Not Exists

- 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')
minus
(select T.course_id
from takes as T
where S.ID = T.ID));
```

Note that $X - Y = \emptyset \iff X \subseteq Y$
Test for Absence of Duplicate Tuples

Find all courses that were offered at most once in 2009:

- **WRONG:**

  ```sql
  select T.course_id
  from course T
  where unique (select R.course_id
  from section R
  where T.course_id = R.course_id and R.year = 2009);
  ```

- **RIGHT:**

  ```sql
  select T.course_id from course T
  where 1 = (select count(R.course_id)
  from section R
  where T.course_id = R.course_id and R.year = 2009);
  ```

Derived Relations

- 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
Triggers

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database.

- Suppose that instead of allowing negative account balances, the bank deals with overdrafts by
  - 1. setting the account balance to zero
  - 2. creating a loan in the amount of the overdraft
  - 3. giving this loan a loan number identical to the account number of the overdrawn account

**Trigger Example in SQL:1999**

```sql
create trigger overdraft-trigger after update on account
    referencing new row as nrow
    for each row
    when nrow.balance < 0
    begin atomic
        actions to be taken
    end
```
Trigger Example in SQL:1999

```
create trigger overdraft-trigger after update on account
    referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
    insert into borrower
    (select customer-name, account-number
     from depositor
     where nrow.account-number = depositor.account-number);
insert into loan values
    (nrow.account-number, nrow.branch-name, nrow.balance);
update account set balance = 0
    where account.account-number = nrow.account-number
end
```

Triggers…

- **External World Actions**
  - How does the DB order something if the inventory is low?

- **Syntax**
  - Every system has its own syntax

- **Careful with triggers**
  - Cascading triggers, Infinite Sequences...

- **More Info/Examples:**
  - Google: “create trigger” oracle download-uk
Views

• In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
• Consider a person who needs to know an instructor's name and department, but not the salary. This person should see a relation described, in SQL, by

```
select ID, name, dept_name
from instructor
```

• A view provides a mechanism to hide certain data from the view of certain users.
• Any relation that is not of the conceptual model but is made visible to a user as a “virtual relation” is called a view.

View Definition

• A view is defined using the create view statement which has the form

```
create view v as < query expression >
```

where <query expression> is any legal SQL expression. The view name is represented by v.
• Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
• View definition is not the same as creating a new relation by evaluating the query expression
  • Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.
Example Views

- A view of instructors without their salary
  
  ```sql
  create view faculty as
  select ID, name, dept_name
  from instructor
  ```

- Find all instructors in the Biology department
  
  ```sql
  select name
  from faculty
  where dept_name = 'Biology'
  ```

- Create a view of department salary totals
  
  ```sql
  create view departments_total_salary(dept_name, total_salary) as
  select dept_name, sum(salary)
  from instructor
  group by dept_name;
  ```

Views Defined Using Other Views

- `create view physics_fall_2009 as`
  
  ```sql
  select course.course_id, sec_id, building, room_number
  from course, section
  where course.course_id = section.course_id
    and course.dept_name = 'Physics'
    and section.semester = 'Fall'
    and section.year = '2009';
  ```

- `create view physics_fall_2009_watson as`
  
  ```sql
  select course_id, room_number
  from physics_fall_2009
  where building= 'Watson';
  ```
View Expansion

- Expand use of a view in a query/another view

```sql
create view physics_fall_2009_watson as
 (select course_id, room_number
  from (select course.course_id, building, room_number
         from course, section
         where course.course_id = section.course_id
           and course.dept_name = 'Physics'
           and section.semester = 'Fall'
           and section.year = '2009')
  where building= 'Watson';
```

Views Defined Using Other Views

- A view relation $v_1$ is said to depend directly on a view relation $v_2$ if $v_2$ is used in the expression defining $v_1$.

- A view relation $v_1$ is said to depend on view relation $v_2$ if either $v_1$ depends directly to $v_2$ or there is a path of dependencies from $v_1$ to $v_2$.

- A view relation $v$ is said to be recursive if it depends on itself.
View Expansion

• A way to define the meaning of views defined in terms of other views.
• Let view $v_i$ be defined by an expression $e_i$ that may itself contain uses of view relations.
• View expansion of an expression repeats the following replacement step:

  repeat
  Find any view relation $v_i$ in $e_i$
  Replace the view relation $v_i$ by the expression defining $v_i$
  until no more view relations are present in $e_i$

• As long as the view definitions are not recursive, this loop will terminate.

Update of a View

Add a new tuple to faculty view which we defined earlier

insert into faculty values ('30765', 'Green', 'Music');

This insertion must be represented by the insertion of the tuple

('30765', 'Green', 'Music', null)

into the instructor relation.
Some Updates Cannot be Translated Uniquely

- create view instructor_info as
  select ID, name, building
  from instructor, department
  where instructor.dept_name = department.dept_name;
- insert into instructor_info values ('69987', 'White', 'Taylor');

1. which department, if multiple departments in Taylor?
2. what if no department is in Taylor?
   - Most SQL implementations allow updates only on simple views
   - The from clause has only one database relation.
   - The select clause contains only attribute names of the relation, and does not have any expressions, aggregates, or distinct specification.
   - Any attribute not listed in the select clause can be set to null
   - The query does not have a group by or having clause.

And Some Not at All

- create view history_instructors as
  select *
  from instructor
  where dept_name = 'History';

- Insert ('25566', 'Brown', 'Biology', 100000) into history_instructors
Join Recap

CREATE TABLE one (a int, b int);
CREATE TABLE two (b int, c int);

INSERT INTO one VALUES (1,2);
INSERT INTO one VALUES (2,3);
INSERT INTO two VALUES (1,2);
INSERT INTO two VALUES (2,1);

SELECT * FROM one;
SELECT * FROM two;

SELECT * FROM one,two;
SELECT * FROM one NATURAL JOIN two;
SELECT * FROM one INNER JOIN two on one.b = two.b;
SELECT * FROM one CROSS JOIN two;
SELECT * FROM one CROSS JOIN two on one.b = two.b;
SELECT * FROM one LEFT OUTER JOIN two on one.b = two.b;
SELECT * FROM one FULL OUTER JOIN two on one.b = two.b;

Semi-Join

- Just for completeness, the semi-join....

SELECT ...
FROM outer_tables
WHERE expr IN (SELECT ... FROM inner_tables ...)
AND ...

- Key point: only columns from outer query returned
Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
  - Entity-Relationship Model
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

Relational Algebra Examples

branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, …)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)
Relational Algebra Examples

Relational Algebra

- Procedural language

- Six basic operators
  - select
  - project
  - union
  - set difference
  - Cartesian product
  - rename

- The operators take one or more relations as inputs and give a new relation as a result.
Select Operation

Relation $r$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>α</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>β</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>12</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>23</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

$\sigma_{A=B \land D > 5} (r)$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>α</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>23</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

SQL Equivalent:

```sql
select *
from r
where A = B and D > 5
```

Unfortunate naming confusion

Project

Relation $r$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>α</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>β</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>12</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>23</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

$\Pi_{A,D} (r)$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

SQL Equivalent:

```sql
select distinct A, D
from r
```
Set Union, Difference

Relation r, s

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

r \cup s:  

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

r – s:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
</tr>
<tr>
<td>β</td>
<td>1</td>
</tr>
</tbody>
</table>

Must be compatible schemas

What about intersection?

Can be derived

\[ r \cap s = r - (r - s); \]

SQL Equivalent:

select * from r
union/except/intersect
select * from s;

This is one case where duplicates are removed.

Cartesian Product

Relation r, s

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>γ</td>
<td>10</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

r \times s:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
<td>α</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>β</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>β</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>γ</td>
<td>10</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>α</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>β</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>β</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>γ</td>
<td>10</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

SQL Equivalent:

select distinct * from r, s

Does not remove duplicates.
Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
  
  Example:
  
  \[ \rho_x(E) \]
  
  returns the expression \( E \) under the name \( X \)

If a relational-algebra expression \( E \) has arity \( n \), then

\[ \rho_{X \{A_1, A_2, \ldots, A_n\}}(E) \]

returns the result of expression \( E \) under the name \( X \), and with the attributes renamed to \( A_1, A_2, \ldots, A_n \).

Relational Algebra

- Those are the basic operations

- What about SQL Joins?
  - Compose multiple operators together
    \[ \sigma_{A=C}(r \times s) \]

- Additional Operations
  - Set intersection
  - Natural join
  - Division
  - Assignment
Additional Operators

- Set intersection ($\cap$)
  - $r \cap s = r - (r - s)$;
  - SQL Equivalent: intersect

- Assignment ($\leftarrow$)
  - A convenient way to right complex RA expressions
  - Essentially for creating “temporary” relations
    - $temp1 \leftarrow \Pi_{R-S}(r)$
  - SQL Equivalent: “create table as…”

Additional Operators: Joins

- Natural join ($\bowtie$)
  - A Cartesian product with equality condition on common attributes
  - Example:
    - if $r$ has schema $R(A, B, C, D)$, and if $s$ has schema $S(E, B, D)$
    - Common attributes: $B$ and $D$
    - Then:
      \[
      r \bowtie s = \Pi_{r.A, r.B, r.C, r.D, s.E}(r.B = s.B \land r.D = s.D \land (r \times s))
      \]
  - SQL Equivalent:
    - select $r.A, r.B, r.C, r.D, s.E$ from $r, s$ where $r.B = s.B$ and $r.D = s.D$
    - OR
    - select * from $r$ natural join $s$