CSc 460
Database Design

Fall 2014

Richard Snodgrass

Why Use a Database?

CSc 460 Database Design

Fall 2014

Acknowledgements

- These slides were written by Richard T. Snodgrass (University of Arizona) and Christian S. Jensen (Aalborg University).
- Michael Soo (amazon.com) provided some of the query processing slides.
- Kristian Torp (Aalborg University) converted the slides from Island Presents to Powerpoint.
- Curtis Dyreson (Washington State University) merged the slides with those from Peter Stewart (Bond University).
- The motivational example was given by Gary Lindstrom and Wei Tao (University of Utah).

Outline

- Why Use a Database?
- The Field
- Overview of the Course

Prevalence of Databases

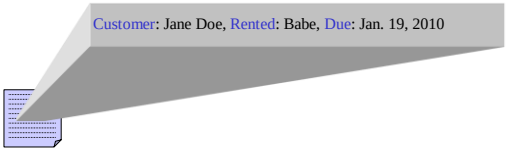
- Behind every successful website, there is a powerful database.
- Examples:
  - Amazon's website
  - American Airlinesz
  - Dell's ordering system
  - UPS / FedEx tracking
  - Wal-Mart's inventory system

Data Management Example

- Scenario
  - You are in a movie web startup.
  - Your customers rent DVD copies of movies.
  - Several copies of each movie.
- Needs
  - Which movies have a customer rented?
  - Are any disks overdue?
  - When will a disk become available?

Solution: A File-based System

- Edit `rented.txt` file



- Advantages
  - Text editors are easy to use
  - Simple to insert a record
  - Simple to delete a record

Complication: Queries

- Does not address needs
  - Query: What movies has Joe Jenkins rented?
  - Execute (not quite right): Search for 'Joe Jenkins'.
  - Execute: Search for
    ```
    ^\s+Customer:\s*Joe\s+Jenkins\s*,\s+Rented:
    ```
  - Query: Are any disks overdue?
  - Execute: ???
- Requirements
  - Robust, sophisticated query language
  - Clear separation between data organization (schema) and data

DBMS Concepts
Schema
DML
SQL

Complication: Integrity

- Lacks data *integrity*, *consistency*
  - Clerk misspells value/field
    ```
    Customer: Jane Doek, Rented: Eraserhead, Deu: Jan. 19, 2010
    ```
  - Inputs improper value, same value differently
    ```
    Customer: Jane Doe, Rented: The Eraserhead, Due: Feb. 29, 2010
    ```
  - Forgets/adds/reorders field
    ```
    Terms: weekly special Due: Jan. 19, 2010, Rented: Eraserhead
    ```
- Requirements
  - Enforce *constraints* to permit only *valid* information to be input.

DBMS Concepts
Integrity constraints
Types

Complication: Update

- Add/delete/update fields in every record
  - Record store location.
    ```
    Customer: Jane Doe, Rented: Babe, Due: Jan. 19, 2010, Store: Robina
    ```
  - Modify customer to first and surname.
    ```
    First: Jane, Surname: Doe, Rented: Eraserhead, Due: Jan. 19, 2010
    ```
- Add/delete/update new information collections
  - `customer.txt` file to record information
    ```
    Customer: Jane Doe, Phone: 557-3344
    ```
- Requirements
  - Ability to manipulate the way data is organized.

DBMS Concepts
DDL

Complication: Multiple Users

- Two clerks edit `rented.txt` file at the same time.
  1) Ben starts to edit `rented.txt`, reads it into memory.
  2) Sarah starts to edit `rented.txt`.
  3) Ben adds a record.
  4) Ben saves `rented.txt` to disk.
  5) Sarah saves `rented.txt` to disk.

  Ben's added record disappears!
- Requirements
  - Must support multiple readers and writers.
  - Updates to data must (appear to) occur in *serial* order.

DBMS Concepts
Serializability
Concurrency control

Why Use a Database? M-10

Complication: Crashes

- Crash during update may lead to inconsistent state.
  - Ben makes 250 of 500 edits to change Jane Doe to her preferred name Jan Doe.
  - Before he saves it, Windows crashes!
- Requirements
  - Must update on all-or-none basis.
  - Implemented by *commit* or *rollback* if necessary.

DBMS Concepts
Transactions
Commit
Rollback
Recovery

Why Use a Database? M-11

Complication: Data in Separate Files

- Need
  - Want to advise Austin Power's fans about new Austin Power's movie.
- Method
  - `customer.txt` contains addresses of customers.
  - Must merge with `rented.txt` to create mailing list.
- Problems
  - Text editors incapable of such a merge (have to write a program)
  - Several Joe Jenkins
  - No information on some customers!?
- Requirements
  - Uniquely identify each customer.
  - Make sure we have information on customers that rent discs.

DBMS Concepts
Joins
Keys
Foreign keys
Referential integrity

Why Use a Database? M-12

Complication: Security

- Customers want to know how many times a movie has been rented.
  - Provide access to `rented.txt`, but not to customer field. How do I do that in an editor?
- Underage clerks should not see history of R-rated rentals.
  - Keep two lists of rentals?
- Requirements
  - Ability to control who has access to what information.

DBMS Concepts
Security
Views

Complication: Efficiency

- Your customer list grows virally.
  - `rented.txt` file gets huge (gigabytes of data).
  - Slow to edit.
  - Slow to query for customer information.
- Requirements
  - New data structures to improve query performance.
  - System automatically modifies queries to improve speed.
  - Ability of system to scale to handle huge datasets.

DBMS Concepts
Indexes
Query optimization
Database tuning

Complication: New Needs

- Your customer list grows virally.
  - What pairs of movies are often rented together?
    Calculate probability of movie combinations.
  - Do we need more copies of the Austin Powers movie anywhere?
    Plot rental history of Austin Powers by city.
- Requirements
  - Collect and analyse summary data.
  - Use computer to *mine* for interesting trends.
  - Support access to data by sophisticated programs.

DBMS Concepts
Data warehouses
Data mining
Database API

Limitations of File-based Systems

- Program must implement
  - Security
  - Concurrency control
  - Support for schema reorganization
  - Performance enhancing data structures, e.g., indexes
- Observation
  - Many applications need these services.
- Solution
  - Build and sell a software system to provide services!

Outline

- Why Use a Database?
- The Field
- Overview of the Course

Major Players in the Field

- Oracle
  - 108,000 employees
  - $38.2B (US dollars) annual revenue (FY 2014)
  - 34% market share
- Microsoft
  - Produces SQLServer
  - 89,000 employees

  $86.7B annual revenue (includes Office, Windows, etc.)
- IBM
  - Produces DB2
  - 105,000 employees in US, 75,000 employees in India
  - $99.8B annual revenue

IBM Information Management Teams

Over 6000 employees worldwide

By C. Mohan, IBM Fellow and IBM India Chief Scientist,

The Field

- Professional Publications
  - *ACM Transactions on Database Systems (TODS)*
    - quarterly, 700 pages per year
  - *IEEE Transactions on Knowledge and Data Engineering (TKDE)*
    - bimonthly, 1000 pages a year
  - *The VLDB Journal (VLDBJ)*
    - quarterly, 450 pages a year
  - *Information Systems (Info Sys)*
    - bimonthly, 600 pages a year
  - Numerous other journals
- Unrefereed Technical Publications
  - *ACM SIGMOD Record*
    - quarterly, 250 pages a year
  - *IEEE Data Engineering Bulletin*
    - quarterly, 250 pages a year

The Field, cont.

- Conferences
  - ACM SIGMOD International Conference on Management of Data (*SIGMOD*)
    - late May or early June, 500 pages a year
  - ACM Principles of Database Systems (*PODS*)
    - In conjunction with SIGMOD, 300 pages a year
  - International Conference on Very Large Data Bases (*VLDB*)
    - Mid-September, 500 pages a year
  - IEEE International Conference on Data Engineering (*ICDE*)
    - February, 600 pages a year
  - Extending Data Base Technology (*EDBT*), International Conference on Database Theory (*ICDT*)
    - alternate years (March and January), 400 pages a year
  - 8-10 specialized conferences a year: $300 \times 8 = 2500$ pages a year
- 8,000 pages per year of research papers

The Field, cont.

- Other
  - Technical reports from university and industrial research labs
  - Trade rags: Data Base Newsletter, Database Review, InfoDB, etc.: thousands of pages a year
  - Perhaps 100 database textbooks
  - Several hundred database user guidebooks

Outline

- Why Use a Database?
- The Field
- Overview of the Course

The Course

- Motivation (**M**)
- The Relational Model (**R**)
  - Definition
  - Integrity constraints
- Relational Query Languages (**E**)
  - Relational algebra
  - Relational calculi: domain and tuple
- Microsoft Access (**A**)
  - DDL
  - DML

The Course, cont.

- SQL DDL and DML (**S**)
  - Schema definition
  - Querying
  - Modifications
  - Views
  - Embedded SQL
  - Transaction management
- Conceptual Design (**C**)
  - Basic Entity-Relationship Model
  - Constraints
  - Specialization and generalization
  - View integration
  - Mapping an ER schema to tables

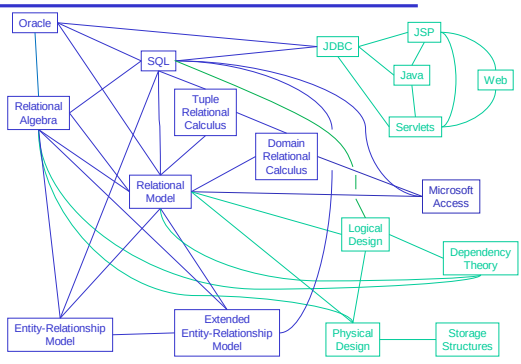
The Course, cont.

- Web/Database Connections (**W**)
  - HTTP
  - Static Database Access
  - Dynamic Database Access
- Application Design and Implementation (**T**)
  - Direct database interfaces
  - Indirect interfaces
  - Web interfaces

The Course, cont.

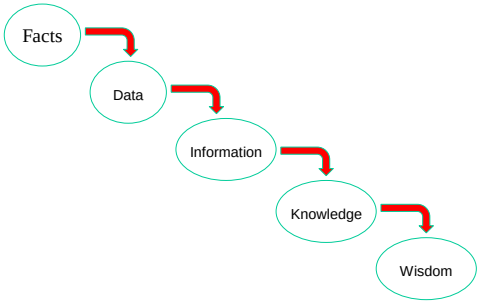
- Logical Design (**L**)
  - Properties of a good design
  - Functional dependencies and keys
  - Normal forms: 3NF, BCNF
  - Decomposition algorithms
- Physical Data Organization (**P**)
  - Relational structures: heap, sorted, compressed
  - Indexes: primary and secondary, B-trees
- The Future (**F**)
  - Data models
  - Query languages
  - Next-generation database technologies

Course Concepts



Why Use a Database? M-28

Your Intellectual Journey



Why Use a Database? M-29

An Example

Facts: Bob started job training on July 1, 2010.

Data: `"Bob", "Job Training", "July 1, 2010"`

Table with foreign key to Client table, foreign key to Task table, and date.

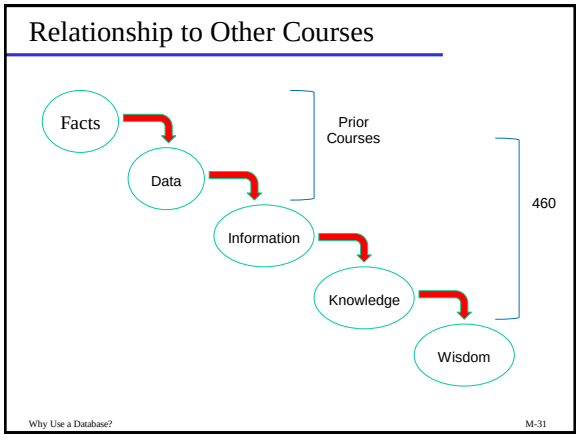
Information: `("14735","377","07-01-2010")`

Knowledge: 198 (79%) of clients are currently in job training.

Clients are in job training for an average of seven months.

Wisdom: Job training, if started within two months, and if monitored closely, can be effective for a majority of people on welfare.

Why Use a Database? M-30



The Relational Model

CSc 460 Database Systems

Fall 2014

Acknowledgements

- These slides were written by Richard T. Snodgrass (University of Arizona) and Christian S. Jensen (Aalborg University).
- Kristian Torp (Aalborg University) converted the slides from island presents to Powerpoint.
- Curtis E. Dyreson (Washington State University) modified the slides.

Overview

- The Relational Model (RM)
  - Relations
  - Properties of relations
- Integrity Constraints

Course Concepts

Sets

Definition: A *set* is an *unordered* collection of *distinct* objects.

- Examples
  - {3, 4, a} is a set
  - {a, 3, 4} is the *same* set as {3, 4, a}
  - {4, 4} is *not* a set

- Set operations include
  - *Intersection*, e.g., {3, 4, a} ⋂ {b, 4} = {4}
  - *Union*, e.g., {3, 4, a} ⋃ {b, 4} = {3, 4, a, b}
  - *Difference*, e.g., {3, 4, a} - {b, 4} = {3, a}

Domains and Attributes

Definition: A *domain* is a set of values of some "type".

- Positive integers = {1, 2, 3, 4, …}
- Alphanumeric characters = {'a', 'b', …, 'Z', '0', …, '9'}

- A common database restriction: Domains are *atomic*.
  - The following programming types are atomic domains.
    `integer`, `char`, `float`, `varchar`
  - Structs and records are associated with a *composite* domain.
    `struct { street: char[], city: char[], state: char[2] } address`

Definition: An *attribute* is the name of a domain.

Relations

Definition: Given sets $A_1, A_2, \ldots, A_n$ a *relation* r is a subset of their Cartesian product.

$$r \subseteq A_1 \times A_2 \times \ldots \times A_n$$

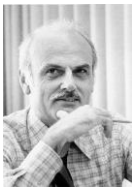
- r is a *set of n-tuples* $(a_1, a_2, \ldots, a_n)$ where $a_i \in A_i$

Definition: $R(A_1, A_2, \ldots, A_n)$ is the *schema* of relation r.

- $A_1, A_2, \ldots, A_n$ are *domains*; their names are *attributes.*
- R is the name of the relation.

- *Notation:* $r(R)$ denotes that r is a relation on the relation schema R.

Edgar F. (Ted) Codd (1923–2003)

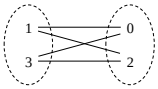


- Codd, E.F., "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM* **13** (6): 377–387, 1970.
- ACM Turing Award winner

Example, The < Relation

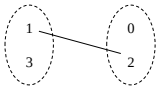
- Let O be the domain {1, 3} and E be {0, 2}.

$$O \times E = \{(1,0), (1,2), (3,0), (3,2)\}$$



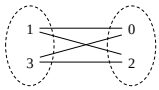
- $<(O,E)$ is the schema, O, E are attributes, {1,3} and {0,2} are domains

$$O < E = \{(1,2)\} \subseteq O \times E$$



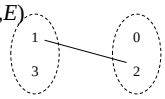
Relations as Tables

- Mathematical relations can be depicted as tables (approx.)
- $\times(O,E)$ where $O = \{1,3\}$ and $E = \{0,2\}$



×	*O*	*E*
	1	0
	1	2
	3	0
	3	2

- $<(O,E)$



<	*O*	*E*
	1	2

Tuples

Definition: An element *t* of a relation *r* is called a *tuple*.

- We refer to component values of a tuple *t* by $t[A_i] = v_i$ (the value of attribute A_i for tuple *t*).
  - Alternatively, use 'dot' notation, e.g., $t.A_i$ is the i^{th} attribute of tuple *t*
- $t[A_i, \ldots, A_k]$ refers to the subtuple of *t* containing the values of attributes $A_i, \ldots, A_k$ respectively.
- Table metaphor
  - Tuple is a row
  - Attribute is a column

Characteristics of Relations

- Tuples in a relation are *unordered.*
- Example: The following two relations have the same information content.

Name	*Age*
Pat	1
Fred	2
Sue	3
Pam	4

Name	*Age*
Sue	3
Pam	4
Fred	2
Pat	1

Characteristics of Relations (cont.)

- Attributes in a tuple/relation are ordered.
  - But, we can change the order at will (to be shown later).
  - Example: The following relations do ***not*** have the same information.

Name	*Age*
Pat	1
Fred	2
Sue	3
Pam	4

Age	*Name*
1	Pat
2	Fred
3	Sue
4	Pam

Characteristics of Relations (cont.)

- Values in a tuple are atomic (indivisible).
- A value cannot be a structure, record, or relation

Example: All atomic values (string or integer).

Name	*Age*
Pat	1
Fred	2

Example: The following is ***not*** a relation, as Name is not atomic.

Name		*Age*
First	*Last*	
Joe	Doe	1
First	*Last*	
Sue	Doe	2

Degree and Cardinality

Ord	***CID***	***Date***	***Salesperson***
102	42	110207	Johnson
103	39	110211	Strong
104	24	110228	Boggs
105	42	110303	Strong
107	34	110308	West
108	46	110308	West
110	24	110312	Boggs
111	21	110318	Strong
112	29	110320	Clark

Tuples (or records, rows)
[*Cardinality* is # of them]

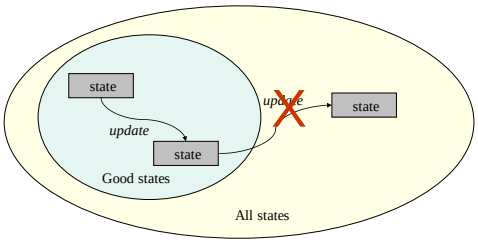
Degree or *arity*

Overview

- The Relational Model (RM)
- Integrity Constraints
  - Overview
  - Key constraints
    - Super, candidate, and primary keys
    - Foreign keys and referential integrity
  - Other constraints

Integrity Constraints

- An *integrity constraint* is a *predicate* that is required to be true of every possible instance of the database schema.
- Purpose: To keep the database in a "good" state.
  - A good state is one that is consistent with the modelled reality.
- How: Check to see if an update will move to a "bad" state.



Integrity Constraints, cont.

- Are specified at the schema level
  - A constraint must be true for all possible instances of a schema.
  - Critical notion: *possible* instances
- Some can be specified (conceptually) with relational algebra.
- In commercial databases, specified with SQL (typically).

Enforcement of Integrity Constraints

- Evaluate on update (insert/delete/modification)
- Cost of enforcement
  - Only allow specification of ICs that can be tested efficiently.
  - How many to evaluate?

Remedy for Violation

- When an integrity violation occurs either
  - Cancel the operation that causes the violation.
  - Perform the operation but inform the user of the violation.
  - Trigger additional updates so the violation is corrected.
  - Execute a user-specified error-correction routine.
- Specified for each individual constraint

Key Constraints

Definition: A *superkey* of R is a set of attributes K of R such that no two tuples in any valid relation instance $r(R)$ will have the same value for K.

- For any distinct tuples t_1 and $t_2 \in r(R)$, $t_1 \neq t_2 \Rightarrow t_1[K] \neq t_2[K]$

Definition: A *candidate key* of R is a *minimal* superkey; that is, a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey.

Definition: A *primary key* is a specified candidate key.

Example

- Consider two relation schemas (with no keys specified).
  - Customer (ID, Name)
  - Reserved (CustomerID, FilmID, ResDate)
- Assumption: No two customers have the same ID
  - Superkeys of Customer
    {ID}
    {ID, Name}
  - Candidate key of Customer (and also the primary key, since only one)
    {ID}
- Assumption: Customer can reserve many films over several days.
  - The superkey of Reserved
    {CustomerID, FilmID, ResDate}
  - One candidate key (see above), primary key (see above)

Keys, Fact and Fiction

- A key is often more than a single attribute.
  - Some DB products impose a single-attribute key restriction
  - Smaller keys are sometimes more efficient.
  - This optimization detracts from semantics.
- An attribute in a key can be of any type.
  - Some DB products restrict keys to integer or auto-number types.
  - Easy for the software to check and maintain
  - Detracts from semantics
- Choosing a key is difficult.
  - Schema may change over time.
    - Assume SSN is a good key, new law makes using SSN illegal.
  - Converting after a "wrong choice" can be costly.
    - Keys are "shared" as foreign keys in other relations.

Entity Integrity

- The primary key attributes $P = \{a_1, \ldots, a_m\}$ of each relation schema R in S cannot have null values in any tuple of $r(R)$.
  - $t[P] \neq Null$ for any tuple $t \in r(R)$
  - Alternatively, $\forall k\ 1 \leq k \leq m\ (\forall t \in r(R)\ t[a_k] \neq Null)$
- This is because primary key values are used to identify individual tuples.

Domain Constraints

- This is similar to programming languages
  1. *Strong* typing or type-by-name in programming languages.
  2. Most DBMSs have only a restricted selection of simple, predefined types available.
     - Each domain creates a separate type.
- Consider these sample relation schemas.

  Film (ID, Title)

  Customer (ID, Name)
- It is not meaningful to compare Film and Customer IDs.
  - Create FilmID domain of type integer
  - Create CustomerID domain of type integer
  - DBMS does not allow IDs of domain FilmIDs to be compared with IDs of domain CustomerIDs.

Referential Integrity

- Referential integrity is the "glue" that keeps the database together, ensuring the consistency of foreign keys.
  - A foreign key *points to* or *references* some relation
  - A foreign key is a key in the other relation
  - Referential integrity says that the information that is pointed to must exist.
- Example:If a CustomerID, e.g., 123456, is associated with a FilmID, e.g., 100001, in a tuple in the Reserved relation, a customer with CustomerID 123456 must exist in the Customer relation, and a film with FilmID 100001 must exist in the Film relation.

Referential Integrity, Definition

Definition: Let r_1 and r_2 be instances of schemas R_1 and R_2, let K be a candidate key for R_1, and let α be a subset of R_2 that is compatible with K. Then, α is a foreign key for R_2 if every value for α in a tuple in r_2 also appears as a value of K in a tuple in r_1.

$$\forall r_1(R_1)\ \forall r_2(R_2)\ \exists t_1 \in r_1\ \forall t_2 \in r_2\ (r_2[\alpha] \neq Null \wedge r_2[\alpha] = r_1[K])$$

- Constraints of this type are termed *referential integrity constraints* or *subset constraints*.

Enforcement of Referential Integrity

- Consider the constraint that every foreign key value in r_2 is in the referenced relation r_1 .
- Insertion into r_2:
  - Check that a corresponding tuple exists in r_1 .
- Deletion from r_1:
  - Check that no matching tuple existed in r_2.
- Updates of tuples in r_2 are treated as insertions above.
- Updates of tuples in r_1 are treated as deletions above.
- Note that cascading deletes are possible.

Assertions

- *Assertions* are more general integrity constraints, expressed directly as predicates which must always be satisfied.
- Think of these as sanity checks.
- Examples
  - No DVD rents for less than $0.00.
  - The spoken and subtitled languages for a foreign film must be different.
  - Reservations must have a date of today.
  - No DVD is being rented by more than one customer.

Relational Query Languages

CSc 460 Database Systems

Fall 2014

Acknowledgements

- These slides were written by Richard T. Snodgrass (University of Arizona) and Christian S. Jensen (Aalborg University).
- Kristian Torp (Aalborg University) converted the slides from island presents to Powerpoint.
- Curtis Dyreson (Washington State University) modified the slides.
- Rick converted to Jun Yang's dbsym font, using George Necula's TexPoint package.
- Lester McCann's article on relational division was helpful: "On Making Relational Division Comprehensible," in *Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference*, pp. F2C-6 to F2C-11, Boulder, CO, November, 2003.

Overview

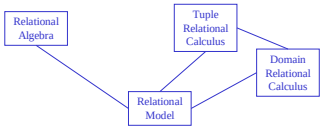
Relational Algebra (RA)

- Operators
  - Complete set – projection, selection, Cartesian product, difference, union
  - Convenient – intersection, theta join, equijoin, natural join, semijoin, antijoin, division
- Example queries
- Limitations
- Extended Relational Algebra Operators

- Tuple Relational Calculus (TRC)

- Domain Relational Calculus (DRC)

Course Concepts



Algebra

- An *algebra* is a mathematical object with
  - A set of *objects*
  - A set of *operations*, each of which takes one or more objects and evaluates to an object
- Some algebras are *closed*: result of each operation on any possible input objects is a valid object.

Relational Algebra

- Five primary operators ("complete" or "query complete")
  - Selection: σ
  - Projection: π
  - Union: $\cup$
  - Difference: –
  - Cartesian Product: $\times$
- Convenient derived operators
  - Intersection: $\cap$
  - Theta join $\bowtie_\theta$, equijoin $\bowtie_=$, and natural join $\bowtie$
  - Semijoins: $\ltimes$ and $\rtimes$
  - Antijoin: $\triangleright$
  - Relational division: $\div$

RA Operations Produce Relations

Name	*Age*
Tom	4
Jane	10
Mel	9

⇨ operation$_1$

Name	*Age*
Jane	10
Mel	9

⇨ operation$_2$

Name
Jane
Mel

Notation: operation$_2$(operation$_1$(*r*))

Unary operators are prefix; binary operators are infix.

Selection, Informal Description

- Choose a row or rows, choice is based on a *condition* or *predicate*
- Result has same schema.
- Remember: rows are unordered so cannot specify row number, e.g., 2nd row or 4th row.

Friends

Name	*Age*
Tom	4
Jane	10
Mel	9

Result

Name	*Age*
Jane	10
Mel	9

Select rows from *Friends* such that ***Age*** > 8: $\sigma_{Age>8}(Friends)$.

Selection, Formal Description

$$\sigma_P(r) = \{t \mid t \in r \wedge P(t)\}$$

- *P* is a formula in *propositional calculus*, dealing with terms of the form:
  - attribute (or constant) = attribute (or constant)
  - attribute ≠ attribute
  - attribute < attribute
  - attribute ≤ attribute
  - attribute ≥ attribute
  - attribute > attribute
  - term ∧ term *(Note: ∧ is AND)*
  - term ∨ term *(Note: ∨ is OR)*
  - ¬(term) *(Note: ¬ is NOT)*

Selection Example

r:

A	B	C
1	y	ω
3	x	γ
2	y	β

$\sigma_{B = y \wedge A > 1}(r)$:

A	B	C
2	y	β

Projection, Informal Description

- Project column(s), choice is based on column name
- Result usually has different schema, fewer columns.
- Remember: column are named, so can use name

Friends

Name	*Age*
Tom	4
Jane	10
Mel	9

Result

Name
Tom
Jane
Mel

Project the *Name* column from *Friends:* $\pi_{Name}(Friends)$.

Projection, Formal Description

$$\pi_X(r) = \{t[X] \mid t \in r\}$$

- Let $X = \{A_{i_1}, A_{i_2}, \ldots, A_{i_n}\}$
- The result is a relation of n columns obtained by keeping the columns that are specified.

r:

A	B	C
1	y	ω
3	x	γ
2	y	β

$\pi_{A,C}(r)$:

A	C
1	ω
3	γ
2	β

$\pi_B(r)$:

B
y
x

Switching Columns

- Assume ***r*** is the following

Name	*Age*
Mel	1
Eric	2
Jane	3
Tom	4

- To switch columns use projection

$\pi_{Age,Name}(r)$ =

Age	*Name*
1	Mel
2	Eric
3	Jane
4	Tom

Union, Informal Description

- Put all tuples in two relations into one, new relation
- Remember: relation has unique elements, must eliminate duplicates

Friends

Name	*Age*
Tom	4
Jane	10
Mel	9

Others

Name	*Age*
Eric	8
Jane	10

Result

Name	*Age*
Tom	4
Jane	10
Mel	9
Eric	8

Perform Union of *Friends* and *Others: Friends* $\cup$ *Others* .

Union, Formal Description

$$r \cup s = \{t \mid t \in r \vee t \in s\}$$

- *r* and *s* must be *union-compatible:*
  - *r* and *s* have the same arity.
  - The attributes of *r* and *s* are over the same domains.

r:

A	*B*	*C*
1	*y*	ω
3	*x*	γ
2	*y*	β

s:

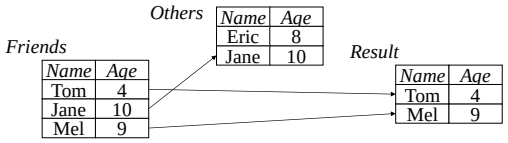
A	*B*	*C*
4	*w*	ζ
3	*x*	γ

$r \cup s$:

A	*B*	*C*
1	*y*	ω
3	*x*	γ
2	*y*	β
4	*w*	ζ

Difference, Informal Description

- Remove same tuples from a relation
- Schemas must be union compatible.



Perform *Friends* minus *Others: Friends – Others*.

Difference, Formal Description

$$r - s = \{t \mid t \in r \wedge \neg(t \in s)\}$$

r:

A	B	C
1	y	ω
3	x	γ
2	y	β

s:

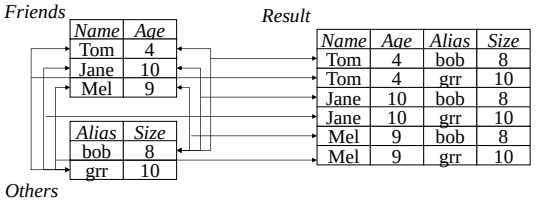
A	B	C
4	w	ζ
3	x	γ

$r - s$:

A	B	C
1	y	ω
2	y	β

Cartesian Product, Informal Description

- All possible combinations of tuples from two relations
- Schema of result is combined schema of both operands



Perform Cartesian Product of *Friends* and *Others: Friends* × *Others* .

Cartesian Product

$$r \times s = \{t \circ q \mid t \in r \wedge q \in s\}$$

- The schema of the resulting relation is $R \circ S$

r:

A	B	C
1	y	ω
3	x	γ
2	y	β

s:

D	A
"Tom"	1
"Eric"	2

$r \times s$:

A	B	C	D	A
1	y	ω	"Tom"	1
3	x	γ	"Tom"	1
2	y	β	"Tom"	1
1	y	ω	"Eric"	2
3	x	γ	"Eric"	2
2	y	β	"Eric"	2

Overview

Relational Algebra (RA)

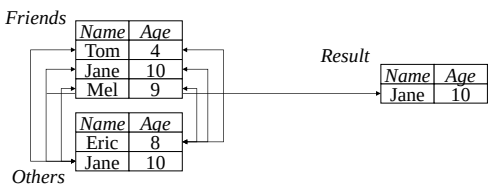
- Operators
  - Complete set – projection, selection, Cartesian product, difference, union
  - Convenient – intersection, theta join, equijoin, natural join, semijoin, division
- Limitations
- Extended Relational Algebra Operators

- Tuple Relational Calculus (TRC)

- Domain Relational Calculus (DRC)

Intersection, Informal Description

- Keep same tuples from two relations
- Schema must be the same for all three



Perform Intersection of *Friends* and *Others*: *Friends* ∩ *Others* .

Intersection, Formal Description

$$r \cap s = \{t \mid t \in r \wedge t \in s\} = r - (r - s)$$

r:

A	B	C
1	y	ω
3	x	γ
2	y	β

s:

A	B	C
4	w	ζ
3	x	γ

$r \cap s$:

A	B	C
3	x	γ

Joins

- Joins are Cartesian products coupled with selections and projections.
- Theta join
  - $r \bowtie_\theta s = \sigma_\theta (r \times s)$
  - Example: $r \bowtie_{A < E} s = \sigma_{A < E} (r \times s)$
- Equijoin
  - A theta join in which θ is an equality predicate
  - Example: $r \bowtie_{A = E} s = \sigma_{A = E} (r \times s)$
- Natural join ⋈
- Semijoins
  - Left semijoin ⋉
  - Right semijoin ⋊
- Antijoin ▷

Theta Join, Informal Description

- Try all possible combinations of tuples from two relations, keep only those that match a condition
- Schema of result is combined schema of both operands

Friends

Name	*Age*
Tom	4
Jane	10
Mel	9

Others

Alias	*Size*
Bob	8
Eric	10

Result

Name	*Age*	*Alias*	*Size*
Tom	4	Bob	8
Tom	4	Eric	10
Mel	9	Eric	10

Perform theta join between *Friends* and *Others* on *Age* < *Size*.

Theta Join, Formal Description

$$r \bowtie_\theta s = \sigma_\theta (r \times s)$$

r:

A	B	C
1	y	ω
0	x	γ
2	y	β

s:

D	E
"Tom"	3
"Eric"	1

$r \bowtie_{A < E} s$:

A	B	C	D	E
1	y	ω	"Tom"	3
0	x	γ	"Tom"	3
0	x	γ	"Eric"	1
2	y	β	"Tom"	3

- Equijoin
  - A theta join in which θ is an equality predicate
  - Example: $r \bowtie_{A=E} s$

Natural Join, Informal Description

- Do equi-join, then eliminate duplicate columns
- Schema of result is combined schema of both operands

Friends

Name	Age
Tom	4
Jane	10
Mel	9

Others

Alias	Age
Bob	8
Eric	10

Result

Name	Age	Alias
Jane	10	grr

Perform natural join of *Friends* and *Others*: *Friends* ⋈ *Others*.

Natural Join, Formal Description

$$r \bowtie s = \pi_{R \cup S} (r \bowtie_= s)$$

- Schema of result is $R \cup S$
- Let t be a tuple in the result.
  - $t[R]$ has the same value as a tuple $t_R \in r$.
  - $t[S]$ has the same value as a tuple $t_s \in s$.
  - Predicate is equality, on *like-named* attributes.

r:

A	B	C
1	y	ω
3	x	γ
2	y	β

s:

D	B
"Tom"	y
"Eric"	x

$r \bowtie s$:

A	B	C	D
1	y	ω	"Tom"
3	x	γ	"Eric"
2	y	β	"Tom"

- Note: notation in textbook is different.

Movie Web Startup Schema (subset)

- Customer (CustomerID, Name, Street, City, State, Zipcode)
- Film (FilmID, Title, RentalPrice, Kind)
- Reserved (CustomerID, FilmID, ResDate)

Movie Web Queries

- List information for films with a rental price over $4.

$$\sigma_{RentalPrice > 4}(Film)$$

- List the titles of films with a rental price over $4.

$$\pi_{Title}(\sigma_{RentalPrice > 4}(Film))$$

Movie Web Queries, cont.

- List outrageously priced films (over $4 or under $1).

$$\pi_{Title}(\sigma_{RentalPrice > 4}(Film) \cup \sigma_{RentalPrice < 1}(Film))$$

$$\pi_{Title}(\sigma_{RentalPrice > 4 \vee RentalPrice < 1}(Film))$$

- List the ID numbers of the films that are expensive and have been reserved.

$$\pi_{FilmId}(\sigma_{RentalPrice > 4}(Film)) \cap \pi_{FilmId}(Reserved)$$

$$\pi_{FilmId}(\sigma_{RentalPrice > 4}(Film)) \bowtie \pi_{FilmId}(Reserved)$$

Movie Web Queries, cont.

- List the IDs of the expensive films that have not been reserved.

$$\pi_{\text{FilmId}} (\sigma_{\text{RentalPrice} > 4}(\text{Film})) - \pi_{\text{FilmId}}(\text{Reserved})$$

Movie Web Queries, cont.

- List the titles of all reserved films.

$$\pi_{\text{Title}} (\sigma_{\text{Film.FilmID} = \text{Reserved.FilmID}}(\text{Film} \times \text{Reserved}))$$

$$\pi_{\text{Title}} (\text{Film} \bowtie \text{Reserved})$$

- List the customers who have reserved a film.

$$\pi_{\text{Name}} (\sigma_{\text{Customer.CustomerID} = \text{Reserved.CustomerID}} (\text{Customer} \times \text{Reserved}))$$

$$\pi_{\text{Name}} (\text{Customer} \bowtie \text{Reserved})$$

Movie Web Queries, cont.

- List the customers who have reserved expensive films.

$$\pi_{\text{Name}} (\sigma_{\text{RentalPrice} > 4} (\text{Customer} \bowtie \text{Reserved} \bowtie \text{Film}))$$

$$\pi_{\text{Name}} (\text{Customer} \bowtie \sigma_{\text{RentalPrice} > 4} (\text{Reserved} \bowtie \text{Film}))$$

$$\pi_{\text{Name}} (\text{Customer} \bowtie \text{Reserved} \bowtie \sigma_{\text{RentalPrice} > 4} (\text{Film}))$$

- List the streets of customers who have reserved foreign films.

$$\pi_{\text{Street}} (\text{Customer} \bowtie \text{Reserved} \bowtie \sigma_{\text{Kind} = \text{“F”}} (\text{Film}))$$

Renaming Operator

- Find the film(s) with the highest rental price.
- We need a renaming operator: ρ_{Name}
- This operator doesn't really do anything, other than make it easier to do self-joins.
- Alternative formulation of the query: Find the film(s) with a rental price for which no other rental price is higher.

$$\pi_{Title}(\text{Film}) - \pi_{F2.Title}(\text{Film} \bowtie_{\text{Film.RentalPrice} > \text{F2.RentalPrice}} \rho_{F2}(\text{Film}))$$

Semijoins

$$r \ltimes s = \pi_R (r \bowtie s)$$

- The result has the same schema as the left-hand argument, *r*.

r:

A	*B*	*C*
1	*y*	ω
3	*x*	γ
2	*y*	β

s:

D	*B*
"Tom"	*y*

$r \ltimes s$:

A	*B*	*C*
1	*y*	ω
2	*y*	β

- Right semijoin: $r \rtimes s = \pi_S (r \bowtie s)$

Semijoins in Distributed Query Processing

- Assume that relation *r* is very large, and resides in a database in Los Angeles.
- Relation *s* is small, and resides in a database in New York City.
- Application desires $r \bowtie s$, in New York City.
- Sending all of *r* from Los Angeles to New York City would be very expensive.
- Alternative processing strategy:
  - New York City sends $\pi_{R \cap S}$ (s) to Los Angeles.
  - Los Angeles computes $r \ltimes s$, and sends the result to New York City. This table is significantly smaller than *r*.
  - New York City computes $(r \ltimes s) \bowtie s = r \bowtie s$

Antijoin

- $r \triangleright s = r - (r \ltimes s)$
- The result has the same schema as the left-hand argument, *r*.

r:

A	*B*	*C*
1	*y*	ω
3	*x*	γ
2	*y*	β

s:

D	*B*
"Tom"	*y*

$r \triangleright s$:

A	*B*	*C*
3	*x*	γ

- Sometimes called the anti-semijoin.

Relational Division

$$r \div s = \{ t \mid \forall u \in s \, (t \circ u \in r) \}$$

r:

A	*B*	*C*
1	*y*	ω
3	*x*	γ
2	*y*	β
1	*z*	ω

s:

B
y
z

$r \div s$:

A	*C*
1	ω

Intuition

Friends:

Name	*Age*
Tom	4
Jane	10
Mel	9

Others:

Alias	*Size*
bob	8
grr	10

Prod = Friends × Others:

Name	*Age*	*Alias*	*Size*
Tom	4	bob	8
Tom	4	grr	10
Jane	10	bob	8
Jane	10	grr	10
Mel	9	bob	8
Mel	9	grr	10

Prod ÷ Friends:

Alias	*Size*
bob	8
grr	10

Prod ÷ Others:

Name	*Age*
Tom	4
Jane	10
Mel	9

Relational Division

- Let r and s be relations on schemes R and S respectively, where
  - $R = (A_1, A_2, ..., A_n, B_1, B_2, ..., B_n)$
  - $S = (B_1, B_2, ..., B_n)$
- The result of r divided by s is a relation on scheme $R \div S = (A_1, A_2, ..., A_n)$
- $(r \div s) \times s \subseteq r$

Movie Web Queries, cont.

- List the customers who have reserved *all* the foreign films.
  - Identify the foreign films.

$$\pi_{\text{FilmID}} (\sigma_{\text{Kind} = \text{“F”}} (\text{Film}))$$

  - Identify those customers who have reserved all foreign films.

$$\pi_{\text{CustomerID,FilmID}} (\text{Reserved}) \div \pi_{\text{FilmID}} (\sigma_{\text{Kind} = \text{“F”}} (\text{Film}))$$

  - Now figure out the names of those customers.

$$\pi_{\text{Name}} (\text{Customer} \bowtie (\pi_{\text{CustomerID,FilmID}} (\text{Reserved}) \div \pi_{\text{FilmID}} (\sigma_{\text{Kind} = \text{“F”}} (\text{Film}))))$$

Hints

1. Make right argument one column of the stuff after “all.”
2. Make left argument two columns, one the stuff before “all” and the other the column above.
3. Both arguments should be IDs of something.

Customers Who Have Reserved All the Foreign Films

- R = { CustomerID, FilmID } "Reservations"
- $r = \pi_{\text{CustomerID,FilmID}}$ (Reserved)

CustomerID	*FilmID*
12345	1
12345	2
12345	7
24680	1
24680	19
36925	1
36925	19
36925	7

- S = { FilmID } "Foreign Films"
- $s = \pi_{\text{FilmID}} (\sigma_{\text{Kind} = \text{"F"}} (\text{Film}))$

FilmID
1
7

- Shared columns: B = { FilmID}
- Extra columns in R: A = { CustomerID }
- Reservations ÷ Foreigh Films is what?

Relational Division

- Let r and s be relations on schemes R and S respectively, where
  - $R = (A_1, A_2, ..., A_n, B_1, B_2, ..., B_n)$
  - $S = (B_1, B_2, ..., B_n)$
- The result of r divided by s is a relation on scheme $R \div S = (A_1, A_2, ..., A_n)$
- Can be done with just the five basic operators (!):

$$r \div s = \pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times s) - r)$$

Division Example, cont.

1. $\pi_{R-S}(r)$: (the A columns of r)

CustomerID
12345
24680
36925

2. $\pi_{R-S}(r) \times s$: (This is the maximum possible result.)

CustomerID	*FilmID*
12345	1
12345	7
24680	1
24680	7
36925	1
36925	7

3. $(\pi_{R-S}(r) \times s) - r$: (the possible tuples that *aren't* in r, termed the *victims*)

CustomerID	*FilmID*
24680	7

Division Example, cont.

4. $\pi_{R-S}((\pi_{R-S}(r) \times s) - r)$:
(the A columns of the victims)

CustomerID
24680

5. $\pi_{R-S}(r)$:
(the A columns of r: the maximum possible result of $r \div s$)

CustomerID
12345
24680
36925

6. $\pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times s) - r)$:
(Step 5 – Step 4: the maximum result minus the victims)

CustomerID
12345
36925

Intuition for Equivalence

- $r \div s = \pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times s) - r)$
- $\pi_{R-S}(r)$: the A columns $(A_1, A_2, ..., A_n)$ of r
- $\pi_{R-S}(r) \times s$: the A and B columns; the maximum possible product
- $(\pi_{R-S}(r) \times s) - r$: the possible tuples that *aren't* in r (termed the *victims*)
- $\pi_{R-S}((\pi_{R-S}(r) \times s) - r)$: the A columns of the victims
- $\pi_{R-S}(r)$: the A columns $(A_1, A_2, ..., A_n)$ of r: the maximum possible *result* of $r \div s$, note: $r \div s \subseteq \pi_{R-S}(r)$
- $\pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times s) - r)$: the resulting tuples (just the A columns) that *aren't* victims

Relational Completeness

- All the operators can be expressed in terms of the five basic operators: σ, π, $-$, $\times$, $\cup$ (along with ρ).
- This set is called a *complete set* of relational algebraic operators.
- Any query language that is at least as powerful as these operators is termed *(query) relationally complete*.

Limitations of the Algebra

- Can't do arithmetic.
  - Find the rental price assuming a 10% increase.
- Can't do aggregates.
  - How many films has each customer reserved?
- Can't handle "missing" data.
  - Make a list of the films, along with who reserved it, if applicable.
- Can't perform transitive closure.
  - For a partof(Part, ConstituentPart) relation, find all parts in the car door.
- Can't sort, or print in various formats.
  - Print a reserved summary, sorted by customer name.
- Can't modify the database.
  - Increase all $3.25 rentals to $3.50.

Overview

Relational Algebra (RA)

- Operators
  - Complete set – projection, selection, Cartesian product, difference, union
  - Convenient – intersection, theta join, equijoin, natural join, semijoin, division
- Example queries
- Limitations
- Extended Relational Algebra Operators

- Tuple Relational Calculus (TRC)

- Domain Relational Calculus (DRC)

Extending the Projection Operator

- The *generalized* projection operator allows expressions in addition to column names in the subscript.

- Find the rental price, assuming a 10% increase.

$$\pi_{\text{Title, RentalPrice*1.1}}(\text{Film})$$

Scalar Aggregate

$$\mathsf{F}_{agg_1, \ldots, agg_k}(r) = \left\{ t \circ a_1 \circ \ldots \circ a_k \mid t \in r \wedge \begin{array}{r} a_1 = agg_1(\{u \mid u \in r\}) \\ \ldots. \\ a_k = agg_k(\{u \mid u \in r\}) \end{array} \right\}$$

- Appends new attributes.
- Each aggregate function agg_i decides which attribute to aggregate over, e.g., agg_2 could be the sum of the third attribute, or the average of the fifth attribute.
- Note: Text does not retain original columns.

Scalar Aggregates, cont.

- How many reservations are there?

$$\pi_{count}(\mathsf{F}_{count}(\text{Reserved}))$$

- What is the maximum rental price for foreign films?

$$\pi_{max}(\mathsf{F}_{max(Rentalprice)}(\sigma_{Kind = \text{“F”}}(\text{Film})))$$

Aggregate Functions

$${}_G\mathsf{F}_{agg_1, \ldots, agg_k}(r) = \left\{ t \circ a_1 \circ \ldots \circ a_k \mid t \in r \wedge \begin{array}{r} a_1 = agg_1(\{u \mid u \in r \wedge u[G] = t[G]\}) \\ \ldots. \\ a_k = agg_k(\{u \mid u \in r \wedge u[G] = t[G]\}) \end{array} \right\}$$

- The grouping attribute(s) G in aggregate functions apply the aggregate separately to each group.
- The original tuple t determines which group is associated with the tuple.

Aggregates Functions, cont.

- How many reservations are there at each rental price?

$$\pi_{\text{RentalPrice, count}}({}_{\text{RentalPrice}}\mathsf{F}_{\text{count}}\ (\text{Reserved} \bowtie \text{Film}))$$

Outer Joins

- In a regular equijoin or natural join, tuples in *r* or *s* that do not having matching tuples in the other relation do not appear in the result.
- The outer joins retain these tuples, and place nulls in the missing attributes.
- Left outer join:

$$r \mathbin{⟕} s = r \bowtie s \cup ((r - (r \ltimes s)) \times (null, ..., null))$$

- Right outer join:

$$r \mathbin{⟖} s = r \bowtie s \cup ((null, ..., null) \times (s - (s \ltimes r)))$$

- Full outer join:

$$r \mathbin{⟗} s = r \bowtie s \cup ((r - (r \ltimes s)) \times \{(null, ..., null)\}) \cup (\{(null, ..., null)\} \times (s - (s \ltimes r)))$$

- Note: text uses a slightly different notation.

Outer Join Example

r:

A	*B*	*C*
1	*y*	ω
3	*x*	γ
2	*z*	β

s:

D	*B*
"Tom"	*y*
"Eric"	*x*
"Mel"	*w*

$r \mathbin{⟗} s$:

A	*B*	*C*	*D*
1	*y*	ω	"Tom"
3	*x*	γ	"Eric"
2	*z*	β	Null
Null	*w*	Null	"Mel"

Movie Web Query

- Make a list of the films, along with who reserved it, *if applicable*.

$$\pi_{\text{Title, Name}}(\text{Film} \rtimes (\text{Reserved} \ltimes \text{Customer}))$$

Overview

- Relational Algebra (RA)
  - Operators
  - Limitations
  - Extended Relational Algebra Operators
- Tuple Relational Calculus (TRC)
- Domain Relational Calculus (DRC)

Relational Calculus

- The relational algebra is *procedural*, specifying *how*, that is, a sequence of operations to derive the desired results.
- Relational calculus is based on first-order predicate calculus.
- Relational calculus is more *declarative*, specifying *what* is desired.
- The expressive power of the two languages is identical.
  - This implies that relational calculus is relationally complete.
- Many commercial relational query languages are based on the relational calculus.
- The implementations are based on the relational algebra.

Relational Calculus, cont.

- Tuple Relational Calculus (TRC)
- Domain Relational Calculus (DRC)

Tuple Relational Calculus

- A tuple variable ranges over tuples of a particular relation.
- Example: List the information about expensive films.

$$\{t \mid Film(t) \wedge t.RentalPrice > 4\}$$

- *Film(t)* specifies the range relation *Film* for the tuple variable *t*.
- Each tuple satisfying *t.RentalPrice > 4* is retrieved.
- The entire tuple is retrieved.
- List the titles of expensive films.

$$\{t.Title \mid Film(t) \wedge t.RentalPrice > 4\}$$

Syntax of Tuple Relational Calculus

$$\{t_1.A_1, t_2.A_2, \ldots, t_j.A_j \mid P(t_1, t_2, \ldots, t_j)\}$$

- $t_1, t_2, \ldots, t_j$ are tuple variables.
- Each t_k is an tuple of the relation over which it ranges.
- *P* is a predicate, which is made up of atoms, of the following types.
  - $r(t_k)$ identifies *r* as the range of t_k
  - Atoms from predicate calculus: $\wedge, \vee, \neg, \forall, \exists$

Movie Web Queries

- List the outrageously priced films (over $4 or under $1).

$$\{t \mid Film(t) \wedge (t.RentalPrice > 4 \vee t.RentalPrice < 1)\}$$

$$\{t.FilmID, t.Title, t.RentalPrice, t.Kind \mid Film(t) \wedge (t.RentalPrice > 4 \vee t.RentalPrice < 1)\}$$

- List the ID numbers of the films that are expensive and have been reserved.

$$\{t.FilmID \mid Film(t) \wedge (t.RentalPrice > 4 \wedge (\exists r\,(Reserved(r) \wedge t.FilmID = r.FilmID))\}$$

Movie Web Queries, cont.

- List the IDs of the expensive films that have *not* been reserved.

$$\{t.FilmID \mid Film(t) \wedge (t.RentalPrice > 4 \wedge \neg\exists r\,(Reserved(r) \wedge t.FilmID = r.FilmID))\}$$

Movie Web Queries, cont.

- List the titles of all reserved films.

$$\{ f.Title \mid \exists r\,(Film(f) \wedge Reserved(r) \wedge f.FilmID = r.FilmID)\}$$

- List the customers who have reserved a film.

$$\{c.Name \mid \exists r\,(Customer(c) \wedge Reserved(r) \wedge c.CustomerID = r.CustomerID)\}$$

Movie Web Queries, cont.

- List the customers who have reserved expensive films.

$$\{c.Name \mid \exists r\, \exists f\, (Customer(c) \wedge Reserved(r) \wedge Film(f) \wedge f.RentalPrice > 4 \wedge c.CustomerID = r.CustomerID \wedge r.FilmID = f.FilmID)\}$$

- List the streets of customers who have reserved foreign films.

$$\{c.Street \mid \exists r\, \exists f\, (Customer(c) \wedge Reserved(r) \wedge Film(f) \wedge f.Kind = \text{“}F\text{”} \wedge c.CustomerID = r.CustomerID \wedge r.FilmID = f.FilmID)\}$$

Movie Web Queries, cont.

- List the customers who have reserved all the foreign films.
- Rephrased: List the customers for which, for all films, if that film was a foreign film, then that film must have been reserved by that customer.

$$\{c.Name \mid Customer(c) \wedge \forall f\, (Film(f) \wedge f.Kind = \text{“}F\text{”} \Rightarrow (\exists r\, (Reserved(r) \wedge r.FilmID = f.FilmID \wedge r.CustomerID = c.CustomerID)))\}$$

$a \Rightarrow b$ is shorthand for “$\neg a \vee b$” or “**if** a **then** b”

Equivalences

- $\forall x\, (P(x)) \equiv \neg\, \exists x\, (\neg\, P(x))$
- $\exists x\, (P(x)) \equiv \neg\, \forall x\, (\neg\, P(x))$
- $\forall x\, (P(x) \wedge Q(x)) \equiv \neg\, \exists x\, (\neg\, P(x) \vee \neg\, Q(x))$
- $\forall x\, (P(x) \vee Q(x)) \equiv \neg\, \exists x\, (\neg\, P(x) \wedge \neg\, Q(x))$
- $\exists x\, (P(x) \vee Q(x)) \equiv \neg\, \forall x\, (\neg\, P(x) \wedge \neg\, Q(x))$
- $\exists x\, (P(x) \wedge Q(x)) \equiv \neg\, \forall x\, (\neg\, P(x) \vee \neg\, Q(x))$
- $\forall x\, (P(x)) \Rightarrow \exists x\, (P(x))$
- $\neg\, \exists x\, (P(x)) \Rightarrow \neg\, \forall x\, (P(x))$

Movie Web Queries, cont.

- Find the film(s) with the highest rental price.
- Rephrased: Find the film(s) for which all films have a rental price that is not higher.

$$\{ f.FilmID \mid Film(f) \wedge \forall f_2 \, (Film(f_2) \Rightarrow f_2.RentalPrice \leq f.RentalPrice)\}$$

- Alternative formulation: Find the film(s) with a rental price for which no other rental price is higher.
- Rephrased: Find the film(s) for which there does not exist a film with a higher rental price.

$$\{ f.FilmID \mid Film(f) \wedge \neg\exists f_2 \, (Film(f_2) \wedge f_2.RentalPrice > f.RentalPrice)\}$$

Overview

- Relational Algebra (RA)
  - Operators
  - Limitations
  - Extended Relational Algebra Operators
- Tuple Relational Calculus (TRC)
- Domain Relational Calculus (DRC)

Domain Relational Calculus

- Each query is an expression of the form

$$\{ x_1, x_2, \ldots , x_n \mid P \, (x_1, x_2, \ldots , x_n)\}$$

- *P* is a formula similar to the formula in the *predicate calculus*.
- List the information on the expensive films.

$$\{ F, T, R, K \mid Film(F, T, R, K) \wedge R > 4\}$$

- List the titles of the expensive films.

$$\{ T \mid \exists F \, \exists R \, \exists K \, (Film(F, T, R, K) \wedge R > 4)\}$$

Movie Web Queries

- List the outrageously priced films (over $4 or under $1).

$$\{T \mid \exists F\, \exists R\, \exists K\, (Film(F, T, R, K) \wedge (R > 4 \vee R < 1))\}$$

- List the ID numbers of the films that are expensive and have been reserved.

$$\{F \mid \exists T\, \exists R\, \exists K\, (Film(F, T, R, K) \wedge R > 4 \wedge \exists I\, \exists D\, (Reserved(I, F, D)))\}$$

Movie Web Queries, cont.

- List the IDs of the expensive films that have not been reserved.

$$\{F \mid \exists T,R,K\, (Film(F, T, R, K) \wedge R > 4 \wedge \neg(\exists I\, \exists D\, (Reserved(I, F, D))))\}$$

Movie Web Queries, cont.

- List the titles of all reserved films.

$$\{T \mid \exists F,R,K\, (Film(F, T, R, K) \wedge \exists I,D\, (Reserved(I, F, D)))\}$$

- List the customers who have reserved a film.

$$\{N \mid \exists I,S,C,ST,Z\, (Customer(I, N, S, C, ST, Z) \wedge \exists F,D\, (Reserved(I, F, D)))\}$$

Movie Web Queries, cont.

- List the customers who have reserved expensive films.

$$\{N \mid \exists I,S,C,ST,Z\,(Customer(I, N, S, C, ST, Z) \wedge \exists F,D\,(Reserved(I, F, D) \wedge (\exists T,R,K\,(Film(F, T, R, K) \wedge R > 4))\}$$

- List the streets of customers who have reserved foreign films.

$$\{S \mid \exists I,N,C,ST,Z\,(Customer(I, N, S, C, ST, Z) \wedge \exists F,D\,(Reserved(I, F, D) \wedge \exists T,R\,(Film(F, T, R, \text{“}F\text{”}))))\}$$

Movie Web Queries, cont.

- List the customers who have reserved all the foreign films.

$$\{N \mid \exists I,S,C,ST,Z\,(Customer(I, N, S, C, ST, Z) \wedge \forall F\,(\exists T,R\,(Film(F, T, R, \text{“}F\text{”}) \Rightarrow \exists D\,(Reserved(I, F, D))))\}$$

Movie Web Queries, cont.

- Find the film(s) with the highest rental price.
- Alternative formulation: Find the film(s) with a rental price for which no other rental price is higher.

$$\{T \mid \exists F,R,K\,(Film(F, T, R, K) \wedge \neg(\exists F_2,T_2,R_2,K_2\,(Film(F_2,T_2,R_2,K_2) \wedge R_2 > R)))\}$$

Assertions

- Assertions are more general integrity constraints, expressed directly as predicates which must always be satisfied.
- These can be expressed in the algebra or calculi, of the form: "There does not exist an offending tuple."

No video rents for less than $0.

- Algebra
  - $\sigma_{RentalPrice < 0}(Film) = \{\}$
- Tuple Relational Calculus
  - $\forall f(Film(f) \Rightarrow f.RentalPrice \geq 0)$
  - $\neg \exists f(Film(f) \wedge f.RentalPrice < 0)$
- Domain Relational Calculus
  - $\forall F\ (\exists T, \exists R\ \exists K(Film(F,T,R,K)) \Rightarrow R \geq 0)$
  - $\neg \exists F,T,R,K(Film(F,T,R,K) \wedge R < 0)$

Another Assertion Example

- The spoken and subtitled languages for a film must be different.
- Algebra

  $\sigma_{spokenLanguage = subtitleLanguage}(Film) = \{\}$
- Tuple Relational Calculus

  $\neg \exists f(Film(f) \wedge f.spokenLanguage = f.subtitleLanguage)$
- Domain Relational Calculus

  $\neg (\exists F,T,R,K,SP,SL\ (Film(F,T,R,K,SP,SL) \wedge SP = SL))$

Summary

- Relational algebra
  - Objects are relations (sets of n-tuples).
- Tuple relational calculus
  - Variables range over relations (sets of tuples).
  - Each variable is associated with an individual tuple.
- Domain relational calculus
  - Variables range over domains (sets of values).
  - Each variable is associated with an individual value.
- Misnamed
  - Tuple relational calculus and Value relational calculus
  - Relational relational calculus and Domain relational calculus

Expressive Power

- Theorem: The following three languages define *exactly* the same class of functions.
  - Relational algebra expressions
  - Safe relational tuple calculus formulas
  - Safe relational domain calculus formulas
- Proof:
  - relational algebra $\preceq$ (safe) tuple relational calculus
    - Show via induction that every relational algebra expression has a counterpart in the tuple relational calculus.
  - tuple relational calculus $\preceq$ domain relational calculus
  - domain relational calculus $\preceq$ relational algebra
- Corollary: All three languages are relationally complete.

Completeness

Theorem: The tuple relational calculus is complete.

Proof: First convert algebraic expression to use just the five basic operators. Then, if algebraic expression is just one operator, substitute the calculus form.

- Selection $\sigma_P(r) \Rightarrow \{ t \mid r(t) \wedge P(t) \}$
- Projection $\pi_X(r) \Rightarrow \{ t[X] \mid r(t) \}$
- Union $r \cup s \Rightarrow \{ t \mid r(t) \vee s(t) \}$
- Difference $r - s \Rightarrow \{ t \mid r(t) \wedge \neg s(t) \}$
- Cartesian product $r \times s \Rightarrow \{ t \circ q \mid r(t) \wedge s(q) \}$
- For algebraic expressions with more than one operator, induct on length of the algebraic expression.

Completeness, Example

- List the titles of all reserved films.

$$\pi_{\text{Title}}\ (\text{Film} \bowtie \text{Reserved})$$

- Replace with the five basic operators:

$$\pi_{\text{Title}}\ (\sigma_{\text{Film.FilmID = Reserved.FilmID}}(\text{Film} \times \text{Reserved}))$$

- Replace projection:

$$\{t.Title \mid \sigma_{\text{Film.FilmID = Reserved.FilmID}} (\text{Film} \times \text{Reserved})\ (t)\}$$

Completeness Example, Cont.

- Replace selection:

$$\{t.Title \mid (Film \times Reserved)\ (t) \wedge t[Film.FilmID] = t[Reserved.FilmID]\ \}$$

- Replace the Cartesian product:

$$\{t.Title \mid \exists f\ (Film(f) \wedge \exists r\ (Reserved(r) \wedge f.FilmID = r.FilmID \wedge t = f \circ r\ ))\ \}$$

- Simplify

$$\{f.Title \mid Film(f) \wedge \exists r\ (Reserved(r) \wedge f.FilmID = r.FilmID\ )\ \}$$

Safety

- It is possible to write tuple calculus expressions that generate infinite relations.
- $\{t \mid \neg\ r(t)\}$ results in an infinite relation if the domain of any attribute of relation r is infinite.
- We wish to ensure that an expression in relational calculus yields only a *finite* number of tuples.
- The *domain of a tuple relational calculus expression* is the set of all values that either appear as constant values in the expression or that exist in any tuple of the relations referenced in the expression.
- An expression is *safe* if all values in its result are from the domain of the expression.

Equivalence of Expressive Power

- Theorem: The relational algebra is as expressive as the (safe) tuple relational calculus.
- (Informal) Proof: by induction on the number of operators in the calculus predicate
  - $\exists X(...) \Rightarrow \pi(...)$
  - $X \vee Y \Rightarrow \pi(X \times Y)$
  - $P(r) \Rightarrow \sigma_P(...)$
  - $\neg P(r) \Rightarrow U - r$
  - $X \wedge Y \Rightarrow \neg(\neg X \vee \neg Y)$
  - $\forall X(P(r)) \Rightarrow \neg \exists X(\neg P(r))$

