Recap

- Data is central to our lives
- Data-savviness is the future!
  - Classical database systems: today's focus!
  - Non-traditional database systems and the data life-cycle
- Notion of a DBMS
- Requirements for a DBMS
- Example DBMSs, and their complexity
- **Next**: the relational data model

Fundamentals: The Relational Data Model

- A DBMS stores many databases
- Each database has many relations
- A relation = a set of tuples, each w/ a predefined set of attributes

Table/Relation (an instance of)

Attributes/Columns

Tuples/ Rows/ Records

Name	Price	Category	Manufacturer
Gizmo	$19.99	gadgets	GizmoWorks
Power Gizmo	$29.99	gadgets	GizmoWorks
SnapPea Camera	$149.99	photography	Canon
Hi Phone	$700.99	phone	Snapple

Relational Schema & Instance

- **Schema**: The structure, format, or scaffolding
  - Schema for a Relation:
    - Relation names plus attribute names & domains
    - Product (Name String, Price Float, Category String, Manuf. String)
  - Schema for a Database:
    - Schemas for many relations
    - Product (…)
    - Vendor (Name String, Address String, Phone Integer)
- **Instance**: Specific instantiation of the Relation or Database
  - A Relation or Database with "values filled in"

Name	Price	Category	Manuf.
Hi Phone	$700.99	phone	Snapple
SnapPea Camera	$149.99	photography	Canon

Relational Schema & Instance

- **Schema**: The structure, format, or scaffolding
  - Schema for a Relation:
    - Relation names plus attribute names & domains
    - Product (Name String, Price Float, Category String, Manuf. String)
  - Schema for a Database:
    - Schemas for many relations
    - Product (…)
    - Vendor (Name String, Address String, Phone Integer)
- **Instance**: Specific instantiation of the Relation or Database
  - A Relation or Database with "values filled in"
- **Schema = Metadata**
- **Instance = Data**

Name	Price	Category	Manuf.
Hi Phone	$700.99	phone	Snapple
SnapPea Camera	$149.99	photography	Canon

Metadata

Data

Recap: The Relational Data Model

- Data is stored in relations
- Relations are defined by a set of attributes/columns w/ specific domains
  - Together this defines the "schema" of a relation
- At any time, relations comprise of a set of tuples/rows/records
  - This constitutes the "instance" of a relation
- Databases are collections of relations

- Today, we will define the Relational Algebra: the operations that manipulate relations (the operands)
  - Parallels to linear algebra or set algebra

Set Operations: Union and Difference

- Union & Difference
  - Binary operations from set algebra
  - Recall that relations are sets
- Union
- Notation: $R1 \cup R2$
  - R1 and R2 must have same schema
- Output schema: same as input
- Example: CurrentStudents (Name, Addr) U Alumni (Name, Addr)
- Difference is similar $R1 - R2$: tuples in R1 and not in R2

Unary Op.: Selection (Cross out rows)

- Returns all tuples that satisfy a condition (also called filter)
- Notation $\sigma_c(R)$
  - Condition c can involve attributes in R with =, >, <, AND, OR, etc.
  - Also called predicate
- Output schema: same as input
- Find all employees with salary greater than $40,000 from an Employee relation

Unary Op.: Selection (Cross out rows)

- Returns all tuples that satisfy a condition (also called filter)
- Notation $\sigma_c(R)$
  - Condition c can involve attributes in R with =, >, <, AND, OR, etc.
  - Also called predicate
- Output schema: same as input
- Find all employees with salary greater than $40,000 from an Employee relation

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

Another Selection Example

SSN	Name	DepID	Salary
839930303	Avi	1	100000
182902033	Sarah	2	90000
383930032	Farida	2	120000
123233849	Jacob	1	70000

- Find employees with Salary greater than 100000 and Department ID 2.

Another Selection Example

SSN	Name	DepID	Salary
~~839930303~~	~~Avi~~	~~1~~	~~100000~~
~~182902033~~	~~Sarah~~	~~2~~	~~90000~~
383930032	**Farida**	**2**	**120000**
~~123233849~~	~~Jacob~~	~~1~~	~~70000~~

- Find employees with Salary greater than 100000 and Department ID 2.

$$\sigma_{Salary>100000\ AND\ DepID=2}(Employee)$$

Unary Op.: Projection (Cross Out Columns)

- Notation $\Pi_{A_1,A_2,\ldots,A_n}(R)$
- Where $R(B_1, B_2, \ldots, B_m)$

$$A_1, A_2, \ldots, A_n \subseteq B_1, B_2, \ldots, B_m$$

- That is, you can only sub select from columns that are present in R
- Output Schema $(A_1, A_2, \ldots, A_m)$
- Example: project out only the names from the Employee (SSN, Name, DepID, Salary)

Unary Op.: Projection (Cross Out Columns)

- Notation $\Pi_{A_1,A_2,\ldots,A_n}(R)$
- Where $R(B_1, B_2, \ldots, B_m)$

$$A_1, A_2, \ldots, A_n \subseteq B_1, B_2, \ldots, B_m$$

- That is, you can only sub select from columns that are present in R
- Output Schema $(A_1, A_2, \ldots, A_m)$
- Example: project out only the names from the Employee (SSN, Name, DepID, Salary)

$$\Pi_{Name}(Employee)$$

- Note: **Deletes any duplicate rows to return a set!**

Another Projection (+ Selection) Example

SSN	Name	DepID	Salary
839930303	Avi	1	100000
182902033	Sarah	2	90000
383930032	Farida	2	120000
123233849	Jacob	1	70000

- Find names and salaries of Employees whose Department ID is 2.

Another Projection (+ Selection) Example

SSN	Name	DepID	Salary
839930303	Avi	1	100000
182902033	Sarah	2	90000
383930032	Farida	2	120000
123233849	Jacob	1	70000

Name	Salary
Sarah	90000
Farida	120000

- Find names and salaries of Employees whose Department ID is 2.

$$\Pi_{Name,Salary}(\sigma_{DepID=2}(Employee))$$

Cartesian (or Cross) Product

- Notation $R_1 \times R_2$
- Where $R_1(A_1, A_2, \ldots, A_n)$

  $R_2(B_1, B_2, \ldots, B_m)$
- Output Schema $S(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$
- Meaning: associate each tuple on LHS with RHS

- One nuance:

  If $A_i = B_j$, then rename as $R_1 . A_i$ and $R_2 . B_j$

SSN	Name	DepID	Salary
839930303	Avi	1	100000
182902033	Sarah	2	90000
383930032	Farida	2	120000

Name	Title
Sarah	Manager
Farida	Programmer

SSN	Name	DepID	Salary
839930303	Avi	1	100000
182902033	Sarah	2	90000
383930032	Farida	2	120000

Name	Title
Sarah	Manager
Farida	Programmer

SSN	R. Name	DepID	Salary	S.Name	Title
839930303	Avi	1	100000	Sarah	Manager
182902033	Sarah	2	90000	Sarah	Manager
383930032	Farida	2	120000	Sarah	Manager
839930303	Avi	1	100000	Farida	Programmer
182902033	Sarah	2	90000	Farida	Programmer
383930032	Farida	2	120000	Farida	Programmer

Renaming

- Does not change the data/instance, only the metadata/schema (the relation and attribute names)
- Notation $\rho_{S(A_1,A_2,\ldots,A_n)}(R)$
- Input schema $R(B_1, B_2, \ldots, B_n)$
- Output schema $S(A_1, A_2, \ldots, A_n)$
- Example:

  rename $Emp(Name, Comp)$ using $\rho_{E(N,C)}(Emp)$

Example

rename $Emp(Name, Comp)$ using $\rho_{E(N,C)}(Emp)$

Emp

Name	Comp
Sarah	90000
Farida	120000

E

N	C
Sarah	90000
Farida	120000

Recap: Relational Algebra Operations

- Set ops. — union, difference (expand/shrink vertically)
- Selection (cross out rows)
- Projection (cross out columns)
- Cartesian product (expand horizontally and vertically)
- Renaming (change schema)

Derived Operations

- Set operations: can derive intersection from union and difference (Exercise!)
- Joins!
  - Special case of cartesian product and selection that appears frequently in queries

Theta Join

- A Cartesian product followed by a selection
- Notation

$$R_1 \bowtie_\theta R_2 = \sigma_\theta(R_1 \times R_2)$$

- Input Schemas $R_1(A_1, A_2, \ldots, A_n)$

$R_2(B_1, B_2, \ldots, B_m)$

- Output Schema

$$S(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$$

- Perform the cross-product, remove tuples that don't satisfy θ
- Special case θ is an equality condition: Equi join
- Like cross-product use prefixes to distinguish common attribs

SSN	Name	DepID	Salary
839930303	Avi	1	100000
182902033	Sarah	2	90000
383930032	Farida	2	120000

Name	Title
Sarah	Manager
Farida	Programmer

$$R \bowtie_{R.Name=S.Name} S$$

SSN	Name	DepID	Salary
839930303	Avi	1	100000
182902033	Sarah	2	90000
383930032	Farida	2	120000

Name	Title
Sarah	Manager
Farida	Programmer

$$R \bowtie_{R.Name=S.Name} S$$

SSN	R.Name	DepID	Salary	S.Name	Title
182902033	Sarah	2	90000	Sarah	Manager
383930032	Farida	2	120000	Farida	Programmer

SSN	Name	DepID	Salary
839930303	Avi	1	100000
182902033	Sarah	2	90000
383930032	Farida	2	120000

Name	Title
Sarah	Manager
Farida	Programmer

$$R \bowtie_{R.Name=S.Name} S$$

SSN	R.Name	DepID	Salary	S.Name	Title
182902033	Sarah	2	90000	Sarah	Manager
383930032	Farida	2	120000	Farida	Programmer

In many cases, we are selecting on the same attributes having the same values, and we don't need two copies of those attributes

SSN	Name	DepID	Salary
839930303	Avi	1	100000
182902033	Sarah	2	90000
383930032	Farida	2	120000

Name	Title
Sarah	Manager
Farida	Programmer

Natural Join $R \bowtie S$

SSN	Name	DepID	Salary	Title
182902033	Sarah	2	90000	Manager
383930032	Farida	2	120000	Programmer

Natural Join

- Notation $R_1 \bowtie R_2$
- Input Schemas $R_1(A_1, A_2, \ldots, A_n)$ $R_2(B_1, B_2, \ldots, B_m)$
- Output Schema

$$S(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$$

- But with duplicate attributes removed
- Sequence:
  - cross product
  - match tuples based on values of shared attributes
  - remove duplicate attributes

Exercise: Express Natural Join using other operators

- R (A, B), S (B, C, D)

Exercise: Express Natural Join using other operators

- R (A, B), S (B, C, D)

$$R \bowtie S = \Pi_{A,B,C,D}(\sigma_{R.B=S.B}(R \times S))$$

Other Exercises (for home!)

- Say R (A, B), S (A, B) are to be natural joined. What would the join evaluate to?

- Say R (A, B), S (C, D) are to be natural joined. What would the join evaluate to?

Recap: Relational Algebra Operations

- Set ops. — union, difference (expand/shrink vertically)
- Selection (cross out rows)
- Projection (cross out columns)
- Cartesian product (expand horizontally and vertically)
- Renaming (change schema)

- Intersection
- Theta join and Natural join (match and "join" tuples across tables)

Exercise

- Given Products (ProductName, Category) and Store (StoreName, Zip, ProductName, Price), find the stores that are in 94705 or sell books (a category) for less than $10.

Exercise

- Given Products (ProductName, Category) and Store (StoreName, Zip, ProductName, Price), find the stores that are in 94705 or sell books (a category) for less than $10.

$$\Pi_{StoreName}\sigma_{((Category='books' \text{ AND } Price<10) \text{ OR } Zip=94705)}(Stores \bowtie Products)$$

Exercise

- Find Stores that sell two different books for the same price from Store (StoreName, Zip, ProductName, Price)

Exercise

- Find Stores that sell two different books for the same price from Store (StoreName, Zip, ProductName, Price)

$$\Pi_{StoreName}(\sigma_{ProductName!=PN2}(Store \bowtie \rho_{StoreName,Zip,PN2,Price}(Store)))$$

Recap: Relational Algebra Operations

- Set ops. — union, difference (expand/shrink vertically)
- Selection (cross out rows)
- Projection (cross out columns)
- Cartesian product (expand horizontally and vertically)
- Renaming (change schema)

- Intersection
- Theta join and Natural join (match and "join" tuples across tables)

What was the point?

- Relational algebra forms the basics of data processing, i.e., "querying" your data
- As you can see, a small # of operators lets you do a LOT!
- Relational databases are built on relational algebra
- Understanding and being able to effectively use relational algebra sets you up for
  - being able to query your data
  - quantifying the capabilities of data systems

Recap

- Data is central to our lives
- Data-savviness is the future!
  - Classical database systems: today's focus!
  - Non-traditional database systems and the data life-cycle
- Notion of a DBMS
- Requirements for a DBMS
- Example DBMSs, and their complexity
- The relational data model and algebra

Sets vs. Bags

- So far, we have focused on set-oriented relational algebra
- There is also an analogous "bag"-oriented relational algebra
  - Where you can have multiple copies of each tuple
  - Multi-set instead of a set
- Relational databases actually use bags instead of sets for most operations
  - Easier to implement, no need to constantly check for multiple copies

Operations on Bags

- Selection: preserve # of occurrences
  - Roughly same speed as sets
- Projection: preserve # of occurrences
  - **Faster than sets**, no elimination of duplicates
  - Projection is a very common operation, so we want this to be fast
- Cartesian product, join: preserve # of occurrences
  - Every copy "matches" with every copy
  - Roughly same speed as sets

Operations on Bags

- Union {a, b, b, c} U {a, b, c, d} = {a, a, b, b, b, c, c, d}
  - Add number of occurrences
  - **Faster than sets**, no need to look for duplicate tuples
- Difference {a, a, a, b, c} — {a, b, b} = {a, a, c}
  - Subtract number of occurrences
  - Similar speed
  - Intersection is similar