Relational Algebra (Part 1)

Two Questions

1. What is the difference be **declarative** languages?

2. What is **the foundation for relational que** like SQL?

Why Relational Algebra?

- SQL is a (fairly) **declarative** query language:
  - SQL queries describe the set of tuples you want to get.
  - To be efficiently implemented, SQL queries need to be translated into **procedural** pro
- **Relational algebra** (RA) provides an inte r evaluating SQL.
  - RA is a **query language** for relational databases.
  - RA is **not visible** from the user interface, but at the **core of SQL**.
  - RA is used by relational DBMSs internally for **representing and optimising SQL queries**.

What is Relational Algebra?

- It's an algebra, like elementary algebra in math.
- An **algebra** is **a set** A together with **a collection of operators** on this set:
  - each operator has an **arity** n, i.e., the number of its arguments;
  - each operator on A of arity n is a (possibly partial) **function**

Example: $\{1, 2, \dots\}$, $\{+, -, \times, /\}$.
$((1 + 2) \times 4) - 7$
$((9 - 3) \times 5)$

- **Relational algebra** is an algebra that is
  - a set of all possible **relations** for a database, together with
  - a collection of **relational operators** for processing relations.

Example: $\{R_1, R_2, \dots\}$, $\{\sigma, \pi, \cup, \cap, \bowtie, \dots\}$, $\sigma_{A=2}(\pi_A(R_1 \bowtie R_2))$

Relational Operators

- Relational algebra (RA) provides a number of **relational operators**:
  - **Selection**: choose certain tuples (i.e., rows).
  - **Projection**: cho s).
  - **Renaming**: cha ation name.
  - **Union**, **intersection** and **di** on two relations that have the same relational schema.
  - **Cartesian product** and **join** (several variations): combine tuples from multiple relations together.
- The operators are applied on one or two relations and the result is always a relation.

Question One

- Consider the relation SOCCER:

HomeT	re		GuestTeam
Kiel			Munich
Munic			Freiburg
Frankfurt	1		Hamburg
Kiel	1		Frankfurt

- What if we only want to know the matches in which the home and guest teams had a tie?

Selection - Choose Rows

- **Selection** $\sigma_{\varphi}(R)$ chooses tuples that satisfy the condition φ from a relation R (i.e., the condition φ acts as a filter).
  - φ is a **condition**:
    - $<$ *attribut*
    - $<$ *attribute_name* $>$ $<$ *op* $>$ $<$ *attri* *_name* $>$,

    and *op* is normally one of the operators {= /

Example:

- $\sigma_{Semester=\text{`2016 S2'}}(Course)$
- $\sigma_{Name\neq\text{`Tom'}}(Employee)$
- $\sigma_{Mark>50}(Exam)$

Selection – Example

- Consider the relation SOCCER:

HomeTeam	HomeScore	GuestScore	GuestTeam
Kiel	1	3	Munich
Munich	0	0	Freiburg
Frankf			Hamburg
Kiel			Frankfurt

- For $\sigma_{HomeScore=GuestScore}$(SOCCER), we ha

HomeTeam	HomeScore	GuestScore	GuestTeam
Munich	0	0	Freiburg
Frankfurt	1	1	Hamburg

- For $\sigma_{HomeScore=1}(\sigma_{HomeScore=GuestScore}$(SOCCER)), we have:

HomeTeam	HomeScore	GuestScore	GuestTeam
Frankfurt	1	1	Hamburg

Selection – Properties

- Each selection $\sigma_{\varphi}(R)$ yields a relation that has **the same attributes** as R (i.e., their relation schemas are the same).
- Selection is **commutative**:

$$\sigma_{\varphi_1}(\sigma_{\varphi_2}(R)) = \sigma_{\varphi_2}(\sigma_{\varphi_1}(R)).$$

- A sequence of selection operations can be combined into a single selection operation with a conjunction of all the conditions:

$$\sigma_{\varphi_2}(\sigma_{\varphi_1}(R)) = \sigma_{\varphi_1 \wedge \varphi_2}(R).$$

Example:

$$\sigma_{Semester=\text{`2016 S2'}}(\sigma_{Name=\text{`Tom'}}(\text{ENROLMENT})) = \sigma_{Semester=\text{`2016 S2'} \wedge Name=\text{`Tom'}}(\text{ENROLMENT})$$

Question Two

- Consider the relation SOCCER:

HomeT	re		GuestTeam
Kiel	0		Munich
Munich	0		Freiburg
Frankfurt	1		Hamburg
Kiel	1		Frankfurt

- What if we only want the names of guest and home teams?

Projection - Choose Columns

- **Projection** $\pi_{A_1,\ldots,A_n}$ ed attributes $A_1, \ldots, A_n$ (columns butes are discarded.

Projection – Example

- Still consider the relation SOCCER:

HomeTeam	HomeScore	GuestScore	GuestTeam
Kiel	1	3	Munich
Munic			Freiburg
Frankf			Hamburg
Kiel			Frankfurt

- For $\pi_{GuestTeam,HomeTeam}$**(SOCCER)**, we hav

GuestTeam	HomeTeam
Munich	Kiel
Freiburg	Munich
Hamburg	Frankfurt
Frankfurt	Kiel

Projection – Duplicates

- Suppose that one more tuple is added into the relation SOCCER:

HomeTeam	HomeScore	GuestScore	GuestTeam
Kiel	1	3	Munich
Munich	3	0	Freiburg
Frankf			Hamburg
Kiel			Frankfurt
Kiel			Munich

- For $\pi_{GuestTeam,HomeTeam}$(SOCCER), is the following correct? Why or why not?

GuestTeam	HomeTeam
Munich	Kiel
Freiburg	Munich
Hamburg	Frankfurt
Frankfurt	Kiel
Munich	Kiel

Projection – Duplicates

- Suppose that one more tuple is added into the relation SOCCER:

HomeTeam	HomeScore	GuestScore	GuestTeam
Kiel	1	3	Munich
Munich	3	0	Freiburg
Frankf			Hamburg
Kiel			Frankfurt
Kiel			Munich

- For $\pi_{GuestTeam,HomeTeam}$(SOCCER), is the following correct? Why or why not?

GuestTeam	HomeTeam
Munich	Kiel
Freiburg	Munich
Hamburg	Frankfurt
Frankfurt	Kiel
Munich	Kiel

Incorrect

GuestTeam	HomeTeam
Munich	Kiel
Freiburg	Munich
Hamburg	Frankfurt
Frankfurt	Kiel

Correct

Projection – Duplicates

1. Projection can introduce duplicates that did not exist before, but that has to be eliminated. **Why?**

   Answer: Relations are sets. The value of an RA expression is a relation, which does not include duplicates.

2. DBMSs often permit duplicates, but you want them removed. **How to do this in SQL?**

   Answer: Using `DISTINCT`.

3. The number of tuples in the resulting relation $\pi_{A_1,\ldots,A_n}(R)$ is always less than or equal to the number of tuples in R. **What happens when $\{A_1, \ldots, A_n\}$ is a superkey of R?**

   Answer: The number of tuples in the resulting relation $\pi_{A_1,\ldots,A_n}(R)$ is **equal** to the number of tuples in R.

Projection – Properties

- The result of $\pi_{A_1,\ldots,A_n}(R)$ is a relation with only the attributes in $A_1, \ldots, A_n$, and in that order.

$\pi_{GuestTeam,HomeT}$

GuestTeam	
Munich	Kiel
Freiburg	Munich
Hamburg	Frankfurt
Frankfurt	Kiel

$_{am,GuestTeam}$(SOCCER)

ea	GuestTeam
	Munich
	Freiburg
Frankfurt	Hamburg
Kiel	Frankfurt

- Projection can be used to **reorder attributes** (i.e.,columns).

Projection – Properties

- Projection is **not commutative**:

$$\pi_{B_1,\ldots,B_m}(\pi_{A_1,\ldots,A_n}(R)) = \pi_{A_1,\ldots,A_n}(\pi_{B_1,\ldots,B_m}(R))$$ **does not hold** in general

HomeT	re		GuestTeam
Kiel			Munich
Munich	0		Freiburg
Frankfurt	1		Hamburg
Kiel	1	3	Frankfurt

- Consider the relation SOCCER, are the following expressions correct?
  - $\pi_{HomeTeam}(\pi_{GuestTeam,HomeTeam}(\text{SOCCER}))$
  - $\pi_{GuestTeam,HomeTeam}(\pi_{HomeTeam}(\text{SOCCER}))$

Projection – Properties

- Projection is **not commutative**:

$$\pi_{B_1,\ldots,B_m}(\pi_{A_1,\ldots,A_n}(R)) = \pi_{A_1,\ldots,A_n}(\pi_{B_1,\ldots,B_m}(R))$$ **does not hold** in general

HomeT	re		GuestTeam
Kiel			Munich
Munich	0		Freiburg
Frankfurt	1		Hamburg
Kiel	1	3	Frankfurt

- Consider the relation SOCCER, are the following expressions correct?
  - $\pi_{HomeTeam}(\pi_{GuestTeam,HomeTeam}(\text{SOCCER}))$ **Correct**
  - $\pi_{GuestTeam,HomeTeam}(\pi_{HomeTeam}(\text{SOCCER}))$ **Incorrect**

Projection – Properties

- If $A_1, \ldots, A_n$ **contains all the attributes** in $B_1, \ldots, B_m$, then

$$\pi_{B_1,\ldots,B_m}(\pi_{A_1,\ldots,A_n}(R)) = \pi_{B_1,\ldots,B_m}(R) \text{ holds.}$$

HomeTeam			GuestTeam
Kiel	1		Munich
Munich	0		Freiburg
Frankfurt	1		Hamburg
Kiel	1	3	Frankfurt

- The following expression holds:
  - $\pi_{HomeTeam}(\pi_{GuestTeam,HomeTeam}(\text{SOCCER}))=\pi_{HomeTeam}(\text{SOCCER})$

Selection and Projection

- "Selection chooses rows."
- "Projection chooses columns."

Union, Intersection and Difference

- Since relations are sets (of tuples), they are standard operations on sets.
  - **Union**, denoted as $R_1 \cup R_2$, results in a relation that includes all tuples either in R_1 or in R_2. Duplicate tuples are eliminated.
  - **Intersection**, denoted as $R_1 \cap R_2$, results in a relation that includes all tuples that are in both R_1 and R_2.
  - **Difference**, denoted as $R_1 - R_2$, results in a relation that includes all tuples that are in R_1 but not in R_2.
- **Type compatibility**: R_1 and R_2 must have **the same type**, i.e.,
  - the same number of attributes, and
  - the same domains for the attributes (the order is important).

Union – Example

- Consider the relation SOCCER:

HomeTeam	HomeScore	GuestScore	GuestTeam
Kiel	1	3	Munich
Munic			Freiburg
Frankf			Hamburg
Kiel	1		Frankfurt

- For $\sigma_{HomeScore=0}(\textbf{SOCCER}) \cup \sigma_{GuestScore=3}(\textbf{SOCCER})$, we have:

HomeTeam	HomeScore	GuestScore	GuestTeam
Munich	0	0	Freiburg
Kiel	1	3	Munich
Kiel	1	3	Frankfurt

Intersection – Example

- Consider the relation SOCCER:

HomeTeam	HomeScore	GuestScore	GuestTeam
Kiel			Munich
Munic			Freiburg
Frankf			Hamburg
Kiel	1		Frankfurt

- For $\sigma_{HomeScore=1}(\textbf{SOCCER}) \cap \sigma_{HomeTeam='Kiel'}(\textbf{SOCCER})$, we have:

HomeTeam	HomeScore	GuestScore	GuestTeam
Kiel	1	3	Munich
Kiel	1	3	Frankfurt

Difference – Example

- Consider the relation SOCCER:

HomeTeam	HomeScore	GuestScore	GuestTeam
Kiel	1	3	Munich
Munic			Freiburg
Frankf			Hamburg
Kiel	1		Frankfurt

- For **SOCCER** $- \sigma_{GuestTeam = \text{`}Frankfurt\text{'}}$ **(SOCCER)**, we have:

HomeTeam	HomeScore	GuestScore	GuestTeam
Kiel	1	3	Munich
Munich	0	0	Freiburg
Frankfurt	1	1	Hamburg

Cartesian Product (Cross Product)

- **Cartesian product** $R_1 \times R_2$ combines tuples from two relations in a combinatorial fashion.
- The result has one tuple for **each combination of two tuples – one from R_1 and the other from**

  i.e., if R_1 has n attributes and p tuples and R ttributes and q tuples, then $R_1 \times R_2$ has
  - n + m attributes, and
  - p $\times$ q tuples.
- Cartesian product is **expensive**, which would result in a very large relation when R_1 and R_2 are large!

Cartesian Product – Example

- Consider the relations HOME and GUEST:

HomeTeam	HomeScore
Kiel	1
Frankfurt	1

GuestScore	GuestTeam
3	Munich
1	Hamburg

- For **HOME**$\times$**GUEST**, we have:

HomeTeam	HomeScore	GuestScore	GuestTeam
Kiel	1	3	Munich
Frankfurt	1	3	Munich
Kiel	1	1	Hamburg
Frankfurt	1	1	Hamburg

Cartesian Product – Example

- Consider the slightly modified relations HOME and GUEST:

HomeTeam	Score
Kiel	1
Frankfurt	1

Score	GuestTeam
3	Munich
1	Hamburg

- For **HOME** $\times$ **GUEST**,

HomeTeam	Home.Score		GuestTeam
Kiel	1		Munich
Frankfurt	1	3	Munich
Kiel	1	1	Hamburg
Frankfurt	1	1	Hamburg

Observations: For $R_1 \times R_2$,

- R_1 and R_2 **do not share any attribute names**. If an attribute occurs in both relations, it occurs twice in the result (prefixed by relation name);
- the relations R_1 and R_2 do **NOT** have to be type compatible

Cartesian Product – Example

- Consider the slightly modified relations HOME and GUEST:

HomeTeam	Score
Kiel	1
Frankfurt	1

Score	GuestTeam
3	Munich
1	Hamburg

- For **HOME** × **GUEST**,

HomeTeam	Home.Score		GuestTeam
Kiel	1		Munich
Frankfurt	1	3	Munich
Kiel	1	1	Hamburg
Frankfurt	1	1	Hamburg

Problem:

- **Many of the tuples in the result do not make sense!**

Join

- To remove the nonsense tuples generated by Cartesian product, we can use **selection** with **Cartesian product**.
- However this is not convenient since two operators have to be used.
- **Join** $R_1 \bowtie_\varphi R_2$ is introduced as the **combination of Cartesian product and selection**. That is

$$_1 \bowtie_\varphi \quad _2 = \sigma$$

Examples: φ may contain $\{=, <, \leq, >, \geq, \neq\}$ suc

  - $(HomeTeam = GuestTeam) \wedge (Home.Score = Guest.Score)$
  - $(Home.Score = Guest.Score) \vee (HomeTeam = GuestTeam)$
    where $\wedge$ means AND and $\vee$ means OR.

- Join combines tuples from two relations whenever **the combination of tuples satisfies the join condition** φ (different from Cartesian product which includes **all** combinations of tuples).

Two Variations of Join

- Two common variations of join:

  - **Join** $R_1 \bowtie_\varphi R$

  - **Natural Join** $R_1 \bowtie R_2$

    1. Implicitly apply the join condition on **equality comparisons of attributes that have the same name** in both relations.
    2. Project out one copy of the attributes that have the same name in both relations.

Join – Example

- Consider the relations MATCH and TEAM.

HomeTeam	GuestTeam
Kiel	Munich
Frankfurt	Munich
Kiel	
Frankfurt	

TeamName	Coach
Kiel	Sven
Munich	Tim
amburg	Martin
Frankfurt	Kai

- For **MATCH** $\bowtie_{HomeTeam=TeamName}$ **TEAM**,

HomeTeam	GuestTea	e	Coach
Kiel	Munich	Kiel	Sven
Frankfurt	Munich	Frankfurt	Kai
Kiel	Hamburg	Kiel	Sven
Frankfurt	Hamburg	Frankfurt	Kai

Note that, the tuples (Munich, Tim) and (Hamburg, Martin) in TEAM are **filtered out** because they do not satisfy the join condition.

- **What will we have for MATCH ⋈ TEAM?**

Natural Join – Example

- Still consider the relations MATCH and TEAM.

HomeTeam	GuestTeam
Kiel	Munich
Frankfurt	Munich
Kiel	
Frankfurt	

HomeTeam	Coach
Kiel	Sven
Munich	Tim
amburg	Martin
Frankfurt	Kai

- For **MATCH** ⋈ **TEAM**, we have:

HomeTeam	GuestTeam	Coach
Kiel	Munich	Sven
Frankfurt	Munich	Kai
Kiel	Hamburg	Sven
Frankfurt	Hamburg	Kai

- Note that, the attribute *HomeTeam* shared by **TEAM** and **MATCH** occurs **only once** in the result.

Attribute Names in Join

- What if two attributes in different relations **have the same name but we don't want them to match**?

Example:

'**TeamName**' in the rela**e**' in the relation PROJECT

- What if two attributes in different relations **e the same name but we do want them to match**?

Example:

'**HomeTeam**' in the relation MATCH and '**TeamName**' in the relation TEAM

Renaming

- **Renaming** is used to rename either the relation name or the attribute names, or both.
- Renaming is denoted as
  - $\rho_{R'(A_1,\ldots,A_n)}(R)$: renaming the relation name to R' and the attribute names to $A_1, \ldots, A_n$,
  - $\rho_{R'}(R)$: renaming the relation name to R' keeping the attribute names unchanged, or
  - $\rho_{(A_1,\ldots,A_n)}(R)$: renaming the attribute names to $A_1, \ldots, A_n$ and keeping the relation name unchanged.
- Renaming is useful for giving names to the relations that hold the intermediate results.

Renaming – Example

- Consider the relation SOCCER:

HomeTeam	HomeScore	GuestScore	GuestTeam
Kiel	1	3	Munich
Munich	0	0	Freiburg
Frankf			Hamburg
Kiel			Frankfurt

- For $\rho_{Football}$(SOCCER), we have a relation F that has the same attributes and tuples as ones in SOCC
- For $\rho_{(HTeam, HScore, GScore, GTeam)}$(SOCCER), we have the relation below:

HTeam	HScore	GScore	GTeam
Kiel	1	3	Munich
Munich	0	0	Freiburg
Frankfurt	1	1	Hamburg
Kiel	1	3	Frankfurt

Renaming – Exercise

COURSE		
Code	Name	Unit
COMP2400	Relational Databases	6
COMP3600	Algorithms	6

StudentID				EnrolDate
456	Tom	COMP2400		02-Jul-2010
458	Mike	COMP2400		28-Jun-2010
458	Mike	COMP3600	2010 S2	05-Aug-2010

Exercise:

- Who did enrol the course “Relational Databases”?

$$\pi_{StudentID,Name}(\sigma_{CName=\text{`}Relational\ \ Databases\text{'}}(\rho_{(CourseNo,CName,Unit)}(\text{COURSE}) \bowtie \text{ENROL}))$$

Relational Operators [1]

. . . . . .
. . . . . .
. . . . . .

[1] http://merrigrove.blogspot.com.au/2011/12/another-introduction-to-algebraic-data.html (with some changes)