

COMP9120

Week 4: Relational Algebra & SQL

Semester 2, 2022

Dr Mohammad Polash
School of Computer Science



THE UNIVERSITY OF
SYDNEY



Acknowledgement of Country

I would like to acknowledge the Traditional Owners of Australia and recognise their continuing connection to land, water and culture. I am currently on the land of the Gadigal People of the Eora nation and pay my respects to their Elders, past, present and emerging.

I further acknowledge the Traditional Owners of the country on which you are on and pay respects to their Elders, past, present and future.



COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

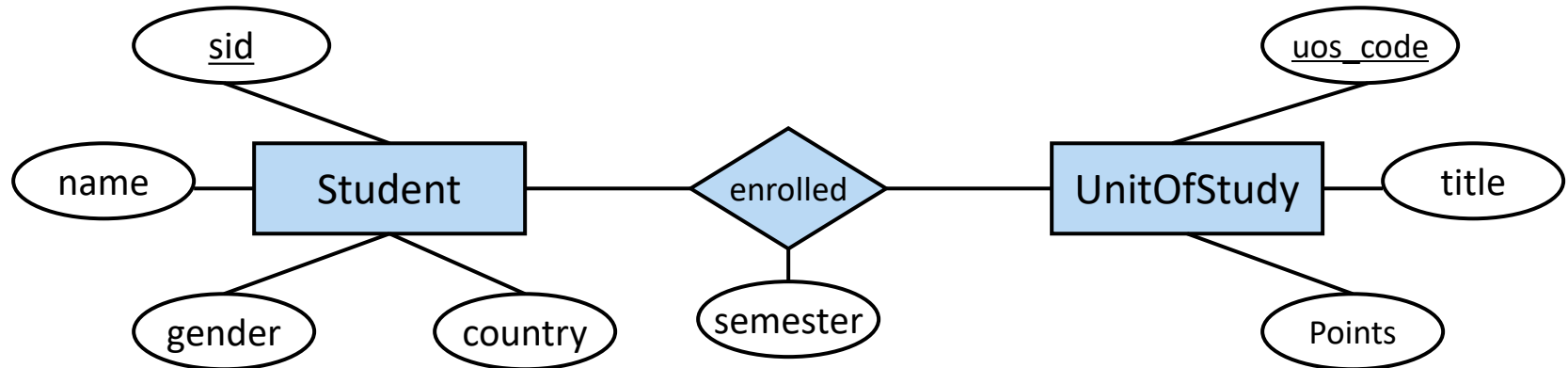
Do not remove this notice.

› **Relational Algebra**: an algebra for relational model

- Six basic operators

› Introduction to **SQL**

- Basic SQL queries
- Join queries
- Set operations



Student			
<u>sid</u>	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		
<u>sid</u>	<u>uos_code</u>	semester
1001	COMP5138	2020-S2
1002	COMP5702	2020-S2
1003	COMP5138	2020-S2
1006	COMP5318	2020-S2
1001	INFO6007	2020-S1
1003	ISYS3207	2020-S2

UnitOfStudy		
<u>uos_code</u>	title	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

Relational Algebra



THE UNIVERSITY OF
SYDNEY

Exercise: Evaluating a Simple Query

Student			
<u>sid</u>	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	Franziska	F	GER

Enrolled		
<u>sid</u>	<u>uos_code</u>	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		
<u>uos_code</u>	title	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

Using the above database instance, **find the titles of all units worth 6 credit points.**
Think about the steps we have to take.

title
Relational DBMS
Data Mining
IT Project Management

How does a RDBMS get the answer?

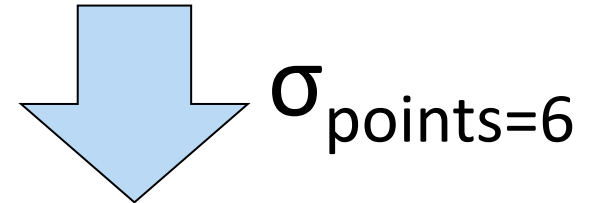
Find the titles of all units worth 6 credit points

Relational Algebra expression:

$$\pi_{\text{title}}(\sigma_{\text{points}=6}(\text{UnitOfStudy}))$$

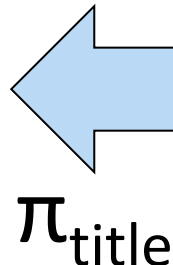
UnitOfStudy

<u>uos_code</u>	title	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



<u>uos_code</u>	title	points
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6

title
Relational DBMS
Data Mining
IT Project Management



Student			
<u>sid</u>	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	Franziska	F	GER

Enrolled		
<u>sid</u>	<u>uos_code</u>	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		
<u>uos_code</u>	title	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

Find the student id of all students who are enrolled in COMP5138.

$\pi_{sid}(\sigma_{uos_code='COMP5138'}(Enrolled))$

sid
1001
1003

› Relational algebra (RA) is an algebra for ***relational model***

- Relational algebra operates on **sets**!
- It is composed using a collection of ***operators*** (e.g., *selection*, *projection*, *join*)

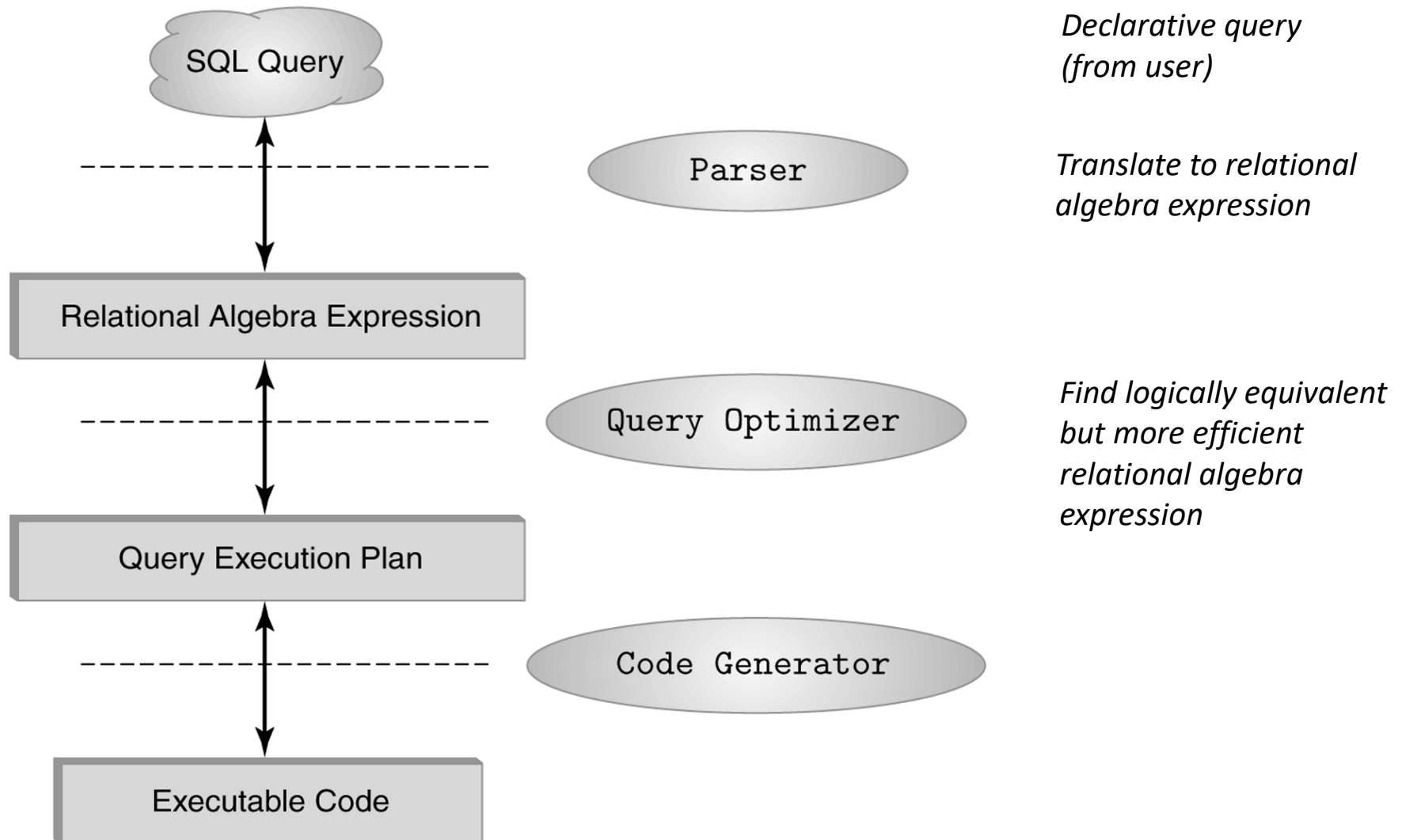
$$\pi_{\text{title}}(\sigma_{\text{points}=6}(\text{UnitOfStudy}))$$

- It describes a step-by-step procedure (i.e., describes ***how***) for computing the desired answer

› RA allows us to translate declarative (SQL) queries into **precise** and **optimizable** expressions!



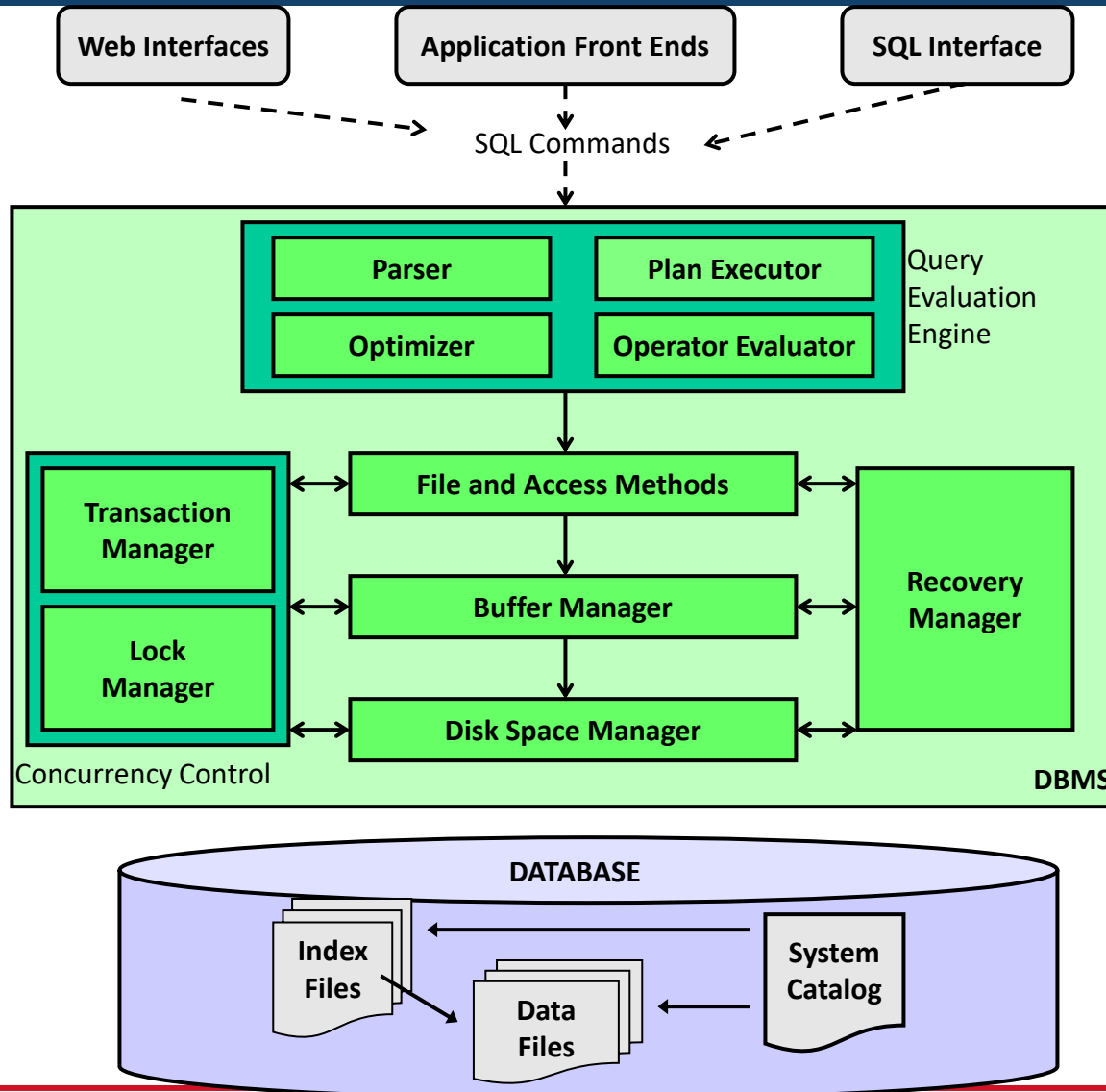
The Role of RA in RDBMS



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



The Role of RA in RDBMS



- › Each operator takes one/more relations as inputs and gives a new relation as result
- › Operators can be chained together to form expressions (queries)

$\pi_{\text{title}}(\sigma_{\text{points}=6}(\text{UnitOfStudy}))$

1. Operators that remove parts of a relation

- **Selection** (σ) selects a subset of rows from relation.
- **Projection** (π) deletes unwanted columns from relation.

2. Operators that combine tuples from two relations

- **Cross-product** (\times) combines *every* tuple from two relations.
- **Join** (\bowtie) combines *matching* tuples from two relations.

3. Set Operators

- **Union** (\cup) returns tuples in relation 1 or in relation 2.
- **Intersection** (\cap) returns tuples in relation 1, as well as in relation 2.
- **Difference** ($-$) returns tuples in relation 1, but not in relation 2.

4. A schema-level 'rename' operator

- **Rename** (ρ) allows us to rename an attribute or a relation.

- › ‘Extracts’ columns for attributes that are in *projection* list.
 - Removes columns that are not in the projection list, then eliminates duplicate rows
 - Schema of the result contains exactly the attributes in the projection list.
- › Examples:

 $\pi_{name, country} (Student)$

name	country
Ian	AUS
Ha Tshi	ROK
Grant	AUS
Simon	GBR
Jesse	CHN
Franziska	GER

 $\pi_{country} (Student)$

country
AUS
ROK
GBR
CHN
GER

› Selects rows that satisfy a *selection condition*.

- Schema of the result is the same as the schema of the input relation
- Example:

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

<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:

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

name
Ian
Grant



<i>Student</i>			
<u>sid</u>	name	gender	country
1001	Ian	M	AUS
1002	Ha Tshi	F	ROK
1003	Grant	M	AUS
1004	Simon	M	GBR
1005	Jesse	F	CHN
1006	Franziska	F	GER

- › Will the following work to “Find the name of the students who live in Australia”?

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

name
Ian
Ha Tshi
Grant
Simon
Jesse
Franziska

- › Selection condition is a Boolean combination of terms
 - Each term has the form *attribute* op *constant*, or *attribute1* op *attribute2*
 - op can be <, >, <=, >=, ≠, =
 - Terms are connected by logical connectives:
 - ∧ - means AND
 - ∨ - means OR

Student			
<u>sid</u>	name	gender	country
1001	Ian	M	AUS
1002	Ha Tschì	F	ROK
1003	Grant	M	AUS
1004	Simon	M	GBR
1005	Jesse	F	CHN
1006	Franzisca	F	GER

Find out the male Australian students

$$\sigma_{\text{gender}='M' \wedge \text{country}='AUS'}(\text{Student})$$

- › Defined as: $R \times S = \{t \mid t \in R \wedge t \in S\}$
 - Each tuple of R is paired with each tuple of S .
 - If R or S is empty, then $R \times S$ is also empty.
 - 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			S								
A	B		C	D	E		A	B	C	D	E
α	1	\times	α	10	a	$=$	α	1	α	10	a
β	2		β	10	a		α	1	β	10	a
			β	20	b		α	1	β	20	b
			γ	10	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

› **Conditional Join:** $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$

Example:

Student \bowtie

Lecturer

Student.f_name = Lecturer.last_name \wedge Student.sid < Lecturer.empid

sid	given	f_name	gender	country	empid	lecturer_name	last_name	room
1001	Ian	Chung	M	AUS	47112344	Vera	Chung	321
1004	Simon	Poon	M	GBR	12345678	Simon	Poon	431
1004	Simon	Poon	M	GBR	99004400	Josiah	Poon	482
...
...
...	

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

› Natural Join: $R \bowtie S$

- Equijoin on all common fields, followed by a projection
- Result schema is similar to cross-product, but retains only one (specifically, the first) copy of fields for which equality is specified.

Enrolled		\bowtie	UnitOfStudy			$=$				
<u>sid</u>	<u>uos_code</u>		<u>uos_code</u>	title	points		sid	uos_code	title	points
1001	COMP5138		COMP5138	Relational DBMS	6		1001	COMP5138	Relational DBMS	6
1002	COMP5702		COMP5318	Data Mining	6		1002	COMP5702	MIT Research Project	18
1003	COMP5138		INFO6007	IT Project Mgmt.	6		1003	COMP5138	Relational DBMS	6
1006	COMP5318		SOFT1002	Algorithms	12		1006	COMP5318	Data Mining	6
1001	INFO6007		ISYS3207	IS Project	4		1001	INFO6007	IT Project Mgmt.	6
1003	ISYS3207		COMP5702	MIT Research Project	18		1003	ISYS3207	IS Project	4

- › Given schemas $R(A, B, C, D)$, $S(A, C, E)$, what is the schema of $R \bowtie S$?
 - (A, B, C, D, E)

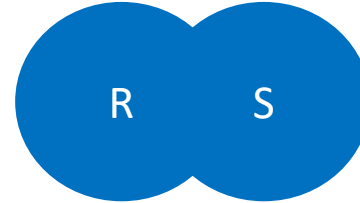
- › Given $R(A, B, C)$, $S(D, E)$, what is $R \bowtie S$?
 - (A, B, C, D, E)

- › Given $R(A, B)$, $S(A, B)$, what is $R \bowtie S$?
 - (A, B)

› These operations take two input relations R and S

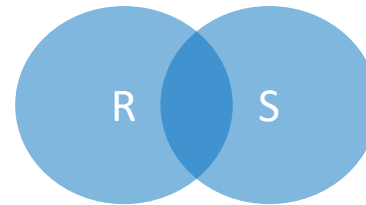
- Set Union $R \cup S$

- Definition: $R \cup S = \{t \mid t \in R \vee t \in S\}$



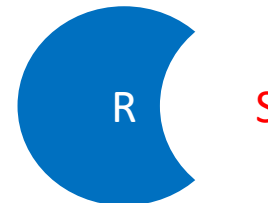
- Set Intersection $R \cap S$

- 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 compatible schema

- R, S have the *same arity* (same number of fields)
- ‘Corresponding’ fields must have the same domains



Exercise: Set Operations

- › Suppose you have the following relations. Use a set operation in an RA expression to **return all students who are not postgraduates**.

<i>Student</i>			
<u>sid</u>	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

<i>Postgraduate</i>	
<u>sid</u>	
1003	
1004	
1005	

$\pi_{\text{sid}}(\text{Student}) - \text{Postgraduate}$

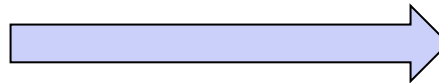
<u>sid</u>
1001
1002
1006

- › Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- › Allows us to refer to a relation by more than one name.
- › Notation 1: $\rho_X(E)$
 - returns the expression E under the name X
- › Notation 2: $\rho_X(A1, A2, \dots, An)(E)$
 - (assumes that the relational-algebra expression E has arity n)
 - returns the result of expression E under the name X , and with the attributes renamed to $A1, A2, \dots, An$.
- › Note that rename only modifies the schema of a relation!

Example: Rename Operation

$$\rho_{UOS(ucode,title,credits)}(UnitOfStudy)$$

<i>UnitOfStudy</i>		
<u>uos_code</u>	title	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

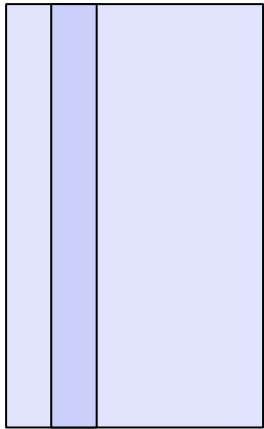


<i>UOS</i>		
<u>ucode</u>	title	credits
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	MIT Research Project	18

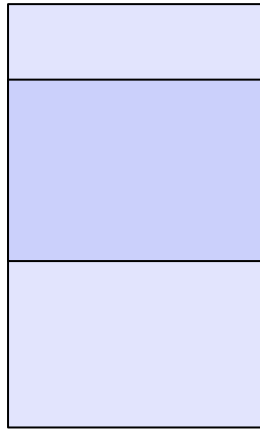


Visualisation of Relational Algebra Operators

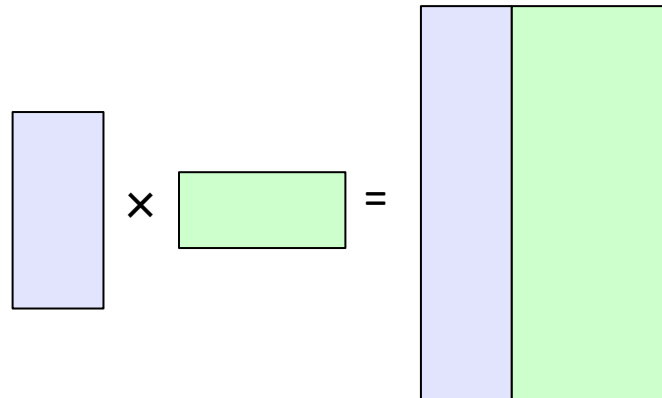
Projection (π)



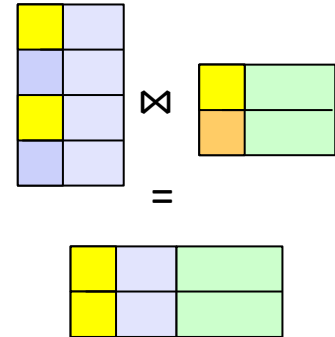
Selection (σ)



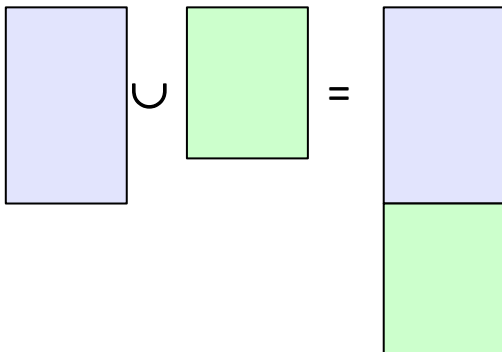
Cross-product (\times)



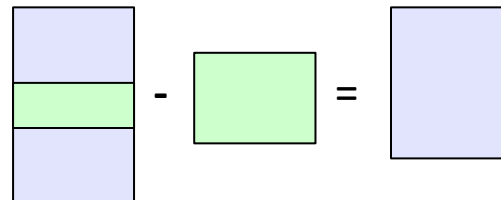
Join (\bowtie)



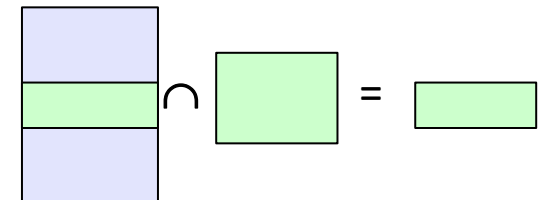
Set Union (\cup)



Set Minus ($-$)



Set Intersection (\cap)



- › We can distinguish between basic and derived RA operators
 - › Only 6 basic operators are required to express everything else:
 - **Selection** (σ) selects a subset of rows from relation.
 - **Projection** (π) deletes unwanted columns from relation.
 - **Cross-product** (\times) allows us to fully combine two relations.
 - **Union** (\cup) returns tuples in relation 1 or in relation 2.
 - **Set Difference** ($-$) returns tuples in relation 1, but not in relation 2.
 - **Rename** (ρ) allows us to rename one field to another name.
 - › Additional (derived) operations:
 - E.g.: intersection, join. [Not essential, but (very!) useful]
 - E.g., Intersection: $R \cap S = R - (R - S)$
- Join: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$

- › Different relational algebra expressions can be equivalent but with different execution costs
 - Example: List the names of all students enrolled in 'Relational DBMS'
 - $\pi_{\text{name}} (\sigma_{\text{title}='Relational DBMS'} ((\text{Student} \bowtie \text{Enrolled}) \bowtie \text{UnitOfStudy}))$
 - $\pi_{\text{name}} (\text{Student} \bowtie (\text{Enrolled} \bowtie (\sigma_{\text{title}='Relational DBMS'} (\text{UnitOfStudy}))))$

<i>Student</i>			
<u>sid</u>	name	gender	country
1001	Ian	M	AUS
1002	Ha Tschì	F	ROK
1003	Grant	M	AUS
1004	Simon	M	GBR
1005	Jesse	F	CHN
1006	Franziska	F	GER

<i>Enrolled</i>		
<u>sid</u>	<u>uos_code</u>	semester
1001	COMP5138	2020-S2
1002	COMP5702	2020-S2
1003	COMP5138	2020-S2
1006	COMP5318	2020-S2
1001	INFO6007	2020-S1
1003	ISYS3207	2020-S2

<i>UnitOfStudy</i>		
<u>uos_code</u>	title	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

- › Ramakrishnan/Gehrke (3rd edition - the 'Cow' book (2003))
 - **Chapter 4** (you can skip the sections on Relational Calculus)
one compact section on RA, including a discussion of relational division

- › Kifer/Bernstein/Lewis (2nd edition – 2006)
 - Chapter 5.1: *one section on RA that covers everything as discussed here in the lecture*

- › Ullman/Widom (3rd edition – 2008)
 - Chapter 2.4
a nice and gentle introduction to basic RA, leaves out relational division though
 - Chapters 5.1 and 5.2
goes beyond what we cover here in the lecture by extending RA and also introduces grouping, aggregation and sorting operators

Let's take a break!



THE UNIVERSITY OF
SYDNEY

Introduction to SQL



THE UNIVERSITY OF
SYDNEY



› **Basic SQL Queries**

› Join Queries

› Set Operations

- › Relational algebra is a **lower-level procedural** query language
 - Hard to understand and write by non-experts
- › SQL is the standard **high-level declarative** query language for RDBMS
 - Describing *what* data we are interested in, but *not how* to retrieve it.
 - Based on SEQUEL, introduced in the mid-1970's as the query language for IBM's System (Structured English Query Language)
- › RDBMS internally maps SQL to equivalent relational algebra expressions.
- › Many standards out there
 - ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ...
 - RDBMS vendors support various subsets

› **DDL** (Data Definition Language)

- Create, drop, or alter the relation schema

› **DML** (Data Manipulation Language)

- The insertion of new information into the database
- The deletion of information from the database
- The modification of information stored in the database
- The retrieval of information stored in the database
 - A **Query** is a statement requesting the retrieval of information
 - The portion of a DML that involves information retrieval is called a **query language**

› **DCL** (Data Control Language)

- Commands that control a database, including administering privileges and users

- › Used for queries on single or multiple tables
- › keywords:
 - **SELECT** Lists the columns (and expressions) that should be returned from the query
 - **FROM** Indicate the table(s) from which data will be obtained
 - **WHERE** Indicate the conditions to include a tuple in the result
 - **GROUP BY** Indicate the categorization of tuples
 - **HAVING** Indicate the conditions to include a category
 - **ORDER BY** Sorts the result according to specified criteria
- › Note: the result of an SQL query is also a table / relation
 - The result table can contain duplicate rows

Select-From-Where (SFW) Queries

- › List the names of all students.

```
SELECT name  
FROM Student
```

- › * in select denotes “all attributes”.


```
SELECT *  
FROM Student
```

- › List the names of all Australian students.

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

- › General form of SFW query:

```
SELECT <attributes>  
FROM <one or more tables>  
WHERE <conditions>
```



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

A Select-From-Where query is equivalent to the relational algebra expression:

$$\pi_{A_1, A_2, \dots, A_n} (\sigma_{condition} (R_1 \times R_2 \times \dots \times R_m))$$

- **SELECT** corresponds to projection (π) in RA
- **FROM** corresponds to Cartesian product (\times) in RA
- **WHERE** corresponds to selection (σ) in RA

- › SQL **commands** are not case sensitive:
 - Same: **SELECT**, **Select**, **select**
 - Same: Student, student

- › Use single quotes for string constants:
 - 'aus' – yes
 - "aus" - no

- › String constants are case sensitive:
 - Different: 'AUS', 'aus', 'Aus'

Remove Duplicates in Select Clause

- › RDBMS allows duplicates in tables as well as in query results.
 - Query result preserves duplicates by default.
 - Example:
- › To eliminate the duplicates, use the keyword **DISTINCT** after **SELECT**.
 - Example: List the distinct countries where students come from.

SELECT country
FROM Student

country
AUS
ROK
AUS
GBR
CHN
GER

SELECT DISTINCT country
FROM Student

country
AUS
ROK
GBR
CHN
GER

Arithmetic Expressions in Select Clause

- › The **SELECT** clause can contain arithmetic expressions involving the operations +, -, * and /, and operating on constants or attributes of tuples.
- › The query:

```
SELECT uos_code, title, points*2  
FROM UnitOfStudy
```

would return a table which is the same as the UnitofStudy table except that the credit-point-values are doubled.

- › The **WHERE** clause specifies conditions that the result must satisfy
- › Comparison operators in SQL: = , > , >= , < , <= , <> (or !=)
- › Comparison results can be combined using the logical connectives **AND**, **OR**, and **NOT**.
- › Comparisons can be applied to results of arithmetic expressions
- › Example: Find all UoS codes for units taken by student 1001 in 2022-S1:

```
SELECT uos_code  
  FROM Enrolled  
 WHERE sid = 1001 AND Semester = '2022-S1'
```


- › SQL includes a string-matching operator for comparisons on character strings.

- **LIKE** is used for string matching
- List the titles of all “COMP” unit of studies.

```
SELECT title  
  FROM UnitOfStudy  
 WHERE uos_code LIKE 'COMP%'
```

- › Patterns are described using two special characters (“wildcards”):

- percent (%). The % character matches any substring.
- underscore (_). The _ character matches any single character.

- › SQL supports a variety of string operations such as

- concatenation (using “||”)
- converting from upper to lower case (and vice versa)
- finding string length, extracting substrings, etc.

- › SQL allows renaming table attributes using the **AS** clause:

old_name **AS** new_name

- › This is very useful to give, e.g., result columns of expressions a meaningful name.
- › This can also assign names to tables as shown in example below

- › Example:

- Find the uos_code, credit_points for COMP5318,

rename the column name uos_code as course_code.

```
SELECT a.uos_code AS course_code, a.credit_points  
  FROM unitofstudy a  
 WHERE a.uos = 'COMP5318'
```

- › List all students (name) from Australia in alphabetical order.

```
SELECT name  
FROM Student  
WHERE country='AUS'  
ORDER BY name
```

- › Two options (per attribute):
 - **ASC** ascending order (default)
 - **DESC** descending order
- › You can order by more than one attribute
 - e.g., **ORDER BY** country **DESC**, name **ASC**



› Basic SQL Queries

› **Join Queries**

› Set Operations

- › A join query combines two or more tables into a single table
- › The **FROM** clause lists the tables involved in the query
 - corresponds to the Cartesian product of the tables.
 - join-predicates explicitly stated in the **WHERE** clause

› Examples:

- Find the Cartesian product *Student x UnitOfStudy*

```
SELECT *  
FROM Student, UnitofStudy
```

- Find the student ID, name, and gender of all students enrolled in INFO6007:


```
SELECT sid, name, gender  
FROM Student, Enrolled  
WHERE Student.sid = Enrolled.sid AND  
      uos_code = 'INFO6007'
```

- › Which students did enroll in what semester?

Join involves multiple tables in
FROM clause



```
SELECT S.sid, S.name, E.semester  
FROM Student S, Enrolled E  
WHERE S.sid = E.sid
```



WHERE clause performs the
equality check for common
columns of the two tables

$$\pi_{S.sid, S.name, E.semester} (\sigma_{S.sid = E.sid} (\rho_S (Student) \times \rho_E (Enrolled)))$$

- › Some queries need to refer to the same table twice
- › In this case, aliases are given to the table name
- Example: For each academic, retrieve the academic's name, and the name of his or her immediate supervisor.

SELECT	<i>L.name, M.name</i>
FROM	Lecturer L, Lecturer M
WHERE	<i>L.manager = M.empid</i>

- We can think of L and M as two different copies of Lecturer;
L represents lecturers in role of supervisees and M represents lecturers in role of supervisors (managers)

- › **Inner join:** *a join in which rows in the result table must have matching rows in both joining tables. Corresponds to joins in relational algebra*
 - By default, joins in SQL are “**inner joins**”. The word “**inner**” can be omitted.
 - **Theta join:** R **JOIN** S **ON** <join condition>
 - **Equi-join:** R **JOIN** S **USING** (<list of attributes>)
 - The specified attributes appear only once in the result table
 - **Natural join:** R **NATURAL JOIN** S
- **Outer join** – *a join in which rows that do not have matching values are nonetheless also included (exactly once) in the result relation*
 - **Left outer join:** *includes rows that would be found in an inner join as well as rows from the left table that don't have matches (padded with NULL values)*
 - **Right outer join:** *includes rows that would be found in an inner join as well as rows from the right table that don't have matches (padded with NULL values)*
 - **Full outer join:** *includes rows that would be found in an inner join as well as rows from the left table and rows from the right table that don't have matches*

- › Note that Join operators (a) are specified in the **FROM** clause, and (b) must have both a **join type** and a **join condition**
 - Available join types: **JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN**
 - Available join conditions: **NATURAL, ON** <join condition>, **USING** (<list of attributes>)

e.g: **SELECT * FROM** Student **JOIN** Enrolled **USING** (sid)

<i>inner join result</i>					
<u>sid</u>	name	birthdate	country	<u>uos_code</u>	semester
112	'A'	01.01.84	India	SOFT1	1
200	'B'	31.5.79	China	COMP2	2

e.g: **SELECT * FROM** Student **NATURAL LEFT OUTER JOIN** Enrolled

<i>left outer join result</i>					
<u>sid</u>	name	birthdate	country	<u>uos_code</u>	semester
112	'A'	01.01.84	India	SOFT1	1
200	'B'	31.5.79	China	COMP2	2
210	'C'	29.02.82	Australia	null	null



› Basic SQL Queries

› Join Queries

› **Set Operations**

- › The set operations **UNION**, **INTERSECT**, and **EXCEPT** operate on tables and correspond to the relational algebra operations \cup , \cap , $-$.
- › Each of the above operations automatically eliminates duplicates.
 - First eliminate duplicates from the input tables, and then do the set operation
 - *Suppose a tuple occurs 3 times in R and 1 times in S, then it occurs 0 times in R **EXCEPT** S*
- › To retain all duplicates, use the corresponding multiset versions **UNION ALL**, **INTERSECT ALL** and **EXCEPT ALL**.
- › Suppose a tuple occurs m times in R and n times in S , then it occurs:
 - $m + n$ times in R **UNION ALL** S
 - $\min(m, n)$ times in R **INTERSECT ALL** S
 - $\max(0, m - n)$ times in R **EXCEPT ALL** S

Example: Set Operations

- › Find all customer names that have a loan, an account, or both:

```
SELECT customer_name FROM depositor  
UNION  
SELECT customer_name FROM borrower
```

```
Depositor(customer_name, account_balance)  
Borrower(customer_name, loan_amount)
```

- › Find all customer names that have both a loan and an account

```
SELECT customer_name FROM depositor  
INTERSECT  
SELECT customer_name FROM borrower
```

- › Find all customer names that have an account but no loan

```
SELECT customer_name FROM depositor  
EXCEPT  
SELECT customer_name FROM borrower
```

- › Find students who enrolled in either 'COMP5138' or 'INFO6007'.

```
SELECT sid FROM Enrolled WHERE uos_code='COMP5138'  
UNION  
SELECT sid FROM Enrolled WHERE uos_code='INFO6007'
```

- This is equivalent to the following SQL command without set operation

```
SELECT sid  
FROM Enrolled  
WHERE uos_code='COMP5138' OR uos_code='INFO6007'
```

- › Find students who enrolled in both 'COMP5138' and 'INFO6007'.

```
SELECT sid FROM Enrolled WHERE uos_code='COMP5138'  
INTERSECT  
SELECT sid FROM Enrolled WHERE uos_code='INFO6007'
```

- This is not equivalent to

```
SELECT sid  
FROM Enrolled  
WHERE uos_code='COMP5138' AND uos_code='INFO6007'
```

- › Ramakrishnan/Gehrke (3rd edition - the 'Cow' book (2003))
 - **Chapter 5**
uses the famous 'Sailor-database' as examples
- › Kifer/Bernstein/Lewis (2nd edition – 2006)
 - Chapter 5
includes some helpful visualisations on how complex SQL is evaluated
- › Ullman/Widom (3rd edition – 2008)
 - Chapter 6
up-to 6.5 good introduction and overview of all parts of SQL querying
- › Silberschatz/Korth/Sudarshan (5th edition - 'sailing boat')
 - Sections 3.1-3.6
- › Elmasri/Navathe (5th edition)
 - Sections 8.4 and 8.5.1

› ...formulate basic SQL Queries

- Select-From-Where Query
- Join queries
- Set operations

› ...know how SQL relates to the relational algebra

- Write equivalent relational algebra expressions for SQL queries

› Advanced SQL

- Nested Queries
- Aggregation & Grouping

› NULL Values

› Readings:

- **Ramakrishnan/Gehrke (Cow book), Chapter 5 & Chapter 4.2.5**
- Kifer/Bernstein/Lewis book, Chapter 5 & 3.2-3.3
- Ullman/Widom, Chapter 6

See you next week!



THE UNIVERSITY OF
SYDNEY