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
Clarification

(a) LOTS1A

<table>
<thead>
<tr>
<th>Property_id#</th>
<th>County_name</th>
<th>Lot#</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FD2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FD5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BCNF Normalization

LOTS1AX

<table>
<thead>
<tr>
<th>Property_id#</th>
<th>Area</th>
<th>Lot#</th>
</tr>
</thead>
</table>

LOTS1AY

| Area | County_name |

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:

```
Students(sid: string, name: string, gpa: float)
```

- Relation name: Students
- Attributes: String, float, int, etc. are the **domains** of the attributes
### The Relational Model: Data

#### Student

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Bob</td>
<td>3.2</td>
</tr>
<tr>
<td>002</td>
<td>Joe</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>Mary</td>
<td>3.8</td>
</tr>
<tr>
<td>004</td>
<td>Alice</td>
<td>3.5</td>
</tr>
</tbody>
</table>

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

The number of attributes is the **arity** of the relation.
The Relational Model: Data

### Student

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Bob</td>
<td>3.2</td>
</tr>
<tr>
<td>002</td>
<td>Joe</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>Mary</td>
<td>3.8</td>
</tr>
<tr>
<td>004</td>
<td>Alice</td>
<td>3.5</td>
</tr>
</tbody>
</table>

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

The number of tuples is the **cardinality** of the relation.
### The Relational Model: Data

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

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Bob</td>
<td>3.2</td>
</tr>
<tr>
<td>002</td>
<td>Joe</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>Mary</td>
<td>3.8</td>
</tr>
<tr>
<td>004</td>
<td>Alice</td>
<td>3.5</td>
</tr>
</tbody>
</table>

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

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

Note that the schemas impose effective domain/type constraints, i.e. Gpa can’t be “Apple”

<table>
<thead>
<tr>
<th>Sid</th>
<th>Name</th>
<th>Gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Bob</td>
<td>3.2</td>
</tr>
<tr>
<td>123</td>
<td>Mary</td>
<td>3.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>564</td>
<td>564-2</td>
<td>4</td>
</tr>
<tr>
<td>308</td>
<td>417</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>564</td>
<td>A</td>
</tr>
</tbody>
</table>
"Find names of all students with GPA > 3.5"

```
SELECT S.name
FROM Students S
WHERE S.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?

1. **SQL Query**
   - Declarative query (from user)

2. **Relational Algebra (RA) Plan**
   - Translate to relational algebra expression

3. **Optimized RA Plan**
   - Find logically equivalent but more efficient RA expression

4. **Execution**
   - Execute each operator of the optimized plan!
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: \( \sigma \)
  2. Projection: \( \Pi \)
  3. Cartesian Product: \( \times \)
  4. Union: \( \cup \)
  5. Difference: \( - \)

• **Derived or auxiliary operators:**
  – Intersection
  – Joins (natural, equi-join, theta join, semi-join)
  – Renaming: \( \rho \)
  – 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 ($\sigma$)

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

SQL:
SELECT * FROM Students WHERE gpa > 3.5;

RA:
$\sigma_{gpa > 3.5}(Students)$
Another example:

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>200000</td>
</tr>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>500000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>500000</td>
</tr>
</tbody>
</table>
2. Projection (\(\Pi\))

- 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: \(Answer(SSN, Name)\)

SQL:
\[
\text{SELECT DISTINCT sname, gpa}
\text{FROM Students;}
\]

RA:
\[
\Pi_{sname, gpa}(Students)
\]
Another example:

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>200000</td>
</tr>
<tr>
<td>5423341</td>
<td>John</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>John</td>
<td>200000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>200000</td>
</tr>
<tr>
<td>John</td>
<td>600000</td>
</tr>
</tbody>
</table>
Note that RA Operators are Compositional!

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

π_{sname, gpa}(σ_{gpa > 3.5}(Students))

σ_{gpa > 3.5}(π_{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

SQL:
SELECT *
FROM Students, People;

RA:
Students × People
Another example:

<table>
<thead>
<tr>
<th>ssn</th>
<th>pname</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>216 Rosse</td>
</tr>
<tr>
<td>5423341</td>
<td>Bob</td>
<td>217 Rosse</td>
</tr>
</tbody>
</table>

Students × People

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ssn</th>
<th>pname</th>
<th>address</th>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>216 Rosse</td>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>5423341</td>
<td>Bob</td>
<td>217 Rosse</td>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>1234545</td>
<td>John</td>
<td>216 Rosse</td>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
<tr>
<td>5423341</td>
<td>Bob</td>
<td>216 Rosse</td>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
</tbody>
</table>
Renaming ($\rho$)

- 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!):

- $\rho_{A_1\rightarrow B_1,\ldots,A_n\rightarrow B_n}(R)$

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

SQL:

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

RA:

$$\rho_{\text{studId}, \text{name}, \text{gradePtAvg}}(\text{Students})$$
Another example:

\[
\rho_{\text{studId,name,gradePtAvg}}(\text{Students})
\]

<table>
<thead>
<tr>
<th>studId</th>
<th>name</th>
<th>gradePtAvg</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
</tbody>
</table>

CSC 261, Fall 2017, UR
Natural Join (⋈)

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: \( R_1 \bowtie R_2 = \Pi_{A \cup B} (\sigma_{C=D} (\rho_{C \rightarrow D} (R_1 \times R_2))) \)
- Where:
  - The rename \( \rho_{C \rightarrow D} \) 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:

<table>
<thead>
<tr>
<th>Students S</th>
<th>People P</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>ssn</td>
</tr>
<tr>
<td>001</td>
<td>1234545</td>
</tr>
<tr>
<td>002</td>
<td>5423341</td>
</tr>
<tr>
<td>S.name</td>
<td>P.name</td>
</tr>
<tr>
<td>John</td>
<td>John</td>
</tr>
<tr>
<td>Bob</td>
<td>Bob</td>
</tr>
<tr>
<td>gpa</td>
<td>address</td>
</tr>
<tr>
<td>3.4</td>
<td>216 Rosse</td>
</tr>
<tr>
<td>1.3</td>
<td>217 Rosse</td>
</tr>
</tbody>
</table>

Students \(\bowtie\) People

<table>
<thead>
<tr>
<th>sid</th>
<th>S.name</th>
<th>gpa</th>
<th>ssn</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>John</td>
<td>3.4</td>
<td>1234545</td>
<td>216 Rosse</td>
</tr>
<tr>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
<td>5423341</td>
<td>216 Rosse</td>
</tr>
</tbody>
</table>
Natural Join

• Given schemas \( R(A, B, C, D), S(A, C, E) \), what is the schema of \( R \bowtie S \) ?

• Given \( R(A, B, C), S(D, E) \), what is \( R \bowtie S \) ?

• Given \( R(A, B), S(A, B) \), what is \( R \bowtie S \) ?
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;

How do we represent this query in RA?

Π_{gpa, address}(\sigma_{gpa>3.5}(S \bowtie P))
Given relations $R(A, B)$ and $S(B, C)$:

- Here, projection & selection commute:
  \[ \sigma_{\alpha=5}(\Pi_A(R)) = \Pi_A(\sigma_{\alpha=5}(R)) \]

- What about here?
  \[ \sigma_{\alpha=5}(\Pi_B(R)) \neq \Pi_B(\sigma_{\alpha=5}(R)) \]
How does a SQL engine work?

We saw how we can transform declarative SQL queries into precise, compositional RA plans.
How does a SQL engine work?

SQL Query \rightarrow Relational Algebra (RA) Plan \rightarrow Optimized RA Plan \rightarrow Execution

We’ll look at how to then optimize these plans later in this lecture.
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: \( \sigma \)
  2. Projection: \( \Pi \)
  3. Cartesian Product: \( \times \)
  4. Union: \( \cup \)
  5. Difference: \( - \)

- **Derived or auxiliary operators:**
  - Intersection
  - Joins (natural, equi-join, theta join, semi-join)
  - Renaming: \( \rho \)
  - Division

We’ll look at these

And also at some of these derived operators
1. Union (∪) and 2. Difference (−)

- $R_1 \cup R_2$
  - Example: $\text{ActiveEmployees} \cup \text{RetiredEmployees}$

- $R_1 - R_2$
  - Example: $\text{AllEmployees} - \text{RetiredEmployees}$
What about Intersection (\( \cap \))?

- It is a derived operator
- \( R_1 \cap R_2 = R_1 - (R_1 - R_2) \)
- Also expressed as a join!
- Example
  - UnionizedEmployees \( \cap \) RetiredEmployees
Fancier RA
Theta Join ($\bowtie_\theta$)

- A join that involves a predicate
- $R_1 \bowtie_\theta R_2 = \sigma_\theta (R_1 \times R_2)$
- Here $\theta$ can be any condition

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

**SQL:**
```
SELECT *
FROM Students, People
WHERE \theta;
```

**RA:**
```
Students $\bowtie_\theta$ People
```
Equi-join ($\bowtie_{A=B}$)

- A theta join where $\theta$ is an equality
- $R_1 \bowtie_{A=B} R_2 = \sigma_{A=B} (R_1 \times R_2)$
- Example:
  - Employee $\bowtie_{SSN=SSN}$ Dependents

Most common join in practice!
Semijoin (⨉)

- \( R \ ⨉ \ S = \Pi_{A_1, \ldots, A_n} (R \ ⨉ \ S) \)
- Where \( A_1, \ldots, A_n \) are the attributes in \( R \)
- Example:
  - Employee \( \times \) Dependents

**SQL:**

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

**RA:**

\( Students \ ⨉ \ People \)
Division ($\div$)

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

- $Y$ 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

\[
\begin{array}{c|c}
R(Y,X) & S(X) & T(Y) \\
\hline
\text{PilotSkills} & \text{Hangar} & \text{PilotSkills DIVIDED BY Hangar} \\
pilot\_name & plane\_name & pilot\_name \\
\hline
'Celko' & 'Piper Cub' & 'Smith' \\
'Higgins' & 'B-52 Bomber' & 'Wilson' \\
'Higgins' & 'F-14 Fighter' & \\
'Higgins' & 'Piper Cub' & \\
'Jones' & 'B-52 Bomber' & \\
'Jones' & 'F-14 Fighter' & \\
'Smith' & 'B-1 Bomber' & \\
'Smith' & 'B-52 Bomber' & \\
'Smith' & 'F-14 Fighter' & \\
'Wilson' & 'B-1 Bomber' & \\
'Wilson' & 'B-52 Bomber' & \\
'Wilson' & 'F-14 Fighter' & \\
'Wilson' & 'F-17 Fighter' & \\
\end{array}
\]

\[
\begin{aligned}
R(Y,X) & \div \; S(X) = T(Y) \\
\text{SELECT PS1.pilot\_name} & \text{ FROM PilotSkills AS PS1, Hangar AS H1} \\
& \text{ WHERE PS1.plane\_name} = \text{H1.plane\_name} \\
& \text{ GROUP BY PS1.pilot\_name} \\
& \text{ HAVING COUNT(PS1.plane\_name) =} \\
& \text{(SELECT COUNT(plane\_name) FROM Hangar);} \\
\end{aligned}
\]

Multisets
Recall that SQL uses Multisets

Note: In a set all counts are \{0,1\}.
Generalizing Set Operations to Multiset Operations

For sets, this is intersection.

\[ \lambda(Z) = \min(\lambda(X), \lambda(Y)) \]
Generalizing Set Operations to Multiset Operations

For sets, this is union

\[ \lambda(Z) = \lambda(X) + \lambda(Y) \]
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 $\sigma$,
  - Project $\pi$
  - Union $\cup$
  - Difference $-$
  - Rename $\rho$, and
  - Cartesian Product $\times$

  - 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>
<thead>
<tr>
<th>OPERATION</th>
<th>PURPOSE</th>
<th>NOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Selects all tuples that satisfy the selection condition from a relation $R$.</td>
<td>$\sigma_{&lt;\text{selection condition}&gt;}(R)$</td>
</tr>
<tr>
<td>PROJECT</td>
<td>Produces a new relation with only some of the attributes of $R$, and removes duplicate tuples.</td>
<td>$\pi_{&lt;\text{attribute list}&gt;}(R)$</td>
</tr>
<tr>
<td>THETA JOIN</td>
<td>Produces all combinations of tuples from $R_1$ and $R_2$ that satisfy the join condition.</td>
<td>$R_1 \bowtie_{&lt;\text{join condition}&gt;} R_2$</td>
</tr>
<tr>
<td>EQUIJOIN</td>
<td>Produces all the combinations of tuples from $R_1$ and $R_2$ that satisfy a join condition with only equality comparisons.</td>
<td>$R_1 \bowtie_{&lt;\text{join condition}&gt;} R_2$, OR $R_1 \bowtie_{(&lt;\text{join attributes 1}&gt;)} R_2$, $R_1 \bowtie_{(&lt;\text{join attributes 2}&gt;)} R_2$</td>
</tr>
<tr>
<td>NATURAL JOIN</td>
<td>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.</td>
<td>$R_1 \bowtie_{&lt;\text{join condition}&gt;} R_2$, OR $R_1 \bowtie_{(&lt;\text{join attributes 1}&gt;)} R_2$, $R_2$ OR $R_1 \bowtie_{(&lt;\text{join attributes 2}&gt;)} R_2$</td>
</tr>
</tbody>
</table>

*continued on next slide*
### Table 8.1  Operations of Relational Algebra (continued)

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>PURPOSE</th>
<th>NOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNION</td>
<td>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.</td>
<td>( R_1 \cup R_2 )</td>
</tr>
<tr>
<td>INTERSECTION</td>
<td>Produces a relation that includes all the tuples in both ( R_1 ) and ( R_2 ); ( R_1 ) and ( R_2 ) must be union compatible.</td>
<td>( R_1 \cap R_2 )</td>
</tr>
<tr>
<td>DIFFERENCE</td>
<td>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.</td>
<td>( R_1 - R_2 )</td>
</tr>
<tr>
<td>CARTESIAN PRODUCT</td>
<td>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 ).</td>
<td>( R_1 \times R_2 )</td>
</tr>
<tr>
<td>DIVISION</td>
<td>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 ).</td>
<td>( R_1(Z) \div R_2(Y) )</td>
</tr>
</tbody>
</table>
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.
<table>
<thead>
<tr>
<th>EMPLOYEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fname</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>DEPARTMENT</td>
</tr>
<tr>
<td>Dname</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>DEPT_LOCATIONS</td>
</tr>
<tr>
<td>Dnumber</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>PROJECT</td>
</tr>
<tr>
<td>Pname</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>WORKS_ON</td>
</tr>
<tr>
<td>Essn</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>DEPENDENT</td>
</tr>
<tr>
<td>Essn</td>
</tr>
</tbody>
</table>
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: $\sigma$
    • Projection: $\Pi$
    • Renaming: $\rho$
  – Binary relational operators (5)
    • Union: $\cup$
    • Intersect: $\cap$
    • Set difference: $-$
    • Cartesian Product (Join): $\times$, $\bowtie$
    • Division: $\div$

• 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.