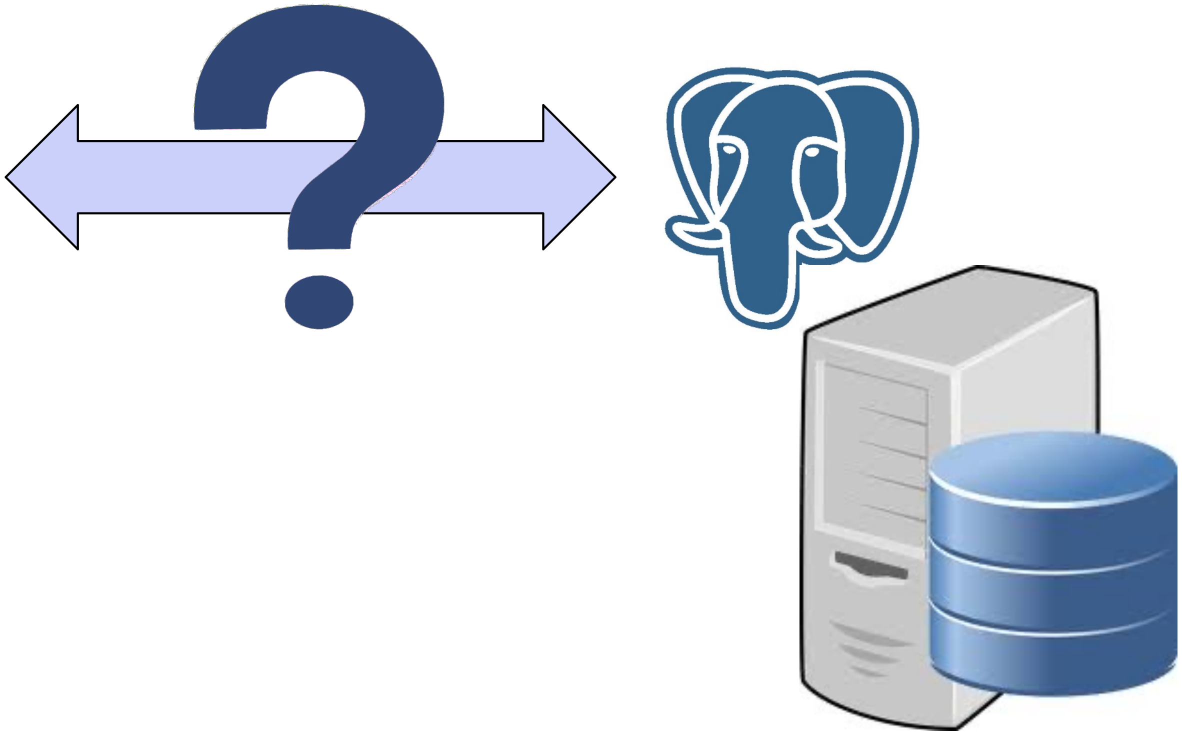
ISYS2120 – Data & Information Management

Week 4A: Relational Algebra

Based on slides from Kifer/Bernstein/Lewis (2006) “Database Systems”
and from Ramakrishnan/Gehrke (2003) “Database Management Systems”,
and also including material from Fekete and Röhm.

Prof Alan Fekete

How Can We Talk to a Database?



Database Querying with SQL

"List all students (by SID) who are enrolled in ***Database Systems I***"

Enter SQL, PL/SQL and SQL*Plus statements.

```
SELECT studID
  FROM Enrolled E, UnitOfStudy U
 WHERE E.uosCode = U.uosCode
     AND U.uosName= 'Database Systems I'
```

Execute | Load Script | Save Script | Cancel

STUDID
305422153
305678453
307088592
316424328
309187546
309145324

6 rows selected.

Many Ways to Write this Query…

"List all students (by SID) who are enrolled in ***Database Systems I***"

```
SELECT studID
  FROM Enrolled E, UnitOfStudy U
 WHERE E.uosCode = U.uosCode
   AND uosName = 'Database Systems I'
```

```
SELECT sid AS studID
  FROM (SELECT u.uosCode AS u1,  e.uosCode AS u2,
               e.studId  AS sid, u.uosName AS title
          FROM Enrolled e, UnitOfStudy u) SubQ
 WHERE u1=u2 AND title = 'Database Systems I'
```

```
SELECT studID
  FROM (SELECT studID, uosName
          FROM Enrolled NATURAL JOIN UnitOfStudy)R
 WHERE uosName= 'Database Systems I'
```

```
SELECT studID
  FROM Enrolled
 WHERE uosCode IN (SELECT uosCode
                     FROM UnitOfStudy
                    WHERE uosName = 'Database Systems I')
```

How do we know what a DBMS is doing?

- All the SQL queries on the previous slide are *equivalent*
  - On the same data set, they produce the same result
  - Even when run on different database systems by various vendors, their results will be the same

- Why?

- Why do we know that this is the case?
- How can database system vendors even guarantee this?

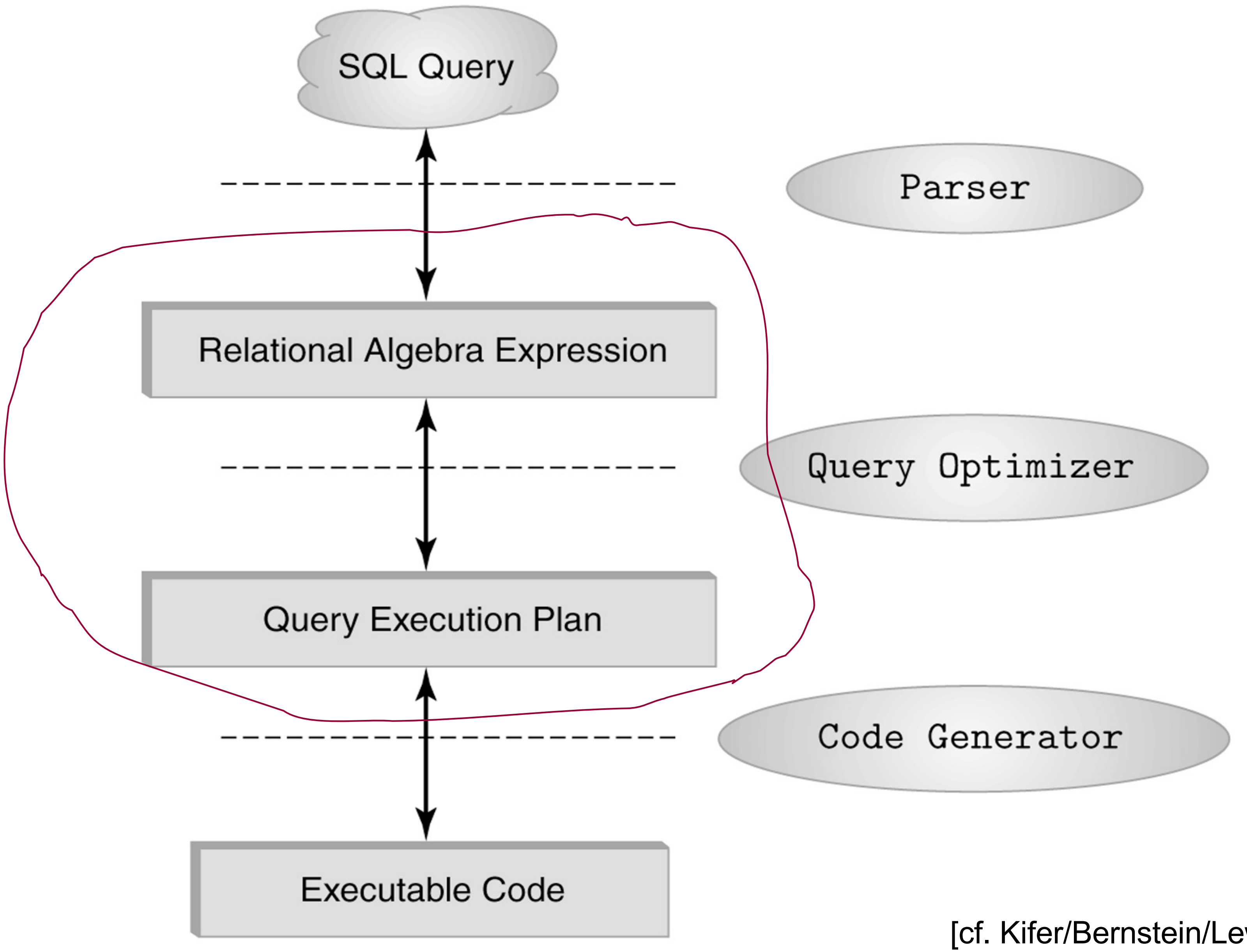
- And why are some not equivalent?
  - Wrong:

```
SELECT studID
  FROM Enrolled E, UnitOfStudy U
 WHERE uosName = 'Database Systems I'
```

The Big Idea

- Users request information from a database using a **query language**
- A query that extracts information can be seen as calculating a relation from combining one or more relations in the current state of the database
  - SQL expresses a query declaratively (“what to return” not “how to calculate”)
- **Relational algebra** (RA) can be used to express how to do a calculation of that kind (give a relation, based on existing relations), step-by-step
  - Each operator takes one or more relation instances, and produces a relation instance *(the operation part of the relational data model)*
- Why is it important?
  - Helps us understanding the precise meaning of declarative SQL queries.
  - Intermediate language used within DBMS (cf. chapter on db tuning)

The Role of Relational Algebra in a DBMS



[cf. Kifer/Bernstein/Lewis, Figure 5.1]

What is an Algebra?

- A language based on operators and a domain of values:
  - basic operators
  - atomic operands (constants and variables)
  - rules to form complex *expressions* from operators and operands, typically using nesting with parentheses

- Example: Algebra of Arithmetic
  - Operators for usual arithmetic operations: + - * /
  - Operands: variables (*x, y, z, …)* and numerical constants
  - Example arithmetic expression: 100 – *((x + y) / 2)*

- In databases, operators and operands are finite relations, which leads to the **Relational Algebra**
  - We refer to expression as a *query* and the value produced as *query result*

Relational Algebra as a model

- In a SQL DBMS, the data are in tables
- Mathematical relation is an idealised mathematical concept that is very similar to a database table
  - However, in a relation, duplicate rows are not allowed, and rows are not ordered
- So the calculation of a relational algebra expression, is an idealised way to express a calculation done in a SQL DBMS
- A real system has some extra complexity in how it deals with duplicates and order

Relational Algebra (RA)

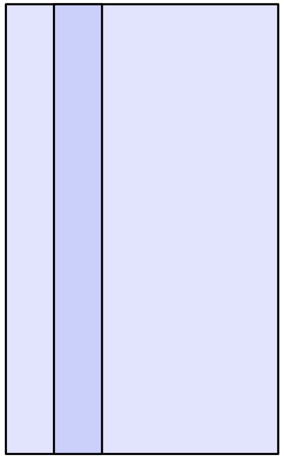
- 1. Set Operations
  - **Union** ($\cup$) tuples in relation 1 or in relation 2.
  - **Intersection** ($\cap$) tuples in relation 1, as well as in relation 2.
  - **Difference** (**-**) tuples in relation 1, but not in relation 2.
- 2. Operations that remove parts of a relation
  - **Selection** (σ) selects a subset of rows from relation.
  - **Projection** (π) deletes unwanted columns from relation.
- 3. Operations that combine tuples from two relations
  - **Cross-product** (**X**) allows us to fully combine two relations.
  - **Join** ($\bowtie$) to combine *matching* tuples from two relations.
- 4. A schema-level ‘rename’ operation
  - **Rename** (ρ) allows us to rename one field or even whole relation

- Domain: set of relations
  - operators take one/more relations as inputs and gives a new relation as result

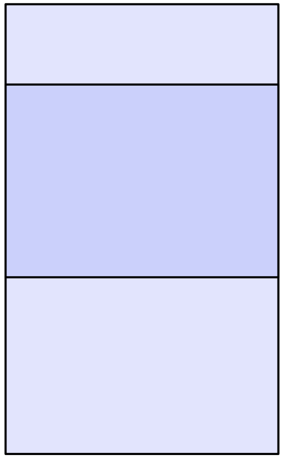
=> **RA queries by nesting of multiple operators** (cf. composition rules at end)

Visualisation of Relational Algebra

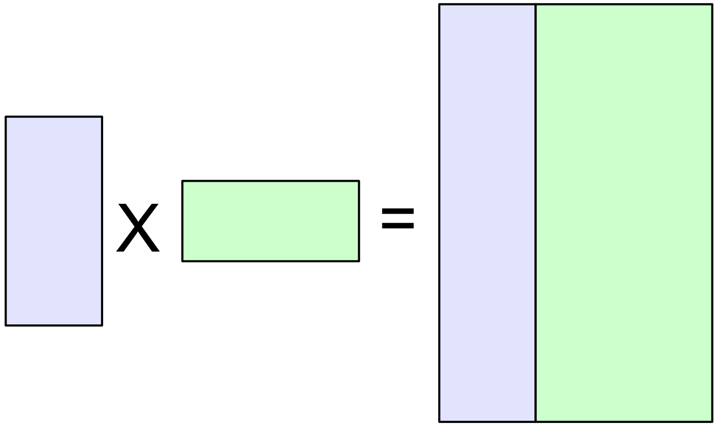
Projection (π)



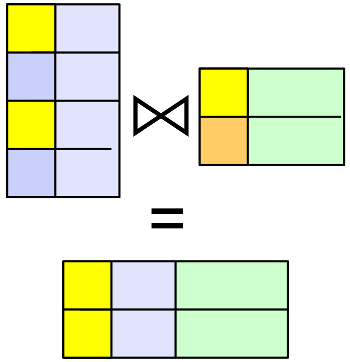
Selection (σ)



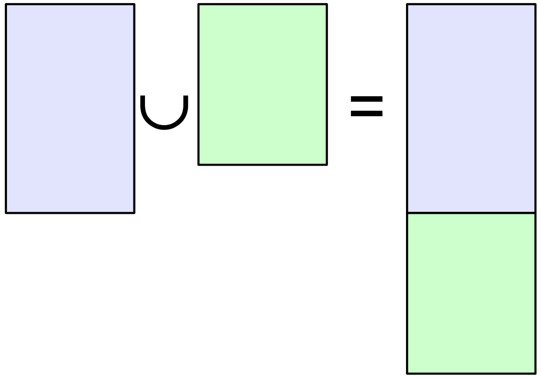
Cross-product (X)



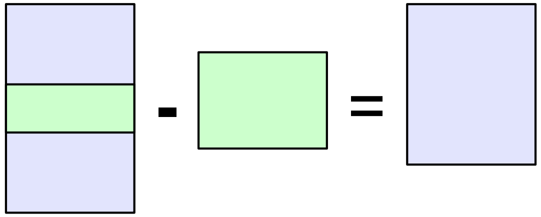
Join ($\bowtie$)



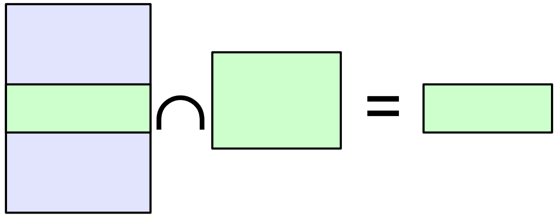
Set Union ($\cup$)



Set Minus (-)

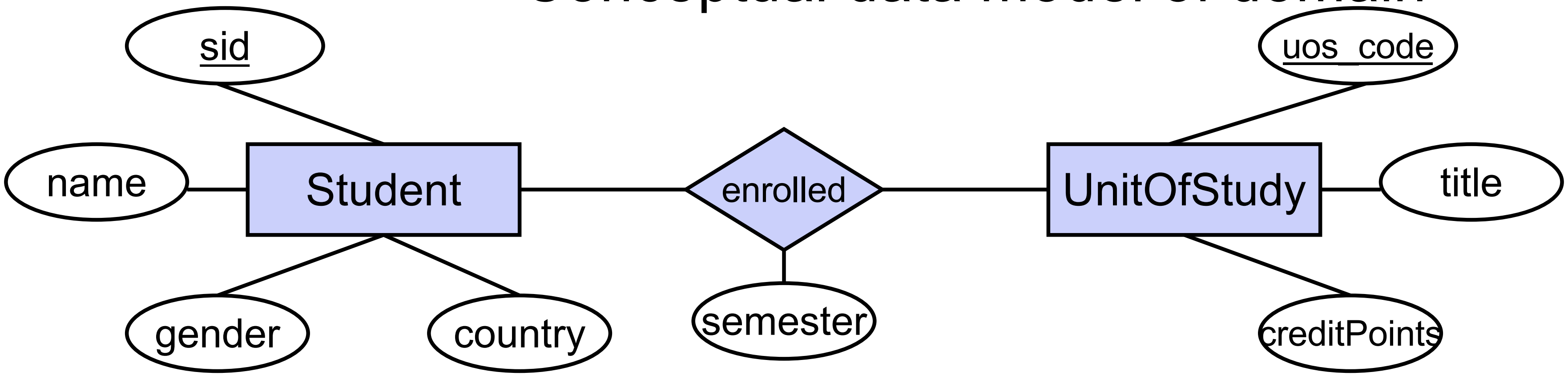


Set Intersection ($\cap$)



Running Example

Conceptual data model of domain



Relational schema and example instance

Student

sid	**name**	**gender**	**country**
1001	Ian	M	AUS
1002	Ha Tschi	F	ROK
1003	Grant	M	AUS
1004	Simon	M	GBR
1005	Jesse	F	CHN
1006	Franzisca	F	GER

Enrolled

sid	**uos_code**	**semester**
1001	COMP5138	2005-S2
1002	COMP5702	2005-S2
1003	COMP5138	2005-S2
1006	COMP5318	2005-S2
1001	INFS6014	2004-S1
1003	ISYS3207	2005-S2

UnitOfStudy

uos_code	**title**	**credit Points**
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	MIT Research Project	18

Set Operations

- These operations take two input relations *R* and *S*
  - Set Union $R \cup S$
    - Definition: $R \cup S = \{ t \mid t \in R \vee t \in S \}$
  - Set Intersection $R \cap S$
    - Definition: $R \cap S = \{ t \mid t \in R \wedge t \in S \}$
  - Set Difference $R - S$
    - Definition: $R - S = \{ t \mid t \in R \wedge t \notin S \}$

- Important constraint: *R* and *S* have the same schema
  - *R*, *S* have the *same arity* (same number of fields)
  - \`Corresponding' fields must have the same names and domains

Projection

- '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 input relation.
- Examples:

$\Pi_{\textit{name, country}}$ (*Student*)

Student	
name	**country**
Ian	AUS
Ha Tschi	ROK
Grant	AUS
Simon	GBR
Jesse	CHN
Franzisca	GER

$\Pi_{\textit{title}}$ (*UnitOfStudy*)

UnitOfStudy
title
Relational DBMS
Data Mining
IT Project Management
Algorithms
IS Project
MIT Research Project

Selection

- Selects rows that satisfy a ***selection condition***.
  - Example:

$$\sigma_{country='AUS'}(Student)$$

Student			
<u>sid</u>	**name**	**gender**	**country**
1001	Ian	M	AUS
1003	Grant	M	AUS

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

$$\Pi_{name}(\ \sigma_{country='AUS'}(Student)\ )$$

Student
name
Ian
Grant

Cross-Product

- Defined as: $R \times S = \{t\, s \mid t \in R \wedge s \in S\}$
  - each tuple of *R* is paired with each tuple of *S*.
  - Resulting schema has one field per field of *R* and *S*, with field names `inherited' if possible.
    - It might end in a conflict with two fields of the same name -> rename needed
- Sometimes also called *Cartesian product*
- Example:

R

A	B
α	1
β	2

x

S

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

=

result

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Joins

- **Conditional Join**:
  - Example:

$$R \bowtie_c S = \sigma_c (R \times S)$$

$$Student \bowtie_{family\_name = last\_name} Lecturer$$

sid	given	family_name	gender	country	empid	first_name	last_name	room
1001	Cho	Chung	M	AUS	47112344	Vera	Chung	321
1004	Ciao	Poon	M	CHN	12345678	Simon	Poon	431
1004	Ciao	Poon	M	CHN	99004400	Josiah	Poon	482
1111	Alice	Poon	F	AUS	12345678	Simon	Poon	431
1111	Alice	Poon	F	AUS	99004400	Josiah	Poon	482
…		…	…	…	…		…	…

- Result schema same as the cross-product's result schema.
- Sometimes called ***theta-join***.
- ***Equi-Join***: Special case where the condition *c* contains only equalities.

Natural Join

- **Natural Join**: $R \bowtie S$
- Like Equijoin on all common fields
  - Result schema has only one copy of fields for which equality is specified.

Enrolled	
<u>sid</u>	<u>uos_code</u>
1001	COMP5138
1002	COMP5702
1003	COMP5138
1006	COMP5318
1001	INFO6007
1003	ISYS3207

$\bowtie$

UnitOfStudy		
<u>uos_code</u>	title	points
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Mgmt.	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	MIT Research Project	18

=

result			
sid	uos_code	title	points
1001	COMP5138	Relational DBMS	6
1002	COMP5702	MIT Research Project	18
1003	COMP5138	Relational DBMS	6
1006	COMP5318	Data Mining	6
1001	INFO6007	IT Project Mgmt.	6
1003	ISYS3207	IS Project	4

Rename Operation

- Allows to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows to refer to a relation by more than one name.
- Notation 1: $\rho_X(E)$
  - returns the expression *E* under the name *X* (rename whole relation)
- Notation 1: $\rho_{X\,(a1 \rightarrow b1)}(E)$
  - returns the result of expression *E* under the name *X*, and with the attributes a_1 …. renamed to b_1 … (rename individual attributes)
  - (assumes that the relational-algebra expression *E* has arity *n*)
- Example:

$$\rho_{\text{Classlist}(\,sid\,\rightarrow\,student\,)}\ (\ \sigma_{uos\_code='ISYS2120'}\ (Enrolled)\ )$$

Basic versus Derived Operators

- We can distinguish between basic and derived RA operators
- Only 6 basic operators are required to express everything else:
  - **Union** ($\cup$) tuples in relation 1 or in relation 2.
  - **Set Difference** (**-**) tuples in relation 1, but not in relation 2.
  - **Selection** (σ) selects a subset of rows from relation.
  - **Projection** (π) deletes unwanted columns from relation.
  - **Cross-product** (**X**) allows us to fully combine two relations.
  - **Rename** (ρ) allows us to rename one field to another name.

- Additional (derived) operations:
  - intersection, join, division:
    - Not essential, but (very!) useful.
  - Cf. Join: $R \bowtie_c S = \sigma_c(R \times S)$

Relational Expressions

- A basic expression in the relational algebra consists of either one of the following:
  - Variable that refers to a relation of the same name in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; then the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $E_1 \bowtie E_2$
  - $\sigma_P (E_1)$, P is a (complex) predicate on attributes in E_1
  - $\pi_S (E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_X (E_1)$, x is the new name for the result of E_1

Example: Basic SQL Query

- List the names of all Australian students.

  SELECT name **FROM** Student **WHERE** country='AUS'

- Corresponding relational algebra expression

  $\pi_{name} (\sigma_{country='AUS'} (\text{Student}))$

- Note: SQL does not permit the '-' character in names, and SQL names are case insensitive, i.e. you can use capital or small letters.
  - You may wish to use upper case where-ever we use bold font.

SQL -> Relational Algebra

- Given a Select-From-Where (SFW) query

SELECT A1, A2, …, An
FROM R1, R2, …, Rm
WHERE condition

- This can be calculated by the relational algebra expression:

$\Pi_{A1,\ A2,\ ...,\ An}\ (\ \sigma_{condition}\ (R_1\ \ x\ R_2\ \ \ x\ \ ...\ \ x\ \ R_m)\ )$

Equivalence Rules

The following equivalence rules hold:

- Commutation rules
  1. $\pi_A (\sigma_p (R)) = \sigma_p (\pi_A (R))$
  2. $R \bowtie S = S \bowtie R$
- Association rule
  1. $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$
- Idempotence rules
  1. $\pi_A (\pi_B (R)) = \pi_A (R)$ if $A \subseteq B$
  2. $\sigma_{p1} (\sigma_{p2} (R)) = \sigma_{p1 \wedge p2} (R)$
- Distribution rules
  1. $\pi_A (R \cup S) = \pi_A (R) \cup \pi_A (S)$
  2. $\sigma_P (R \cup S) = \sigma_P (R) \cup \sigma_P (S)$
  3. $\sigma_P (R \bowtie S) = \sigma_P (R) \bowtie S$ if P only references R
  4. $\pi_{A,B}(R \bowtie S) = \pi_A (R) \bowtie \pi_B (S)$ if *join-attr.* in $(A \cap B)$
  5. $R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$
- These are the basis for the automatic optimisation of relational queries

You should now be able …

- to understand the **Foundations of SQL**
  - basic operations of Relational Algebra:
    Projection, Selection, Rename, Set Operations, Cross Product
  - the meaning of a relational join
  - differences between an equi-join, a theta-join and a natural join
- to give a relational algebra expression that will calculate the result for a (simple) English data-request on a given schema
- to give an English expression of the purpose of a given calculation in relational algebra
- to translate a relational algebra expression into a SQL query
  - and vice versa…

References

- Kifer/Bernstein/Lewis (2nd edition – 2006)
  - Chapter 5.1
    one section on RA that covers everything as discussed here in the lecture
- Ramakrishnan/Gehrke (3rd edition - the ‘Cow’ book (2003))
  - Chapter 4.2
    one compact section on RA, including a discussion of relational division
- Ullman/Widom (3rd edition – 2008)
  - Chapter 2.4
    a nice and gentle introduction to the basic RA operations,
    leaves out relational division though
  - Chapters 5.1 and 5.2
    goes beyond what we cover here in the lecture by extending RA to bags and also introduces grouping, aggregation and sorting operators