CSE 232

Principles of Database Systems

1

Database

- Collection of highly structured data
- Along with a set of access and control mechanisms

Databases vs. Information Retrieval vs. Expert Systems

- Airline reservations: database
- Computerized library: information retrieval
- Medication advisor: expert system

2

Top Level Goals of a Database System

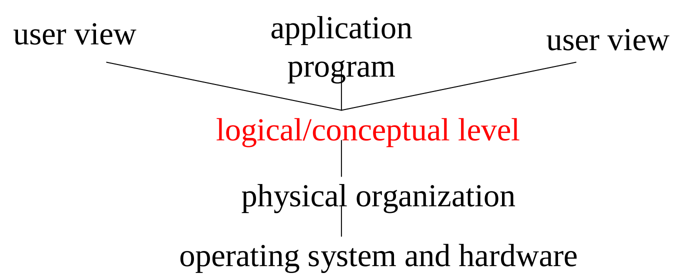
- Provide users with a meaning-based view of data
  - shield from irrelevant detail → abstract view
- Support operations on data
  - queries, updates
- Provide data control
  - integrity, protection
  - concurrency, recovery

3

Data Independence

- user view
- implementation view
  - separate!

Basic DB System Architecture



4

Database System

- Tailored to specific application

Database Management System

- Generalized DB system
  - used in variety of application environments
  - common approach to
    - data organization
    - data storage
    - data access
    - data control
  - e.g. Ingres, Oracle, O2, etc.

5

The Logical Level

- Interface provides a data model
- e.g. relational. hierarchical, object-oriented, etc.
- Specifying data: data definition language (DDL)
- Manipulating data: data manipulation language (DML)

6

Entity-Relationship Model (ER)

- entities
- attributes
- relationships

7

Relational Model

- tables
- attributes

customer name address

account number balance

cust-acct name number

8

Database Schema

- Description of permanent aspects of DB structure

Database Instance

- Content of DB at current time

Integrity Constraints

- Additional characteristics of valid data
  - "Each student is a person"
  - "Each course has only one instructor"
    - course $\rightarrow$ instructor
    - functional dependency

9

Main Database Issues

- The core issues
  - data models and query languages
  - query processing
    - optimization
    - file organization and access methods (B-trees, …)
  - constraints
  - schema design
  - concurrency control

10

Beyond the Core

- New models
  - object-oriented
  - semantic
  - deductive
- "Smarter" databases
  - deduction
  - incomplete information
- Temporal databases
- Views
  - materialized/virtual
  - maintenance
- Distributed databases

11

Overlap with other Areas

- Operating systems
  - file management, concurrency
- Data structures
  - access methods
- Hardware
  - DB machines
- AI
  - knowledge bases, semantic models
- Logic programming
  - deductive databases
- Programming Languages
  _ database programming languages, object-oriented databases
- Theory
  - finite model theory: theory of query languages
  - descriptive complexity: complexity of queries

12

Recent Developments in DB

- The WEB
  - heterogeneous, semistructured data
  - data integration
  - distributed query evaluation "agents"
- Data warehouses
  - OLAP
  - data mining
- Multimedia databases
  - video/image retrieval
  - mix of DB and image processing techniques

13

Recent Developments in DB (2)

- Electronic commerce
  - database support
  - active databases
- Geographic information systems
  - efficient representation
  - efficient querying of spatial data

14

Databases at UCSD

- Yannis Papakonstantinou
- Victor Vianu

- Intersections with other groups
  - storage
  - multimedia
  - AI
  - IR
  - Supercomputer center

What kind of research?

- The whole spectrum

- implementation/systems

Indexing
Query processing
Interfaces
Data on the Web
...

- theory

Logic and complexity
Theory of query languages
Electronic Commerce
Spatial databases
Semi-structured data, XML

The Relational Model

- Relational DB schema
  - set of table names
  - each table has set of attributes
- Specification of a table
  - <name-of-table> : <list of attributes>
  - movie : title, director, actor
  - schedule : theater, title

movie	title	director	actor

schedule	theater	title

  - no repeating relation names
  - no repeating attributes in one table

17

Queries

- Consider:

movie	title	director	actor
	wild	lynch	winger
	sky	berto	winger
	reds	beatty	beatty
	tango	berto	brando
	tango	berto	sneyder

schedule	theater	title
	odeon	wild
	forum	reds
	forum	sky

18

Example Queries

- find titles of current movies:

answer	title
	wild
	reds
	sky

- find titles of all movies by Berto:

answer	title
	sky
	tango

- find titles and directors of currently playing movies:

answer	title	director
	wild	lynch
	reds	beatty
	sky	berto

19

Result of Query

- Table constructed from DB tables
- Query languages
  - procedural: "how"
    relational algebra
  - declarative: "what"
    relational calculus
- Commercial query languages: mix
  - w.g. SQL, QBE, etc.

20

SQL: A Commercial Relational DBMS Language

- "Structured Query Language"
- Creating a table
  - create table R(A int 1, B int 4, C char 16)
- Inserting tables
  - insert into R values (0, 1, "John")
- Removing a table
  - drop R
- Command buffer editor
  - \e
- Quit
  - \quit

21

SQL Queries: The Basic From

- Basic form
- SELECT *a1*, …, *aN* FROM
- *R1*, …, *RM* WHERE *condition*
-
- WHERE clause is optional

- When more than one relation of the FROM has an attribute named *A,* we refer to a specific *A* attribute as *<RelationName>.A*

Find titles of currently playing movies

```
SELECT Title
FROM Schedule
```

Find the titles of all movies by "Berto"

```
SELECT Title
FROM Movie
WHERE Director="Berto"
```

Find the titles and the directors of all currently playing movies

```
SELECT Movie.Title, Director
FROM Movie, Schedule
WHERE Movie.Title=Schedule.Title
```

22

SQL Queries: Aliases

- Use the same relation more than once in the FROM clause
- Example: find actors who are also directors

```
SELECT  t.Actor
FROM  Movie  t  s
WHERE t.Actor = s.Director
```

23

SQL Queries: Nesting

- The WHERE clause can contain predicates of the form
  - *attr/value* IN <*SQL query*>
  - *attr/value* NOT IN <*SQL query*>
- The predicate is satisfied if the *attr* or *value* appears in the result of the nested <*SQL query*>
- Queries involving nesting but no negation can always be un-nested, unlike queries with nesting and negation

Typical use:"find objects that always satisfy property X", e.g., *find actors playing in every movie by "Berto":*

```
SELECT Actor FROM Movie
WHERE Actor NOT IN
        (SELECT m2.Actor
          FROM Movie m1 m2,
          WHERE m1.Director="Berto"
             AND m2.Actor NOT IN
                    (SELECT Actor
                      FROM Movie
                      WHERE Title=m1.Title))
```

The shaded query finds actors NOT playing in some movie by "Berto"

24

The top lines complement the shaded part

SQL:Union, Intersection, Difference

- *Union*
  - <*SQL query 1*> UNION <*SQL query 2*>
- *Intersection*
  - <*SQL query 1*> INTERSECT <*SQL query 2*>
- *Difference*
  - <*SQL query 1*> MINUS <*SQL query 2*>

Find all actors or directors

```
(SELECT Actor
 FROM Movie)
UNION
(SELECT Director
 FROM Movie)
```

Find all actors who are not directors

```
(SELECT Actor
 FROM Movie)
MINUS
(SELECT Director
 FROM Movie)
```

25

Nested Queries: Existential and Universal Quantification

- *A op* ANY <*nested query*> is satisfied if ***there is*** a value *X* in the result of the <*nested query*> and the condition *A op X* is satisfied
  - ANY aka SOME
- *A op* ALL <*nested query*> is satisfied if ***for every*** value *X* in the result of the <*nested query*> the condition *A op X* is satisfied

Find directors of currently playing movies

```
SELECT Director
FROM Movie
WHERE Title = ANY
   SELECT Title
   FROM Schedule
```

Find the employees with the highest salary

```
SELECT Name
FROM Employee
WHERE Salary >= ALL
   SELECT Salary
   FROM Employee
```

26

Nested Queries: Set Comparison

- <*nested query 1*> CONTAINS <*nested query 2*>

Find actors playing in every movie by "Berto"

```
SELECT m1.Actor
FROM Movie m1
WHERE
        (SELECT Title
         FROM Movie m2
         WHERE m1.Actor = m2.Actor)
        CONTAINS
        (SELECT Title
         FROM Movie
         WHERE Director = "Berto")
```


SQL Queries: Aggregation and Grouping

- There is no relational algebra equivalent for aggregation and grouping
- Aggregate functions: AVG, COUNT, MIN, MAX, SUM, ... (user defined functions)
- Group-by

Employee

Name	Dept	Salary
Joe	Toys	45
Nick	PCs	50
Jim	Toys	35
Jack	PCs	40

Find average salary of all employees

```
SELECT AvgSal=Avg(Salary)
FROM Employee
```

AvgSal
42.5

Find the average salary for each department

```
SELECT Dept, AvgSal=Avg(Salary)
FROM Employee
GROUP-BY Dept
```

Dept	AvgSal
Toys	40
PCs	45

SQL Grouping: Conditions that Apply on Groups

- *HAVING* clause

Find the average salary of for each department that has more than 1 employee

SELECT Dept, AvgSal= Avg(Salary)
FROM Employee
GROUP-BY Dept
HAVING COUNT(Name)>1

29

SQL: More Bells and Whistles ...

- Select all attributes using *
- Pattern matching conditions
  - <*attr*> LIKE <*pattern*>

Retrieve all movie attributes of currently playing movies

SELECT Movie.*
FROM Movie, Schedule
WHERE Movie.Title=Schedule.Title

Retrieve all movies where the title starts with "Ta"

SELECT *
FROM Movie
WHERE Title LIKE "Ta"

30

…and a Few "Dirty" Points

- *Duplicate elimination* must be explicitly requested
  - SELECT DISTINCT … FROM … WHERE …
- *Null values*
  - all comparisons involving NULL are *false* by definition
  - all aggregation operations, except *count*, ignore NULL values

SELECT Title
FROM Movie

Title
Tango
Tango
Tango

SELECT DISTINCT Title
FROM Movie

Title
Tango

Title	Director	Actor
Wild	Lynch	Winger
Sky	Berto	Winger
Reds	NULL	Beatty
Tango	Berto	Brando
Tango	Berto	Winger
Tango	Berto	NULL

31

SQL as a Data Manipulation Language: Insertions

- inserting tuples
  - INSERT INTO *R* VALUES (*v1*,…,*vk*);
- some values may be left NULL
- use results of queries for insertion
  - INSERT INTO *R* SELECT … FROM … WHERE

INSERT INTO Movie
VALUES ("Brave", "Gibson", "Gibson");

INSERT INTO Movie(Title,Director)
VALUES ("Brave", "Gibson");

INSERT INTO EuroMovie
SELECT * FROM Movie
WHERE Director = "Berto"

32

SQL as a Data Manipulation Language: Updates and Deletions

- ***Deletion*** basic form: delete every tuple that satisfies *<cond>*
  - DELETE FROM *R* WHERE *<cond>*
- ***Update*** basic form: update every tuple that satisfies *<cond>* in the way specified by the SET clause
  - UPDATE *R*
    SET *A1*=*<exp1>*, …,
    Ak=*<expk>*
    WHERE *<cond>*

Delete the movies that are not currently playing

```
DELETE FROM Movie
WHERE Title NOT IN SELECT Title
                   FROM Schedule
```

Change all "Berto" entries to "Bertoluci"

```
UPDATE Movie
SET Director="Bertoluci"
WHERE Director="Berto"
```

Increase all salaries in the Toys dept by 10%

```
UPDATE Employee
SET Salary = 1.1 * Salary
WHERE Dept = "Toys"
```

The "rich get richer" exercise:
Increase by 10% the salary of the employee with the highest salary

33

QBE

- Query-By-Example
  - provides a graphic interface for queries and updates
- Examples: movie database queries
  - query 1

schedule	theater	title
		P.

  - P. : "print value"
  - query 2

movie	title	director	actor
__	P.	Berto	

34

QBE (2)

– query 3

movie	title	director	actor
	_t	_d	

schedule	theater	title
		_t

answer	title	director
I.	_t	_d

- Note:
  - answer table explicitly specified
  - underscore _x means _x can take any value, like a variable
  - I. means insert

35

QBE (3)

- "Find all actors playing in every movie by Berto"
  - requires multi-stage query, creating intermediate answers
  - analog of nested queries in SQL
- I stage:

schedule	title	director	actor
			_a
	_t	Berto	
¬	_t		_a

temp	actor
I.	_a

- Semantics of ¬
  - for _t and _a fixed, satisfying positive part of pattern, there is no tuple occurring with _t and _a as in the negated tuple

36

QBE (4)

- II stage
  - (complement of temp computed in stage I)

movie	title	director	actor
			_a

temp	actor
¬	_a

answer	actor
I.	_a

37

Updates in QBE

- Deletions: similar to inserts
  - D.
  - “Delete all movies by Berto”:

movie	title	director	actor
D.		Berto	

  - “Delete all movies by directors who are also actors”:

movie	title	director	actor
D.		_d	
			_d

38

Updates in QBE (2)

- Updates: using key attributes
  - key: set of attributes which uniquely identify the tuple
  - keys are explicitly declared

  - "Sally gets a 5% salary raise"

employee	name	salary
U.	Sally	_x * 1.05
	Sally	_x

39

Updates in QBE (3)

- "All employees who make less than 2000 receive a 5% raise"

employee	name	salary
U.	_u	_x * 1.05
	_u	_x

Condition box

_x < 2000

- Note: QBE allows explicit specification of conditions using condition boxes

40

Relational Calculus

- Models data manipulation in declarative languages like SQL, QBE
  - idea: specify “what” not “how”
- General form
  - {t | property (t)}
- property (t) is described by a language based on predicate calculus (first-order logic)

41

Tuple Relational Calculus

- In the style of SQL: language talks about tuples
- What you can say:
  - refer to tuples: tuple variables t, s, …
  - a tuple t belongs to a relation R: R(t)
  - conditions on attributes of a tuple t and s:
    - $t(A) = (\neq)(\geq)$ constant
    - $t(A) = s(B)$
    - $t(A) \neq s(B)$
    - etc.
- Simple expressions above: Atoms

42

Tuple Relational Calculus (2)

- Combine properties using boolean operators
  - $\wedge, \vee, \neg$
  - (abbreviation: $p \to q \equiv \neg p \vee q$)
- Quantifiers
  - there exists t: $\exists t$: <att> $\varphi(t)$
  - for every t: $\forall t$: <att> $\varphi(t)$
    - <att> specifies attributes of t
    - $\varphi(t)$ is a property of t
  - abbreviations
    - $\exists t \in R\ \varphi(t)$ means $\exists t$: att(R) $[R(t) \wedge \varphi(t)]$
    - $\forall t \in R\ \varphi(t)$ means $\forall t$: att(R) $[R(t) \to \varphi(t)]$

43

Rules for correct queries

- Compatible attributes
  - if R: AB, S: BC then $R(t) \wedge S(t)$ does not use t consistently
- Correct quantification
  - scope of quantifier:
    - scope of $\exists t\ \varphi(t)$ is φ
    - scope of $\forall t\ \varphi(t)$ is φ
  - free variable:
    - not in scope of any quantifier
  - Rule: in quantification $\exists t\ \varphi(t)$, $\forall t\ \varphi(t)$, t must be free in φ

44

Semantics of Tuple Calculus

- Active domain:
  - set of values in database, or mentioned in query
  - tuple variables range over active domain
- Note: a query without free variables always evaluates to true or false
- Example: “Sky is by Berto” is expressed without free variables:
  - $\exists m \in$ movie [m(title) = “Sky” $\wedge$ m(director) = “Berto”]
  - this statement is true or false

45

Tuple Calculus Query

- {t: <att> | φ(t)}
  - where φ is a formula with only one free variable t
  - produces as answer a table with attributes <att> consisting of all tuples v in active domain with make φ(v) true
  - note: φ(v) has no free variables so it evaluates to true or false

46

Examples (Movie Database Queries)

- Query 1
  - $\{t: title \mid \exists s \in schedule\ [s(title) = t(title)]\}$
- Query 2
  - $\{t: title \mid \exists m \in movie\ [m(director) = \text{“Berto”} \wedge t(title) = m(title)]\}$
- Query 3
  - $\{t: title, director \mid \exists s \in schedule\ \exists m \in movie\ [s(title) = m(title) \wedge t(title) = m(title) \wedge t(director) = m(director)]\}$

47

Tuple Calculus and SQL

- Example: “Find directors of all currently playing movies”:
  - SQL:
    - SELECT m.director
      FROM movie m, schedule s
      WHERE m.title = s.title
  - tuple calculus:
    - $\{t: director \mid \exists m \in movie\ \exists s \in schedule\ [t(director) = m(director) \wedge m(title) = s(title)]\}$

48

Basic SQL Query

- SELECT $A_1, \ldots, A_n$
  FROM $R_1, \ldots, R_k$
  WHERE $cond(R_1, \ldots, R_k)$

Tuple Calculus

- $\{t: A_1, \ldots, A_n \mid \exists r_1 \in R_1 \ldots \exists r_k \in R_k [\wedge_j\ t(A_j) = r_{ij}(A_j) \wedge cond(r_1, \ldots, r_k)]\}$
- Note: basic SQL query uses only $\exists$;
- no explicit construct for $\forall$

49

Using Tuple Calculus to Formulate SQL Queries

- Example: "Find actors playing in every movie by Berto"
- Tuple calculus
  - $\{a: actor \mid \forall m \in movie\ [m(director) = \text{“Berto”} \rightarrow \exists t \in movie\ [m(title) = t(title) \wedge t(act) = a(actor)]]\}$
- Eliminate $\forall$:
  - $\{a: actor \mid \neg\exists m \in movie\ [m(dir) = \text{“Berto”} \wedge \neg\exists t \in movie\ [m(title) = t(title) \wedge t(actor) = a(actor)]]\}$
- Rule: $\forall x\ \varphi(x) \equiv \neg\exists x\ \neg\varphi(x)$

50

Convert to SQL Query

- Basic rule: one level of nesting for each “¬∃”
- SELECT ACTOR FROM MOVIE
  WHERE ACTOR NOT IN
  (SELECT S.ACTOR
  FROM MOVIE S, MOVIE M
  WHERE M.DIR = “BERTO”
  AND S.ACTOR NOT IN
      (SELECT T.ACTOR
      FROM MOVIE T
      WHERE M.TITLE = T.TITLE))
- Note: Calculus is more flexible than SQL

51

Another Possibility

- SELECT A.ACTOR FROM MOVIE A
  WHERE NOT EXISTS
      (SELECT * FROM MOVIE M
      WHERE M.DIR = BERTO AND
      NOT EXISTS
          (SELECT *
          FROM MOVIE T
          WHERE M.TITLE = T.TITLE
          AND T.ACTOR = A.ACTOR))

52

Domain Relational Calculus

- Style of QBE: variables represent values, not tuples
- Difference with tuple calculus: instead of tuple variable t use different variable for each value t(A)
- Examples (movie database)

  Query 1.

  $\{t: \text{TITLE} \mid \exists s: \text{THEATER } (\text{SCHEDULE}(s, t))\}$

Examples, continued

- Query 2.

  $\{t: \text{TITLE} \mid \exists a: \text{ACTOR}(\text{MOVIE}(t, \text{“BERTO”}, a))\}$
- Query 3.

  $\{t: \text{TITLE } d: \text{DIRECTOR} \mid \exists x: \text{THEATER}$
  $\exists a: \text{ACTOR}(\text{Movie}(t\ d\ a) \wedge$
  $\text{SCHEDULE}(x\ t))\}$
- Rules: similar to tuple calculus
  - Simplest formulas: $R(x_1 \ldots x_n)$
    x comp y
    x comp <const>
  - Boolean combinations: $\wedge, \vee, \neg$
  - Quantifiers: $\exists x: A\varphi(x)$, $\forall x: A\varphi(x)$ where x is free in φ

Rules, continued

- Convert to QBE
  - Basic rule: one stage for each “¬∃”
  - Stage I:

MOVIE	TITLE	DIR	ACTOR
	-t		

TEMP	NAME
I.	-n

LIKES	NAME	TITLE
¬	-n	-t
	-n	

  - Stage II:

LIKES	NAME	TITLE
	-n	

TEMP	NAME
¬	-n

ANSWER	NAME
I.	-n

55

Using Domain Calculus to Formulate QBE Queries

- Similar to tuple calculus and SQL
- Example: Movie DB with an additional table

LIKES	NAME	TITLE

Query: “Find people who like all movies”

- Domain calculus:

{u: NAME | ∀t: TITLE ∀d: DIR ∀a:
ACTOR(MOVIE(tda) → LIKES(ut))}

- Eliminate ∀:

{u: NAME | ¬∃t: TITLE ∃d: DIR ∃a:
ACTOR(MOVIE(tda) ∧ ¬LIKES(ut))}

56

Relational Algebra

- Simple set of algebraic operations on relations
- Projection: eliminate some columns

$\pi_X(R)$ R table name

X $\subseteq$ attributes(R)

– e.g. Query1: π_{TITLE}(SCHEDULE)

Another example:

R	A	B	C
	0	1	2
	0	2	2
	1	3	1
	0	1	3

$\pi_A(R)$ =

A
0
1

$\pi_{AB}(R)$ =

A	B
0	1
0	2
1	3

No repetitions of tuples!

57

Relational Algebra

- Selection: select tuples satisfying condition of the form

att = value

att $\neq$ value

$att_1 = (\neq)\ att_2$

other comparators possible

$\sigma_{cond}(R)$ where cond is a condition involving only attributes of R

e.g. $\sigma_{A=0}(R)$ =

A	B	C
0	1	2
0	2	2
0	2	2

$\sigma_{B=C}(R)$ =

A	B	C
0	2	2

$\sigma_{A\neq 0}(R)$ =

A	B	C
1	3	1

58

Relational Algebra

- Union, Difference

  R, S: tables with same attributes

  $R \cup S$, $R - S$: union, difference of sets of tuples in R and S.

- Cross-product

  R, S: tables with disjoint attributes

  $R \times S$: cross-product of tuples

e.g.

R	A	B
	0	1
	2	2
	5	0

S	C	D
	c_1	d_1
	c_2	d_2

$R \times S$	A	B	C	D
	0	1	c_1	d_1
	0	1	c_2	d_2
	2	2	c_1	d_1
	2	2	c_2	d_2
	5	0	c_1	d_1
	5	0	c_2	d_2

59

Dealing with Common Attributes

- Prefix by relation name

R	A	B

S	B	C

$R \times S$	R.A	R.B	S.B	S.C

- Cleaner solution: renaming

  $\delta_{B \to D}(S)$ =

S	D	C

  B is renamed to D in S

  Now can take cross-product

  $R \times \delta_{B \to D}(S)$

60

Relational Algebra

- Basic set of operations:
  $\pi, \sigma, \cup, -, \times, \delta$

- Back to movie example queries:

  Query 1: $\pi_{TITLE}(SCHEDULE)$

  Query 2: $\pi_{TITLE}(\sigma_{DIR=BERTO}(MOVIE))$

  Query 3: $\pi_{TITLE,\ DIR}\, \sigma_{TITLE=TITLE1}(MOVIE\ X\ \delta_{TITLE \rightarrow TITLE1}(SCHEDULE))$

61

Relational Algebra

- Note: Query 3 awkward because of renaming for common attribute
- New operation forcing matching on common attributes
  - Join: $R \bowtie S$

    e.g.

R	A	B

S	B	C

    $R \bowtie S = \pi_{ABC}\ (\sigma_{B=B'}(R \times \delta_{B \rightarrow B'}(S)))$

- Note: more than one common attribute allowed
  - If there are no common attributes, Join is $\times$ (cross-product)

62

Relational Algebra

- Note: Query 3 awkward because of renaming for common attribute
- New operation forcing matching on common attributes
  - Join: R ⊳⊲ S

    e.g. R(A B) S(B C)

    $R \rhd\lhd S = \pi_{ABC}(\sigma_{B=B'}(R \times \delta_{B \to B'}(S)))$

- Note: more than one common attribute allowed
  - If there are no common attributes, Join is × (cross-product)

63

Relational Algebra

Examples:

R

A	B
0	1
0	2
5	3

S

B	C
1	2
1	3
2	2

R⊳⊲S =

A	B	C
0	1	2
0	1	3
0	2	2

Note that <5, 3> is "lost".

If S is

S

C
2
3

then R⊳⊲S =

A	B	C
0	1	2
0	1	3
0	2	2
0	2	3
5	3	2
5	3	3

64

Relational Algebra

- Query 3 is now simpler:
  - $\pi_{TITLE,\ DIR}(\text{MOVIE} \bowtie \text{SCHEDULE})$
- Other useful operations:
  - Intersection $R \cap S$ att(R) = att(S)
  - Division (Quotient) $R \div S$

R	A	B

S	B

$R \div S$: $\{a \mid \langle a, b\rangle \in R$ for <u>every</u> $b \in S\}$

e.g.:

R	A	B
	0	α
	0	β
	1	α
	1	β
	1	γ
	2	α

S	B
	α
	β

R÷S	A
	0
	1

65

Relational Algebra

- Note:
  - π is like $\exists$ "there exists"…
  - $\div$ is like $\forall$ "for all"…
- Expressing $\div$ using logic operations:
  - $R \div S = \pi_A(R) - \pi_A(\pi_A(R) \bowtie S - R)$
  - Similar to:

$$\forall x\ \varphi(x) \equiv \neg\exists x\ \neg\varphi(x)$$

66

Calculus Vs. Algebra

- Underline: Theorem: Calculus and Algebra are equivalent

Basic Correspondence

Algebra Operation	Calculus Operator
π	$\exists$
σ	x comp y
$\cup$	$\vee$
$\bowtie$	$\wedge$
$-$	$\neg$
$\div$	$\forall$

67

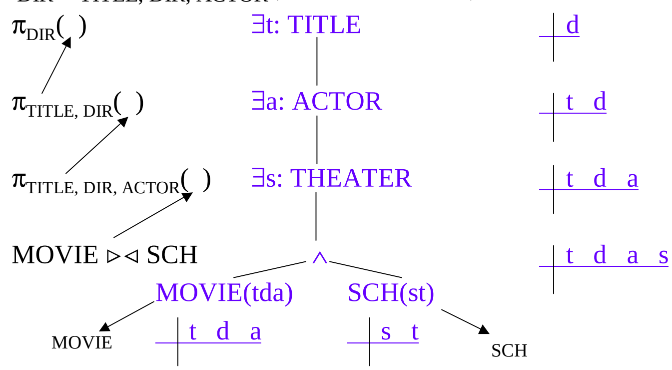
Examples

- Find directors of currently playing movies
  - Domain calculus:
    {d: DIR | ∃t: TITLE ∃a: ACTOR ∃s: THEATER
    (MOVIE(tda) ∧ SCHEDULE(st))}
  - Algebra:
    π_{DIR}(MOVIE $\bowtie$ SCHEDULE)

68

Examples

- $\pi_{DIR}\ \pi_{TITLE,\ DIR,\ ACTOR}(MOVIE \bowtie sd)$



- {tdas | MOVIE(tda) ∧ SCH(st)}
- π_{DIR} (MOVIE▷◁SCH)
- $\pi_X\ \pi_{XY}(\ ) \rightarrow \pi_X(\ )$

69

Examples

- Database schema:
  - PERSON : NAME AGE
- Query:
  - FIND OLDEST PERSON(S)

- Domain calculus

  {n: NAME | ∃a: AGE(PERSON(na) ∧
  ¬∃a’: AGE ∃n’: NAME (PERSON(n’a’) ∧ a<a’))}
- Algebra

  $\pi_{NAME}(PERSON) -$
  $\pi_{NAME}(\sigma_{AGE<AGE'}(PERSON \bowtie \delta_{AGE \rightarrow AGE'}(\pi_{AGE}(PERSON))))$

70