Extended Projection

Allow the columns in the projection to be functions of one or more columns in the argument relation.

Example

\[
R = \begin{array}{cc}
A & B \\
1 & 2 \\
3 & 4 \\
\end{array}
\]

\[
\pi_{A+B,A,A}(R) = \begin{array}{ccc}
A+B & A1 & A2 \\
3 & 1 & 1 \\
7 & 3 & 3 \\
\end{array}
\]
Sorting

• $\tau_L(R)$ = list of tuples of $R$, ordered according to attributes on list $L$.
• Note that result type is outside the normal types (set or bag) for relational algebra.
  – Consequence: $\tau$ cannot be followed by other relational operators.

Example

$R = \begin{array}{|c|c|}
    \hline
    A & B \\
    \hline
    1 & 3 \\
    3 & 4 \\
    5 & 2 \\
    \hline
\end{array}$

$\tau_B(R) = [(5,2), (1,3), (3,4)]$. 

Aggregate Functions

- These functions operate on the multiset of values of a column of a relation, and return a value
  - **avg**: average value
  - **min**: minimum value
  - **max**: maximum value
  - **sum**: sum of values
  - **count**: number of values
Aggregation Operators

• These are not relational operators; rather they summarize a column in some way.
• Five standard operators: Sum, Average, Count, Min, and Max.
• Use with grouping (later today) or shorthand as “special” projection:
• \[ R: A \quad B \]
  \[
  \begin{array}{cc}
  1 & 2 \\
  3 & 4 \\
  \end{array}
  \]
  \[ \pi_{\text{Max}(A), \text{Min}(B)}(R) = \text{Max}(A) \quad \text{Min}(B) \]
  \[
  \begin{array}{cc}
  3 & 2 \\
  \end{array}
  \]
• Remember: Aggregations return a single row – can’t combine with non-aggregates in projection

Aggregate Functions (Cont.)

- Find the average salary of instructors in the Computer Science department
  - select avg (salary)
    from instructor
    where dept_name= 'Comp. Sci.';
- Find the total number of instructors who teach a course in the Spring 2010 semester
  - select count (distinct ID)
    from teaches
    where semester = 'Spring' and year = 2010;
- Find the number of tuples in the course relation
  - select count (*)
    from course;
Aggregate Example

- **select avg (salary) from instructor where dept_name = 'Finance';**
- **Returns how many rows?**
  A. 0  
  B. 1  
  C. 2  
  D. 12  
  E. Not a valid query

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>80000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>Finance</td>
<td>80000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
</tr>
<tr>
<td>58583</td>
<td>Califieri</td>
<td>History</td>
<td>62000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>Physics</td>
<td>87000</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
</tbody>
</table>

Aggregate Functions – Group By

- Find the average salary of instructors in each department
  - **select dept_name, avg (salary) as avg_salary from instructor group by dept_name;**

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>80000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>Finance</td>
<td>80000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
</tr>
<tr>
<td>58583</td>
<td>Califieri</td>
<td>History</td>
<td>62000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>Physics</td>
<td>87000</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dept_name</th>
<th>avg_salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biology</td>
<td>72000</td>
</tr>
<tr>
<td>Comp. Sci.</td>
<td>77333</td>
</tr>
<tr>
<td>Elec. Eng.</td>
<td>80000</td>
</tr>
<tr>
<td>Finance</td>
<td>85000</td>
</tr>
<tr>
<td>History</td>
<td>61000</td>
</tr>
<tr>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>Physics</td>
<td>91000</td>
</tr>
</tbody>
</table>
Grouping Operator

$\gamma_L(R)$, where $L$ is a list of elements that are either

a) Individual (grouping) attributes or

b) Of the form $\theta(A)$, where $\theta$ is an aggregation operator
and $A$ the attribute to which it is applied,
is computed by:

1. Group $R$ according to all the grouping attributes on list $L$.
2. Within each group, compute $\theta(A)$, for each element $\theta(A)$ on list $L$.
3. Result is the relation whose columns consist of one tuple for each group. The components of that tuple are the values associated with each element of $L$ for that group.

Aggregation (Cont.)

- Attributes in select clause outside of aggregate functions must appear in group by list
  
  /* erroneous query */
  
  select dept_name, ID, avg(salary)
  from instructor
  group by dept_name;
Aggregate Functions – Having Clause

- Find the names and average salaries of all departments whose average salary is greater than 42000

```sql
select dept_name, avg(salary)
from instructor
group by dept_name
having avg(salary) > 42000;
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups
Null Values and Aggregates

- Total all salaries
  ```sql
  select sum(salary)
  from instructor
  ```
  - Above statement ignores null amounts
  - Result is null if there is no non-null amount
- All aggregate operations except `count(*)` ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - `count` returns 0
  - all other aggregates return null

How many events this weekend?

<table>
<thead>
<tr>
<th>Title</th>
<th>Date</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corporate Partner UG</td>
<td>Sep 11</td>
<td>7:30pm</td>
<td>LWSN Commons</td>
</tr>
<tr>
<td>Mixer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS Career Fair</td>
<td>Sep 12</td>
<td>5:00pm</td>
<td>CoRec</td>
</tr>
<tr>
<td>Boiler Bridge Walk</td>
<td>Sep 9</td>
<td>5:45pm</td>
<td>Myers Bridge</td>
</tr>
<tr>
<td>Black Lives Matter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Panel Discussion</td>
<td>Sep 13</td>
<td>6:30pm</td>
<td>STEW Fowler Hall</td>
</tr>
<tr>
<td>Hulu Tech Talk</td>
<td>Sep 12</td>
<td>12:00pm</td>
<td>HAAS 101</td>
</tr>
<tr>
<td>Gary Johnson</td>
<td>Sep 13</td>
<td>5:00pm</td>
<td>CoRec</td>
</tr>
<tr>
<td>CS348 Midterm</td>
<td>11:30am</td>
<td>ARMS 101</td>
<td></td>
</tr>
</tbody>
</table>
How many events each day?

<table>
<thead>
<tr>
<th>Title</th>
<th>Date</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corporate Partner UG Mixer</td>
<td>Sep 11</td>
<td>7:30pm</td>
<td>LWSN Commons</td>
</tr>
<tr>
<td>CS Career Fair</td>
<td>Sep 12</td>
<td>5:00pm</td>
<td>CoRec</td>
</tr>
<tr>
<td>Boiler Bridge Walk</td>
<td>Sep 9</td>
<td>5:45pm</td>
<td>Myers Bridge</td>
</tr>
<tr>
<td>Black Lives Matter Panel Discussion</td>
<td>Sep 13</td>
<td>6:30pm</td>
<td>STEW Fowler Hall</td>
</tr>
<tr>
<td>Hulu Tech Talk</td>
<td>Sep 12</td>
<td>12:00pm</td>
<td>HAAS 101</td>
</tr>
<tr>
<td>Gary Johnson</td>
<td>Sep 13</td>
<td>5:00pm</td>
<td>CoRec</td>
</tr>
<tr>
<td>CS348 Midterm</td>
<td></td>
<td>11:30am</td>
<td>ARMS 1010</td>
</tr>
</tbody>
</table>

A. select * from events group by Date  
B. select count(*) from events group by Date  
C. $\gamma_{Date}$ events  
D. select count(*) from events group by Date having date between to_date('Sep 10') and to_date('Sep 11')

Question: 1st Midterm

A: September 26
- Will have completed 5 weeks
- Assignment 1, project 1 graded and returned
- Assignment 2 (relational algebra) complete, solution set available
  *No late assignment 2 would be accepted*
- I would hold office hours Saturday, 9/24
  *If no exam, will be guest lecture this day*

B: October 3
- Assignment 1+2, project 1 graded and returned
- Middle of Project 2

C: October 14
- Expect some database design questions
- Graded and returned shortly before drop date
Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries. A subquery is a select-from-where expression that is nested within another query.
- The nesting can be done in the following SQL query:

```sql
select A_1, A_2, ..., A_n
from r_1, r_2, ..., r_m
where P
```

as follows:
- \( A_i \) can be replaced by a subquery that generates a single value.
- \( r_i \) can be replaced by any valid subquery.
- \( P \) can be replaced with an expression of the form:

\[
B \text{ <operation>} (\text{subquery})
\]

Where \( B \) is an attribute and \(<\text{operation}>\) to be defined later.

Subqueries in the Where Clause

- A common use of subqueries is to perform tests:
  - For set membership
  - For set comparisons
  - For set cardinality.
Set Membership

- Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
  course_id in (select course_id
                 from section
                 where semester = 'Spring' and year= 2010);
```

- Find courses offered in Fall 2009 but not in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
  course_id not in (select course_id
                     from section
                     where semester = 'Spring' and year= 2010);
```

Set Membership (Cont.)

- Find the total number of (distinct) students who have taken course sections taught by the instructor with ID 10101

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
  (select course_id, sec_id, semester, year
   from teaches
   where teaches.ID= 10101);
```

- Note: Above query can be written in a much simpler manner. The formulation above is simply to illustrate SQL features.
**Set Comparison – “some” Clause**

- Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

  ```sql
  select distinct T.name
  from instructor as T, instructor as S
  where T.salary > S.salary and S.dept name = 'Biology';
  ```

- Same query using > some clause

  ```sql
  select name
  from instructor
  where salary > some (select salary
  from instructor
  where dept name = 'Biology');
  ```

**Definition of “some” Clause**

- F <comp> some r ⇔ ∃ t ∈ r such that (F <comp> t)
  Where <comp> can be: <, ≤, >, =, ≠

  - `(5 < some 0 5) = true` (read: 5 < some tuple in the relation)
  - `(5 < some 0 5) = false`
  - `(5 = some 0 5) = true`
  - `(5 ≠ some 0 5) = true` (since 0 ≠ 5)

  `(= some) ≡ in
  However, (≠ some) ≠ not in`
Set Comparison – “all” Clause

- Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

```
select name
from instructor
where salary > all (select salary
    from instructor
    where dept name = 'Biology');
```

Definition of “all” Clause

- \( F < \text{comp} > \text{all} \quad r \iff \forall t \in r \; (F < \text{comp} > t) \)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\((5 < \text{all}) = \text{false} \)

\|   |   |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

\((5 < \text{all}) = \text{true} \)

\|   |   |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

\((5 = \text{all}) = \text{false} \)

\|   |   |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

\((5 \neq \text{all}) = \text{false} \) (since 5 \neq 4 and 5 \neq 6)

\((\neq \text{all}) \equiv \text{not in} \)

However, \((= \text{all}) \neq \text{in} \)
Test for Empty Relations

- The `exists` construct returns the value `true` if the argument subquery is nonempty.
- `exists r ⇔ r ≠ Ø`
- `not exists r ⇔ r = Ø`

Use of “exists” Clause

- Yet another way of specifying the query “Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester”

```sql
select course_id
from section as S
where semester = 'Fall' and year = 2009 and
exists (select *
    from section as T
    where semester = 'Spring' and year = 2010
    and S.course_id = T.course_id);
```

- **Correlation name** – variable S in the outer query
- **Correlated subquery** – the inner query
Use of “not exists” Clause

- Find all students who have taken all courses offered in the Biology department.

```sql
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
from course
where dept_name = 'Biology')
except
(select T.course_id
from takes as T
where S.ID = T.ID));
```

- First nested query lists all courses offered in Biology
- Second nested query lists all courses a particular student took

- Note that \( X - Y = \emptyset \iff X \subseteq Y \)
- Note: Cannot write this query using `all` and its variants

Test for Absence of Duplicate Tuples

- The `unique` construct tests whether a subquery has any duplicate tuples in its result.
- The `unique` construct evaluates to “true” if a given subquery contains no duplicates.
- Find all courses that were offered at most once in 2009

```sql
select T.course_id
from course as T
where unique (select R.course_id
from section as R
where T.course_id = R.course_id
and R.year = 2009);
```
Subqueries in the From Clause

- SQL allows a subquery expression to be used in the `from` clause.
- Find the average instructors’ salaries of those departments where the average salary is greater than $42,000.”

```sql
select dept_name, avg_salary
from (select dept_name, avg (salary) as avg_salary
      from instructor
      group by dept_name)
where avg_salary > 42000;
```

- Note that we do not need to use the `having` clause.
- Another way to write above query

```sql
select dept_name, avg_salary
from (select dept_name, avg (salary)
      from instructor
      group by dept_name) as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

With Clause

- The `with` clause provides a way of defining a temporary relation whose definition is available only to the query in which the `with` clause occurs.
- Find all departments with the maximum budget

```sql
with max_budget (value)
as
(select max(budget)
from department)
select department.name
from department, max_budget
where department.budget = max_budget.value;
```
Complex Queries using With Clause

- Find all departments where the total salary is greater than the average of the total salary at all departments

```sql
with dept_total (dept_name, value) as
    (select dept_name, sum(salary) from instructor
     group by dept_name),
  dept_total_avg(value) as
    (select avg(value) from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.value;
```

Scalar Subquery

- Scalar subquery is one which is used where a single value is expected
- List all departments along with the number of instructors in each department

```sql
select dept_name,
       (select count(*)
        from instructor
        where department.dept_name = instructor.dept_name)
       as num_instructors
from department;
```
- Runtime error if subquery returns more than one result tuple
Outerjoin

The normal join can “lose” information, because a tuple that doesn’t join with any from the other relation (dangles) has no vestage in the join result.

- The null value $\perp$ can be used to “pad” dangling tuples so they appear in the join.
- Gives us the outerjoin operator $\bowtie$.
- Variations: theta-outerjoin, left- and right-outerjoin (pad only dangling tuples from the left (respectively, right).

Left Outer Join

- course natural left outer join prereq

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
<td>null</td>
</tr>
</tbody>
</table>
Right Outer Join

- course natural right outer join prereq

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

Joined Relations

- Join operations take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the from clause
- Join condition – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- Join type – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.
Full Outer Join

- course natural full outer join prereq

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
<th>course_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
<td>null</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

Joined Relations – Examples

- course inner join prereq on
course.course_id = prereq.course_id

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
<th>course_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
<td>null</td>
<td>CS-190</td>
</tr>
<tr>
<td>CS-347</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

- What is the difference between the above, and a natural join?
- course left outer join prereq on
course.course_id = prereq.course_id

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
<th>course_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>CS-347</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Joined Relations – Examples

- **course natural right outer join** `prereq`

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

- **course full outer join** `prereq using (course_id)`

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
<td>null</td>
</tr>
<tr>
<td>CS-347</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

**Extended ("Nonclassical") Relational Algebra**

Add features needed for SQL, bags.

1. Duplicate-elimination operator \( \delta \).
2. Extended projection.
3. Sorting operator \( \tau \).
4. Grouping-and-aggregation operator \( \gamma \).
5. Outerjoin operator \( \bowtie \).