CS2Bh: Current Technologies

Introduction to XML and Relational Databases

Spring 2005

Relational Algebra

Relational Algebra

Relational algebra is a set of 6 operators that act on tables to produce tables. Just as we operate on numbers with arithmetic, we operate on tables with relational algebra

Key to understanding SQL and query processing, optimization

- ✓ SQL is, roughly speaking, a generalization of relational algebra
- ✓ Internal languages: an SQL query is rewritten as a relational-algebra expression, which can in turn be rewritten into a more efficient form and evaluated using a bunch of well-developed algorithms

Queries and query languages

- ✓ **Query**: a question about the data in a database.

A statement requesting the retrieval of information from a database.

Example: find the names of students who are taking CS2.

- ✓ Query language: language in which queries are expressed.
- ✓ Query languages **<>** programming languages!
  - QLs are not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

Preliminaries

- ✓ Review: a **relational database** is a collection of tables.
- ✓ A query is applied to relation instances, and the result of a query is also a relation instance.
  - Schemas of input relations for a query are fixed (the query will run regardless of instances!)
  - The schema for the result of a given query is also fixed!

Determined by the query.

Example schemas:

Students (<u>sid: string</u>, sname: string, gpa: real)

Courses (<u>cid: string</u>, cname: string, credit: integer, teacher: string)

Enroll (<u>sid: string, cid: string</u>, grade: string)

Example instances

S:

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0
004	john	3.0

E:

sid	cid	grade
001	166	B
002	CS2	C
003	166	A
003	CS2	A

C:

cid	cname	credits	instructor
166	math	3	poe
CS2	db	4	fan

Question: both John and Joe have a GPA of 3.0. Is S legal?

Relational Algebra

A set of operations (functions), each of which takes a relation (or relations) as input and produces a *relation* as output.

Basic operations: using these we can build up sophisticated database queries.

- ✓ Projection
- ✓ Selection
- ✓ Union
- ✓ Difference
- ✓ Product
- ✓ Renaming

Additional operations: intersection, join, division.

Projection

Given a list of column names *A* and a relation *R*, $\pi_A(R)$ extracts the columns in A from the relation.

Example: given S

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0

$\pi_{\text{sid, gpa}}$ (S) is

sid	gpa
001	3.0
002	2.8
003	4.0

Question:

- What is the schema of the result? Recall S has schema
  Students (sid: string, sname: string, gpa: real)
- What is the query (in English)?

Projection - continued

Suppose the result of $\pi_A(R)$ has duplicate values.

Example: given S1

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0
004	john	3.0

$\pi_{\textbf{gpa}}$ (S1) is

gpa
3.0
2.8
4.0

- ✓ In relational algebra, the answer is always a ***set*** (has to eliminate duplicates).
- ✓ However, SQL and some other languages return, by default, a bag (don't eliminate duplicates).

Selection

Given a condition *C* and a relation *R*, $\sigma_C(R)$ extracts those rows from the relation *R* that satisfy *C*.

Example: given S

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0
004	john	3.0

$\sigma_{gpa >= 3.0}$ (S) is

sid	sname	gpa
001	joe	3.0
003	grace	4.0
004	john	3.0

Question:

- What is the schema of the result? Recall S has schema *Students (<u>sid: string</u>, sname: string, gpa: real)*
- What is the query to find (in English)?

What can go into a condition?

Condition *C* in σ_C *(R)* is built up from

- Boolean operations on the filed names: <. <=, =, ≠, >=, >.
  Example: gpa >= 3.0, sname = "john".
- Predicates constructed from these using ∧ (and), ∨ (or), ¬ (not).

Question: what is the result of $\sigma_{gpa >= 3.0 \wedge sname = \text{"john"}}$ (S)?

given S1

sid	sname	gpa
001	john	2.0
002	mary	2.8
003	grace	4.0
004	john	3.0

$\sigma_{gpa >= 3.0 \wedge sname = \text{"john"}}$ (S1) is

sid	sname	gpa
004	john	3.0

Set operations

- ✓ Set operations: S1 ∪ S2, S1 – S2, S1 ∩ S2.
  What are these?
- ✓ **Condition**: all these operations must be **union-compatible**:
  - Same number of fields (same arity)
  - ‘Corresponding’ fields have the same domain (same type).

  Union-compatibility is a database terminology.
  What is the corresponding programming language terminology?
  Having the same type!
  Question: recall S and C given above. Can we write S ∪ C? Why?
- ✓ What is the schema of the result of a set operation?

Set operations – union

Given S1:

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0
004	john	3.0

S2:

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0
005	peter	4.0

S1 ∪ S2 is:

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0
004	john	3.0
005	peter	4.0

Set operations – set difference

Given S1:

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0
004	john	3.0

S2:

sid	sname	Gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0
005	peter	4.0

S1 – S2 is:

sid	sname	gpa
004	john	3.0

Question: what is S2 – S1?

Set operations – intersection

Given S1:

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0
004	john	3.0

S2:

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0
005	peter	4.0

S1 ∩ S2 is:

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0

Set operations – intersection

In relational algebra, basic set operations are union and set difference only. It turns out that we can implement the other set operations using those basic operations. For example, for any relations (sets) S1 and S2, we can already express S1 ∩ S2.

How?

$S1 \cap S2 = S1 - (S1 - S2)$

However, we have to be careful. Although it is mathematically nice to have fewer operators, operations like set differences may be less efficient than intersection.

Product

Product $R \times S$ connects two relations *R* and *S* that are not necessarily union compatible.

Example: given S:

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0

C:

cid	cname	credits	instructor
166	math	3	poe
CS2	db	4	fan

S × C is:

sid	sname	gpa	cid	cname	credits	instructor
001	joe	3.0	166	math	3	poe
002	mary	2.8	166	math	3	poe
003	grace	4.0	166	math	3	poe
001	joe	3.0	CS2	db	4	fan
002	mary	2.8	CS2	db	4	fan
003	grace	4.0	CS2	db	4	fan

Cartesian product

- ✓ Each row of S is paired with each row of R.
- ✓ Schema of the result has one field per field of S and R.

  Example: the schema of S × C is

  (sid: string, sname: string, gpa: real,

  cid: string, cname: string, credits: integer,

  instructor: string)

  Question: what is the primary key of R × S in general?
- ✓ Cardinality. Suppose that S has n rows and R has m rows. What is the cardinality of R × S?

Product – continued

What happens when we form a product of two relations with columns having the same name?

Example: given S

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0

E:

sid	cid	grade
001	166	B
002	CS2	C

Details may vary among systems/formalisms but a common answer is to suffix the attribute names with 1 and 2.

Example: S × E is

sid:1	sname	gpa	sid: 2	cid	grade
001	joe	3.0	001	166	B
002	mary	2.8	001	166	B
003	grace	4.0	001	166	B
001	joe	3.0	002	CS2	C
002	mary	2.8	002	CS2	C
003	grace	4.0	002	CS2	C

Product – continued

Products are hardly used alone; they are typically used in conjunction with a selection.

Example: $\sigma_{\textbf{sid:1 = sid: 2} \wedge \textbf{cid = CS2}}$ (S × E)

sid:1	sname	gpa	sid: 2	cid	grade
002	mary	2.8	002	CS2	C

Question: what does this query do (in English)?

Suppose we want to find the names and grades of students who are taking CS2. How to write the query?

$\pi_{\textbf{sname, grades}}(\sigma_{\textbf{sid:1 = sid: 2} \wedge \textbf{cid = CS2}}$ (S × E))

sname	grade
mary	C

Question: why is this possible?

Joins – conditional join

The combination of a selection and a join is so common that we give it a special symbol (and name).

$\mathbf{R} \bowtie_{C} \boldsymbol{S}$ is defined to be σ_C (R × S)

Example: S $\bowtie_{\textbf{sid:1 = sid: 2}}$ E is

sid:1	sname	gpa	sid: 2	cid	grade
001	joe	3.0	001	166	B
002	mary	2.8	002	CS2	C

Questions:

- ✓ What is the result schema?
- ✓ Conditional join is in general more efficient than cross product. Why?

The condition C in a conditional join is usually an equality or conjunction of equalities.

natural join

R $\bowtie$ *S* : a 'special case' of conditional join: equality on common fields of R and S

- ✓ Equality condition only
- ✓ On all common fields
- ✓ Leave only one copy of these fields in the resulting relation.

R

A	B	C
1	2	a
1	2	b
1	3	c
2	1	g

S

A	B	D
1	2	d
1	2	e
1	4	d

R $\bowtie$ S

A	B	C	D
1	2	a	d
1	2	a	e
1	2	b	d
1	2	b	e

Question: what if R and S have no fields in common? -- Cartesian product!

Example – natural join

Example: given S:

sid	sname	gpa
001	joe	3.0
002	mary	2.8
003	grace	4.0

E:

sid	cid	grade
001	166	B
002	CS2	C

Question: what is S $\bowtie$ E?

Answer:

sid	sname	gpa	cid	grade
001	joe	3.0	166	B
002	mary	2.8	CS2	C

Question: what is S $\bowtie$ C where C is an instance of *Courses (cid: string, cname: string, credits: integer, instructor: string)* ?

Example – queries (1)

Students (sid: string, sname: string, gpa: real)
Courses (cid: string, cname: string, credits: integer, instructor: string)
Enroll (sid: string, cid: string, grade: string)

1. Find the names of Students.
   $\pi_{\textbf{sname}}$ (Students)
2. Find the courses taught by Fan.
   $\sigma_{\textbf{instructor = “fan”}}$ (Courses)
3. Find the titles of courses taught by Fan.
   $\pi_{\textbf{cname}}$ ($\sigma_{\textbf{instructor = “fan”}}$ (Courses))

- ✓ These queries involve a single relation
- ✓ The result of a query is also a relation and therefore, can be used as input of another query

Example – queries (2)

Students (sid: string, sname: string, gpa: real)
Courses (cid: string, cname: string, credits: integer, instructor: string)
Enroll (sid: string, cid: string, grade: string)

Find the names of Students who are taking CS2.

- ✓ Two relations. Therefore, use (natural) join or product
- ✓ Fields: projection
- ✓ Condition: selection

Solutions:

1. $\pi_{\textbf{sname}}$ ($\sigma_{\textbf{cid = “CS2”}}$ (Students $\bowtie$ Enroll))

2. $\pi_{\textbf{sname}}$ (Students $\bowtie$ $\sigma_{\textbf{cid = “CS2”}}$ (Enroll))

renaming – another operator (1)

renaming operator:

- ρ (R, E): define temporary relation R to hold the result of query E
- ρ (R (A $\rightarrow$ B), E): in addition, rename A attribute in E as B.

Example. Find the names of Students who are taking CS2.

π_{sname} (Students $\bowtie$ $\sigma_{cid = \text{"CS2"}}$ (Enroll))

is equivalent to:

✓ ρ (temp1, $\sigma_{cid = \text{"CS2"}}$ (Enroll))

✓ ρ (temp2, Students $\bowtie$ temp1)

✓ π_{sname} (temp2)

renaming – another operator (2)

Students (<u>sid: string</u>, sname: string, gpa: real)

Courses (<u>cid: string</u>, cname: string, credits: integer, instructor: string)

Enroll (<u>sid: string, cid: string</u>, grade: string)

Consider Students × Enroll. Note we have name conflict.

(<u>sid1 string</u>, sname: string, gpa: real, <u>sid2 string, cid: string</u>, grade: string)

To eliminate name conflict, we can do the following:

✓ ρ (temp1 (sid $\rightarrow$ sid1), Students)

✓ ρ (temp2 (sid $\rightarrow$ sid2), Enroll)

✓ temp1 × temp2

Example – queries (3)

Students (<u>sid: string</u>, sname: string, gpa: real)

Courses (<u>cid: string</u>, cname: string, credits: integer, instructor: string)

Enroll (<u>sid: string, cid: string</u>, grade: string)

Find the names of Students who are taking a course taught by Fan.

✓ Information about instructor only available in Courses; so we need an extra join. Solutions:

1. π_{sname} (Students $\bowtie$ Enroll $\bowtie$ $\sigma_{\text{instructor = "fan"}}$ (Courses))
   Remark. Associative!
   $R \bowtie S \bowtie T = (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
2. A more efficient solution:

π_{sname} (Students $\bowtie$ π_{sid} (Enroll $\bowtie$ π_{cid} ($\sigma_{\text{instructor = "fan"}}$ (Courses))))

Remark. A query optimizer can find this given the first solution.

Exercises – queries

Students (<u>sid: string</u>, sname: string, gpa: real)

Courses (<u>cid: string</u>, cname: string, credits: integer, instructor: string)

Enroll (<u>sid: string, cid: string</u>, grade: string)

1. Find the GPAs of Grace.
2. Find the ids of the courses being taken by Grace.
3. Find the instructors of the courses being taken by Grace.
4. Find the names of students who are taking at least one course.

Solutions:

1. π_{gpa} ($\sigma_{\text{sname = "grace"}}$ Students **)**
2. π_{cid} (($\sigma_{\text{sname = "grace"}}$ Students **)** $\bowtie$ Enroll **)**
3. $\pi_{\text{instructor}}$ (Courses $\bowtie$ Enroll $\bowtie$ ($\sigma_{\text{sname = "grace"}}$ Students **))**
4. π_{sname} (Students $\bowtie$ Enroll **)**

Example – queries (4)

Students (<u>sid: string</u>, sname: string, gpa: real)

Courses (<u>cid: string</u>, cname: string, credits: integer, instructor: string)

Enroll (<u>sid: string, cid: string</u>, grade: string)

Find the names of Students who are taking a course taught by Poe or Fan.

- ✓ $\pi_{\textbf{sname}}$ (Students $\bowtie$ Enroll $\bowtie$ ($\sigma_{\textbf{instructor = “poe”} \vee \textbf{instructor = “fan”}}$ (Courses)))
- ✓ How to do this using union?
  - ρ (temp, $\sigma_{\textbf{instructor = “poe”}}$ (Courses) $\cup$ $\sigma_{\textbf{instructor = “fan”}}$ (Courses))
  - $\pi_{\textbf{sname}}$ (Students $\bowtie$ Enroll $\bowtie$ temp)

Example – queries (5)

Students (<u>sid: string</u>, sname: string, gpa: real)

Courses (<u>cid: string</u>, cname: string, credits: integer, instructor: string)

Enroll (<u>sid: string, cid: string</u>, grade: string)

Find the ids of Students who are taking a course taught by Poe and a course taught by Fan.

- ✓ $\pi_{\textbf{sid}}$ (Enroll $\bowtie$ $\sigma_{\textbf{instructor = “poe”}}$ (Courses)) $\cap$ $\pi_{\textbf{sid}}$ (Enroll $\bowtie$ $\sigma_{\textbf{instructor = “fan”}}$ (Courses))
- ✓ The following is incorrect!

$\pi_{\textbf{sid}}$ (Enroll $\bowtie$ ($\sigma_{\textbf{instructor = “poe”} \wedge \textbf{instructor = “fan”}}$ (Courses)))

Question: Find the names of Students who are taking a course taught by Poe and a course taught by Fan.

$\pi_{\textbf{sname}}$ (Students $\bowtie$ $\pi_{\textbf{sid}}$ (Enroll $\bowtie$ $\sigma_{\textbf{instructor = “poe”}}$ (Courses)) $\cap$ $\pi_{\textbf{sid}}$ ((Enroll $\bowtie$ $\sigma_{\textbf{instructor = “fan”}}$ (Courses))))

Example – queries (6)

Students (<u>sid: string</u>, sname: string, gpa: real)

Courses (<u>cid: string</u>, cname: string, credits: integer, instructor: string)

Enroll (<u>sid: string, cid: string</u>, grade: string)

Find the sids of Students who are not taking CS2.

Solution:

$\pi_{\textbf{sid}}$ (Students) $-$ $\pi_{\textbf{sid}}$ ($\sigma_{\textbf{cid = "CS2"}}$ (Enroll)

Exercise: Find the names of Students who are not taking CS2.

Solution:

- ✓ ρ (temp, $\pi_{\textbf{sid}}$ (Students) $-$ $\pi_{\textbf{sid}}$ ($\sigma_{\textbf{cid = "CS2"}}$ (Enroll))
- ✓ $\pi_{\textbf{sname}}$ (temp $\bowtie$ Students)

Division -- introduction

Students (<u>sid: string</u>, sname: string, gpa: real)

Courses (<u>cid: string</u>, cname: string, credits: integer, instructor: string)

Enroll (<u>sid: string, cid: string</u>, grade: string)

Find the sids of students who are taking all courses.

1. Build a relation with all possible pairs of sids and cids.

   Let Allpairs be $\pi_{\textbf{sid}}$ (Enroll) $\times$ $\pi_{\textbf{cid}}$ (Courses)
2. Find the set of (sid, cid) pairs for which sid is not taking cid.

   Let temp1 be Allpairs $-$ $\pi_{\textbf{sid, cid}}$ (Enroll)
3. Find the set of sids of students who are not taking some course.

   Let temp2 be $\pi_{\textbf{sid}}$ (temp1)
4. Find the set of sids of students who are taking all courses.

   $\pi_{\textbf{sid}}$ (Enroll) $-$ temp2

Division – another operator

Find the sids of students who are taking all courses.

$\pi_{sid,\ cid}$(Enroll) / π_{cid}(Courses)

In general, A / B.

- The schema of B must be a proper subset of the schema of A, i.e., B $\subset$A (check: {cid} $\subset$ {cid, sid})
- The schema of the result is A – B, i.e., the set difference of the schema of A and the schema of B (check: result is {sid})
- For every tuple *t* in the result and every tuple *s* in B, *t s* (t appended onto s) is in the first relation A.

Division – examples of A/B

sno	pno
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

A

pno
P2

B1

sno
S1
S2
S3
S4

A/B1

pno
P2
P4

B2

sno
S1
S4

A/B2

pno
P1
P2
P4

B3

sno
S1

A/B3

Example – queries (7)

Students (sid: string, sname: string, gpa: real)

Courses (cid: string, cname: string, credits: integer, instructor: string)

Enroll (sid: string, cid: string, grade: string)

1. Find the names of students who are taking all courses.

✓ ρ (tempid, $\pi_{\textbf{sid, cid}}$ (Enroll) / $\pi_{\textbf{cid}}$ (Courses))

✓ $\pi_{\textbf{sname}}$ (Students $\bowtie$ tempid)

1. Find the names of students who are taking all courses taught by Fan.

✓ ρ (tempid, $\pi_{\textbf{sid, cid}}$ (Enroll) / $\pi_{\textbf{cid}}$ ($\sigma_{\textbf{instructor = “fan”}}$ Courses)))

✓ π_{sname} (Students $\bowtie$ tempid)

What we can’t compute with relational algebra?

✓ Arithmetic expressions, e.g., 4 + 3.

✓ Aggregate operations, e.g., “the number of students who are taking CS2”, or “the average GPA of students”.

In SQL, these are possible – SQL has numerous extensions to relational algebra.

✓ Recursive queries, e.g., given a relation *Parent(parent, child)*, compute the *ancestor* relation. This appears to call for an arbitrary number of joins.

These are not possible in SQL either.

✓ Complex structures, e.g., lists, arrays, bags, or nested relations.

SQL can’t handle complex structures either, but they are possible in object-oriented data models and query languages.

Summary

- ✓ What are query languages?
- ✓ Relational algebra. A set of operations (functions), each of which takes a relation (or relations) as input and produces a *relation* as output.

  Basic operations: using these we build up queries.
  - Projection
  - Selection
  - Union
  - Difference
  - Product
  - Renaming.
  - Additional operations: intersection, join, Very useful.
- ✓ What we cannot do with relational algebra?
- ✓ **SKILL: “programming” in relational algebra!**