv$
  $\{(x_1, \ldots, x_n, y_1, \ldots, y_m) \mid \varphi(x_1, \ldots, x_n) \wedge \psi(y_1, \ldots, y_m)\}$

$\square$

Domain-independent FO $\subseteq$ RA

Theorem

Every domain-independent FO query can be rewritten as an equivalent relational algebra query.

Proof.

Omitted (but surprisingly straightforward!) $\square$

Equivalence of relational algebra queries

Fundamental problem relevant to query optimization: given relational algebra queries Q, Q', are Q and Q' **equivalent**? That is, do their answers agree on all databases?

Theorem

Equivalence of relational algebra queries is undecidable.

Proof.

Follows from a well-known result in mathematical logic, **Trakhtenbrot's Theorem**, which says that **validity** of first-order sentence on "finite structures" (i.e., databases) is undecidable. A first-order sentence - a formula with no free variables - φ is **valid** if it's true for any database. $\square$

Summary

- Take-home message: SQL is just first-order logic in disguise
- Although, we ignored features of "real" SQL like aggregation, user-defined functions, ...
- Next time: some beautiful results about fragments of SQL where equivalence **is** decidable.