COMP9120

Week 4: Relational Algebra & SQL

Semester 1, 2025

Professor Athman Bouguettaya
School of Computer Science

THE UNIVERSITY OF SYDNEY

Contents co-developed by Dr Lijun Chang

Warming up

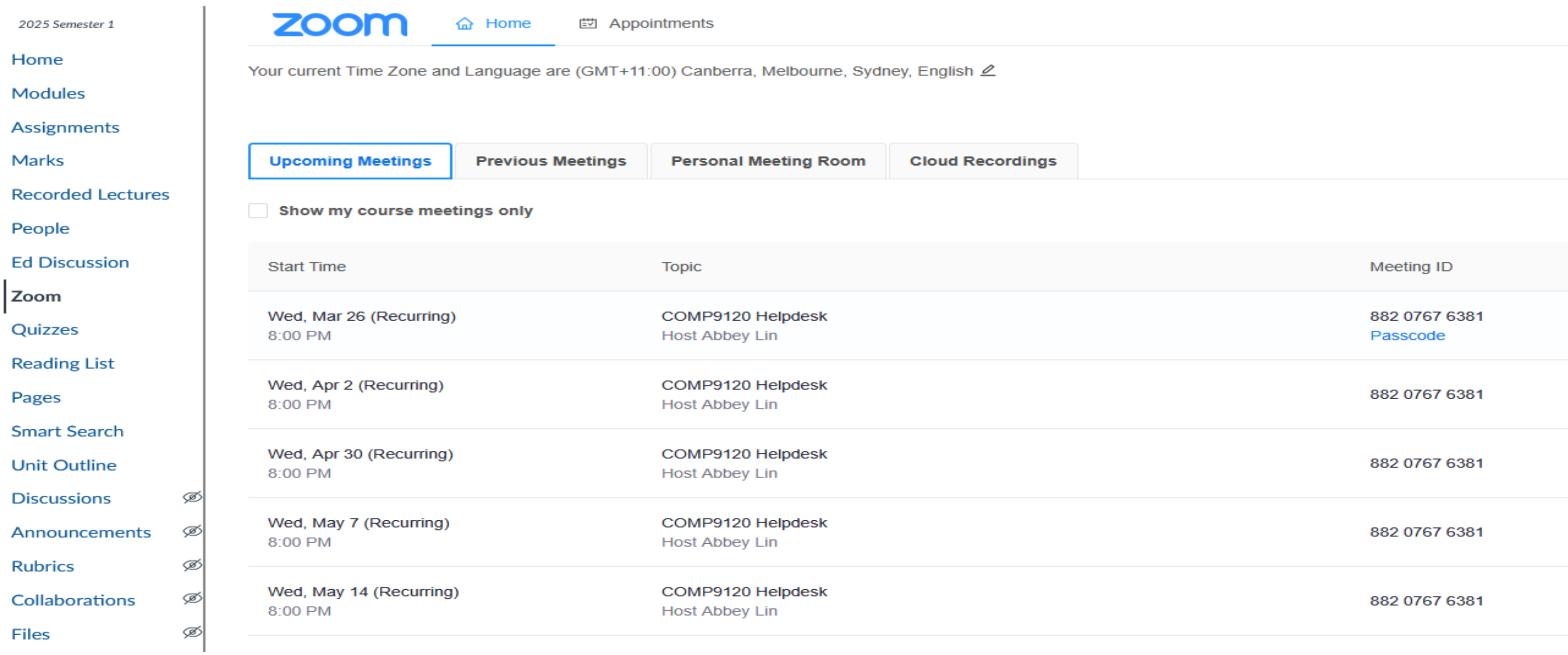
Acknowledgement of Country

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

Copyright warning

Schedule of Zoom Drop-in Helpdesk

- **Accessible from The Zoom link on Canvas.**
- **Drop-in Helpdesk**:
  - Week **5, 6** and **9, 10, 11**



THE UNIVERSITY OF SYDNEY

REMINDER

WEEKLY LIVE HELP ON ED: *Started last week!*

To access the **weekly schedule**, please go to Canvas:

https://canvas.sydney.edu.au/courses/63042/modules

under Modules, under NEW! LIVE ED SCHEDULE S1 2025

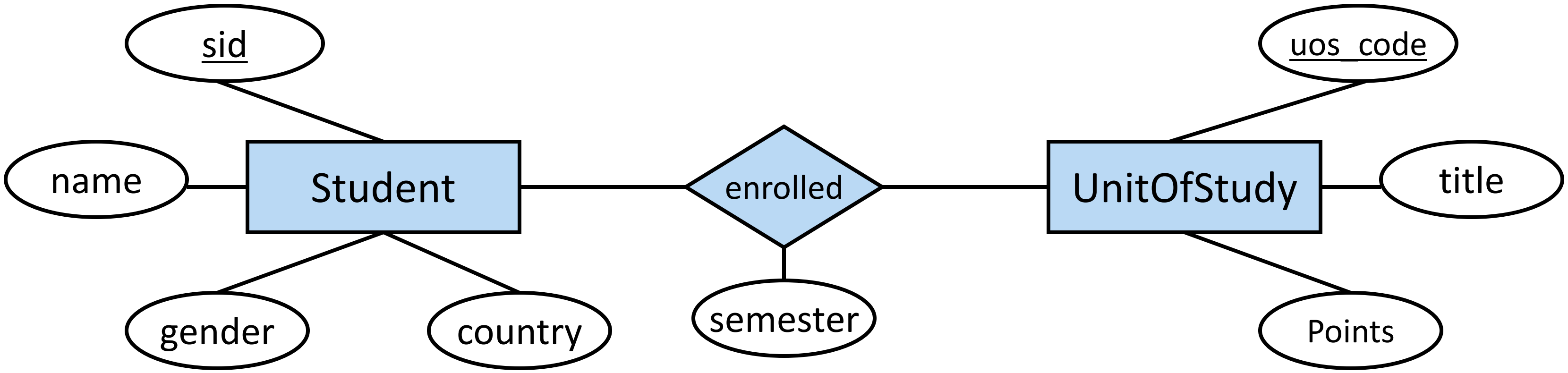
How: The indicated tutor (e.g., Dipankar) will be on Ed during the hours mentioned in the schedule to answer your questions **live**!

Week	Weekday	FROM	TO	Name
3	TUE	6:00 pm	8:00 pm	Dipankar
3	THUR	11:00 am	1:00 pm	Dipankar
4	TUE	6:00 pm	8:00 pm	Dipankar
4	THUR	11:00 am	1:00 pm	Dipankar
5	TUE	6:00 pm	8:00 pm	Dipankar
5	WED	6:00 pm	8:00 pm	Dipankar
5	THUR	11:00 am	1:00 pm	Dipankar
6	TUE	6:00 pm	8:00 pm	Dipankar
6	WED	6:00 pm	8:00 pm	Dipankar
6	THUR	11:00 am	1:00 pm	Dipankar
6	WED	6:00 pm	8:00 pm	Abbey
6	THUR	11:00 am	12:00 pm	Abbey
7	TUE	6:00 pm	8:00 pm	Dipankar
7	THUR	11:00 am	1:00 pm	Dipankar
8	TUE	6:00 pm	8:00 pm	Dipankar
8	THUR	11:00 am	1:00 pm	Dipankar
9	TUE	6:00 pm	8:00 pm	Dipankar
9	WED	6:00 pm	8:00 pm	Dipankar
9	THUR	11:00 am	1:00 pm	Dipankar
10	TUE	6:00 pm	8:00 pm	Dipankar
10	WED	6:00 pm	8:00 pm	Dipankar
10	THUR	11:00 am	1:00 pm	Dipankar
10	WED	6:00 pm	8:00 pm	Abbey
11	THUR	11:00 am	12:00 pm	Abbey
11	TUE	6:00 pm	8:00 pm	Dipankar
11	WED	6:00 pm	8:00 pm	Dipankar
11	THUR	11:00 am	1:00 pm	Dipankar
11	WED	6:00 pm	8:00 pm	Abbey
11	THUR	11:00 am	12:00 pm	Abbey
12	TUE	6:00 pm	8:00 pm	Dipankar
12	THUR	11:00 am	1:00 pm	Dipankar
13	TUE	6:00 pm	8:00 pm	Dipankar
13	WED	6:00 pm	8:00 pm	Dipankar
13	THUR	11:00 am	1:00 pm	Dipankar

Outline

- **Relational Algebra**: an algebra for relational model
  - Six basic operators
- **Data Manipulation Language:** Introduction to **SQL**
  - Basic SQL queries
  - Join queries
  - Set operations

Running Example



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	2023-S2
1002	COMP5702	2023-S2
1003	COMP5138	2023-S2
1006	COMP5318	2023-S2
1001	INFO6007	2023-S1
1003	ISYS3207	2023-S2

UnitOfStudy		
uos_code	**title**	**points**
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	Thesis	18

Relational Algebra

Exercise: Evaluating a Simple Query

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	2023-S2
1002	COMP5702	2023-S2
1003	COMP5138	2023-S2
1006	COMP5318	2023-S2
1001	INFS6014	2023-S1
1003	ISYS3207	2023-S2

UnitOfStudy		
uos_code	title	points
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	Thesis	18

Using the above database instance, **find the titles of all units worth 6 credit points**.

Think about the steps we have to take to get the output.

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:

Two steps:

Relational Algebra expression:

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

UnitOfStudy

uos_code	title	points
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	Thesis	18

$\sigma_{points=6}$

uos_code	title	points
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6

π_{title}

title
Relational DBMS
Data Mining
IT Project Management

Example

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	2023-S2
1002	COMP5702	2023-S2
1003	COMP5138	2023-S2
1006	COMP5318	2023-S2
1001	INFS6014	2023-S1
1003	ISYS3207	2023-S2

UnitOfStudy		
uos_code	title	points
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	Thesis	18

Find the id of all students who are enrolled in COMP5138

$$\pi_{sid}(\sigma_{uos\_code='COMP5138'}(\text{Enrolled}))$$

sid
1001
1003

Relational Algebra (RA)

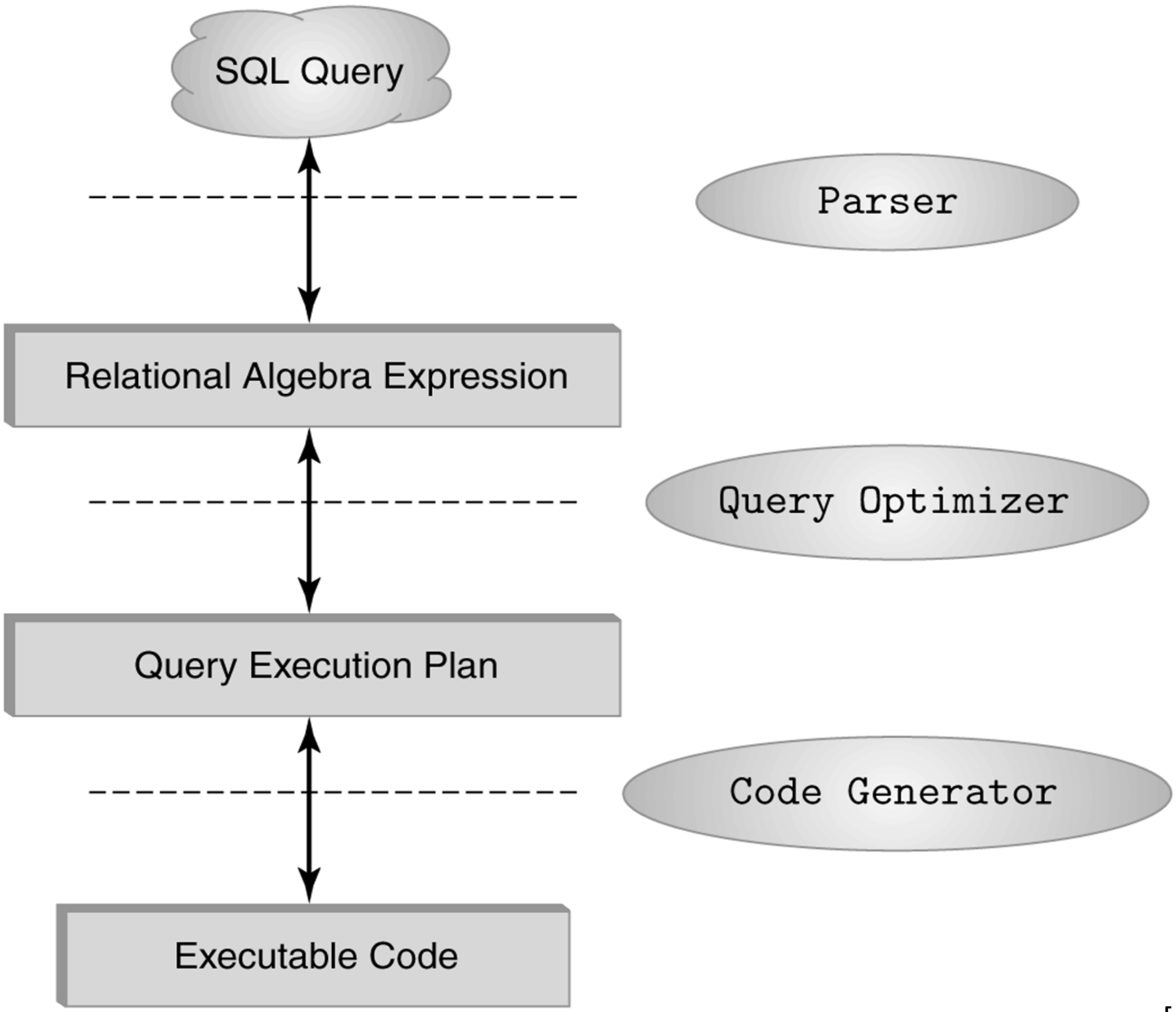
- **Relational algebra** (RA) is an algebra for the ***relational model***
  - Relational algebra operates on **sets**!
  - It is composed of a set of ***operators*** *(e.g., selection, projection, join)*

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

  - It describes a step-by-step procedure (i.e., describes the ***how***) for computing the desired answer - **imperative**
  - **Contrast** with *tuple calculus* which specifies the *constraints* the *results* should *satisfy* (i.e., describes the ***what***) - **declarative**

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

The Role of RA in RDBMS



Declarative query
(from user)

Translate to ***relational algebra expression***

Find logically equivalent but more efficient relational algebra expression

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

RA Operators

Unary operators

1. Operators that *focus* on *one single* relation

› Each operator takes *one* or *more relations* as input and gives *a new relation* as an output

- **Selection** (σ) *selects a subset of rows* from a relation.

› Operators can be *chained* together to form expressions that *represent* queries

- **Projection** (π) *selects a subset of columns* from a relation.

2. A schema-level 'rename' operator $\pi_{title}(\sigma_{points=6}(UnitOfStudy))$

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

Binary operators

1. Operators that *matches* tuples from *two* relations

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

2. Set Operators: the relations must have the *same number* of attributes and *be compatible*

- **Union** ($\cup$) of relations A and B returns all tuples in relation A plus those that are in relation B.
- **Intersection** ($\cap$) of relations A and B returns all tuples in relation A that are matched in relation B (or vice versa).
- **Difference** (-) of relations A and B returns tuples in relation A that have no match in relation B.

Projection (π)

- ‘Extracts’ columns for attributes that are in the *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:

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

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

$\pi_{\mathit{country}}\ (\mathit{Student})$

country
AUS
ROK
GBR
CHN
GER

Selection (σ)

- 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!
  - (called ***Operator composition***)
    - Example:

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

name
Ian
Grant

Exercise

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

- Will the following RA expression work to "Find the name of the students who live in Australia"?

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

name
Ian
Ha Tschi
Grant
Simon
Jesse
Franzisca

Selection Condition

› *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			
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

Find out the male Australian students:

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

sid	**name**	**gender**	**country**
1001	Ian	M	AUS
1003	Grant	M	AUS

Cross Product (x)

› Sometimes also called *Cartesian 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 is the *concatenation* of the fields of *R* and *S*.
  - We might end up in a conflict with two fields having the same name -> use the *rename* operation
- If R or S is empty, then *R* x *S* is also empty.

› 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

=

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

Theta Join: $\bowtie_\theta$

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

Example:

$$\text{Student} \bowtie_{\mathit{Student.f\_name = Lecturer.last\_name} \land \mathit{Student.sid < Lecturer.empid}} \text{Lecturer}$$

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
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
…		…	…	…	…		…	…

› Resulting *schema* is the *cross-product* schema.

› ***Equi-Join***: Special case of *theta join* where the *condition Θ* contains *only* ***equalities***.

Natural Join (⋈)

- **Natural Join**: R ⋈ S
  - ***Equivalent*** *to Equi-join on all common attributes* (i.e., same named attributes), followed by a *projection*
    - Resulting schema is similar to equi-join, however the difference is that we retain *only one common field for which equality is specified*.

Enrolled

sid	uos_code
1001	COMP5138
1002	COMP5702
1003	COMP5138
1006	COMP5318
1001	INFO6007
1003	ISYS3207

⋈

UnitOfStudy

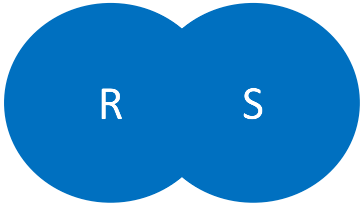
uos_code	title	points
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	Thesis	18

=

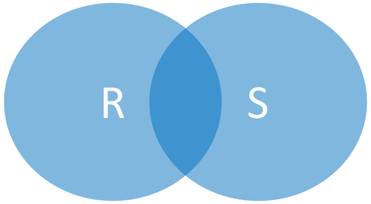
sid	uos_code	title	points
1001	COMP5138	Relational DBMS	6
1002	COMP5702	Thesis	18
1003	COMP5138	Relational DBMS	6
1006	COMP5318	Data Mining	6
1001	INFO6007	IT Project Management	6
1003	ISYS3207	IS Project	4

Set Operations

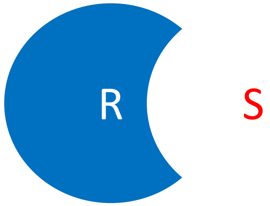
- The **set operations** take two input relations *R* and *S* of the *same arity* (i.e., number of attributes) *and* same attribute *domains (*called *union-compatible)*
  - Set Union $R \cup S$
    - Definition: $R \cup S = \{ t \mid t \in R \lor t \in S \}$



  - Set Intersection $R \cap S$
    - Definition: $R \cap S = \{ t \mid t \in R \land t \in S \}$



  - Set Difference $R - S$
    - Definition: $R - S = \{ t \mid t \in R \land t \notin S \}$



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

Exercise: Set Operations

› Consider the following relations: Use a set operation in an RA expression to **return the *id* of all students who are not postgraduates**.

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

~~Student – Postgraduate~~ ?

Postgraduate
sid
1003
1004
1005

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

sid
1001
1002
1006

Rename Operation (ρ)

- Allows us to rename a relation and its attributes:
- Notation 1: $\rho_X(E)$ (*pronounced* Ro)
  - returns the relation *E* under the name *X*
- Notation 2: $\rho_{X\,(A1,\, A2,\, \ldots,\, An)}(E)$
  - (assumes that the relational-algebra expression *E* has arity *n*)
  - returns the result of relation *E* under the name *X*, and with the attributes renamed to *A1, A2, …., An*.
- Note that rename *only modifies* the schema of a relation!

Example: Rename Operation

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

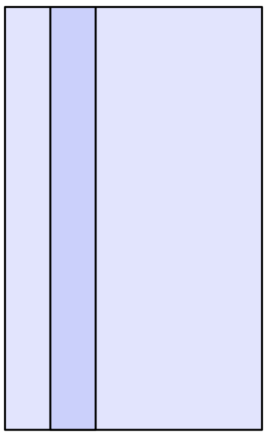
UnitOfStudy		
uos_code	**title**	**points**
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	Thesis	18

UOS		
ucode	**title**	**credits**
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	Thesis	18

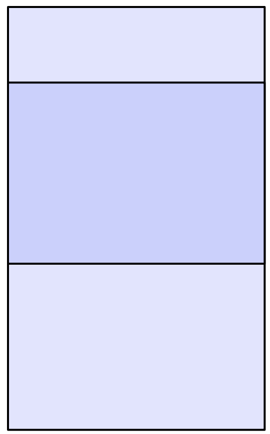
THE UNIVERSITY OF SYDNEY

Visualisation of Relational Algebra Operators

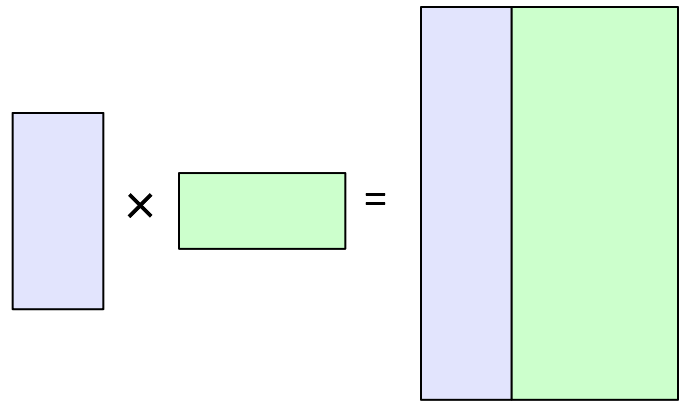
Projection (π)



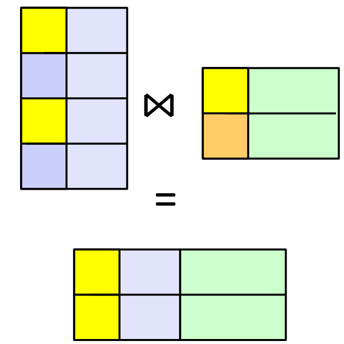
Selection (σ)



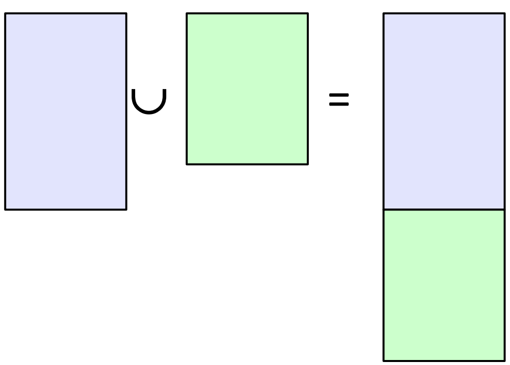
Cross-product (×)



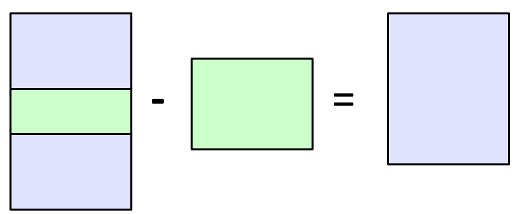
Join (⋈)



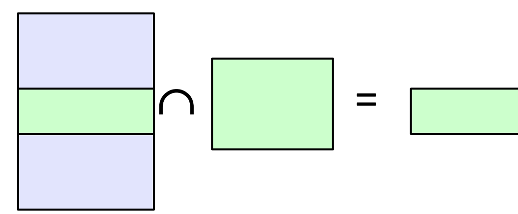
Set Union (∪)



Set Minus (-)



Set Intersection (∩)



Basic Versus Derived Operations

› 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) removes unwanted columns from relation.
- **Cross-product** (**X**) allows us to fully combine two relations.
- **Union** ($\cup$) returns tuples that are in relation 1 or relation 2.
- **Set Difference** (**-**) returns tuples that are in relation 1, but not in relation 2.
- **Rename** (ρ) allows us to rename one field/table name to another name.

› Additional (derived) operations:

- E.g.: *intersection*, *join*. [Not essential, but very useful]
- Equivalence: Intersection: $R \cap S = R - (R-S)$

  Join: $R \bowtie_{\Theta} S = \sigma_{\Theta}(R \times S)$

Intersection of
R = {1, 2, 3}
S = {3, 4} = {3}

R – S = {1, 2 }
R – (R - S) = {3}

Remarks

- Different relational algebra expressions *can be equivalent* but have *different execution costs*
- Example: List the names of all students enrolled in 'Relational DBMS'
  - One way is
    - $\pi_{name}(\sigma_{title='Relational\ DBMS'}((Student \bowtie Enrolled) \bowtie UnitOfStudy))$
  - Another (*more efficient*) way is:
    - $\pi_{name}(Student \bowtie (Enrolled \bowtie (\sigma_{title='Relational\ DBMS'}(UnitOfStudy))))$

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	2023-S2
1002	COMP5702	2023-S2
1003	COMP5138	2023-S2
1006	COMP5318	2023-S2
1001	INFO6007	2023-S1
1003	ISYS3207	2023-S2

UnitOfStudy

uos_code	title	points
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	Thesis	18

Let’s take a break!

and

have some fun!

Introduction to SQL

THE UNIVERSITY OF SYDNEY

Outline

- **Basic SQL Queries**
- Join Queries
- Set Operations

SQL: The Structured Query Language

- Relational algebra is a *theory-based* **procedural** query language
  - May be challenging to non-experts to understand and use
- Reminder: SQL is a 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

SQL Overview

- **DDL** (Data Definition Language)
  - Create, drop, or alter the relation schema

- **DML** (Data Manipulation Language)
  - The <u>insertion</u> of new information into the database
  - The <u>deletion</u> of information from the database
  - The <u>modification</u> of information stored in the database
  - The <u>retrieval</u> 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 access privileges and users

SQL

- 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
```

- List the names of all Australian students.

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

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

- * in select denotes "all attributes".

```
SELECT *
  FROM Student
```

- General form of SFW query:

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

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

$$\pi_{A1,\ A2,\ \ldots,\ An}\ (\sigma_{\ condition}\ (R_1\ \times R_2\ \ \times\ \ldots\ \times\ R_m)\ )$$

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

A Few Details

- SQL **commands** are <u>not</u> case sensitive:
  - **SELECT**, **Select**, **select,** all okay to use
  - 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 and query results.
  - Query result preserves duplicates by default.
  - Example:

SELECT country
FROM Student

country
AUS
ROK
AUS
GBR
CHN
GER

- To eliminate the duplicates, use the keyword **DISTINCT** after **SELECT**.
  - Example: List the distinct countries where students come from.

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

- 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 2024-S2:

```
SELECT uos_code
  FROM Enrolled
 WHERE sid = 1001 AND Semester = '2024-S2'
```

String Operations

- 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.

The *Rename* Operation

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

old_name **AS** new_name

› This is very useful: e.g., give result *columns of expressions* a meaningful name.

› We can also assign names (called *aliases*) to tables as shown in the following example:

› Example:

- Find the uos_code, credit_points for COMP5318,

  and 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_code = 'COMP5318'
```

Order By Clause*

› List all students (i.e., their names) 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**

Outline

- Basic SQL Queries
- **Join Queries**
- Set Operations

Join Query

- An *<u>implicit join </u>* 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’
```

Join Example

› Which students did enroll in what semester?

FROM clause: This is an implicit Join which involves multiple tables

```
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

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

Aliases

Why do we need **aliases**?

› Some queries need to refer to the same table twice

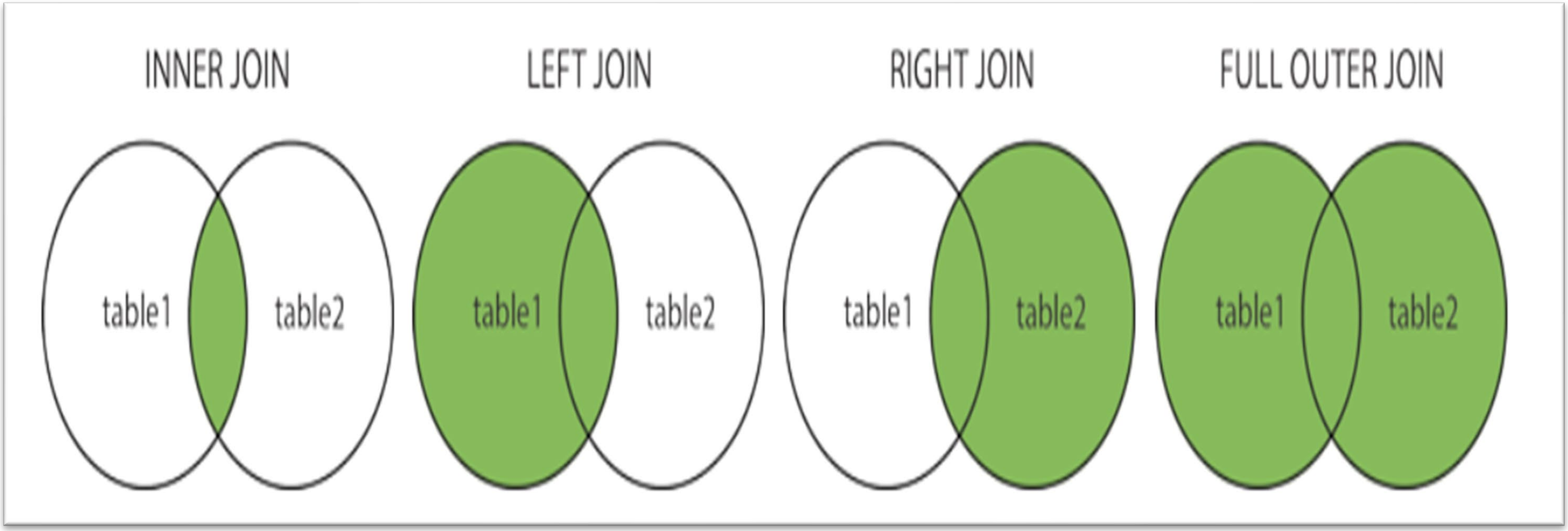
› In this case, aliases are given to the same table name

- Example: For each academic staff, retrieve their name, and the name of his/her immediate supervisor.

  SELECT L.*name*, M.*name*
  FROM Lecturer L, Lecturer M
  WHERE L.*manager* = M.*empid*

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

Using Joins *Explicitly*



- ***Outer join*** *– a join in which rows that do not have matching values are 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*

SQL Join Operators*

› 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)

sid	name	gender	country	uos_code	semester
1001	*Ian*	*M*	*AUS*	*COMP5138*	*2023-S2*
1002	*Ha Tschi*	*F*	*ROK*	*COMP5702*	*2023-S2*
…	…	…	…	…	…

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

sid	name	gender	country	uos_code	semester
1001	*Ian*	*M*	*AUS*	*COMP5138*	*2023-S2*
1002	*Ha Tschi*	*F*	*ROK*	*COMP5702*	*2023-S2*
…	…	…	…	…	…
1005	*Jesse*	*F*	*CHN*	*Null*	*Null*

Outline

- Basic SQL Queries
- Join Queries
- **Set Operations**

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
  - *Example: Suppose a tuple occurs 3 times in R and 1 time 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.**

› Example: 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

R = {a, b, a, c, d, a}
S = {a, e, f, a}

THE UNIVERSITY OF SYDNEY

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(<u>customer_name</u>,account_balance)
Borrower(<u>customer_name</u>, 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
```

THE UNIVERSITY OF SYDNEY

Example: Set Operations*

› Find students who enrolled in either ‘COMP5138’ or ‘ISYS3207’.

```
SELECT sid FROM Enrolled WHERE uos_code='COMP5138'
UNION
SELECT sid FROM Enrolled WHERE uos_code='ISYS3207'
```

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

```
SELECT sid
FROM Enrolled
WHERE uos_code='COMP5138' OR uos_code='ISYS3207'
```

› Find students who enrolled in both ‘COMP5138’ and ‘ISYS3207’.

```
SELECT sid FROM Enrolled WHERE uos_code='COMP5138'
INTERSECT
SELECT sid FROM Enrolled WHERE uos_code='ISYS3207'
```

- This is **not** equivalent to

```
SELECT sid
FROM Enrolled
WHERE uos_code='COMP5138' AND uos_code='ISYS3207'
```

You should now be able to…

- **…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

References

- 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

Next Week

- Integrity Constraints
  - Domain and CHECK constraints
  - ON DELETE and ON UPDATE actions, deferred constraints
  - Assertions

- Readings :
  - Ramakrishnan/Gehrke (3rd edition - the 'Cow' book)
    - **Sections 3.2-3.3 and Sections 5.7-5.9**
    - *Integrity constraints are covered in different parts of the SQL discussion;*
  - Kifer/Bernstein/Lewis (2nd edition)
    - Sections 3.2.2-3.3 and Chapter 7
    - *Integrity constraints are covered as part of the relational model, but a good dedicated chapter (Chap 7) on triggers*
  - Ullman/Widom (3rd edition)
    - Chapter 7
    - *Has a complete chapter dedicated to both integrity constraints&triggers.*

See you next week!