Relational Algebra

- The Relational Model consists of the elements: relations, which are made up of attributes.

Relational Algebra

- A relation is a set of attributes with values for each attribute such that:
  - Each attribute value must be a single value only (atomic).
  - All values for a given attribute must be of the same type (or domain).
  - Each attribute name must be unique.
  - The order of attributes is insignificant
  - No two rows (tuples) in a relation can be identical.
  - The order of the rows (tuples) is insignificant.

Relational Algebra

- Relational Algebra is a *collection of operations* on Relations.

- Relations are *operands* and the result of an operation is another relation.

Relational Algebra

- Two main collections of relational operators:
  - Set theory operations:
    - Union, Intersection, Difference and Cartesian product.
  - Specific Relational Operations:
    - Selection, Projection, Join, Division.

Set Theoretic Operations

Consider the following relations **R** and **S**.

R

First	Last	Age
Bill	Smith	22
Sally	Green	28
Mary	Keen	23
Tony	Jones	32

S

First	Last	Age
Forrest	Gump	36
Sally	Green	28
DonJuan	DeMarco	27

Union: R∪S

- Result: Relation with tuples from R and S with duplicates removed.

First	Last	Age
Bill	Smith	22
Sally	Green	28
Mary	Keen	23
Tony	Jones	32
Forrest	Gump	36
DonJuan	DeMarco	27

Difference: R - S

- Result: Relation with tuples from R but not from S

First	Last	Age
Bill	Smith	22
Mary	Keen	23
Tony	Jones	32

Intersection: R∩S

- Result: Relation with tuples that appear in both R and S.

First	Last	Age
Sally	Green	28

Union Compatible Relations

- Attributes of relations need not be identical to perform union, intersection and difference operations.
- However, they must have the same number of attributes or *arity* and the *domains* for corresponding attributes must be identical.

Union Compatible Relations

- **Domain** is the datatype and size of an attribute.
- The **degree** of relation R is the number of attributes it contains.
- **Definition:** Two relations R and S are *union compatible* if and only if they have the same degree and the domains of the corresponding attributes are the same.

Additional properties

- Union, Intersection and difference operators may only be applied to Union Compatible relations.
- Union and Intersection are commutative operations
- Difference operation is NOT commutative.

Cartesian Product: R×S

- Produce all combinations of tuples from two relations.

R

First	Last	Age
Bill	Smith	22
Mary	Keen	23
Tony	Jones	32

S

Dinner	Dessert
Steak	Ice Cream
Lobster	Cheesecake

Cartesian Product: R×S

- R×S:

First	Last	Age	Dinner	Dessert
Bill	Smith	22	Steak	Ice Cream
Bill	Smith	22	Lobster	Cheesecake
Mary	Keen	23	Steak	Ice Cream
Mary	Keen	23	Lobster	Cheesecake
Tony	Jones	32	Steak	Ice Cream
Tony	Jones	32	Lobster	Cheesecake

Relational Algebra

- Two main collections of relational operators:
  - Set theory operations:
    - Union, Intersection, Difference and Cartesian product.
  - Specific Relational Operations:
    - Selection, Projection, Join, Division.

Selection Operator

- Selection and Projection are *unary* operators.
- The selection operator is sigma: σ
- The selection operation acts like a *filter* on a relation by returning only a certain number of tuples.

Selection Operator

- The resulting relation will have the same degree as the original relation.
- The resulting relation may have fewer tuples than the original relation.
- The tuples to be returned are dependent on a *condition* that is part of the selection operator.

Selection Operator

- $\sigma_C(R)$ Returns only those tuples in R that satisfy condition **C**
- A condition C can be made up of any combination of comparison or logical operators that operate on the attributes of R.
  - Comparison operators:

    $=$ $<$ $>$ $\geq$ $\leq$ $\neq$

  - Logical operators:

    $\wedge$ $\vee$ $\neg$

$\wedge$	**T**	**F**
T	T	F
F	F	F

$\vee$	**T**	**F**
T	T	T
F	T	F

$\neg$	**T**	**F**
	F	T

Selection Examples

- Assume the following relation EMP has the following tuples:

Name	Office	Dept	Rank
Smith	400	CS	Assistant
Jones	220	Econ	Adjunct
Green	160	Econ	Assistant
Brown	420	CS	Associate
Smith	500	Fin	Associate

Selection Examples

Select only those Employees in the CS department:

$\sigma_{\text{Dept = 'CS'}}$ (EMP)

Result:

Name	Office	Dept	Rank
Smith	400	CS	Assistant
Brown	420	CS	Associate

Selection Examples

- Select only those Employees with last name Smith who are assistant professors:

  $\sigma_{\text{Name = 'Smith'} \wedge \text{Rank = 'Assistant'}}$ (EMP)

  Result:

Name	Office	Dept	Rank
Smith	400	CS	Assistant

Selection Examples

- Select only those Employees who are either Assistant Professors or in the Economics department:

  $\sigma_{\text{Rank = 'Assistant'} \vee \text{Dept = 'Econ'}}(\text{EMP})$

  Result:

Name	Office	Dept	Rank
Smith	400	CS	Assistant
Jones	220	Econ	Adjunct
Green	160	Econ	Assistant

Selection Examples

- Select only those Employees who are not in the CS department or Adjuncts:

  $\sigma_{\neg (Rank = 'Adjunct' \vee Dept = 'CS')} (EMP)$

  Result:

Name	Office	Dept	Rank
Green	160	Econ	Assistant
Smith	500	Fin	Associate

Projection Operator

- Projection is also a Unary operator.
- The Projection operator is pi: π
- Projection limits the *attributes* that will be returned from the original relation.
- The general syntax is: $\pi_{\text{attributes}} R$
  Where *attributes* is the list of attributes to be displayed and R is the relation.

Projection Operator

- The resulting relation will have the same number of tuples as the original relation (unless there are duplicate tuples produced).
- The degree of the resulting relation may be equal to or less than that of the original relation.

Projection Examples

Project only the names and departments of the employees:

$\pi_{\text{name, dept}}(\text{EMP})$

Results:

Name	Dept
Smith	CS
Jones	Econ
Green	Econ
Brown	CS
Smith	Fin

Combining Selection and Projection

- The selection and projection operators can be combined to perform both operations.
- Show the names of all employees working in the CS department:

$\pi_{\text{name}} \sigma (_{\text{Dept = 'CS'}} (\text{EMP}))$

Results:

Name
Smith
Brown

Combining Selection and Projection

- Show the name and rank of those Employees who are not in the CS department or Adjuncts:

  $\pi_{\text{name, rank}}\ \sigma(\neg\ (_{\text{Rank = 'Adjunct'} \vee\ \text{Dept = 'CS'}})\ (\text{EMP})\ )$

  Results:

Name	Rank
Green	Assistant
Smith	Associate

Aggregate Functions

- We can also apply *Aggregate functions* to attributes and tuples:
  - SUM
  - MINIMUM
  - MAXIMUM
  - AVERAGE, MEAN, MEDIAN
  - COUNT

Aggregate Functions

- Assume the relation EMP has the following tuples:

Name	Office	Dept	Salary
Smith	400	CS	45000
Jones	220	Econ	35000
Green	160	Econ	50000
Brown	420	CS	65000
Smith	500	Fin	60000

Aggregate Functions Examples

- Find the minimum Salary: $\mathcal{F}_{\text{MIN (salary)}}$ (EMP)
  Results:

MIN(salary)
35000

Aggregate Functions Examples

- Find the average Salary: $F_{AVG\ (salary)}$ (EMP)
  Results:

AVG(salary)
51000

Aggregate Functions Examples

- Count the number of employees in the CS department: $F_{COUNT\ (name)}\ \sigma\ (_{Dept = 'CS'}\ (EMP)\ )$
  Results:

COUNT(name)
2

Aggregate Functions Examples

- Find the total payroll for the Economics department: $\mathrm{F}_{\text{SUM (salary)}}\ \sigma\ (_{\text{Dept = 'Econ'}}(\text{EMP}))$
  Results:

SUM(salary)
85000

Join Operation

- Join operations bring together two relations and combine their attributes and tuples in a specific fashion.
- The generic join operator (called the *Theta Join* is: 
- It takes as arguments the attributes from the two relations that are to be joined.

Join Operation

For example assume we have the EMP relation as above and a separate DEPART relation with (Dept, MainOffice, Phone) :

EMP $\bowtie_{\text{EMP.Dept = DEPART.Dept}}$ DEPART

- The join condition can be $=$ $<$ $>$ $\geq$ $\leq$ $\neq$
- When the join condition operator is $=$ then we call this an *Equijoin*
- Note that the attributes in common are repeated.

Join Examples

- Assume we have the EMP relation from above and the following DEPART relation:

Name	Office	Dept	Salary
Smith	400	CS	45000
Jones	220	Econ	35000
Green	160	Econ	50000
Brown	420	CS	65000
Smith	500	Fin	60000

Dept	MainOffice	Phone
CS	404	555-1212
Econ	200	555-1234
Fin	501	555-4321
Hist	100	555-9876

Join Examples

- Find all information on every employee including their department info:
  EMP $\bowtie_{\text{emp.Dept = depart.Dept}}$ DEPART

Join Examples

- EMP ⋈ emp.Dept = depart.Dept DEPART

Name	Office	EMP.Dept	Salary	DEPART.Dept	MainOffice	Phone
Smith	400	CS	45000	CS	404	555-1212
Jones	220	Econ	35000	Econ	200	555-1234
Green	160	Econ	50000	Econ	200	555-1234
Brown	420	CS	65000	CS	404	555-1212
Smith	500	Fin	60000	Fin	501	555-4321

Join Examples

- Find all information on every employee including their department info, where the employee works in an office numbered less than the department main office:

  $EMP \bowtie_{(emp.office < depart.mainoffice) \wedge (emp.dept = depart.dept)} DEPART$

Join Examples

- EMP $\bowtie$ (emp.office < depart.mainoffice) $\wedge$ (emp.dept = depart.dept) DEPART

Name	Office	EMP.Dept	Salary	DEPART.Dept	MainOffice	Phone
Smith	400	CS	45000	CS	404	555-1212
Green	160	Econ	50000	Econ	200	555-1234
Smith	500	Fin	60000	Fin	501	555-4321

Natural Join

- Notice in the generic (Theta) join operation, any attributes in common (such as dept above) are repeated.
- The *Natural Join* operation removes these duplicate attributes.
- The natural join operator is: *****
- We can also assume using ***** that the join condition will be **=** on the two attributes in common.

Natural Join Example

- Example: EMP * DEPART

Results:

Name	Office	Dept	Salary	MainOffice	Phone
Smith	400	CS	45000	404	555-1212
Jones	220	Econ	35000	200	555-1234
Green	160	Econ	50000	200	555-1234
Brown	420	CS	65000	404	555-1212
Smith	500	Fin	60000	501	555-4321

Outer Join

- Often in joining two relations, a tuple in one relation does not have a matching tuple in the other relation: there is no matching value in the join attributes.
- To display rows in the result that do not have matching values in the join column, use Outer join.
- Types of outer joins:
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join

Left Outer Join

R ⟕ S

- (Left) outer join is a join in which tuples from R that do not have matching values in common columns of S are also included in result relation.

- Missing values in the second relation are set to null.

- The advantage of an Outer join is that information is preserved, that is, the Outer join preserves tuples that would have been lost by other types of tuples.

Outer Join Examples

- Assume we have two relations:

PEOPLE:

Name	Age	Food
Alice	21	Hamburger
Bill	24	Pizza
Carl	23	Beer
Dina	19	Shrimp

MENU:

Food	Day
Pizza	Monday
Hamburger	Tuesday
Chicken	Wednesday
Pasta	Thursday
Tacos	Friday

Left Outer Join

- PEOPLE $\overset{\text{people.food = menu.food}}{⟕}$ MENU

Name	Age	people.Food	menu.Food	Day
Alice	21	Hamburger	Hamburger	Tuesday
Bill	24	Pizza	Pizza	Monday
Carl	23	Beer	*NULL*	*NULL*
Dina	19	Shrimp	*NULL*	*NULL*

Right Outer Join

- PEOPLE ⟖ $_{\text{people.food = menu.food}}$ MENU

Name	Age	people.Food	menu.Food	Day
Bill	24	Pizza	Pizza	Monday
Alice	21	Hamburger	Hamburger	Tuesday
NULL	*NULL*	*NULL*	Chicken	Wednesday
NULL	*NULL*	*NULL*	Pasta	Thursday
NULL	*NULL*	*NULL*	Tacos	Friday

Full Outer Join

- PEOPLE ⟗ $_{\text{people.food = menu.food}}$ MENU

Name	Age	people.Food	menu.Food	Day
Alice	21	Hamburger	Hamburger	Tuesday
Bill	24	Pizza	Pizza	Monday
Carl	23	Beer	*NULL*	*NULL*
Dina	19	Shrimp	*NULL*	*NULL*
NULL	*NULL*	*NULL*	Chicken	Wednesday
NULL	*NULL*	*NULL*	Pasta	Thursday
NULL	*NULL*	*NULL*	Tacos	Friday

Relational algebra and SQL

- SELECT statement

SELECT [DISTINCT | ALL]
{* | [columnExpression [AS newName]] [,...] }
FROM TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList] [HAVING condition]
[ORDER BY columnList]

SELECT Statement

FROM	**Specifies table(s) to be used.**
WHERE	**Filters rows.**
GROUP BY	**Forms groups of rows with same column value.**
HAVING	**Filters groups subject to some condition.**
SELECT	**Specifies which columns are to appear in output.**
ORDER BY	**Specifies the order of the output.**

Relational algebra and SQL

- Projection
- Example: The table **E** (for **EMPLOYEE**)

nr	**name**	**salary**
1	John	100
5	Sarah	300
7	Tom	100

Relational algebra and SQL - Projection

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td><pre>select distinct
salary from E</pre></td>
<td>
<table>
<tr><th>salary</th></tr>
<tr><td>100</td></tr>
<tr><td>300</td></tr>
</table>
</td>
<td>$\pi_{\text{salary}}(E)$</td>
</tr>
<tr>
<td><pre>select nr,
salary from E</pre></td>
<td>
<table>
<tr><th>nr</th><th>salary</th></tr>
<tr><td>1</td><td>100</td></tr>
<tr><td>5</td><td>300</td></tr>
<tr><td>7</td><td>100</td></tr>
</table>
</td>
<td>$\pi_{\text{nr, salary}}(E)$</td>
</tr>
</table>

Relational algebra and SQL - Selection

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td><pre>select * from
E where salary
< 200</pre></td>
<td>
<table>
<tr><th>nr</th><th>name</th><th>salary</th></tr>
<tr><td>1</td><td>John</td><td>100</td></tr>
<tr><td>7</td><td>Tom</td><td>100</td></tr>
</table>
</td>
<td>$\sigma_{\text{salary} < 200}(E)$</td>
</tr>
<tr>
<td><pre>select * from
E where salary
< 200 and
nr >= 7</pre></td>
<td>
<table>
<tr><th>nr</th><th>name</th><th>salary</th></tr>
<tr><td>7</td><td>Tom</td><td>100</td></tr>
</table>
</td>
<td>$\sigma_{\text{salary} < 200 \text{ and } nr >= 7}(E)$</td>
</tr>
</table>

Combination of projection and selection

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td>

```
select name,
salary from
E where
salary < 200
```

</td>
<td>
<table>
<tr><th>name</th><th>salary</th></tr>
<tr><td>John</td><td>100</td></tr>
<tr><td>Tom</td><td>100</td></tr>
</table>
</td>
<td>Π name, salary (σ salary < 200(E))</td>
</tr>
</table>

Cartesian Product

table **E** (for **EMPLOYEE**)

enr	ename	edept
1	Bill	A
2	Sarah	C
3	John	A

table **D** (for **DEPARTMENT**)

dnr	dname
A	Marketing
B	Sales
C	Legal

Cartesian Product

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td><code>select *from E, D</code></td>
<td>
<table>
<tr><th>enr</th><th>ename</th><th>edept</th><th>dnr</th><th>dname</th></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>B</td><td>Sales</td></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>C</td><td>Legal</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>A</td><td>Marketing</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>B</td><td>Sales</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>C</td><td>Legal</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>B</td><td>Sales</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>C</td><td>Legal</td></tr>
</table>
</td>
<td>E × D</td>
</tr>
</table>

Join ("inner join")

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td>

```
select *
from E, D
where
dept = dnr
```

</td>
<td>
<table>
<tr><th>enr</th><th>ename</th><th>dept</th><th>dnr</th><th>dname</th></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>C</td><td>Legal</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>A</td><td>Marketing</td></tr>
</table>
</td>
<td>$\sigma_{dept = dnr} (E \times D)$
or, using the equivalent join operation
$E \bowtie_{dept = dnr} D$</td>
</tr>
</table>

Aggregate functions

- Table **E** (for **EMPLOYEE**)

nr	**name**	**salary**	**dept**
1	John	100	A
5	Sarah	300	C
7	Tom	100	A
12	Anne	**null**	C

Sum

SQL	Result	Relational algebra
`select sum(salary) from E`	**sum** 500	$F_{sum(salary)}(E)$

- Count:
  - Duplicates are not eliminated.
  - **Null** values are ignored.

Count

SQL	Result	Relational algebra
`select count(salary) from E`	**count** 3	$F_{count(salary)}(E)$
`select count(distinct salary)from E`	**count** 2	$F_{count(salary)}(\pi_{salary}(E))$

Aggregate Functions

- We can calculate aggregates "grouped by" something:

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td><pre>select
sum(salary)
from E group
by dept</pre></td>
<td>
<table>
<tr><th>dept</th><th>sum</th></tr>
<tr><td>A</td><td>200</td></tr>
<tr><td>C</td><td>300</td></tr>
</table>
</td>
<td>$_{dept}F_{sum(salary)}(E)$</td>
</tr>
</table>

Aggregate Functions

- Several aggregates simultaneously:

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td>

```
select
sum(salary),
count(*)from
E group by
dept
```

</td>
<td>
<table>
<tr><th>dept</th><th>sum</th><th>count</th></tr>
<tr><td>A</td><td>200</td><td>2</td></tr>
<tr><td>C</td><td>300</td><td>1</td></tr>
</table>
</td>
<td>$_{dept}F_{sum(salary),\ count(*)}(E)$</td>
</tr>
</table>

Outer join

- Example: Table **E** (for **EMPLOYEE**); table **D** (for **DEPARTMENT**)

enr	ename	dept
1	Bill	A
2	Sarah	B
3	John	A

dnr	dname
A	Marketing
B	Sales
C	Legal

- List each employee together with the department he or she works at.

Outer join

- What if we want to know the number of employees at each department?

Outer join

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td>

```
select *
from (E
right outer
join D on
dept = dnr)
```

</td>
<td>
<table>
<tr><th>enr</th><th>ename</th><th>dept</th><th>dnr</th><th>dname</th></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>2</td><td>Sarah</td><td>B</td><td>B</td><td>Sales</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>null</td><td>null</td><td>null</td><td>C</td><td>Legal</td></tr>
</table>
</td>
<td>$E \bowtie_{edept = dnr} D$</td>
</tr>
</table>

Outer Join

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td>

```
select dnr,
dname,
count(*)from
(E right outer
join D on dept
= dnr)group by
dnr, dname
```

</td>
<td>
<table>
<tr><th>dnr</th><th>dname</th><th>count</th></tr>
<tr><td>A</td><td>Marketing</td><td>2</td></tr>
<tr><td>B</td><td>Sales</td><td>1</td></tr>
<tr><td>C</td><td>Legal</td><td>1</td></tr>
</table>
</td>
<td>dnr, dnameFcount(*)(E ⋈ dept = dnr D)</td>
</tr>
</table>

Outer Join

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td>

```
select dnr,
dname,
count(enr)from
(E right outer
join D on edept
= dnr) group by
dnr, dname
```

</td>
<td>
<table>
<tr><th>dnr</th><th>dname</th><th>count</th></tr>
<tr><td>A</td><td>Marketing</td><td>2</td></tr>
<tr><td>B</td><td>Sales</td><td>1</td></tr>
<tr><td>C</td><td>Legal</td><td>0</td></tr>
</table>
</td>
<td>$_{dnr,\ dname}\mathbf{F}_{count(enr)}(E \underset{dept = dnr}{\bowtie} D)$</td>
</tr>
</table>