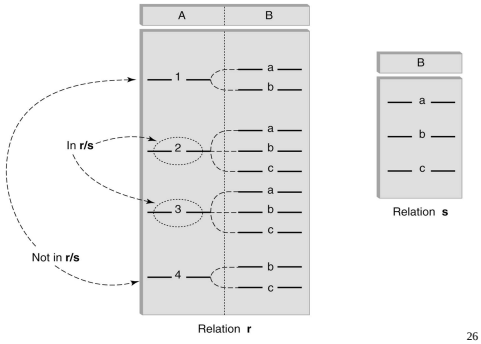
Division

- Goal: Produce the tuples in one relation, r, that match *all* tuples in another relation, s
  - *r* ($A_1, \ldots A_n, B_1, \ldots B_m$)
  - *s* ($B_1 \ldots B_m$)
  - *r/s*, with attributes $A_1, \ldots A_n$, is the set of all tuples <*a*> such that for every tuple <*b*> in *s*, <*a,b*> is in *r*
- Can be expressed in terms of projection, set difference, and cross-product

Division (con't)



Division - Example

- List the Ids of students who have passed *all* courses that were taught in spring 2000
- *Numerator*:
  - *StudId* and *CrsCode* for every course passed by every student:

    $\pi_{StudId,\ CrsCode}\ (\sigma_{Grade \neq 'F'}\ (\text{Transcript})\ )$
- *Denominator*:
  - *CrsCode* of all courses taught in spring 2000

    $\pi_{CrsCode}\ (\sigma_{Semester='S2000'}\ (\text{Teaching})\ )$
- Result is *numerator/denominator*

Schema for Student Registration System

Student (*Id, Name, Addr, Status*)
Professor (*Id, Name, DeptId*)
Course (*DeptId, CrsCode, CrsName, Descr*)
Transcript (*StudId, CrsCode, Semester, Grade*)
Teaching (*ProfId, CrsCode, Semester*)
Department (*DeptId, Name*)

Query Sublanguage of SQL

```
SELECT C.CrsName
FROM Course C
WHERE C.DeptId = 'CS'
```

- *Tuple variable* C ranges over rows of Course.
- Evaluation strategy:
  - FROM clause produces Cartesian product of listed tables
  - WHERE clause assigns rows to C in sequence and produces table containing only rows satisfying condition
  - SELECT clause retains listed columns
- Equivalent to: $\pi_{CrsName}\sigma_{DeptId='CS'}(\text{Course})$

Join Queries

```
SELECT C.CrsName
FROM Course C, Teaching T
WHERE C.CrsCode=T.CrsCode AND T.Sem='S2000'
```

- List CS courses taught in S2000
- Tuple variables clarify meaning.
- Join condition "C.*CrsCode*=T.*CrsCode*"
  - eliminates garbage
- Selection condition " T.*Sem*='S2000' "
  - eliminates irrelevant rows
- Equivalent (using natural join) to:

$\pi_{CrsName}(\text{Course} \bowtie \sigma_{Sem='S2000'}(\text{Teaching})\ )$

$\pi_{CrsName}\ (\sigma_{Sem='S2000'}\ (\text{Course} \bowtie \text{Teaching})\ )$

Correspondence Between SQL and Relational Algebra

```
SELECT C.CrsName
FROM Course C, Teaching T
WHERE C.CrsCode=T.CrsCode AND T.Sem='S2000'
```

Also equivalent to:

$\pi_{CrsName}\ \sigma_{C_CrsCode=T_CrsCode\ AND\ Sem='S2000'}$ (Course [*C_CrsCode, DeptId, CrsName, Desc*] × Teaching [*ProfId, T_CrsCode, Sem*])

This is the simple evaluation algorithm for SELECT. Relational algebra expressions are procedural. Which of the two equivalent expressions is more easily evaluated?

31

Self-join Queries

Find Ids of all professors who taught at least two courses in the same semester:

```
SELECT T1.ProfId
FROM Teaching T1, Teaching T2
WHERE T1.ProfId = T2.ProfId
   AND T1.Semester = T2.Semester
   AND T1.CrsCode <> T2.CrsCode
```

Tuple variables essential in this query!

Equivalent to:

$\pi_{ProfId}(\sigma_{T1.CrsCode \neq T2.CrsCode}$(Teaching[*ProfId, T1.CrsCode, Sem*] ⋈ Teaching[*ProfId, T2.CrsCode, Sem*]))

Duplicates

- Duplicate rows not allowed in a relation
- However, duplicate elimination from query result is costly and not automatically done; it must be explicitly requested:

```
SELECT DISTINCT .....
FROM .....
```

33

Use of Expressions

Equality and comparison operators apply to strings (based on lexical ordering)

```
WHERE S.Name < 'P'
```

Concatenate operator applies to strings

```
WHERE S.Name || '--' || S.Address = ....
```

Expressions can also be used in SELECT clause:

```
SELECT S.Name || '--' || S.Address AS NmAdd
FROM Student S
```

34

Set Operators

- SQL provides UNION, EXCEPT (set difference), and INTERSECT for union compatible tables
- Example: Find all professors in the CS Department and all professors that have taught CS courses

```
(SELECT P.Name
 FROM Professor P, Teaching T
 WHERE P.Id=T.ProfId AND T.CrsCode LIKE 'CS%')
UNION
(SELECT P.Name
 FROM Professor P
 WHERE P.DeptId = 'CS')
```

35

Nested Queries

List all courses that were not taught in S2000

```
SELECT C.CrsName
FROM Course C
WHERE C.CrsCode NOT IN
    (SELECT T.CrsCode     --subquery
     FROM Teaching T
     WHERE T.Sem = 'S2000')
```

Evaluation strategy: subquery evaluated once to produces set of courses taught in S2000. Each row (as C) tested against this set.

36

Correlated Nested Queries

Output a row <*prof, dept*> if *prof* has taught a course in *dept*.

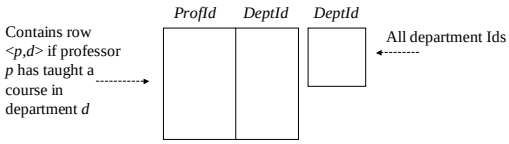
```
SELECT  P.Name, D.Name                  --outer query
  FROM Professor P, Department D
  WHERE  P.Id  IN
          -- set of  all ProfId's who have taught a course in D.DeptId
          (SELECT T.ProfId                    --subquery
           FROM Teaching T, Course C
           WHERE T.CrsCode=C.CrsCode  AND
                    C.DeptId=D.DeptId       --correlation
          )
```


Correlated Nested Queries (con't)

- Tuple variables T and C are *local* to subquery
- Tuple variables P and D are *global* to subquery
- *Correlation*: subquery uses a global variable, D
- The value of D.*DeptId* parameterizes an evaluation of the subquery
- Subquery must (at least) be re-evaluated for each distinct value of D.*DeptId*
- *Correlated queries can be expensive to evaluate*

Division

- *Query type*: Find the subset of items in one set that are related to *all* items in another set
- *Example*: Find professors who have taught courses in *all* departments
  - Why does this involve division?



$\pi_{ProfId,DeptId}(Professor) \ / \ \pi_{DeptId}(Department)$

Division

- *Strategy for implementing division in SQL*:
  - Find set, A, of all departments in which a particular professor, *p*, has taught a course
  - Find set, B, of all departments
  - Output *p* if A $\supseteq$ B, or, equivalently, if B–A is empty

Division – SQL Solution

```
SELECT P.Id
FROM Professor P
WHERE NOT EXISTS
   (SELECT D.DeptId          -- set B of all dept Ids
    FROM Department D
       EXCEPT
    SELECT C.DeptId          -- set A of dept Ids of depts in
                             -- which P has taught a course
    FROM Teaching T, Course C
    WHERE T.ProfId=P.Id      -- global variable
          AND T.CrsCode=C.CrsCode)
```