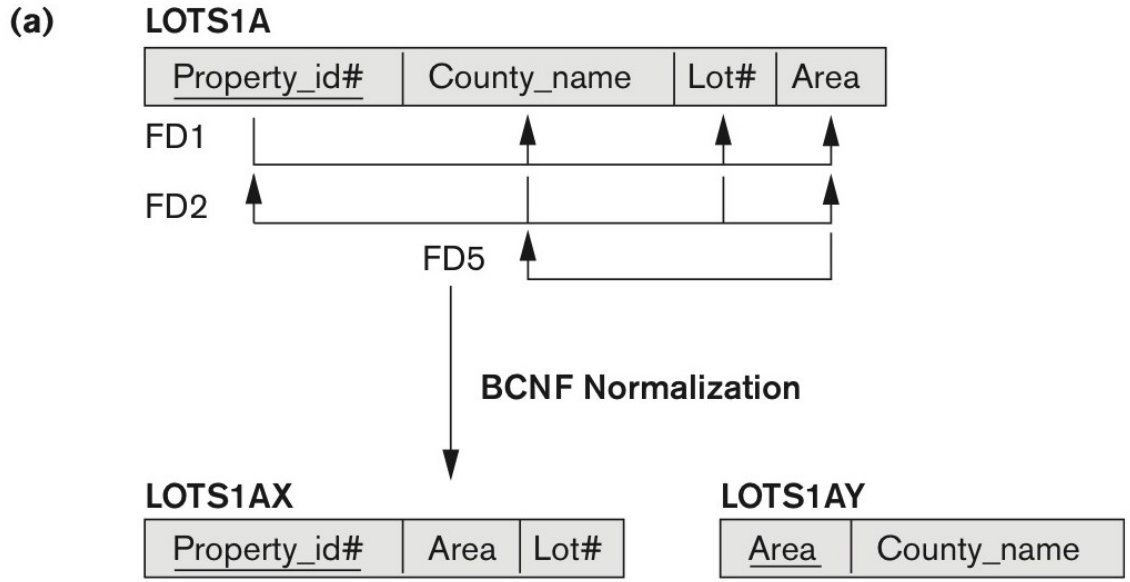
CSC 261/461 – Database Systems
Lecture 13

Fall 2017

Announcement

- Start learning HTML, CSS, JavaScript, PHP + SQL
  - We will cover the basics next week
  - https://www.w3schools.com/php/php_mysql_intro.asp
- Project 1 Milestone 3 will be out soon
  - Combination of Theory and Application
    - BCNF decomposition and PHP and MySQL
- Term Paper for Grad Students
  - http://www.cs.rochester.edu/courses/261/fall2017/projects/grad-topics.html

Clarification



If we remove FD2, then LOTS1A is in _____ form?

2NF

Agenda

- Relational Algebra (Today)
- Relational Calculus (We will not cover)

RELATIONAL ALGEBRA

Motivation

Relational Algebra provides a formal foundation for relational model operations

It is the basis for implementing and optimizing queries in any RDBMS

The core operations of most relational systems are based on Relational Algebra

The Relational Model: Schemata

- Relational Schema:

Relation name

String, float, int, etc. are the **domains** of the attributes

Attributes

The Relational Model: Data

An **attribute** (or **column**) is a typed data entry present in each tuple in the relation

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

The number of attributes is the **arity** of the relation

The Relational Model: Data

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

The number of tuples is the **cardinality** of the relation

A **tuple** or **row** (or *record)* is a single entry in the table having the attributes specified by the schema

The Relational Model: Data

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

Recall: In practice DBMSs relax the set requirement, and use multisets.

A **relational instance** is a ***set*** of tuples all conforming to the same *schema*

Relation Instances

- *Relation DB Schema*
  - Students(sid: *string*, name: *string*, gpa: *float*)
  - Courses(cid: *string*, cname: *string*, credits: *int*)
  - Enrolled(sid: *string*, cid: *string*, grade: *string*)

Note that the schemas impose effective domain / type constraints, i.e. Gpa can't be "Apple"

Relation Instances

Sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

sid	cid	Grade
123	564	A

Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses

Querying

```
SELECT S.name
FROM Students S
WHERE S.gpa > 3.5;
```

"Find names of all students with GPA > 3.5"

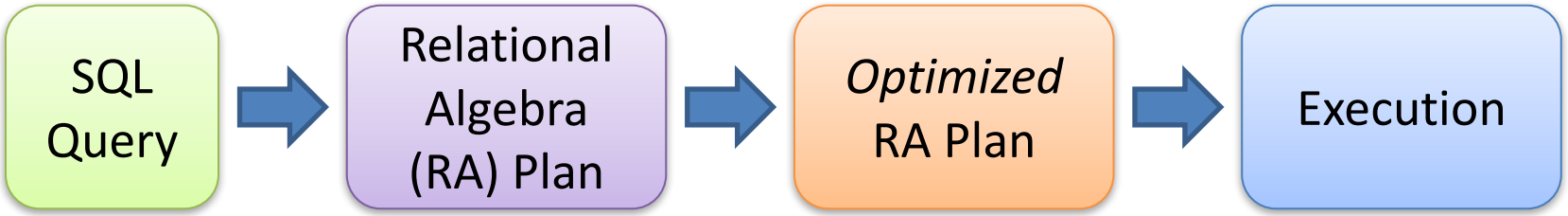
We don't tell the system *how* or *where* to get the data- **just what we want**, **i.e., Querying is *declarative***

To make this happen, we need to translate the *declarative* query into a series of operators… we'll see this next!

Relational Algebra

RDBMS Architecture

How does a SQL engine work ?



Declarative query (from user)

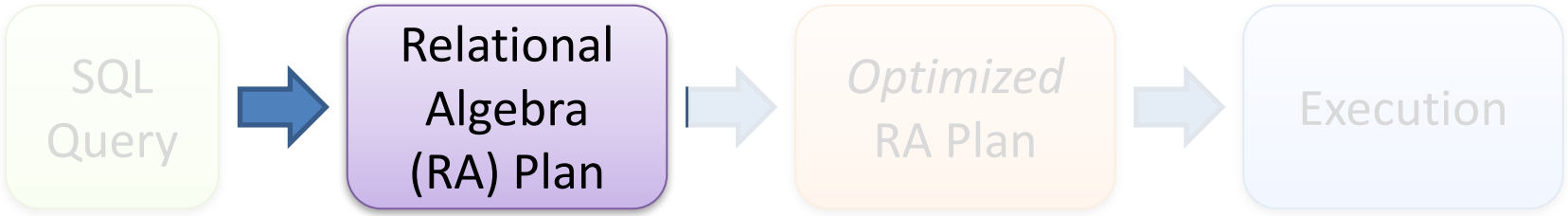
Translate to relational algebra expresson

Find logically equivalent- but more efficient- RA expression

Execute each operator of the optimized plan!

RDBMS Architecture

How does a SQL engine work ?



Relational Algebra allows us to translate declarative (SQL) queries into precise and optimizable expressions!

Relational Algebra (RA)

- Five **basic** operators:
  1. Selection: σ
  2. Projection: Π
  3. Cartesian Product: ×
  4. Union: ∪
  5. Difference: -
- Derived or auxiliary operators:
  - Intersection
  - Joins (natural, equi-join, theta join, semi-join)
  - Renaming: ρ
  - Division

We'll look at these first!

And also at one example of a derived operator (natural join) and a special operator (renaming)

Keep in mind: RA operates on sets!

- RDBMSs use multisets, however in relational algebra formalism we will consider **sets**!

- Also: we will consider the ***named perspective***, where every attribute must have a unique name
  - →attribute order does not matter…

Now on to the basic RA operators…

1. Selection (σ)

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Examples
  - $\sigma_{\text{Salary} > 40000}(\text{Employee})$
  - $\sigma_{\text{name} = \text{“Smith”}}(\text{Employee})$
- The condition c can be $=$, $<$, $\leq$, $>$, $\geq$, $\neq$

```
Students(sid,sname,gpa)
```

SQL:

```
SELECT *
FROM Students
WHERE gpa > 3.5;
```

RA:

$\sigma_{gpa>3.5}(Students)$

Another example:

SSN	Name	Salary
1234545	John	200000
5423341	Smith	600000
4352342	Fred	500000

$\sigma_{\text{Salary} > 40000}$ (Employee)

SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

2. Projection (Π)

- Eliminates columns, then removes duplicates
- Notation: $\Pi_{A_1,\ldots,A_n}(R)$
- Example: project social-security number and names:
  - $\Pi_{SSN,\ Name}$ (Employee)
  - Output schema: Answer(SSN, Name)

```
Students(sid,sname,gpa)
```

SQL:

```
SELECT DISTINCT
   sname,
   gpa
FROM Students;
```

RA:

$\Pi_{sname,gpa}(Students)$

Another example:

SSN	Name	Salary
1234545	John	200000
5423341	John	600000
4352342	John	200000

$\Pi_{\text{Name,Salary}}$ (Employee)

Name	Salary
John	200000
John	600000

Note that RA Operators are Compositional!

```
Students(sid,sname,gpa)
```

```
SELECT DISTINCT
   sname,
   gpa
FROM Students
WHERE gpa > 3.5;
```

$$\Pi_{sname,gpa}(\sigma_{gpa>3.5}(Students))$$

$$\sigma_{gpa>3.5}(\Pi_{sname,gpa}(Students))$$

How do we represent this query in RA?

Are these logically equivalent?

3. Cross-Product (×)

- Each tuple in R1 with each tuple in R2
- Notation: R1 × R2
- Example:
  - Employee × Dependents
- Rare in practice; mainly used to express joins

```
Students(sid,sname,gpa)
People(ssn,pname,address)
```

SQL:

```
SELECT *
FROM Students, People;
```

RA:

$Students \times People$

Another example:

People

ssn	pname	address
1234545	John	216 Rosse
5423341	Bob	217 Rosse

×

Students

sid	sname	gpa
001	John	3.4
002	Bob	1.3

$Students \times People$

ssn	pname	address	sid	sname	gpa
1234545	John	216 Rosse	001	John	3.4
5423341	Bob	217 Rosse	001	John	3.4
1234545	John	216 Rosse	002	Bob	1.3
5423341	Bob	216 Rosse	002	Bob	1.3

Renaming (ρ)

```
Students(sid,sname,gpa)
```

- Changes the schema, not the instance
- A 'special' operator- neither basic nor derived
- Notation: $\rho_{B_1,\ldots,B_n}(R)$

- **Note: this is shorthand for the proper form (since names, not order matters!):**
  - $\rho_{A_1 \to B_1,\ldots,A_n \to B_n}(R)$

SQL:

```
SELECT
  sid AS studId,
  sname AS name,
  gpa AS gradePtAvg
FROM Students;
```

RA:

$\rho_{studId,name,gradePtAvg}(Students)$

We care about this operator *because* we are working in a *named perspective*

Another example:

Students

sid	sname	gpa
001	John	3.4
002	Bob	1.3

$\rho_{studId,name,gradePtAvg}(Students)$

Students

studId	name	gradePtAvg
001	John	3.4
002	Bob	1.3

Natural Join (⋈)

Note: Textbook notation is *

```
Students(sid,name,gpa)
People(ssn,name,address)
```

- Notation: $R_1 \bowtie R_2$
- Joins R_1 and R_2 on *equality of all shared attributes*
  - If R_1 has attribute set A, and R_2 has attribute set B, and they share attributes $A \cap B = C$, can also be written: $R_1 \bowtie_C R_2$
- Our first example of a *derived* RA operator:
  - Meaning: $R_1 \bowtie R_2 = \Pi_{A \cup B}(\sigma_{C=D}(\rho_{C \to D}(R_1) \times R_2))$
  - Where:
    - The rename $\rho_{C \to D}$ renames the shared attributes in one of the relations
    - The selection $\sigma_{C=D}$ checks equality of the shared attributes
    - The projection $\Pi_{A \cup B}$ eliminates the duplicate common attributes

SQL:

```
SELECT DISTINCT
  sid, S.name, gpa,
  ssn, address
FROM
  Students S,
  People P
WHERE S.name = P.name;
```

RA:

$Students \bowtie People$

Another example:

Students S

sid	S.name	gpa
001	John	3.4
002	Bob	1.3

$\bowtie$

People P

ssn	P.name	address
1234545	John	216 Rosse
5423341	Bob	217 Rosse

$$Students \bowtie People$$

sid	S.name	gpa	ssn	address
001	John	3.4	1234545	216 Rosse
002	Bob	1.3	5423341	216 Rosse

Natural Join

- Given schemas R(A, B, C, D), S(A, C, E), what is the schema of R ⋈ S ?
- Given R(A, B, C), S(D, E), what is R ⋈ S ?
- Given R(A, B), S(A, B), what is R ⋈ S ?

Example: Converting SFW Query -> RA

```
Students(sid,name,gpa)
People(ssn,name,address)
```

```
SELECT DISTINCT
  gpa,
  address
FROM Students S,
     People P
WHERE gpa > 3.5 AND
   S.name = P.name;
```

$$\Pi_{gpa,address}(\sigma_{gpa>3.5}(S \bowtie P))$$

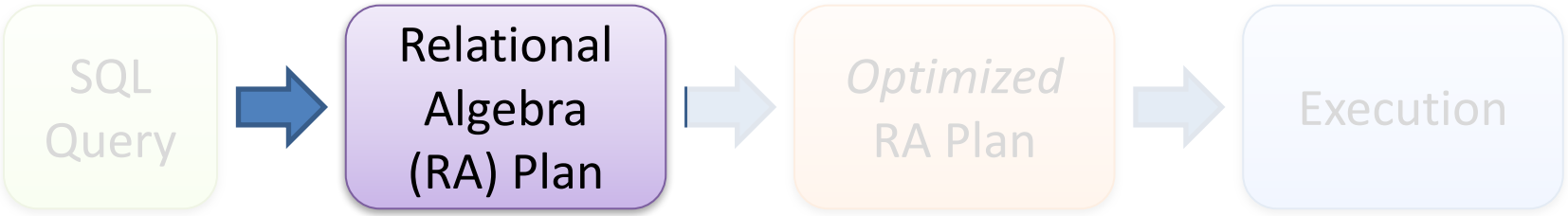
How do we represent this query in RA?

Logical Equivalece of RA Plans

- Given relations R(A,B) and S(B,C):
  - Here, projection & selection commute:
    - $\sigma_{A=5}(\Pi_A(R)) = \Pi_A(\sigma_{A=5}(R))$
  - What about here?
    - $\sigma_{A=5}(\Pi_B(R)) \,?= \Pi_B(\sigma_{A=5}(R))$

RDBMS Architecture

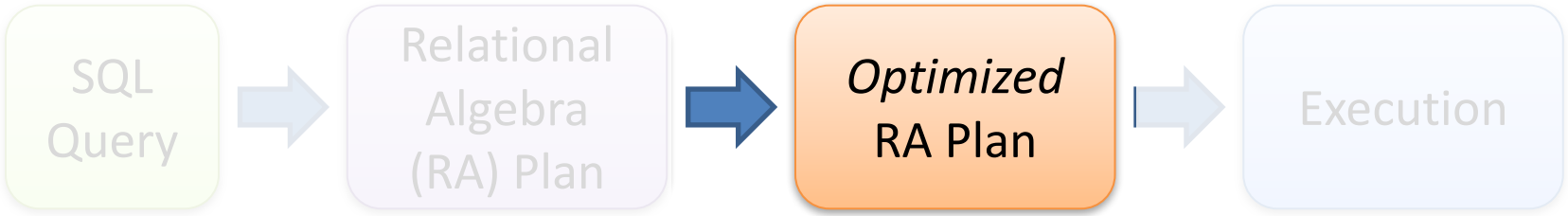
How does a SQL engine work ?



We saw how we can transform declarative SQL queries into precise, compositional RA plans

RDBMS Architecture

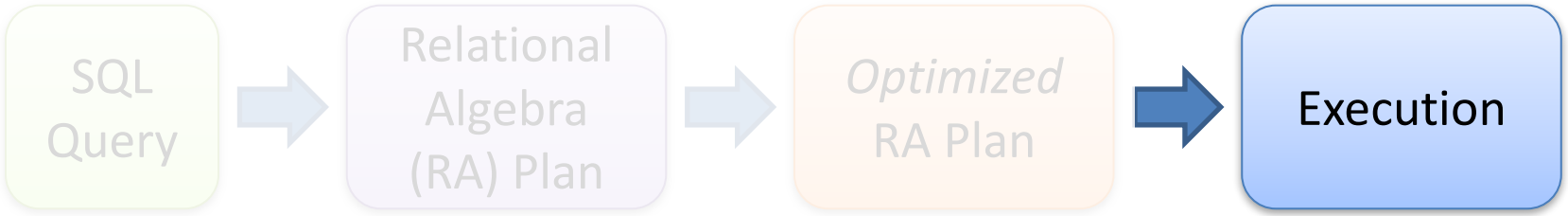
How does a SQL engine work ?



We'll look at how to then optimize these plans later in this lecture

RDBMS Architecture

How is the RA "plan" executed?



We already know how to execute all the basic operators!

2. ADV. RELATIONAL ALGEBRA

What you will learn about in this section

1. Set Operations in RA

2. Fancier RA

Relational Algebra (RA)

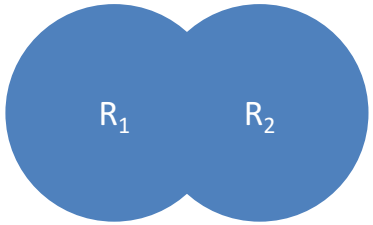
- Five **basic** operators:
  1. Selection: σ
  2. Projection: Π
  3. Cartesian Product: ×
  4. Union: ∪
  5. Difference: -
- Derived or auxiliary operators:
  - Intersection
  - Joins (natural,equi-join, theta join, semi-join)
  - Renaming: ρ
  - Division

We'll look at these

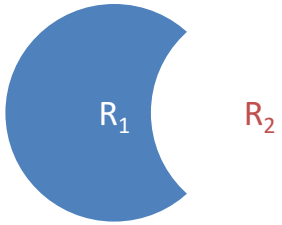
And also at some of these derived operators

1. Union (∪) and 2. Difference (–)

- R1 ∪ R2
- Example:
  - ActiveEmployees ∪ RetiredEmployees

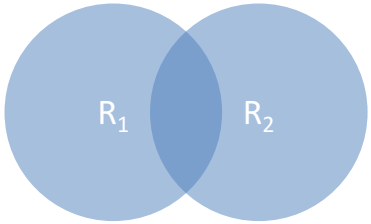


- R1 – R2
- Example:
  - AllEmployees -- RetiredEmployees



What about Intersection (∩) ?

- It is a derived operator
- $R_1 \cap R_2 = R_1 - (R_1 - R_2)$
- Also expressed as a join!
- Example
  - UnionizedEmployees ∩ RetiredEmployees



Fancier RA

Theta Join ($\bowtie_\theta$)

- A join that involves a predicate
- $R1 \bowtie_\theta R2 = \sigma_\theta (R1 \times R2)$
- Here θ can be any condition

Note that natural join is a theta join + a projection.

```
Students(sid,sname,gpa)
People(ssn,pname,address)
```

SQL:

```
SELECT *
FROM
  Students,People
WHERE θ;
```

RA:

$$Students \bowtie_\theta People$$

Equi-join ($\bowtie_{A=B}$)

- A theta join where θ is an equality
- $R1 \bowtie_{A=B} R2 = \sigma_{A=B} (R1 \times R2)$
- Example:
  - Employee $\bowtie_{SSN=SSN}$ Dependents

```
Students(sid,sname,gpa)
People(ssn,pname,address)
```

SQL:

```
SELECT *
FROM
  Students S,
  People P
WHERE sname = pname;
```

RA:

$$S \bowtie_{sname=pname} P$$

Most common join in practice!

Semijoin (⋉)

- $R \ltimes S = \Pi_{A_1,\ldots,A_n}(R \bowtie S)$
- Where $A_1, \ldots, A_n$ are the attributes in R
- Example:
  - Employee ⋉ Dependents

```
Students(sid,sname,gpa)
People(ssn,pname,address)
```

SQL:

```
SELECT DISTINCT
  sid,sname,gpa
FROM
  Students,People
WHERE
  sname = pname;
```

RA:

$$Students \ltimes People$$

Divison (÷)

- $T(Y) = R(Y,X) \div S(X)$

- Y is the set of attributes of R that are not attributes of S.

- For a tuple t to appear in the result T of the Division, the values in t must appear in R in combination with *every* tuple in S.

Example

R(Y,X) ÷ S(X) = T(Y)

```
PilotSkills
pilot_name    plane_name
=========================
'Celko'     'Piper Cub'
'Higgins'   'B-52 Bomber'
'Higgins'   'F-14 Fighter'
'Higgins'   'Piper Cub'
'Jones'     'B-52 Bomber'
'Jones'     'F-14 Fighter'
'Smith'     'B-1 Bomber'
'Smith'     'B-52 Bomber'
'Smith'     'F-14 Fighter'
'Wilson'    'B-1 Bomber'
'Wilson'    'B-52 Bomber'
'Wilson'    'F-14 Fighter'
'Wilson'    'F-17 Fighter'
```

```
Hangar
plane_name
==========
'B-1 Bomber'
'B-52 Bomber'
'F-14 Fighter'
```

```
PilotSkills DIVIDED BY Hangar
pilot_name
============================
'Smith'
'Wilson'
```

```
SELECT PS1.pilot_name
   FROM PilotSkills AS PS1, Hangar AS H1
  WHERE PS1.plane_name = H1.plane_name
  GROUP BY PS1.pilot_name
  HAVING COUNT(PS1.plane_name) =
(SELECT COUNT(plane_name) FROM Hangar);
```

https://www.simple-talk.com/sql/t-sql-programming/divided-we-stand-the-sql-of-relational-division/

Multisets

Recall that SQL uses Multisets

$\lambda(X)$= "Count of tuple in X"
(Items not listed have implicit count 0)

Multiset X

Tuple
(1, a)
(1, a)
(1, b)
(2, c)
(2, c)
(2, c)
(1, d)
(1, d)

Equivalent Representations of a **Multiset**

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	1
(2, c)	3
(1, d)	2

Note: In a set all counts are {0,1}.

Generalizing Set Operations to Multiset Operations

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

$\cap$

Multiset Y

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

$=$

Multiset Z

Tuple	$\lambda(Z)$
(1, a)	2
(1, b)	0
(2, c)	2
(1, d)	0

$$\lambda(Z) = min(\lambda(X), \lambda(Y))$$

For sets, this is **intersection**

Generalizing Set Operations to Multiset Operations

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

$\cup$

Multiset Y

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

$=$

Multiset Z

Tuple	$\lambda(Z)$
(1, a)	7
(1, b)	1
(2, c)	5
(1, d)	2

$$\lambda(Z) = \lambda(X) + \lambda(Y)$$

For sets, this is **union**

Operations on Multisets

- $\sigma_C(R)$: preserve the number of occurrences
- $\Pi_A(R)$: no duplicate elimination
- Cross-product, join: no duplicate elimination

This is important-
relational engines work on multisets, not sets!

Complete Set of Relational Operations

- The set of operations including
  - Select σ,
  - Project π
  - Union $\cup$
  - Difference –
  - Rename ρ, and
  - Cartesian Product X

  – is called a *complete set*

  – because any other relational algebra expression can be expressed by a combination of these five operations.
- For example:
  – $R \cap S = (R \cup S) - ((R - S) \cup (S - R))$
  – $R \bowtie_{\text{<join condition>}} S = \sigma_{\text{<join condition>}} (R \times S)$

Table 8.1 Operations of Relational Algebra

Table 8.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$ OR $R_1 * R_2$

continued on next slide

Table 8.1 Operations of Relational Algebra (continued)

Table 8.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Query Tree Notation

- Query Tree
  - An internal data structure to represent a query
  - Standard technique for estimating the work involved in executing the query, the generation of intermediate results, and the optimization of execution
  - Nodes stand for operations like selection, projection, join, renaming, division, ….
  - Leaf nodes represent base relations
  - A tree gives a good visual feel of the complexity of the query and the operations involved
  - Algebraic Query Optimization consists of rewriting the query or modifying the query tree into an equivalent tree.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

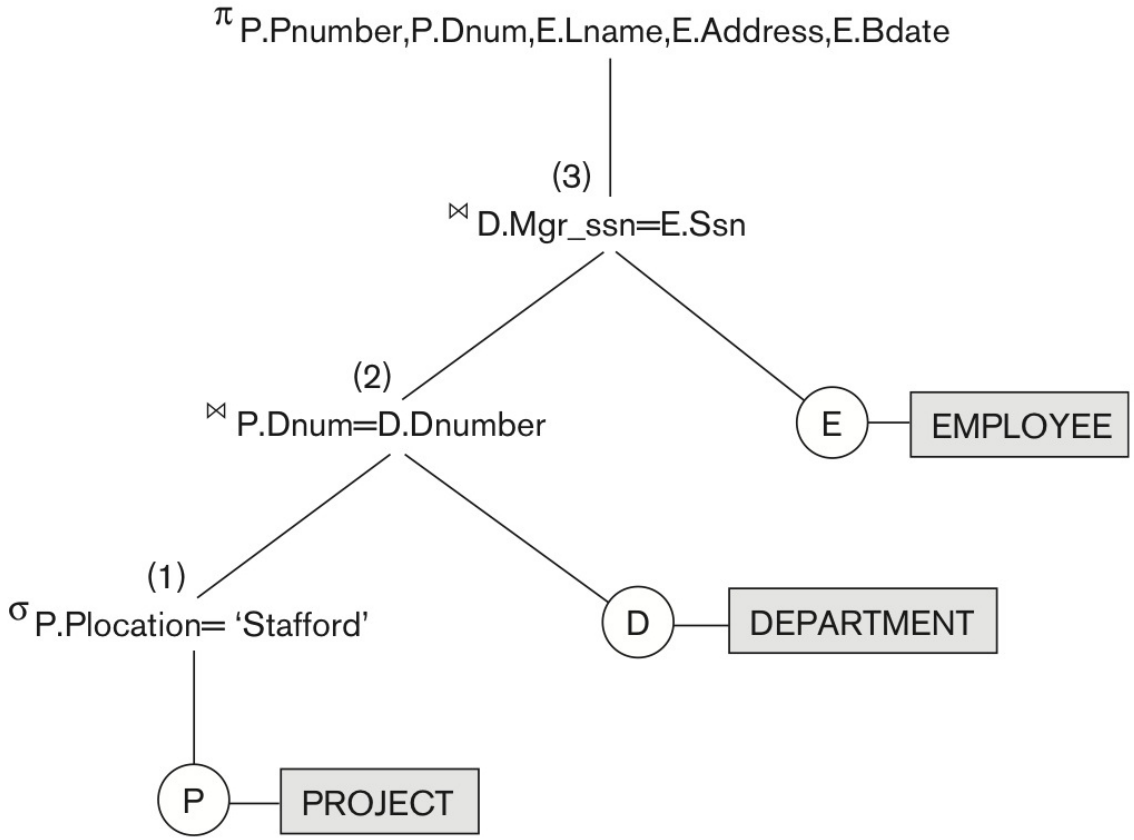
Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Example of Query Tree

- For every project located in Stafford, list the project number, dept. number, manager's last name, address, and birth date
- (page 258)



Summary

- Total 8 basic operators:
  - Unary relational operators (3)
    - Selection: σ
    - Projection: Π
    - Renaming: ρ
  - Binary relational operators (5)
    - Union: ∪
    - Intersect: ∩
    - Set difference: -
    - Cartesian Product (Join): × , ⋈
      - Natural Join, Theta Join, Equi-Join, Semi-Join.
    - Division: ÷
- Tell us: How the query may be executed.

Acknowledgement

- Some of the slides in this presentation are taken from the slides provided by the authors.
- Many of these slides are taken from cs145 course offered by Stanford University.