CS 4604: Introduction to Database Management Systems

Relational Model and Relational Algebra

Virginia Tech CS 4604 Sprint 2021

Instructor: Yinlin Chen

Today’s Topics

- Relational Model
- Relational Algebra

The Relational Model

- Simple: Built around a single concept for modeling data: the relation or table.
  - A relational database is a collection of relations.
  - Each relation is a table with rows and columns.
- Supports high-level programming language (SQL).
  - Limited but very useful set of operations
- Has an elegant mathematical design theory.
- Most current DBMS are relational (Oracle, IBM DB2, MS SQL)

The Relational Model

CoursesTaken

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- Structure: Table (like an array of structs)
- Operations: Relational algebra (selection, projection, conditions, etc)
- Constraints: E.g., grades can be only {A, B, C, D, F}

The Semi-structured model

```
<CoursesTaken>
        <Student>Hermione Grainger</Student>
        <Course>Potions</Course>
        <Grade>A</Grade>
        <Student>Draco Malfoy</Student>
        <Course>Potions</Course>
        <Grade>B</Grade>
...
</CoursesTaken>
```

- **Structure**: Trees or graphs, tags define role played by different pieces of data
- **Operations**: Follow paths in the implied tree from one element to another
- **Constraints**: E.g., can express limitations on data types

Relations

- A relation is a two-dimensional table:
  - Relation ⇔ table
  - Attribute ⇔ column name
  - Tuple ⇔ row (not the header row)
- Database ⇔ collection of relations
- A relation has two parts:
  - Schema defines column heads of the table (attributes)
  - Instance contains the data rows (tuples, rows, or records) of the table

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

CoursesTaken

Schema

CoursesTaken

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- The schema of a relation is the name of the relation followed by a parenthesized list of attributes

  CoursesTaken (Student, Course, Grade)
- A design in a relational model consists of a set of schemas.
- Such a set of schemas is called a relational database schema.

Relation and Schema

- Relation is a set of tuples
  - Order in which we present the tuples does not matter
- The attributes in a schema are also a set (not a list)
  - Schema is the same irrespective of order of attributes.

    CoursesTaken(Student, Grade, Course)
  - We specify a “standard” order when we introduce a schema

CoursesTaken

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

Degree and Cardinality

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- Degree/Arity is the number of fields/attributes in schema
  - (=3 in the table above)
- Cardinality is the number of tuples in relation
  - (=4 in the table above)

Keys of Relations

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- Keys are one form of integrity constraints (IC)
  - No pair of tuples should have identical keys
- What is the key for CoursesTaken?
  - Student if only one course in the relation
  - Pair (Student, Course) if multiple courses
  - What if student takes same course many times?

Keys of Relations

- Keys help associate tuples in different relations

SID	CID	Grade
123	15-401	A
111	15-401	B
123	14-501	B
….	….	….

SID	Student	GPA
123	Hermione Grainger	3.9
111	Draco Malfoy	3.0
234	Harry Potter	3.7
456	Ron Weasley	3.1

Types of Keys

- Superkeys, (Candidate) keys
- Primary keys, Alternative keys
- Foreign keys

Superkeys

- A superkey is defined as a subset of attribute types of a relation *R* with the property that no two tuples in any relation state should have the same combination of values for these attribute types
- A superkey specifies a **uniqueness constraint**
- A superkey can have redundant attribute types
  - Example: (Studentnr, Name, HomePhone)

(Candidate) Keys

- A key *K* of a relation scheme *R* is a superkey of *R* with the additional property that removing any attribute type from *K* leaves a set of attribute types that is no superkey of *R*
- A key does not have any redundant attribute types
  - Example: Studentnr
- The key constraint states that every relation must have at least one key that allows uniquely identifying its tuples
- All super keys can't be candidate keys. All candidate keys are super keys

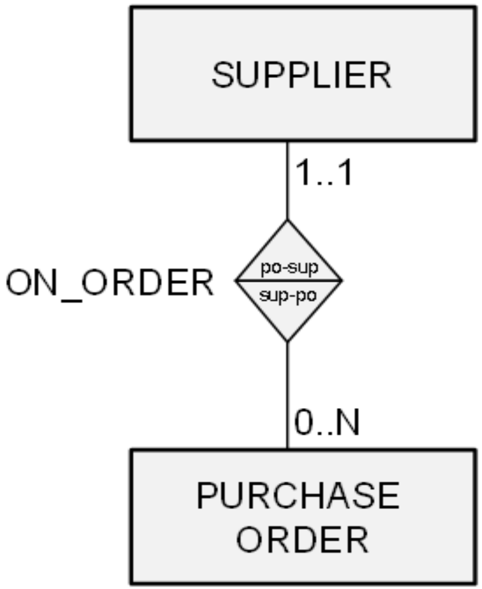
Primary Keys, and Alternative Keys

- A relation may have more than one key (candidate keys)
  - PRODUCT: product number and product name
- The primary key is used to identify tuples in the relation, to establish connections to other relations, and for storage purposes
  - Entity integrity constraint: attribute types that make up the primary key should always satisfy a NOT NULL constraint
- Only one Candidate Key can be Primary Key
- Other candidate keys are then referred to as alternate keys

Foreign Keys

- A set of attribute types FK in a relation R_1 is a foreign key of R_1 if two conditions are satisfied (referential integrity constraint)
  - The attribute types in FK have the same domains as the primary key attribute types PK of a relation R_2
  - A value FK in a tuple t_1 of the current state r_1 either occurs as a value of PK for some tuple t_2 in the current state r_2 or is NULL

Foreign Keys



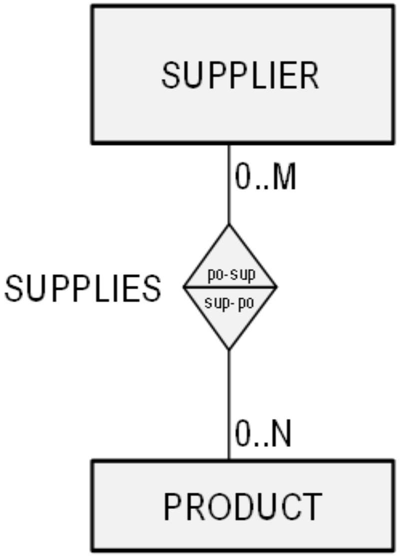
SUPPLIER

SUPNR	SUPNAME	SUPADDRESS	SUPCITY	SUPSTATUS
...				
37	Ad Fundum	82, Wacker Drive	Chicago	95
94	The Wine Crate	330, McKinney Avenue	Dallas	75
...				

PURCHASE_ORDER

PONR	PODATE	*SUPNR*
1511	2015-03-24	37
1512	2015-04-10	94
...		

Foreign Keys



SUPPLIER

SUPNR	SUPNAME	SUPADDRESS	SUPCITY
21	Deliwines	240, Avenue of the Americas	New York
32	Best Wines	660, Market Street	San Francisco
...			

SUPPLIES

SUPNR	*PRODNR*	PURCHASE_PRICE	DELIV_PERIOD
...			
68	0327	56.99	4
...			
21	0289	17.99	1
21	0327	56.00	6
21	0347	16.00	2
...			
69	0347	18.00	4
84	0347	18.00	4
...			

PRODUCT

PRODNR	PRODNAME	PRODTYPE	AVAILABLE_QUANTITY
0119	Chateau Miraval, Cotes de Provence Rose, 2015	rose	126
0154	Chateau Haut Brion, 2008	red	111
...		red	5

Relational Constraints

Domain constraint	The value of each attribute type A must be an atomic and single value from the domain.
Key constraint	Every relation has a key that allows uniquely identifying its tuples.
Entity integrity constraint	The attribute types that make up the primary key should always satisfy a NOT NULL constraint.
Referential integrity constraint	A foreign key FK has the same domain as the primary key PK attribute type(s) it refers to and either occurs as a value of PK or NULL.

Example Relational Data Model

SUPPLIER(SUPNR:integer, SUPNAME:string, SUPADDRESS:string, SUPCITY:string, SUPSTATUS:integer)

PRODUCT(PRODNR:integer, PRODNAME:string, PRODTYPE:string, AVAILABLE QUANTITY:integer)

SUPPLIES(*SUPNR*, *PRODNR*:integer, PURCHASE_PRICE:real, DELIV_PERIOD:integer)

PURCHASE_ORDER(PONR:integer, PODATE:date, *SUPNR*:integer)

PO_LINE(*PONR*:integer, *PRODNR*:integer, QUANTITY:integer)

Example Relational Data Model

Supplier

SUPNR	SUPNAME	SUPADDRESS	SUPCITY	SUPSTATUS
21	Deliwines	240, Avenue of the Americas	New York	20
32	Best Wines	660, Market Street	San Francisco	90
...				

Product

PRODNR	PRODNAME	PRODTYPE	AVAILABLE_QUANTITY
0119	Chateau Miraval, Cotes de Provence Rose, 2015	rose	126
0384	Dominio de Pingus, Ribera del Duero, Tempranillo, 2006	red	38
...			

Supplies

SUPNR	PRODNR	PURCHASE_PRICE	DELIV_PERIOD
21	0119	15.99	1
21	0384	55.00	2
...			

Purchase_Order

PONR	PODATE	SUPNR
1511	2015-03-24	37
1512	2015-04-10	94
...		

PO_Line

PONR	PRODNR	QUANTITY
1511	0212	2
1511	0345	4
...		

Relational Query Languages

<u>Query languages</u>: Allow manipulation and retrieval of data from a database.

Relational model supports simple, powerful QLs:

- Strong formal foundation based on logic.
- Allows for optimization.

Query Languages != programming languages!

- QLs not expected to be "Turing complete".
- QLs not intended to be used for complex calculations.
- QLs support easy, efficient access to large data sets.

Formal Relational Query Languages

Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:

- *Relational Algebra*: More operational (imperative), very useful for representing execution plans. (a procedural programming language)
- *Relational Calculus*: Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative, basis for SQL*.)

VIRGINIA TECH

Preliminaries

A query is applied to **relation instances**, and the result of a query is also a **relation instance**.

- *Schemas* of input relations for a query are fixed (but query will run regardless of instance!)
- The schema for the *result* of a given query is also fixed! Determined by definition of query language constructs.

Positional vs. named-field notation:

- Positional notation easier for formal definitions, named-field notation more readable.
- Both used in SQL

Example Instances

“Sailors” and “Reserves” relations for our examples.

We’ll use positional or named field notation, assume that names of fields in query results are `inherited’ from names of fields in query input relations.

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Relational Algebra

- Operator takes in a relation and output a different relation
- Pure relational algebra has set semantics
  - No duplicate tuples in a relation instance
- Basic operators:
  - Selection (σ) Selects a subset of rows from relation.
  - Projection (π) Deletes unwanted columns from relation.
  - Cross-product ($\times$) Allows us to combine two relations.
  - Set-difference (-) Tuples in reln. 1, but not in reln. 2.
  - Union ($\cup$) Tuples in reln. 1 and in reln. 2.
- Additional operators:
  - Intersection, join, renaming: Not essential, but (very!) useful.
- Since each operator returns a relation, operators can be composed!

Relational Algebra Operators: Unary

- Unary Operators: on **single relation**
- **Projection** (π): Retains only desired columns (vertical)
- **Selection** (σ): Selects a subset of rows (horizontal)
- **Renaming** (ρ): Rename attributes and relations.

Relational Algebra Operators: Binary

- Binary Operators: on **pairs of relations**
- **Union** ($\cup$): Tuples in r1 or in r2.
- **Set-difference** (—): Tuples in r1, but not in r2.
- **Cross-product** ($\times$): Allows us to combine two relations.

Relational Algebra Operators: Compound

- Compound Operators: common "*macros*" for the above
- **Intersection** ($\cap$): Tuples in r1 and in r2.
- **Joins** ($\bowtie_\theta$, $\bowtie$): Combine relations that satisfy predicates

Projection (π)

Deletes attributes that are not in *projection list*.

Schema of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.

No duplicates in result!

Result relation can be the *input* for another relational algebra operation! (*Operator composition.*)

Corresponds to the _____ list in SQL?

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{sname,rating}(S2)$

set semantics

age
35.0
55.5

$\pi_{age}(S2)$

Selection (σ)

- Selects rows that satisfy *selection condition*.
- *Schema* of result identical to schema of (only) input relation.
- No duplicates in result!
- Selects a subset of rows (horizontal)
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)
- Corresponds to the ___ clause in SQL

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$S2$

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$\sigma_{rating>8}(S2)$

sname	rating
yuppy	9
rusty	10

$\pi_{sname,rating}(\sigma_{rating>8}(S2))$

Composing Select and Project

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sname	rating
yuppy	9
rusty	10

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

What about:

$$\sigma_{\text{rating>8}}(\pi_{\text{sname}}(\text{S1}))$$

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

S1

Example Instances

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

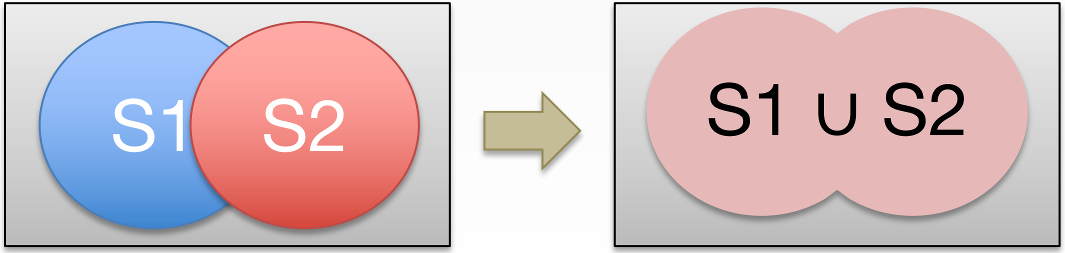
Union (U)

All of these operations take two input relations, which must be *union-compatible*:

- Same number of fields.
- `Corresponding' fields have the same type.

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

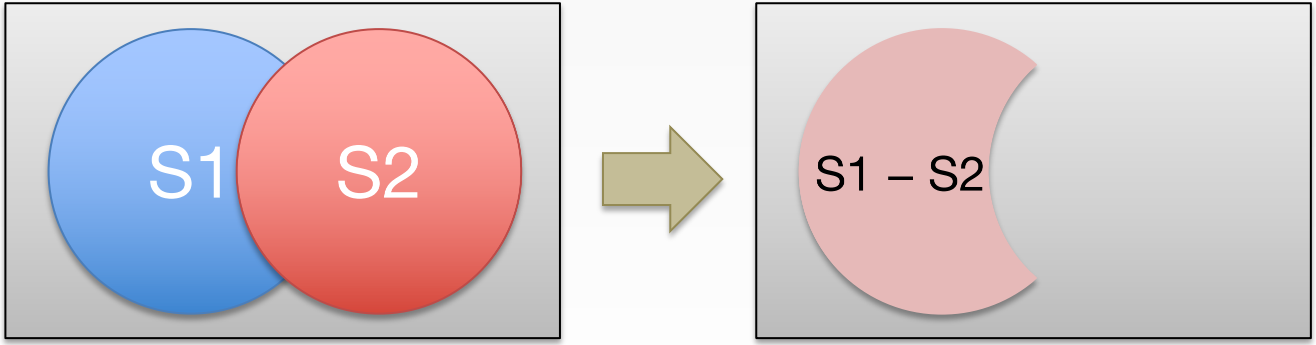


Set Difference (−)

Same as with union, both input relations must be *compatible*.

SQL Expression: EXCEPT

S1 − S2



Set Difference (−), cont.

Relational *Instance* **S1**

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Relational *Instance* **S2**

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S1 − S2

sid	sname	rating	age
22	dustin	7	45

S2 − S1

sid	sname	rating	age
28	yuppy	9	35.0
44	guppy	5	35.0

Cross-Product (×)

- **R1** × **S1:** Each row of **R1** paired with each row of **S1**
- Result schema has one field per field of S1 and R1, with field names `inherited' if possible.
- Conflict: Both S1 and R1 have a field called sid.

R1:

sid	bid	day
22	101	10/10/96
58	103	11/12/96

×

S1:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

=

sid	bid	day	sid	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
22	101	10/10/96	31	lubber	8	55.5
22	101	10/10/96	58	rusty	10	35.0
58	103	11/12/96	22	dustin	7	45.0
58	103	11/12/96	31	lubber	8	55.5
58	103	11/12/96	58	rusty	10	35.0

How many rows in result? |R1|*|S2|
Schema compatability? Not needed.
Duplicates? None generated.

Renaming operator: $\rho\ (C(1 \rightarrow sid1,\ 4 \rightarrow sid2),\ S1 \times R1)$

Renaming (ρ = "rho")

- *Renames relations and their attributes:*
- Note that relational algebra doesn't require names.
  - We could just use positional arguments.

$$\rho(\underbrace{\text{Temp1}}_{\text{Output Relation Name}}(\underbrace{1 \rightarrow \text{sid1}, 4 \rightarrow \text{sid2}}_{\text{Renaming List: position} \rightarrow \text{New Name}}), \underbrace{R1 \times S1}_{\text{Input Relation}})$$

R1 × S1

sid	bid	day	sid	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
22	101	10/10/96	31	lubber	8	55.5
22	101	10/10/96	58	rusty	10	35.0
58	103	11/12/96	22	dustin	7	45.0
58	103	11/12/96	31	lubber	8	55.5
58	103	11/12/96	58	rusty	10	35.0

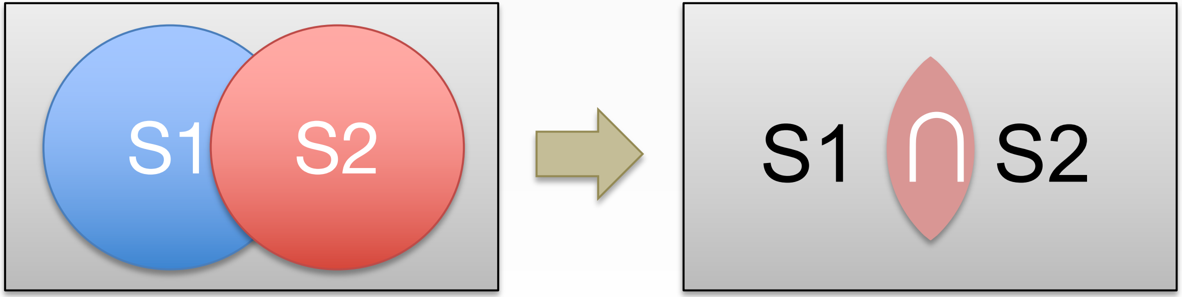
Temp1

sid1	bid	day	sid2	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
22	101	10/10/96	31	lubber	8	55.5
22	101	10/10/96	58	rusty	10	35.0
58	103	11/12/96	22	dustin	7	45.0
58	103	11/12/96	31	lubber	8	55.5
58	103	11/12/96	58	rusty	10	35.0

Intersection

All of these operations take two input relations, which must be *union-compatible*:

- Same number of fields.
- `Corresponding' fields have the same type.
- Equivalent to:
  S1 — (S1 — S2)



Intersection (∩)

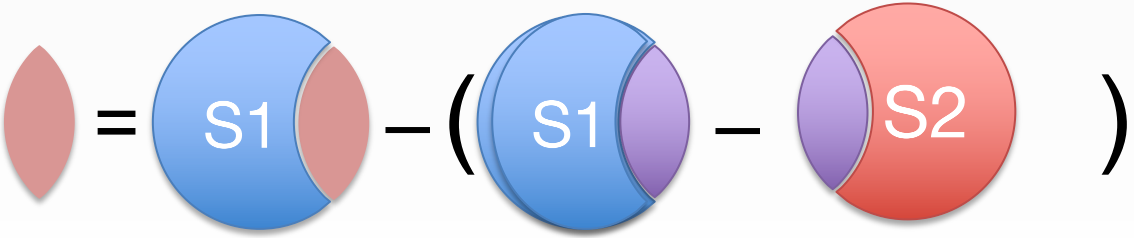
Relational *Instance* **S1**

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Relational *Instance* **S2**

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- S1 ∩ S2 = S1 — (S1 — S2)



S1 ∩ S2

<u>sid</u>	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

Join

- Joins are compound operators (like intersection):
  - Generally, σ_{θ}(R × S)
- Hierarchy of common kinds:
  - **Theta Join** ($\bowtie_{\theta}$): join on **logical expression** θ
    - **Equi-Join:** theta join with theta being a conjunction of equalities
      - **Natural Join** ($\bowtie$): equi-join on all matching column names
- Condition (Theta) Join ⇔ Theta Join ⇔ Condition Join
- Note: we will need to learn a good join algorithm.
- Avoid cross-product if we can!!

Theta Join ($\bowtie_\theta$) Example

- **R1** $\bowtie_{\text{sid=sid}}$ **S1**

R1:

sid	bid	day
22	101	10/10/96
58	103	11/12/96

$\bowtie_{\text{sid=sid}}$

S1:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

=

sid	bid	day	sid	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
58	103	11/12/96	58	rusty	10	35.0

- Note that output needs a rename operator!

Another Theta Join ($\bowtie_\theta$) Example

- **Condition (Theta) Join: R $\bowtie_\theta$ S = σ_θ(R × S)**
- **Example:** *More senior sailors for each sailor.*
- S1 $\bowtie_{f4 < f8}$ S1

S1:

f1	f2	f3	f4
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1				S1			
f1	f2	f3	f4	f5	f6	f7	f8
22	dustin	7	45.0	22	dustin	7	45.0
22	dustin	7	45.0	31	lubber	8	55.5
22	dustin	7	45.0	58	rusty	10	35.0
31	lubber	8	55.5	22	dustin	7	45.0
31	lubber	8	55.5	31	lubber	8	55.5
31	lubber	8	55.5	58	rusty	10	35.0
58	rusty	10	35.0	22	dustin	7	45.0
58	rusty	10	35.0	31	lubber	8	55.5
58	rusty	10	35.0	58	rusty	10	35.0

S1				S1			
sid	sname	rating	age	sid	sname	rating	age2
22	dustin	7	45.0	31	lubber	8	55.5
58	rusty	10	35.0	22	dustin	7	45.0
58	rusty	10	35.0	31	lubber	8	55.5

Equi-Join

Equi-Join: A special case of condition join where the condition *c* contains only ***equalities***.

$$\mathbf{R1} \bowtie_{sid} \mathbf{S1}$$

Theta join with AND of = predicates

Result schema similar to cross-product, but only one copy of fields for which equality is specified.

Equi-Join Example

- R1 $\bowtie_{sid}$ S1

R1:

sid	bid	day
22	101	10/10/96
58	103	11/12/96

$\bowtie_{sid}$

S1:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

=

sid	bid	day	sname	rating	age
22	101	10/10/96	dustin	7	45.0
58	103	11/12/96	rusty	10	35.0

Natural Join (⋈)

- Special case of Equi-join in which equalities are specified for all matching fields and duplicate fields are projected away

$$R \bowtie S = \pi_{\text{unique fld.}}\, \sigma_{\text{eq. matching fld.}}(R \times S)$$

- Compute R × S
- Select rows where fields appearing in both relations have **equal** values
- Project onto the set of all **unique** fields.

Natural Join (⋈) Example

- $R \bowtie S = \pi_{\text{unique fld.}} \sigma_{\text{eq. matching fld.}} (R \times S)$

R1 ⋈ S1

sid	bid	day	sid	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
~~22~~	~~101~~	~~10/10/96~~	~~31~~	~~lubber~~	~~8~~	~~55.5~~
~~22~~	~~101~~	~~10/10/96~~	~~58~~	~~rusty~~	~~10~~	~~35.0~~
~~58~~	~~103~~	~~11/12/96~~	~~22~~	~~dustin~~	~~7~~	~~45.0~~
~~58~~	~~103~~	~~11/12/96~~	~~31~~	~~lubber~~	~~8~~	~~55.5~~
58	103	11/12/96	58	rusty	10	35.0

sid	bid	day	sname	rating	age
22	101	10/10/96	dustin	7	45.0
58	103	11/12/96	rusty	10	35.0

R1:

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

S1:

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Joins Examples

Sid	Sname	Rating	Age
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

Sailors

Bid	Bname	Color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	Green
104	Marine	Red

Boats

Sid	Bid	Day
22	101	10/10/96
58	103	11/12/96

Reserves

Find names of sailors who've reserved boat #103

Solution 1: $\pi_{sname}((\sigma_{bid=103}\text{Reserves}) \bowtie \text{Sailors})$

Solution 2: $\rho(\text{Temp1},(\sigma_{bid=103}\text{Reserves})$

$\rho(\text{Temp2},(\text{Temp1} \bowtie \text{Sailors})$

$\pi_{sname}(\text{Temp2})$

Solution 3: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie \text{Sailors}))$

Find names of sailors who've reserved a red boat

Information about boat color only available in Boats; so need an extra join:

$\pi_{sname}((\sigma_{color='red'}Boats) \bowtie Reserves \bowtie Sailors))$

- A (slightly) more efficient solution:
  $\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'}Boats) \bowtie Reserves) \bowtie Sailors)$

Find sailors who've reserved a red or a green boat

Can identify all red or green boats, then find sailors who've reserved one of these boats:

$\rho(\text{Tempboats}, (\sigma_{color='red' \vee color='green'} \text{Boats}))$

$\pi_{sname}(\text{Tempboats} \bowtie \text{Reserves} \bowtie \text{Sailors})$

- Can also define Tempboats using union!
- What happens if $\vee$ is replaced by $\wedge$ in this query?

Find sailors who've reserved a red and a green boat

Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors):

$\rho(\text{Tempred}, \pi_{sid}((\sigma_{color='red'}\text{Boats}) \bowtie \text{Reserves}))$

$\rho(\text{Tempgreen}, \pi_{sid}((\sigma_{color='green'}) \bowtie \text{Reserves}))$

$\pi_{sname}((\text{Tempred} \cap \text{Tempgreen}) \bowtie \text{Sailors})$

An Example of a “Rewrite”: Push-Down

R1:

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

- Want reservations for sailors whose age > 40

$$\sigma_{age > 40} (R1 \bowtie S1)$$

sid	bid	day	sname	rating	age
22	101	10/10/96	dustin	7	45.0
~~58~~	~~103~~	~~11/12/96~~	~~rusty~~	~~10~~	~~35.0~~

Q: Any other expressions?

Another equiv. exp: $R1 \bowtie \sigma_{age > 40} S1$

➔ This may be cheaper to compute!

An Example of a "Rewrite": Eliminating Nesting

- *Names of sailors who've **not** reserved boat #103:*

One approach:

$$\pi_{sname}\, \mathbf{R} - \pi_{sname}\, ((\sigma_{bid=103}\, \mathbf{R}) \bowtie S))$$

R:

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Extended Relational Algebra

- Group By / Aggregation Operator (γ):
  - $\gamma_{\text{age, AVG(rating)}}$(Sailors)
  - With selection (HAVING clause):
    - $\gamma_{\text{age, AVG(rating), COUNT(*)>2}}$(Sailors)

- Textbook uses two operators:
  - GROUP BY age, AVG(rating) (Sailors)
  - HAVING COUNT(*)>2
    (GROUP BY age, AVG(rating)(Sailors))

Relational Algebra Summary

- Relational Algebra: a small set of operators mapping relations to relations
  - Operational, in the sense that you specify the explicit order of operations
  - A closed set of operators! Mix and match.
- Basic ops include: σ, π, $\times$, $\cup$, —
- Important compound ops: $\cap$, $\bowtie$

Summary

The relational model has rigorously defined query languages that are simple and powerful.

Relational algebra is more operational; useful as internal representation for query evaluation plans.

Several ways of expressing a given query; a query optimizer should choose the most efficient version.

Summary

Relational calculus is non-operational, and users define queries in terms of what they want, not in terms of how to compute it. (Declarative)

Algebra and safe calculus have same expressive power, leading to the notion of relational completeness.

Reading and Next Class

- The Relational Model and Relational Algebra
  - Ch3, Ch4.1 – 4.2
- Next: Entity/Relationship Models I
  - Ch 2