COMP353 Databases

Relational Algebra (RA) for Relational Data Model

Relational Algebra (RA)

- *Database Query languages* are specialized languages to ask for information (**queries**) in DB.
- **Relational Algebra (RA)** is a query language associated with the relational data model.
- Queries in RA are expressions using a collection of operators on relations in the DB.
- The input(s) and output of a RA query are relations
- A query is ***evaluated*** using the ***current instance*** of the input relations to produce the output

Operations in “standard” RA

- The well-known set operations
  - √ **Union ($\cup$)**
  - √ **Intersection ($\cap$)**
  - √ **Difference ($-$)**
- Special DB operations that select “parts” of a relation instance
  - **Selection (σ)** – selects some rows (tuples) & discards the rest
  - **Projection (π)** – selects some columns (attributes) & discards the rest
- Operations that “combine” the tuples from the argument relations
  - √ **Cartesian product ($\times$)** **–** pairs the tuples in all possible ways
  - **Join ($\bowtie$)** – pairs particular tuples from the two input relations
- A unary operation to ***rename*** relations, called **Rename (ρ)**

Note: The output of a RA expression is an “unnamed” relation/set, i.e., **RA** expressions return **sets**, whereas **SQL** returns **multisets** (**bags**)

Compatibility Requirement

- We can apply the **set operators** of **union, intersection,** and **difference** to instances of relations **R** and **S** if **R** and **S** are *compatible*, that is, they have “the same” schemas.
- **Definition**: Relations $\mathbf{S(A_1,\ldots,A_n)}$ and $\mathbf{R(B_1,\ldots,B_m)}$ are compatible if:

  (1) n=m and

  (2) type($\mathbf{A_i}$) **=** type($\mathbf{B_i}$) (or compatible types), for all $\mathbf{1 \leq i \leq n}$**.**

Set Operations on Relations

Let **R** and **S** be relation schemas, and **r** and **s** be any instances of them.

- The **union** of **r** and **s** is the set of all tuples that appear in either one or both. Each tuple **t** appears only once in the union, even if it appears in both; $\mathbf{r} \cup \mathbf{s} = \{t \mid t \in \mathbf{r} \vee t \in \mathbf{s}\}$
- The **intersection** of **r** and **s**, is the set of all tuples that appear in both; $\mathbf{r} \cap \mathbf{s} = \{t \mid t \in \mathbf{r} \wedge t \in \mathbf{s}\}$
- The **difference** of **r** and **s**, is the set of all tuples that appear in **r** but not in **s**; $\mathbf{r} - \mathbf{s} = \{t \mid t \in \mathbf{r} \wedge t \notin \mathbf{s}\}$
  - Commutative operations; **r Op s** = **s Op r**

    Note: Set difference (–) is not commutative, i.e., ($\mathbf{r} - \mathbf{s} \neq \mathbf{s} - \mathbf{r}$)

Example

Relation Schema: **Star** (<u>name</u>, address, gender, birthdate)

Instance r of Star:

Name	Address	Gender	Birthdate
Carrie Fisher	123 Maple	F	9/9/99
Mark Hamill	456 Oak rd.	M	8/8/88

Instance s of Star:

Name	Address	Gender	Birthdate
Carrie Fisher	123 Maple	F	9/9/99
Harrison Ford	789 Palm rd.	M	7/7/77

$r \cup s$:

Name	Address	Gender	Birthdate
Carrie Fisher	123 Maple	F	9/9/99
Mark Hamill	456 Oak rd.	M	8/8/88
Harrison Ford	789 Palm rd.	M	7/7/77

Example

Relation Schema: **Star** (<u>name</u>, address, gender, birthdate)

Instance r of Star:

Name	Address	Gender	Birthdate
Carrie Fisher	123 Maple	F	9/9/99
Mark Hamill	456 Oak rd.	M	8/8/88

Instance s of Star:

Name	Address	Gender	Birthdate
Carrie Fisher	123 Maple	F	9/9/99
Harrison Ford	789 Palm rd.	M	7/7/77

$r \cap s$:

Name	Address	Gender	Birthdate
Carrie Fisher	123 Maple	F	9/9/99

Example

Relation Schema: **Star** (<u>name</u>, address, gender, birthdate)

Instance r of Star:

Name	Address	Gender	Birthdate
Carrie Fisher	123 Maple	F	9/9/99
Mark Hamill	456 Oak rd.	M	8/8/88

Instance s of Star:

Name	Address	Gender	Birthdate
Carrie Fisher	123 Maple	F	9/9/99
Harrison Ford	789 Palm rd.	M	7/7/77

r – s:

Name	Address	Gender	Birthdate
Mark Hamill	456 Oak rd.	M	8/8/88

Example

Relation Schema: **Star** (<u>name</u>, address, gender, birthdate)

Instance r of Star:

Name	Address	Gender	Birthdate
Carrie Fisher	123 Maple	F	9/9/99
Mark Hamill	456 Oak rd.	M	8/8/88

Instance s of Star:

Name	Address	Gender	Birthdate
Carrie Fisher	123 Maple	F	9/9/99
Harrison Ford	789 Palm rd.	M	7/7/77

s – r:

Name	Address	Gender	Birthdate
Harrison Ford	789 Palm rd.	M	7/7/77

Projection (π)

- Let **R** be a relation schema.
- The **projection** operation (π) is used to produce, from any instance **r** of **R,** a new relation that includes listed "columns" of **R**
- The output of $\pi_{A1, A2,\ldots,Aj}$ (**r**) is a relation with columns $A_1, A_2,\ldots, A_j$, in this order.
- Note: The subscript of π is a *list*, which defines the structure of the output as the ordered tuple ($A_1, A_2,\ldots, A_j$).

Example

Relation Schema: **Movie**(<u>title</u>, <u>year</u>, length, filmType, studioName, producer)

Instance movie Of Movie:

title	year	length	filmType	studioName	producer
Star wars	1977	124	color	Fox	12345
Mighty Ducks	1991	104	color	Disney	67890
Wayne's World	1992	95	color	Paramount	99999

Query: $\pi_{\text{title, year, length}}(\text{movie})$

title	year	length
Star wars	1977	124
Mighty Ducks	1991	104
Wayne's World	1992	95

Example

Relation Schema: **Movie**(<u>title</u>, <u>year</u>, length, filmType, studioName, producer)

Instance movie Of Movie:

title	year	length	filmType	studioName	producer
Star wars	1977	124	color	Fox	12345
Mighty Ducks	1991	104	color	Disney	67890
Wayne's World	1992	95	color	Paramount	99999

Query: $\pi_{\text{filmType}}(\text{movie})$

Result:

filmType
color

Selection (σ)

- The **selection** operator (**σ**), applied to an instance **r** of relation **R**, returns a subset of **r**
- We denote this operation/query by $\boldsymbol{\sigma_C(r)}$
- The output includes tuples satisfying condition **C**
- The schema of the output is the same as **R**

Example

Relation Schema: **Movie**(<u>title</u>, <u>year</u>, length, filmType, studioName, producer)

Instance movie of Movie:

title	year	length	filmType	studioName	producer
Star wars	1977	124	color	Fox	12345
Mighty Ducks	1991	104	color	Disney	67890
Wayne's World	1992	95	color	Paramount	99999

Query: $\sigma_{length \geq 100}(movie)$

Result:

title	year	length	filmType	studioName	producer
Star wars	1977	124	color	Fox	12345
Mighty Ducks	1991	104	color	Disney	67890

Example

Relation: **Movie**(<u>title</u>, <u>year</u>, length, filmType, studioName, producer)

Instance movie of Movie:

title	year	length	filmType	studioName	producer
Star wars	1977	124	color	Fox	12345
Mighty Ducks	1991	104	color	Disney	67890
Wayne's World	1992	95	color	Paramount	99999

Query: $\sigma_{\text{length} \geq 100 \text{ AND studioName} = \text{'Fox'}}(\text{movie})$

Result:

title	year	length	filmType	studioName	producer
Star wars	1977	124	color	Fox	12345

Cartesian Product (×)

- Let **R** and **S** be relation schemas, and **r** and **s** be any instances of **R** and **S**, respectively.
- The **Cartesian Product** of **r** and **s** is the set of all tuples obtained by "concatenating" the tuples in **r** and **s**. Formally, $\mathbf{r} \times \mathbf{s} = \{ t_1.t_2 \mid t_1 \in r \wedge t_2 \in s \}$
- The schema of result is the "union" of **R** and **S**
  - If **R** and **S** have some attributes in common, we need to invent new names for identical names, e.g., use **R.B** and **S.B**, if **B** appears in both **R** and **S**

Example

Instance r of R:

A	B
1	2
3	4

Instance s of S:

B	C	D
2	5	6
4	7	8
9	10	11

$r \times s$:

A	R.B	S.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Theta-join (θ)

- Suppose **R** and **S** are relation schemas, **r** is an instance of **R**, and **s** is an instance of **S**. The **theta-join** of **r** and **s** is the set of all tuples obtained from concatenating all $t_1 \in \mathbf{r}$ and $t_2 \in \mathbf{s}$, such that t_1 and t_2 *satisfy some condition* **c**
- We denote θ-join by $\mathbf{r} \rhd\lhd_{\mathbf{c}} \mathbf{s}$
- The schema of the result is the same as the schema of $\mathbf{R} \times \mathbf{S}$ (i.e., the union of **R** and **S**)
- **c** is a Boolean expression, simple or complex, as in operation $\boldsymbol{\sigma}$

Example

Instance r of R:

A	B	C
1	2	3
6	5	8
9	7	11

Instance s of S:

B	C	D
2	3	4
2	3	5
7	8	10

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

A	R.B	R.C	S.B	S.C	D
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	5	8	7	8	10
9	7	11	7	8	10

Equi-join

- The **equi-join** operator, is a special case of θ-join, in which we may only use the equality relation (=) in condition **C**
- It is denoted as $\mathbf{r} \rhd\lhd_{\mathbf{c}} \mathbf{s}$ (i.e., the same as θ-join)
- The schema of the output is the same as that of θ-join

Example

Instance r of R:

A	B	C
1	2	3
6	5	8
9	7	11

Instance s of S:

B	C	D
2	3	4
2	3	5
7	8	10

$r \bowtie_{R.C\,=\,S.C}$ s:

A	R.B	R.C	S.B	S.C	D
1	2	3	2	3	4
1	2	3	2	3	5
6	5	8	7	8	10

Natural Join (⊳⊲)

- **Natural join**, is a special case of equi-join, where the equalities are not explicitly specified, rather they are assumed implicitly on the common attributes of **R** and **S**
- We denote this natural join operation by **r** ⊳⊲ **s**
- The schema of the output is similar to that of equi-join, except that each common attribute appears only once.

Note: If **R** and **S** do not have any common attribute, then the join operation becomes Cartesian product.

Example

Instance r of R:

A	B	C
1	2	3
6	2	8
9	7	3

Instance s of S:

B	C	D
2	3	4
2	3	5
7	8	10

r ▷◁ s:

A	B	C	D
1	2	3	4
1	2	3	5

Expressing Queries in RA

- Every standard RA operation has relation(s) as argument(s) and produces a relation (set) as the output

(Exception is the sort operator τ)

- This property of RA operations (that inputs and outputs are relations) makes it possible to formulate/express any query by *composing*/*nesting*/*grouping* subqueries.
- We can use parentheses for *grouping, in order* to improve clarity and readability

Example: RA Query

- Relation schema:

  Movie (title, year, length, filmType, studioName)

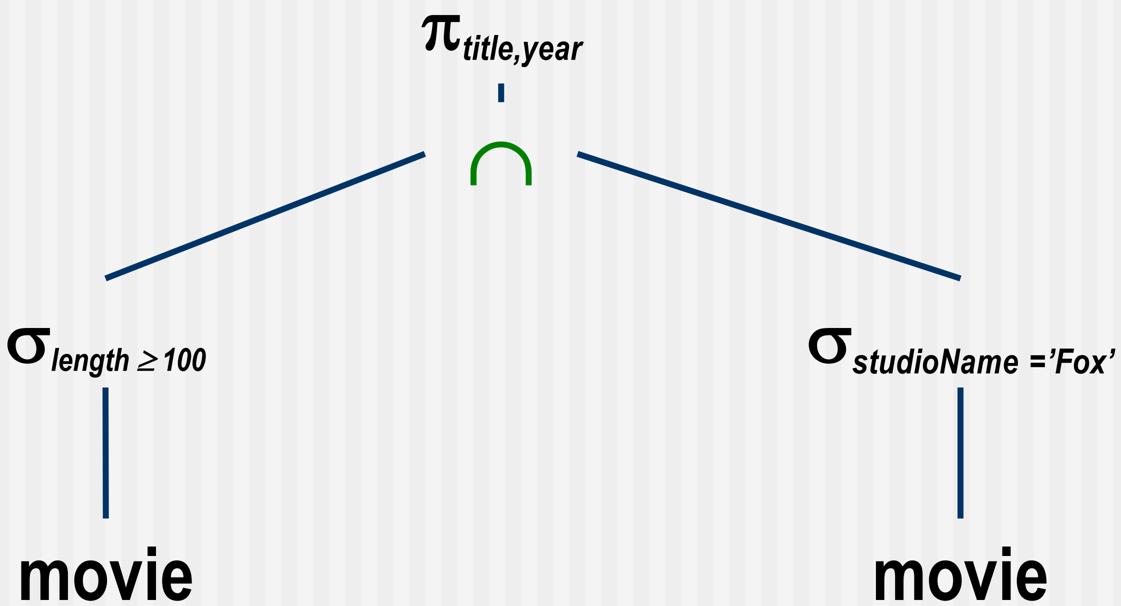
- Query: List the title and year of every movie made by **Fox** studio whose length is at least 100 minutes?
- One way to express this query in RA is:

  $\pi_{title,year}(\sigma_{studioName = 'Fox' \text{ and } length >= 100}(\text{movie}))$

- Another way:
  - Select those **movie** tuples that have *length* ≥100
  - Select those **movie** tuples that have *studioName* = 'Fox'
  - Find the intersection of the above two results
  - Then project on the attributes **title** and **year**

Example: RA Query

$$\pi_{title,year}(\sigma_{studioName = 'Fox' \text{ and } length >= 100}(\text{movie}))$$



$$\pi_{title,year}(\sigma_{length\geq 100}(\text{movie}) \cap \sigma_{studioName = 'Fox'}(\text{movie}))$$

Example: RA Query

- Relation schema:

  Movie (title, year, length, filmType, studioName)

  StarsIn (title, year, starName)

- Query: List the **names** of the stars of movies of length ≥ 100 minutes long.
- One expression in RA for this query:
  - Select **movie** tuples of ***length* ≥ 100**
  - Join the result with relation **StarsIn**
  - Project on the attribute ***starName***
- ***Exp1***: $\pi_{starName}(\sigma_{length \geq 100}(\textbf{movie}) \rhd\lhd \textbf{starsIn})$
- ***Another solution***: $\pi_{starName}(\sigma_{length \geq 100}(\textbf{movie} \rhd\lhd \textbf{starsIn}))$

Renaming Operator (ρ)

- To **control** manipulating the names of the attributes in formulating queries in relational algebra, we may need **renaming** of relations. May do this for *convenience too*
- **The Renaming Operator** is denoted by $\rho_{s(A1,A2,\ldots,An)}(r)$
- The result is a copy of the input relation instance **r**, but renamed to **s** and its attributes to A1, …, An, in that order.
- Use $\rho_s(r)$ to give relation **r** a new name **s (with the same attributes in r)**

  That is, in this case, schema of **s** is the same as that of **r**.

Example

- Query: $\pi_{\textit{starName}}(\sigma_{\textit{length} \geq 100}(\textbf{movie}) \rhd\lhd \textbf{starsIn})$
- This query can be rewritten in 2 steps as follows:

  1. $\rho_{\textbf{M(title, year, length, filmType, studioName)}}(\sigma_{\textit{length}\geq 100}(\textbf{movie}))$

     or even simpler as: $\rho_{\textbf{M}}(\sigma_{\textit{length}\geq 100}(\textbf{movie}))$ if used in the same formula

  2. Or use M := $\sigma_{\textit{length}\geq 100}$(movie) as a separate formula and then formulate the query as: $\pi_{\textit{starName}}(\textbf{M} \rhd\lhd \textbf{starsIn})$

- Consider **takes**(<u>sid</u>, <u>cid</u>, grade)
- Query: Find ID of every student who has taken at least 2 courses.
- $\pi_{\text{takes.sid}}(\sigma_{(\text{takes.sid} = \text{T.sid}) \text{ and } (\text{takes.cid} \neq \text{T.cid})}(\text{takes} \times \rho_{\text{T}}(\text{takes})))$

Dependent and Independent Operations

- Some RA operations can be expressed based on other operations. Examples include:
  - $\mathbf{r} \cap \mathbf{s} = \mathbf{r} - (\mathbf{r} - \mathbf{s})$
  - $\mathbf{r} \bowtie_C \mathbf{s} = \sigma_C (\mathbf{r} \times \mathbf{s})$
  - $\mathbf{r} \bowtie \mathbf{s} = \pi_L(\sigma_{\text{r.A1} = \text{s.A1 AND… AND r.An} = \text{s.An}} (\mathbf{r} \times \mathbf{s}))$,

    where **L** is the list of attributes in **R** *followed* by those attributes in **S** that are not in **R,** and **A1**,…, **An** are the common attributes of **R** and **S**

Relational Algebra with Bag Semantics

- Relations stored in DB are called **base** relations/tables.
- Base relations are normally sets; no duplicates.
- In some situations, e.g., during query processing, it is allowed for relations to have duplicate tuples.
- If duplicates are allowed in a collection, it is called ***bag/multiset***.

Instance r of R:

A	B	C
1	2	3
6	5	8
6	5	8
1	2	3
9	7	11

Here, r is a bag

Why Bags?

- **1. Faster projection operations**
  - Bag projection is faster, since otherwise returning distinct values is expensive (as we need sorting for duplicate elimination.

    Another example: *Computing the bag union* (**r** $\cup^B$ **s**) *is much che*aper than computing the standard set union **r** $\cup$ **s**.

    Formally, if **r** and **s** have *n* and m tuples, then the bag and set union operations will cost **O**($n+m$) and **O**($n*m$), respectively.
- **2. Correct computation** with some aggregation
  - For example, to compute the *average* of values for attribute A in the previous relation, we must consider the bag of those values

Set Operations on Bags

- $\mathbf{r} \cup^{B} \mathbf{s}$, the **bag union** of **r** and **s**, is the bag of tuples that are in **r**, in **s,** or in both. If a tuple *t* appears *n* times in **r**, and *m* times in **s**, then *t* appears *n+m* times in bag $\mathbf{r} \cup^{B} \mathbf{s}$

  $$\mathbf{r} \cup^{B} \mathbf{s} = \{\ \mathrm{t{:}k} \mid t{:}n \in \mathbf{r} \land t{:}m \in \mathbf{s} \land k = n+m\ \}$$

- $\mathbf{r} \cap^{B} \mathbf{s}$, the **bag intersection** of **r** and **s**, is the bag of tuples that appear in both **r** and **s.** If a tuple *t* appears *n* times in **r**, and *m* times in **s**, then the number of occurrences of t in bag $\mathbf{r} \cap^{B} \mathbf{s}$ is *min(n,m)*

  $$\mathbf{r} \cap^{B} \mathbf{s} = \{\ \mathrm{t{:}k} \mid t{:}n \in \mathbf{r} \land t{:}m \in \mathbf{s} \land k = min(n,m)\ \}$$

- $\mathbf{r} -^{B} \mathbf{s}$, the **bag difference** of **r** and **s** is defined as follows:

  $$\mathbf{r} -^{B} \mathbf{s} = \{\ \mathrm{t{:}k} \mid t{:}n \in \mathbf{r} \land t{:}m \in \mathbf{s} \land k = max(0,\ n-m)\ \}$$

  $$\mathbf{s} -^{B} \mathbf{r} = \{\ \mathrm{t{:}k} \mid t{:}n \in \mathbf{r} \land t{:}m \in \mathbf{s} \land k = max(0,\ m-n)\ \}$$

Example

Bag r:

A	B
1	2
3	4
1	2
1	2

Bag s:

A	B
1	2
3	4
3	4
5	6

$r \cup^B s$:

A	B
1	2
3	4
1	2
1	2
1	2
3	4
3	4
5	6

Example

Bag r:

A	B
1	2
3	4
1	2
1	2

Bag s:

A	B
1	2
3	4
3	4
5	6

$r \cap^B s$:

A	B
1	2
3	4

Example

Bag r:

A	B
1	2
3	4
1	2
1	2

Bag s:

A	B
1	2
3	4
3	4
5	6

$r -_B s$:

A	B
1	2
1	2

Example

Bag r:

A	B
1	2
3	4
1	2
1	2

Bag s:

A	B
1	2
3	4
3	4
5	6

$s -^{B} r$:

A	B
3	4
5	6

Bag Projection π^B

- Let **R** be a relation scheme, and **r** be a collection of tuples over **R**, which could have duplicates.

  The **bag projection** operator is used to produce, from **r,** a bag of tuples over some of **R.**
- Even when **r** does not have duplicates, we may get duplicates when projecting on some attributes of **R**.

  That is, π^B does not eliminate the duplicates and hence corresponds exactly to the SELECT clause in SQL.

Example

Bag r:

A	B	C
1	2	5
3	4	6
1	2	7
1	2	8

$\pi^{B}_{A,\,B}\,(r)$:

A	B
1	2
3	4
1	2
1	2

Example

Relation Schema: **movie**(title, year, length, filmType, studioName, producer)

Instance:

title	year	length	filmType	studioName	producer
Star wars	1977	124	color	Fox	12345
Mighty Ducks	1991	104	color	Disney	67890
Wayne's World	1992	95	color	Paramount	99999

$\pi^{B}_{filmType}$**(movie):**

filmType
color
color
color

Selection on Bags

- The selection operator σ_c applied to an instance **r** of relation **R** will return a subset of **r**
  - The tuples returned are those that satisfy the specified condition **C** (which involves attributes of **R**)
  - Duplicates are **not** eliminated from the result of a bag-selection

Note: The selection operation σ in RA is different from the **SELECT** clause in SQL

Example

Bag r:

A	B	C
1	2	5
3	4	6
1	2	7
1	2	7

$\sigma_{C \geq 6}(r)$:

A	B	C
3	4	6
1	2	7
1	2	7

Cartesian Product of Bags

- The **Cartesian Product** of bags **r** and **s** is the bag of tuples that can be formed by concatenating pairs of tuples, the first of which comes from **r** and the second from **s**. In symbols, $\mathbf{r} \times \mathbf{s} = \{ t_1.t_2 \mid t_1 \in \mathbf{r} \wedge t_2 \in \mathbf{s} \}$
  - Each tuple of one relation is paired with each tuple of the other, regardless of whether it is a duplicate or not
  - If a tuple t_1 appears ***m*** times in a relation **r**, and a tuple t_2 appears ***n*** times in relation **s**, then tuple $t_1.t_2$ appears **m*n** times in their bag-product, $\mathbf{r} \times \mathbf{s}$

Example

Bag r:

A	B
1	2
1	2

Bag s:

B	C
2	3
4	5
4	5

$r \times s$:

A	R.B	S.B	C
1	2	2	3
1	2	2	3
1	2	4	5
1	2	4	5
1	2	4	5
1	2	4	5

Join of Bags

- The bag join is computed in the same way as the standard join operation
- Duplicates are not eliminated in a bag join operation

Bag r:

A	B
1	2
1	2

Bag s:

B	C
2	3
4	5

r ⋈ s:

A	B	C
1	2	3
1	2	3

Constraints on Relations

- RA offers a convenient way to express a wide variety of constraints, e.g., referential integrity and FD's.
- There are **two ways** to express constraints in RA

1. If **r** is an expression in RA, then the constraint **r** = $\emptyset$ says:
   "**r** *has no tuples, i.e., or* **r** *is empty*"
2. If **r** and **s** are RA *expressions*, then the constraint **r** $\subseteq$ **s** says:
   "*every tuple in (the result of)* **r** *is in (the result of)* **s**"

These constraints hold also when **r** and **s** are bags.

Constraints on Relations

- Note that these two types of constraints are not independent. Why?
  - The constraint $\mathbf{r} \subseteq \mathbf{s}$ could also be written as $\mathbf{r} - \mathbf{s} = \varnothing$

This follows from the definition of "–", because $\mathbf{r} \subseteq \mathbf{s}$ **iff** $\mathbf{r} - \mathbf{s} = \varnothing$, meaning that there is no tuple in **r** which is not in **s**

Referential Integrity Constraints

- Referential integrity in relational data model means:
  - if there is a value ***v*** in a tuple *t* in a relation **r**,

    then it is expected that ***v*** appears in a particular component (attribute) of some tuple s in relation **s**

  E.g., if tuple *(s,c,g)* is in table ***takes****(sid,cid,grade),* then there must be a **student** with *sid* = *s* and a **course** with *cid* = *c* such that *s* has taken *c*

  IOW, the mentions of values s *and* c *in* ***takes*** *"refers" to some values outside this relation, and these values* ***must*** *exist*

Example

- Relation schemas:

  Movie (<u>title</u>, <u>year</u>, length, filmType)

  StarsIn (<u>title</u>, <u>year</u>, <u>starName</u>)

- Constraint:

  the **title** and **year** of every movie that appears in relation **starsIn** *must* appear also in **movie**; otherwise there is a violation in referencing in **starsIn**

- Query in RA:

  - $\pi_{\text{title, year}}$ **(starsIn)** $\subseteq \pi_{\text{title, year}}$ **(movie)**

  or equivalently

  - $\pi_{\text{title, year}}$ **(starsIn)** $- \pi_{\text{title, year}}$ **(movie)** $= \varnothing$

Functional Dependencies

- **Any** functional dependency X $\longrightarrow$ Y can be expressed as an expression in RA
- Example:

  Consider the relation schema:

  Star (<u>name</u>, address, gender, birthdate)
- How to express the FD: **name $\rightarrow$ address** in RA?

Functional Dependencies

- Relation schema:
  Star (<u>name</u>, address, birthdate)
- With the FD: **name → address**

- The **idea** is that if we construct all pairs of **star** tuples, we **must not** find a pair that agree on **name** but disagree on **address**
- To “construct” the pairs in RA, we use **Cartesian product**, and to find pairs that violate this FD, we use **selection**
- We are then ready to express this **FD** by equating the result to **ø,** as follows…

Example

Star:

Name	Address	Birthdate
Carrie Fisher	123 Maple	9/9/99
Mark Hamill	456 Oak rd.	8/8/88
Harrison Ford	789 Palm rd.	7/7/77

$\rho_{S1(name,\ address,\ birthdate)}(\textbf{star})$

Name	Address	Birthdate
Carrie Fisher	123 Maple	9/9/99
Mark Hamill	456 Oak rd.	8/8/88
Harrison Ford	789 Palm rd.	7/7/77

$\rho_{S2(name,\ address,birthdate)}(\textbf{star})$

Name	Address	Birthdate
Carrie Fisher	123 Maple	9/9/99
Mark Hamill	456 Oak rd.	8/8/88
Harrison Ford	789 Palm rd.	7/7/77

Example

$s1 \times s2$:

S1.Name	S1.Address	S1.Birthdate	S2.Name	S2.Address	S2.Birthdat
Carrie Fisher	123 Maple	9/9/99	Carrie Fisher	123 Maple	9/9/99
Carrie Fisher	123 Maple	9/9/99	Mark Hamill	456 Oak rd.	8/8/88
Carrie Fisher	123 Maple	9/9/99	Harrison Ford	789 Palm rd.	7/7/77
Mark Hamill	456 Oak rd.	8/8/88	Carrie Fisher	123 Maple	9/9/99
Mark Hamill	456 Oak rd.	8/8/88	Mark Hamill	456 Oak rd.	8/8/88
Mark Hamill	456 Oak rd.	8/8/88	Harrison Ford	789 Palm rd.	7/7/77
Harrison Ford	789 Palm rd.	7/7/77	Carrie Fisher	123 Maple	9/9/99
Harrison Ford	789 Palm rd.	7/7/77	Mark Hamill	456 Oak rd.	8/8/88
Harrison Ford	789 Palm rd.	7/7/77	Harrison Ford	789 Palm rd.	7/7/77

$\sigma_{S1.name=S2.name \text{ AND } S1.address \neq S2.address}(s1 \times s2) = \varnothing$

Functional Dependencies

- Relation schema:
  Star (<u>name</u>, address, birthdate)
- With the FD: **name → address**
- In RA:

$$\sigma_{S1.name=S2.name \text{ AND } S1.address \neq S2.address}(\rho_{S1}(star) \times \rho_{S2}(star)) = \emptyset$$

Domain Constraints

- Relation schema:
  Star (<u>name</u>, address, gender, birthdate)
- How to express the following constraint?
  Valid values for **gender** are **'F'** and **'M'**

- In RA:
  - $\sigma_{gender \neq 'F' \text{ AND } gender \neq 'M'}\textbf{(star)} = \varnothing$
  - This is an example of ***domain constraints***

Domain Constraints

- Relation schema:
  Employee (eid, name, address, salary)
- How to express the constraint:
  Maximum employee salaries is $150,000

- In RA:
  - $\sigma_{salary > 150000}$ **(employee)** $= \varnothing$

"For All" Queries (1)

- Given the database schema:

  Student(Sid, Sname, Addr)

  Course(Cid, Cname, Credits)

  Enrolled (<u>Sid</u>, <u>Cid</u>)
- Consider the query:

  "*Find students enrolled in **all** the courses.*"
- A first attempt (below) fails!

  $$\pi_{Sid}(\text{Enrolled})$$
- This RA query returns those students enrolled in some courses.
- So, how to correctly express "For All" types of queries?

"For All" Queries (2)

- A *solution strategy* would be to:
  - First find the list of “all” students (**all guys**), from which we then subtract those who have not taken at least a course (**bad guys**)
- Then “good guys”, would be “all guys” from which we remove the “bad guys”, i.e.,

Answer (**Good guys**) = **All guys** – **Bad guys**

"For All" Queries (3)

- Set of all students that we should consider:

  All Courses $:=$ π_{Cid} **(Course)**

  All Students $:=$ π_{Sid} **(Student)**

- Steps to find students not enrolled in all courses
  1. Create all possible "student-course" pairs:

     All: Student-Course Pairs $:=$ π_{Sid} **(Student)** $\times$ π_{Cid} **(Course)**

  2. Extract the "actual" student-course pairs from **Enrolled**
  3. Using 1 & 2, we then find students not enrolled in all courses:

     Bad : $\pi_{Sid}(\pi_{Sid}(\text{Student}) \times \pi_{Cid}(\text{Course}) - \text{Enrolled})$

- **Answer: All − Bad**

The Division Operation (÷)

- The previous query can be expressed in RA using the ***division*** operator ÷
  - Divide Enrolled by π_{Cid}(Course)

    that is, Enrolled ÷ π_{Cid}(Course)
  - Schema of the result is {Sid, Cid} − {Cid}
- R ÷ S requires that the attributes of S to be a subset of the attributes of R.
  - The schema of the output would be R - S

Example: Enrolled (student, sport)

Find students enrolled in all sports {Hockey, Football}.

Enrolled (Student, sport)

Jim	Hockey
Joe	Football
Jim	Football
Sue	Hockey

Example: Enrolled(student, sport)

$\pi_{student}(\text{Enrolled}) \times \pi_{sport}(\text{Enrolled}) \quad - \quad \text{Enrolled} \quad =$

Jim	Hockey
Jim	Football
Joe	Hockey
Joe	Football
Sue	Hockey
Sue	Football

$-$

Jim	Hockey
Joe	Football
Jim	Football
Sue	Hockey

$=$

Joe	Hockey
Sue	Football

$\pi_{student}(\text{Enrolled}) -$
$\pi_{student}\left((\pi_{student}(\text{Enrolled}) \times \pi_{sport}(\text{Enrolled})) - \text{Enrolled} \right)$

All

Jim
Joe
Sue

$-$

Bad

Joe
Sue

$= ?$ ➔ Jim is the only student enrolled in all sports

Another Example

- $r \div s = \pi_{R-S}(r) - \pi_{R-S}(\pi_{R-S}(r) \times s - r))$
- Consider the following DB schema in a banking application:
  - Customer(cid, name)
  - Branch(bid, district)
  - Account(cid, bid)
- Query: "*Find the names of those customers who have an account in every branch in the Westmount area*"
- Solution?
  - π_{name} **(Customer $\bowtie$ Account $\bowtie$ ($\sigma_{district = "Westmount"}$**(Branch)**))**)?
- **No**, this query returns those customers who have an account at some branch in Westmount, but not necessarily at all such branches.

Database:

Customer(cid, name),   Branch(bid, district), Account(cid, bid)

- We can use the division operator $\div$
  - Find all customer-branch pairs **(cid, bid)** for which customer (cid) has an account at branch (bid):
    $\pi_{cid,bid}$ **(customer $\bowtie$ account)**
  - Divide the above by bid's of all branches in Westmount
    π_{bid} ($\sigma_{district = "Westmount"}$**(branch))**
- **That is:** π_{name} **((customer $\bowtie$ account)$\div$** π_{bid} ($\sigma_{district = "Westmount"}$**(branch)))**
- The division "r $\div$ s" of relation r by s is defined as:
  $r \div s = \pi_{R-S}(r) - \pi_{R-S}(\pi_{R-S}(r) \times s - r)$