ER Examples

- Barber business
- Musical artists

The Relational Algebra

- The relational algebra is a procedural query language
- This means it says what data it wants and how to get it
- It consists of a set of operations that take one or two relations as input and produces a new relation as a result
- The fundamental operations in the relational algebra are *select*, *project*, *union*, *set difference*, *Cartesian product*, and *rename*.
- Others include *set intersection*, *natural join*, and *assignment*.

The Relational Algebra

- Select, project, and rename are called *unary* operations because they operate on one relations
- Union, intersection, set difference, Cartesian product, and natural join operate on pairs of relations and are therefore called *binary* operations.

The Select Operation

- The select operation selects tuples that satisfy a given predicate
- σ
- The predicate appears as a subscript to σ
- The argument **relation** is in parentheses after σ

Say we want the tuples of the following relation where the instructor is in the physics department.

Table 1: Instructor Relation

ID	NAME	DEPT	SALARY
10101	Srinivasan	CS	98822.97
12121	Wu	Finance	92298.75
15151	Mozart	Music	99473.86
22222	Einstein	Physics	98718.90
32343	El Said	History	95757.29
33456	Gold	Phyics	97907.41
45565	Katz	CS	96998.25
58583	Califieri	History	90743.02
76543	Singh	Finance	95514.52
76766	Crick	CS	97777.23
83821	Brandt	BIO	96493.92
98345	Kim	Enig	91779.37

The Select Operation

- We write:
- $\sigma_{DEPT=Physics}(Instructor)$

The Select Operation

Our result is:

Table 2: $\sigma_{DEPT=Physics}(Instructor)$

ID	NAME	DEPT	SALARY
22222	Einstein	Physics	98718.90
33456	Gold	Physics	97907.41

The Project Operator

- The project operator takes all rows *i* from column *j*
- Π
- Again, the predicate appears as a subscript and the argument relation in ()

The Project Operator

- Recall our Instructor relation
- Consider $\Pi_{ID}(Instructor)$
- What is the result?

$\Pi_{ID}(Instructor)$

Table 3: $\Pi_{ID}(Instructor)$

ID
10101
12121
15151
22222
32343
33456
45565
58583
76543
76766
83821
98345

Composition of Relational Operations

- What about a more complicated question?
- "Find the name of all instructors in the Physics department"
- We can compose relational operations together
- Write $\Pi_{name}\left(\sigma_{dept\_name=Physics}\left(instructor\right)\right)$

Composition of Relational Operations

- What did we do here
- $\Pi_{name}\left(\sigma_{dept\_name=Physics}\left(instructor\right)\right)$
- Instead of giving a name of a relation, we used an expression that will create a relation

Composition of Relational Operations

- Since the results of a relational-algebra operation is of the same type (relation) as its inputs, relational algebra operations can be composed into **relational-algebra expressions**!
- Just like composing arithmetic operations (+, -, *, ect).

The Union Operation

- he Union $\cup$ operator takes the union of two tables produced by a query

The Union Operation

petID	petType	owner	dwelling
1	Dog	John	Rent
2	Cat	John	Rent
3	Cat	Sarah	Own
4	Parrot	Michael	Rent
5	Fish	Michael	Rent
6	Fish	Emily	Own

The Union Operation

- Say we want to know all of the cat and fish owners who rent their dwellings
- What would we write to get the names of cat owners who rent?
- $\Pi_{owner}(\sigma_{petType=Cat \wedge dwelling=Rent}(Pets))$
- What would we write to get the fish owners who rent?
- $\Pi_{owner}(\sigma_{petType=Fish \wedge dwelling=Rent}(Pets))$
- The Union operator gives us both sets combined

The Union Operation

- $\Pi_{owner}(\sigma_{petType=Cat \land dwelling=Rent}(Pets)) \cup \Pi_{owner}(\sigma_{petType=Fish \land dwelling=Rent}(Pets))$

The Union Operation

- **Important**!
- Notice we took the union of two sets that had a petID value
- We can only take unions between **compatible relations**
- We need two conditions for the union operation to hold

The Union Operation

- The relations r and s must be of the same arity (same number of attributes)
- domains of the ith attribute of r and ith attribute of s must be the same, for all i
- This is **more restrictive** than set theory

The Set Difference Operation

- $-$ or / like before
- The set difference between A and B (A-B) will give tuples that appear in A and not in B

The Set Difference Operation

Lets look at our pets table again

petID	petType	owner	dwelling
1	Dog	John	Rent
2	Cat	John	Rent
3	Cat	Sarah	Own
4	Parrot	Michael	Rent
5	Fish	Michael	Rent
6	Fish	Emily	Own

How would we write and find owners who have fish but not parrots?

The Set Difference Operation

- $\Pi_{owner}(\sigma_{petType=fish}(Pets)) - \Pi_{owner}(\sigma_{petType=parrots}(Pets))$
- Emily
- IMPORTANT: the same compatibility conditions from union apply

The Cartesian-Product Operation

- $\times$
- Say we have two relations, A and B
- $C = A \times B$
- What tuples appear in C
- Each possible pair of tuples (one from A, and one from B).

The Cartesian-Product Operation

- Say A has n_1 tuples and B n_2 tuples
- C will have $n_1 * n_2$ tuples
- Note it may be the case $t[ID_A] \neq t[ID_b]$
- What tuples appear in C
- Still each possible pair of tuples (one from A, and one from B).

The Cartesian-Product Operation

Imagine we had another table on top of our pet table that represented how different pettypes needed to be kept (called PetTraits)

petType	liveType
Dog	Land
Cat	Land
Fish	Water

The Cartesian-Product Operation

The Cartesian product of *Pets* and *PetTraits* is

petID	petType	owner	dwelling	petType	liveType
1	Dog	John	Rent	Dog	Land
1	Dog	John	Rent	Cat	Land
1	Dog	John	Rent	Fish	Water
2	Cat	John	Rent	Dog	Land
2	Cat	John	Rent	Cat	Land
2	Cat	John	Rent	Fish	Water
3	Fish	Sarah	Own	Dog	Land
3	Fish	Sarah	Own	Cat	Land
3	Fish	Sarah	Own	Fish	Water
4	Dog	Michael	Rent	Dog	Land
4	Dog	Michael	Rent	Cat	Land
4	Dog	Michael	Rent	Fish	Water
5	Cat	Michael	Rent	Dog	Land
5	Cat	Michael	Rent	Cat	Land
5	Cat	Michael	Rent	Fish	Water

The Cartesian-Product Operation

- Notice anything off about this table?
- It contains tuples that don't exist in reality
- How do we deal with this?

The Cartesian-Product Operation

- Naming schema:
- If an attribute name in r and s match, attach the name of the relation to the attribute(s) with duplicates
- petID, petType, owner, dwelling, petType, liveType
- Changes to: petID, Pets.petType, owner, dwelling, PetTraits.petType liveType

The Cartesian-Product Operation

- We have tuples for which $t[Pets.petType] \neq t[PetTraits.petType]$
- At the same time, there are tuples that match!

The Cartesian-Product Operation

- We have tuples for which $t[\mathit{Pets.petType}] \neq t[\mathit{PetTraits.petType}]$
- At the same time, there are tuples that match!

The Cartesian-Product Operation

- We have tuples for which $t[Pets.petType] \neq t[PetTraits.petType]$
- At the same time, there are tuples that match!
- Lets look closer

The Cartesian-Product Operation

petID	Pets.petType	owner	dwelling	PetTraits.petType	liveType
1	**Dog**	John	Rent	**Dog**	Land
1	Dog	John	Rent	Cat	Land
1	Dog	John	Rent	Fish	Water
2	Cat	John	Rent	Dog	Land
2	**Cat**	John	Rent	**Cat**	Land
2	Cat	John	Rent	Fish	Water
3	Fish	Sarah	Own	Dog	Land
3	Fish	Sarah	Own	Cat	Land
3	**Fish**	Sarah	Own	**Fish**	Water
4	Dog	Michael	Rent	Dog	Land
4	Dog	Michael	Rent	Cat	Land
4	Dog	Michael	Rent	Fish	Water
5	Cat	Michael	Rent	Dog	Land
5	**Cat**	Michael	Rent	**Cat**	Land
5	Cat	Michael	Rent	Fish	Water

The Cartesian Product Operator

- We have instances where $Pets.petType$ matches $PetTraits.petType$
- We can use the select operator to refine our table to get rid of the pesky errors
- $\sigma_{Pets.petType=PetTraits.petType}(Pets \times PetTraits)$

The Cartesian Product Operator

petID	petType	owner	dwelling	petType	liveType
1	Dog	John	Rent	Dog	Land
2	Cat	John	Rent	Cat	Land
3	Fish	Sarah	Own	Fish	Water
4	Dog	Michael	Rent	Dog	Land
5	Cat	Michael	Rent	Cat	Land
6	Fish	Emily	Own	Fish	Water

The Rename Operator

- We may, at times, want to rename attributes in a table
- For example, upon combining tables, we may worry that we will produce duplicate attribute names
- This could create ambiguity
- Forbidden

The Rename Operator

- For a relational-algebra expression E, we may use ρ for renaming
- $\rho_x(E)$
- returns the results of E under the name x

The Rename Operator: Example 1

- Assume E has arity n.
- Perform $\rho_{x(A_1,A_2,...A_n)}(E)$
- Attributes in E are now $A_1, A_2, ...A_n$

The Rename Operator: Example 2

- Consider the following query
- “What is the highest salary in the University”

The Rename Operator: Example 2

Table 4: Instructor Relation

ID	NAME	DEPT	SALARY
10101	Srinivasan	CS	98822.97
12121	Wu	Finance	92298.75
15151	Mozart	Music	99473.86
22222	Einstein	Physics	98718.90
32343	El Said	History	95757.29
33456	Gold	Phyics	97907.41
45565	Katz	CS	96998.25
58583	Califieri	History	90743.02
76543	Singh	Finance	95514.52
76766	Crick	CS	97777.23
83821	Brandt	BIO	96493.92
98345	Kim	Enig	91779.37

The Rename Operator: Example 2

- One strategy here is as follows:
  - Combine the instructor table with itself via a Cartesian product
  - This will produce a table with 144 tuples and 8 attributes (duplicates)
  - We can then compare the two salaries in any tuple, but can only do so of the columns are unambiguous.

The Rename Operator: Example 2

- We use the rename operator to remove ambiguity
-

$$\Pi_{\text{salary}}\left(\sigma_{\text{salary} < b.\text{salary}}(\textit{instructor} \times \rho_b(\textit{instructor}))\right)$$

The Rename Operator: Example 2

- We use the rename operator to remove ambiguity
- $$\Pi_{\text{salary}}\left(\sigma_{\text{salary}<b.\text{salary}}(\textit{instructor} \times \rho_b(\textit{instructor}))\right)$$

The Rename Operator: Example 2

- We use the rename operator to remove ambiguity
-
$$\Pi_{\text{salary}}\left(\sigma_{\text{salary}<b.\text{salary}}(\mathit{instructor} \times \rho_b(\mathit{instructor}))\right)$$
- This query will select all tuples for which salary is less than the highest salary

The Rename Operator: Example 2

- Of course, this will give us the 11 instructors who earn LESS than the highest
- What can we do?
- Set subtraction!
- $$\Pi_{salary}(instructor) - \Pi_{\text{salary}}\left(\sigma_{\text{salary}<b.\text{salary}}(instructor \times \rho_b(instructor))\right)$$
- The result is the set of all instructors minus the set of instructors earning less than the highest, which will give us the highest salary.

The Set Intersection Operation

- Consider two entities A and B
- Set intersection is $A \cap B$
- Same restrictions on unions apply!

The Set Intersection Operation

- A useful equivalence result for set intersection comes from set difference
- $A \cap B = A - (A - B)$
- Set intersection is most often used as a shortcut for the above calculation

The Natural Join Operation

- We were able to combine two relations with a Cartesian product before
- But...when we did it...it was kind of taxing!
- This is common operation, calls for a short cut

The Natural Join Operation

- $\bowtie$
- Natural join forms of a Cartesian product, performs a selection forcing equality on those attributes that appear in both relation schemas, and removes duplicate attributes

The Natural Join Operation

- Consider two relations r(R) and s(S). The natural join of r and s, denoted by $r \bowtie s$ is a schema on $R \cup S$ defined as
- $r \bowtie s = \Pi_{R \cup S}(\sigma_{r.A_1=s.A1 \wedge r.A_2=s.A_2 \wedge ... r.A_n=s.A_n}(r \times s)$ where $R \cap S = \{A_1, A_2, ...A_n\}$
- Further note that if $R \cap S = \emptyset$ then $r \bowtie s = r \times s$

The Natural Join Operation

Consider three different relations

- Course(course_id, students)
- Instructor (ID, name, dept_name, salary)
- Teaches (ID, course_id, sec_id, semester, year)

The Natural Join Operation

Consider three different relations

- Course(course_id, students)
- Instructor (ID, name, dept_name, salary)
- Teaches (ID, course_id, sec_id, semester, year)
- $(instructor \bowtie teaches) \bowtie course = instructor \bowtie (teaches \bowtie course)$
- The Natural join is associative

The θ Join

- The θ join is a variant of the natural join that allows us to combine a selection and a Cartesian product into a single operation
- Consider relations r(R) and s(S), and let θ be a predicate on the attributes in the schema $R \cup S$.
- The θ join operation $r \bowtie_\theta s = \sigma_\theta(r \times s)$

Example of Theta Join

Let's consider two tables:

Students

StudentID	Name	Age
1	Alice	20
2	Bob	22
3	Carol	21

Courses

CourseID	CourseName	Instructor
101	Math 101	John Doe
102	English 101	Jane Smith
103	History 101	John Doe

Example of Theta Join (Contd.)

We want to find students who are enrolled in courses taught by the instructor with the name 'John Doe.'

$$\text{Result} = \text{Students} \bowtie_{\text{Instructor}=\text{'John Doe'}} \text{Courses}$$

The result would include students enrolled in courses taught by 'John Doe.'

Result

StudentID	Name	Age	CourseID	CourseName
1	Alice	20	101	Math 101
3	Carol	21	103	History 101

Handling Null Values

- There are situations where some information doesn't exist or may not exist yet
- NULL value: an as yet unknown data value within a table column

Handling Null Values

- There are situations where some information doesn't exist or may not exist yet
- NULL value: an as yet unknown data value within a table column

Handling Null Values

- NULL values also create a new information content of $\mathbb{1}(NULL = TRUE)$
- We therefore have to leave behind binary logic in which any statement is either true or false

Handling Null values

- Lets look at an example: we want to know the set of employees that live in Kentucky or who do not live in Kentucky

E#	Name	Street	City
1	John Doe	123 Main St	New York
2	Jane Smith	456 Elm St	NULL
3	Bob Johnson	789 Oak St	Los Angeles
4	Alice Brown	101 Pine St	NULL

Handling Null values

$$\Pi_{Name}\left(\sigma_{City=Lexington}\right) \cup \Pi_{Name}\left(\sigma_{City \neq Lexington}\right)$$

E#	Name	Street	City
1	John Doe	123 Main St	New York
3	Bob Johnson	789 Oak St	Los Angeles

Handling Null values

- This defies the conventional logic that the union of employees who live in Lexington with its complement (the employs who do not live in Lexington) should be the complete set of employees
- Sentential logic with the values of TRUE, FALSE, and UKNOWN is commonly called three valued logic
- Since this practically makes interpretation of results difficult, we avoid these values when we can

Handling Null values

- Sometimes a default is used
- In SQL, we would use the function COALESE(X,Y), which will replace unknown X with Y

Extended Relational-Algebra Operations

Outer Join Operations

- The outer-join operation is an extension of the join operation to deal with missing information
- Consider the following relational schema
- Instructor(ID, Name, dept_name)
- Teaches(ID, course_id, sec_id, semester, year)
- Lets say not every teacher has a course

Gold On Sabbatical

ID	CourseID	Semester	year
10101	CS101	fall	2010
10101	CS315	fall	2010
10101	CS347	spring	2009
12121	Fin201	spring	2009

ID	NAME	DEPT	SALARY
10101	Srinivasan	CS	90542.86
12121	Wu	Finance	95024.65
15151	Gold	Music	97595.49

Outer Join Operations

- We will lose information if we attempt to join, because Gold isn't teaching these courses

Outer Join Operations

- We will lose information if we attempt to join, because Gold isn't teaching these courses
- Outer joins will allow us to preserve tuples that would be lost in a join by creating tuples in the result with null values

Outer Join Operations

- We will lose information if we attempt to join, because Gold isn't teaching these courses
- Outer joins will allow us to preserve tuples that would be lost in a join by creating tuples in the result with null values
- Similar to a natural join

Left/Right Join Operations

- Left outer join ($⟕$)

Left/Right Join Operations

- Left outer join ($⟕$)
- Right outer join ($⟖$)

Left/Right Join Operations

- Left outer join ($⟕$)
- Right outer join ($⟖$)
- Right is symmetric to left

Left/Right Join Operations

- R ⟕ S
- Take all tuples in the left relation (R) that do not match with any tuple in the right relation (S), and pad them null values

Left/Right Join Operations

- R ⟗ S
- Take all tuples in the left relation (R) that do not match with any tuple in the right relation (S), and pad them null values
- Should there be any tuples from S that do not match with R, pad with null values

Left/Right Join Operations

Instructor ⟕ Teaches

ID	NAME	DEPT	SALARY	CourseID	Semester	year
10101	Srinivasan	CS	90542.86	CS101	fall	2010
10101	Srinivasan	CS	90542.86	CS315	fall	2010
10101	Srinivasan	CS	90542.86	CS347	spring	2009
12121	Wu	Finance	95024.65	Fin201	spring	2009
15151	Gold	Music	97595.49			

Full Outer Join Operations

- Full outer join ($⟗$) does both a right and left outer join, padding tuples from both ends

Full Outer Join Operations

- It is interesting to note that outerjoin operations can be expressed as basic relational algebra operations

Full Outer Join Operations

- It is interesting to note that outerjoin operations can be expressed as basic relational algebra operations
- $$(r \bowtie s) \cup (r - \Pi_R(r \bowtie s)) \times \{(null, ...null)\}$$

Generalized Projection $\Pi_{F_1,F_2,\ldots,F_n}(E)$

- We may generalize the projection operator by allowing for operations such as arithmetic and string functions to be applied

Generalized Projection $\Pi_{F_1,F_2,\ldots,F_n}(E)$

- We may generalize the projection operator by allowing for operations such as arithmetic and string functions to be applied
- $\Pi_{F_1,F_2,\ldots,F_n}(E)$ (where F_i are functions)

Generalized Projection $\Pi_{F_1,F_2,\ldots,F_n}(E)$

- We may generalize the projection operator by allowing for operations such as arithmetic and string functions to be applied
- $\Pi_{F_1,F_2,\ldots,F_n}(E)$ (where F_i are functions)
- $\Pi_{salary*\frac{1}{12}}$

Aggregation $\mathcal{G}$

- Aggregation functions ($\mathcal{G}$) take a collection of values and return a single value as a result

Aggregation $\mathcal{G}$

- Aggregation functions ($\mathcal{G}$) take a collection of values and return a single value as a result
- count, max, sum, average, min

Aggregation $\mathcal{G}$

- Aggregation functions ($\mathcal{G}$) take a collection of values and return a single value as a result
- count, max, sum, average, min
- $\mathcal{G}$**sum(salary)**(*instructor*)

Aggregation $\mathcal{G}$

- Sometimes we may wish to only factor in distinct observations

Aggregation $\mathcal{G}$

- Sometimes we may wish to only factor in distinct observations
- **count-distinct**ID

Aggregation $\mathcal{G}$

- Sometimes we may wish to only factor in distinct observations
- **count-distinct**ID
- Other times we may wish to apply aggregation to a group of sets of tuples

Aggregation $\mathcal{G}$

- Sometimes we may wish to only factor in distinct observations
- **count-distinct**ID
- Other times we may wish to apply aggregation to a group of sets of tuples
- dept_name $\mathcal{G}_{\textbf{average}(\textit{salary})}(\textit{instructor})$

Aggregation $\mathcal{G}$

- A more general expression is

Aggregation $\mathcal{G}$

- A more general expression is
-
$$_{G_1, G_2, \ldots G_n}\mathcal{G}_{F_1(A_1), F_2(A_2), \ldots F_m(A_m)}(E)$$

Aggregation $\mathcal{G}$

- A more general expression is
- $$_{G_1,G_2,\ldots G_n}\mathcal{G}_{F_1(A_1),F_2(A_2),\ldots F_m(A_m)}(E)$$
- The collection $\{G_1, G_2, \ldots G_n\} \in G$ is a list of grouping attributes (department name)

Aggregation $\mathcal{G}$

- A more general expression is
-
$$_{G_1, G_2, \ldots G_n}\mathcal{G}_{F_1(A_1), F_2(A_2), \ldots F_m(A_m)}(E)$$
- The collection $\{G_1, G_2, \ldots G_n\} \in G$ is a list of grouping attributes (department name)
- Each F_i is an aggregation function

Aggregation $\mathcal{G}$

- A more general expression is
- $$_{G_1, G_2, \ldots G_n}\mathcal{G}_{F_1(A_1), F_2(A_2), \ldots F_m(A_m)}(E)$$
- The collection $\{G_1, G_2, \ldots G_n\} \in G$ is a list of grouping attributes (department name)
- Each F_i is an aggregation function
- Each A_i is an attribute name

The Assignment Operator

- Step1 $\leftarrow R \times S$

The Assignment Operator

- Step1 $\leftarrow R \times S$
- Step2 $\leftarrow \sigma_{r.A_1=s.A1 \wedge r.A_2=s.A_2 \wedge \ldots r.A_n=s.A_n}(Step1)$

The Assignment Operator

- Step1 $\leftarrow R \times S$
- Step2 $\leftarrow \sigma_{r.A_1=s.A1 \wedge r.A_2=s.A_2 \wedge \ldots r.A_n=s.A_n}(Step1)$
- result $\leftarrow \Pi_{R \cup S}(Step2)$

The Assignment Operator

- Step1 $\leftarrow R \times S$
- Step2 $\leftarrow \sigma_{r.A_1=s.A1 \wedge r.A_2=s.A_2 \wedge \ldots r.A_n=s.A_n}(Step1)$
- result $\leftarrow \Pi_{R \cup S}(Step2)$
- We rewrote the definition of $\bowtie$ in steps using the assignment operator

How Do We Approach Writing Good Queries?

- Break it into smaller questions
- Think about the sequence of how things must happen
- The very first steps are the ones that are going to be buried the deepest in parentheses

Relational Algebra: Recipes with Meat Products

- **Given:**
  - $\mathit{FoodItems}(\text{item}, \text{type}, \text{calories})$
  - $\mathit{Ingredients}(\text{fooditem}, \text{recipe}, \text{ounces})$
  - $\mathit{stock}(\text{item}, \text{stock})$
- **Task:** Find names of all recipes that contain meat products (food items of type "Meat").
- $\Pi_{\text{recipe}}(\sigma_{\text{type='Meat'}}(\mathit{FoodItems}) \bowtie_{\mathit{fooditem}=\mathit{item}} \mathit{Ingredients})$