COMP9120

Week 4: Relational Algebra and SQL

Semester 2, 2016

(Ramakrishnan/Gehrke – Chapter 4.2 & 5;

Kifer/Bernstein/Lewis – Chapter 5;

Ullman/Widom – Chapter 2.4 & 6)

Based on material by Dr. Bryn Jeffries

THE UNIVERSITY OF SYDNEY

Flashback Question

title	*year*	*length*	*genre*
Gone With the Wind	1939	231	drama
Star Wars	1977	124	sciFi
Wayne's World	1992	95	comedy

Equivalent Relations?

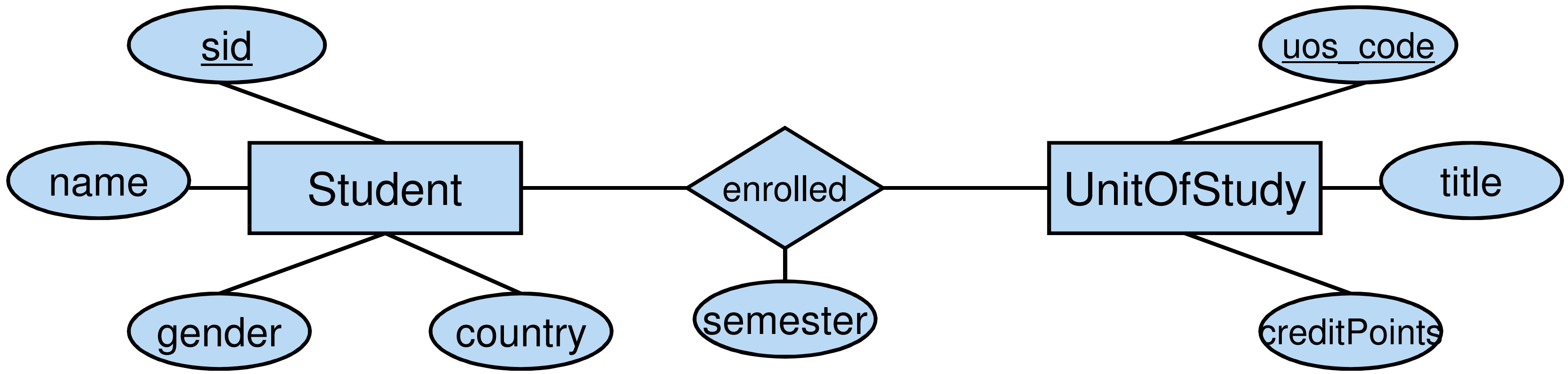
year	*genre*	*title*	*length*
1977	sciFi	Star Wars	124
1992	comedy	Wayne's World	95
1939	drama	Gone With the Wind	231

J. Ulman & J Widom, A First Course in Database Systems, 2008

Lecture Outline

- **Foundations of Declarative Querying**
  - Relational Algebra
  - Six basic operations
  - Join operation
  - Composition and equivalence rules
- Introduction to SQL
  - Overview
  - Basic SQL Queries
  - Joins Queries
  - Aggregate Functions and 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	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**	**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

Exercise 1: 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	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	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, give a corresponding relation that gives the titles of all units worth 6 credit points. Think about the steps you take.

title
Relational DBMS
Data Mining
IT Project Management

How does a RDBMS get the answer?

Give the titles of all units worth 6 credit points

Relational Algebra expression:

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

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	MIT Research Project	18

$$\sigma_{\text{points}=6}$$

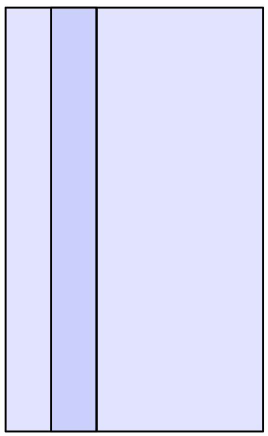
uos_code	title	points
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6

$$\pi_{\text{title}}$$

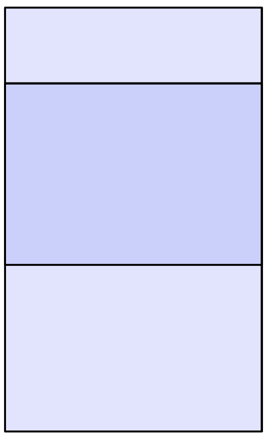
title
Relational DBMS
Data Mining
IT Project Management

Visualisation of Relational Algebra

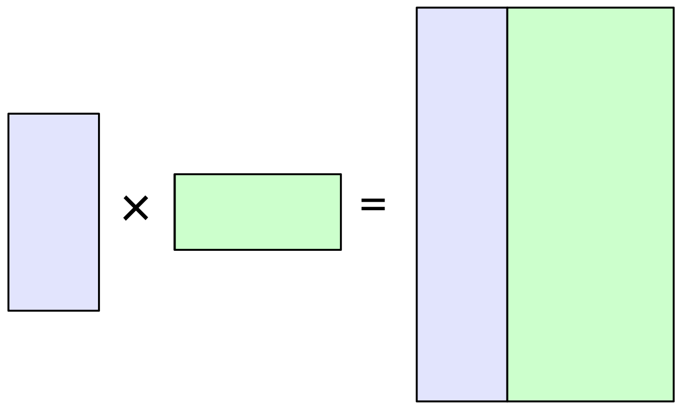
Projection (π)



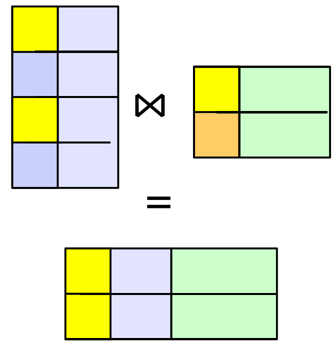
Selection (σ)



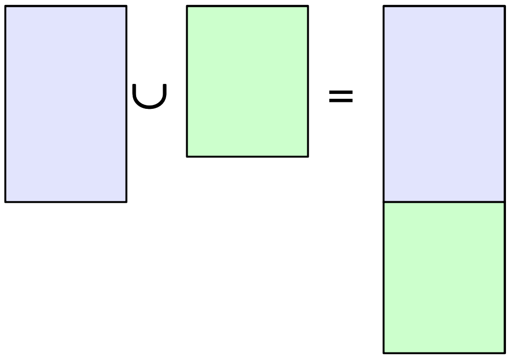
Cross-product (×)



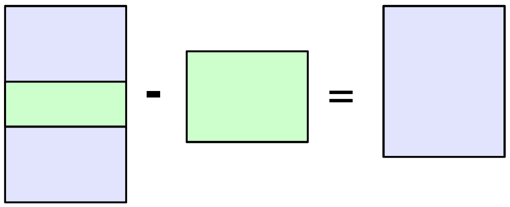
Join (⋈)



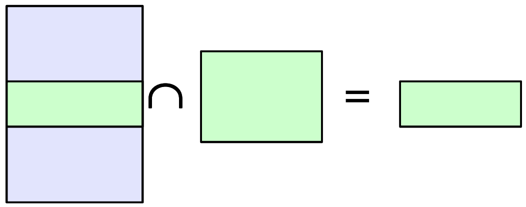
Set Union (∪)



Set Minus (-)



Set Intersection (∩)



Relational Algebra (RA)

- Operators take one/more relations as inputs and give a new relation as result
- Operations can be chained together to form expressions (queries)

1. Operations that remove parts of a relation
   - **Selection** (σ) selects a subset of rows from relation.
   - **Projection** (π) deletes unwanted columns from relation.
2. Operations that combine tuples from two relations
   - **Cross-product** ($\times$) to combine *every* tuple from two relations.
   - **Join** ($\bowtie$) to combine *matching* tuples from two relations.
3. 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.
4. A schema-level 'rename' operation
   - **Rename** (ρ) allows us to rename a field or relation.

Projection

- ‘Extracts’ columns for attributes that are 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:

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

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

$\pi_{title}(UnitOfStudy)$

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			
sid	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: $R \times S = \{t\,s \mid t \in R \land s \in S\}$
  - 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

Conditional Join

› **Conditional Join**:

- Example: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$

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

sid	given	f_name	gender	country	empid	l_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 θ contains only equalities.

Specifying Conditions

- Can compose multiple expressions using logical connectives:
  - $\wedge$ - means AND
  - $\vee$ - means OR
  - ¬ - means NOT
- You can also include comparisons between constants and columns of the input to an operator:
  - Can use <, >, <=, >=, ≠, =

Natural Join

- **Natural Join**: R ⋈ S
  - Equijoin on all common fields.
    - Result schema similar to cross-product, but only one copy of fields for which equality is specified. (according to Ramakrishnan & Gherke also holds for equijoin, but some other texts don't require this for pure equijoin)

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 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

Set Operations

- These operations take two input relations *R* and *S*
  - Set Union $R \cup S$
    - Definition: $R \cup S = \{ t \mid t \in R \lor t \in S \}$
  - 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* have the same schema
  - *R*, *S* have the *same arity* (same number of fields)
  - `Corresponding' fields must have the same names and domains

Exercise 2: Set Operations

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

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

Postgraduate

sid
1003
1004
1005

→

sid
1001
1002
1006

THE UNIVERSITY OF SYDNEY

Rename Operation

- 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: $\rho_{X\,(A1,\,A2,\,\ldots,\,An)}(E)$
  - returns the result of expression *E* under the name *X*, and with the attributes renamed to *A1, A2, …., An*.
  - (assumes that the relational-algebra expression *E* has arity *n*)
- Example: $\rho_{\text{Classlist}(student,cid,title,credits)}(\ Enrolled \times UnitOfStudy\ )$

Basic Versus Derived Operations

- 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:
  - Eg: intersection, join:
    - Not essential, but (very!) useful.
  - E.g., Join: $R \bowtie_{\theta} S = \sigma_{\theta} (R \times S)$

References

- 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 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

Outline

- Foundations of Declarative Querying
  - Relational Algebra
  - Six basic operations
  - Join operation
  - Composition and equivalence rules
- **Introduction to SQL**
  - Overview
  - Basic SQL Queries
  - Joins Queries
  - Aggregate Functions and Set Operations

SQL: The Structured Query Language

- SQL is the standard *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)
- ANSI standard since 1986, ISO-standard since 1987
- 1989: revised to SQL-89
- 1992: more features added – **SQL-92**
- 1999: major rework – **SQL:1999** (SQL 3)
- SQL:2003 – 'bugfix release' of SQL:1999 plus SQL/XML
- SQL:2008 – slight improvements, e.g. INSTEAD OF triggers
- SQL:2011 – quite a few new time based features, eg. time-period definitions

SQL Overview

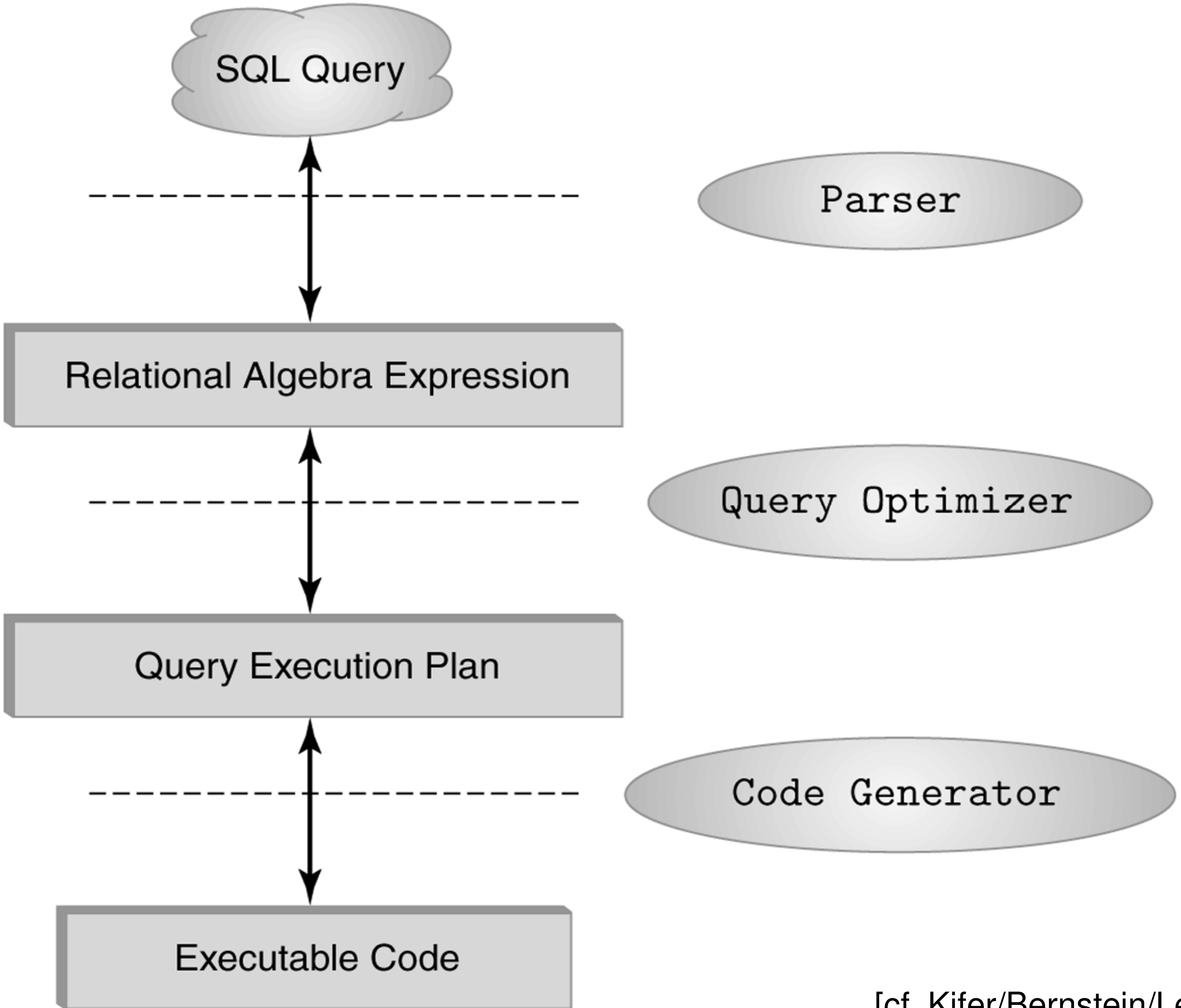
- **DDL** (Data Definition Language)
  - Create, drop, or alter the relation schema
- **DML** (Data Manipulation Language)
  - The retrieval of information stored in the database
    - A **Query** is a statement requesting the retrieval of information
    - The portion of a DML that involves information retrieval is called a **query language**
  - The insertion of new information into the database
  - The deletion of information from the database
  - The modification of information stored in the database
- **DCL** (Data Control Language)
  - Commands that control a database, including administering privileges and users

SELECT Statement

- Used for queries on single or multiple tables
- Clauses of the SELECT statement:
  - **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

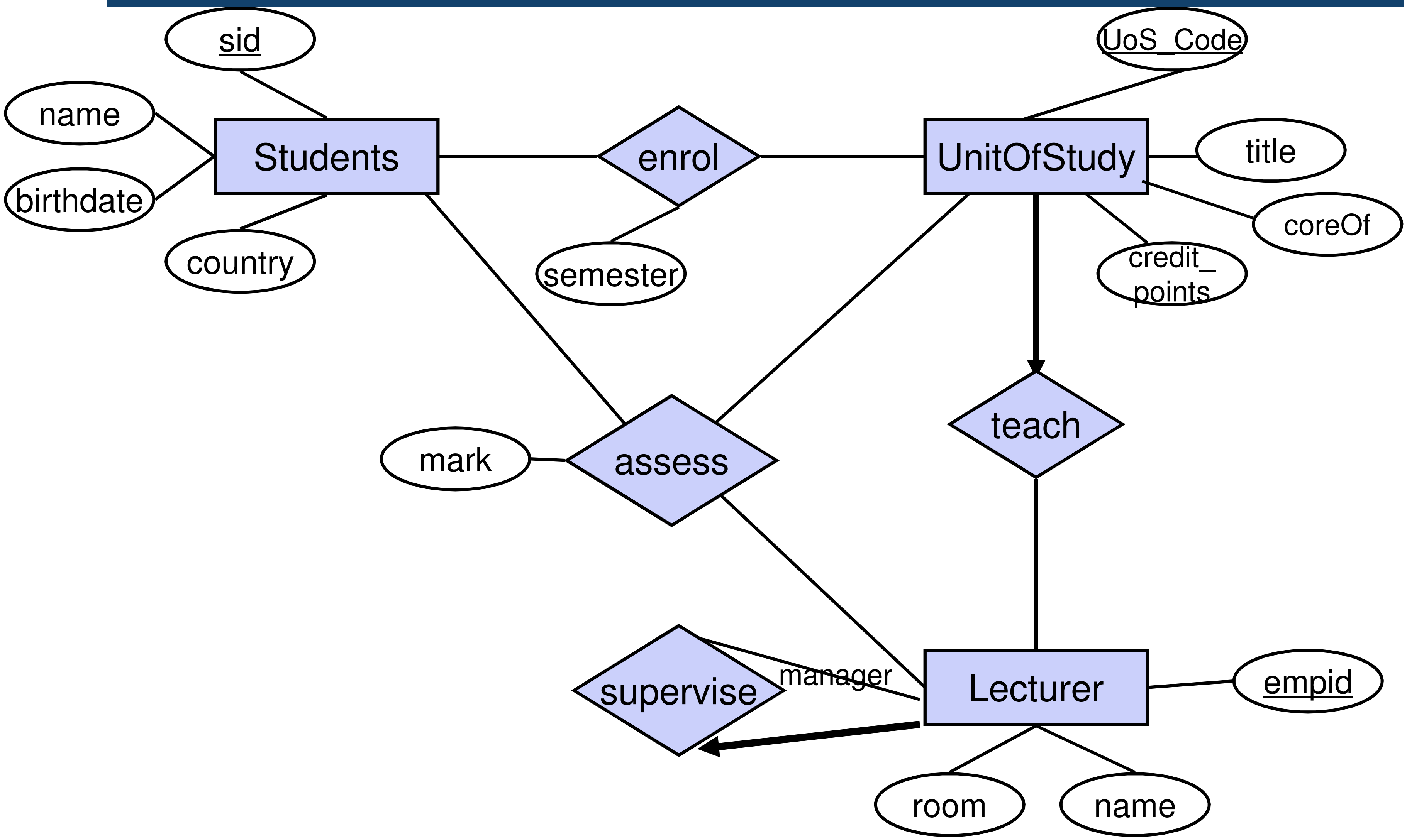
- The result of an SQL query is a relation
- a Select-From-Where (SFW) query is equivalent to the relational algebra expression: $\pi_{A1,\, A2,\, \ldots,\, An}\,(\,\sigma_{condition}\,(R_1 \times R_2 \times \ldots \times R_m)\,)$

The Role of SQL & RA in RDBMS

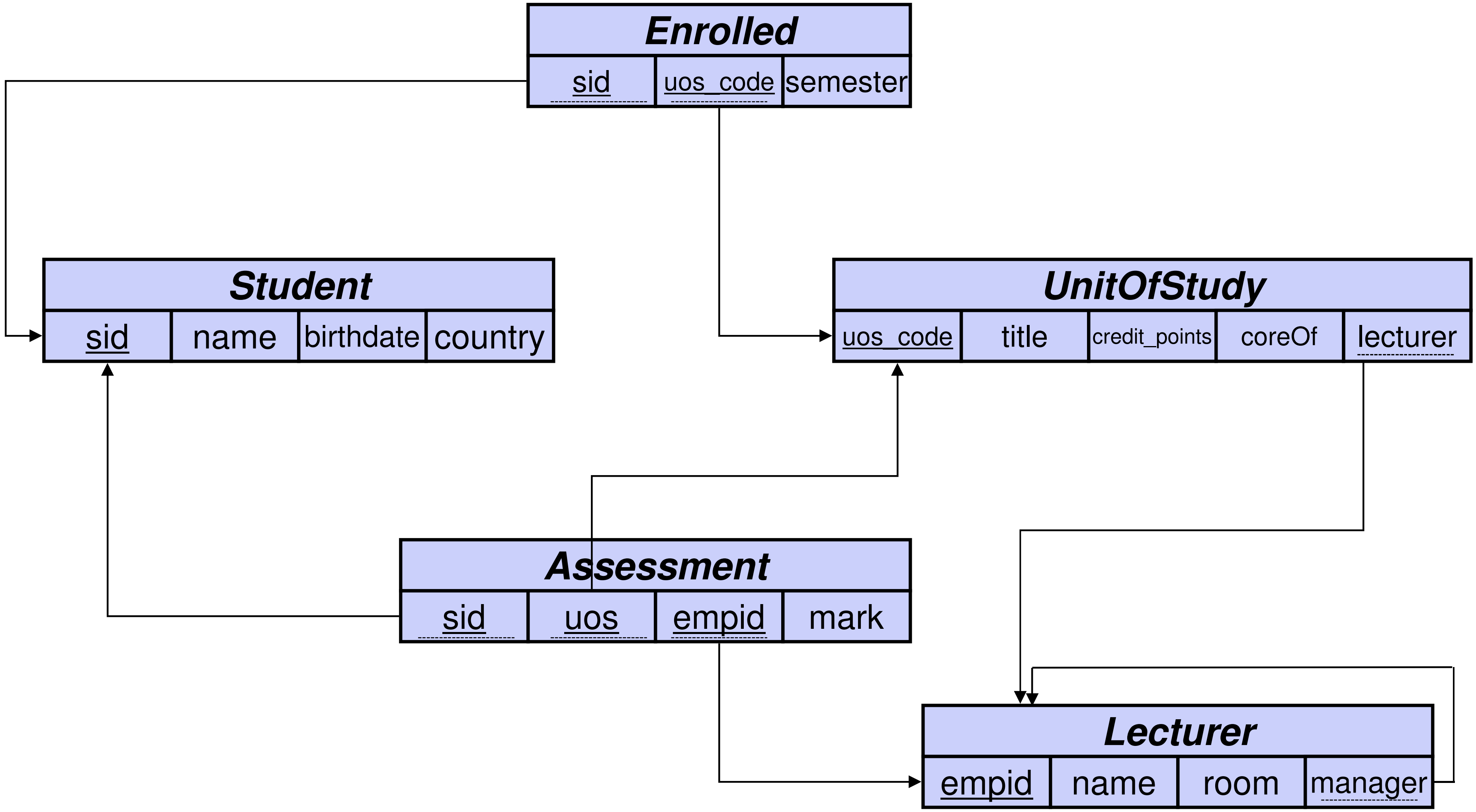


[cf. Kifer/Bernstein/Lewis, Figure 5.1]

Running Example: ER Diagram



Running Example: Database Schema



Example: Basic SQL Query

- List the names of all Australian students.

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

- 'Corresponding' relational algebra expression

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

- Note: RDBMS' can return duplicate rows
  - (eg: having same values in the name column above)
  - Sets containing duplicate entries are sometimes called multi-sets
  - Strict Relational Algebra will only return sets (no duplicate values)

Example: Order By Clause

› 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 oder (default)
- **DESC** descending order

› You can order by more than one attribute

- e.g., **order by** country **desc**, name **asc**

Duplicates

- In contrast to the relational algebra, SQL allows duplicates in relations as well as in query results.

- To force the elimination of duplicates, insert the keyword **distinct** after **select.**

- Example: List the countries where students come from.

```
select distinct country
  from Student
```

- The keyword **all** specifies that duplicates not be removed.

```
select all country
  from Student
```

Arithmetic Expressions in Select Clause

- An asterisk in the select clause denotes „all attributes“

```
SELECT *
  FROM Student
```

- The select clause can obtain arithmetic expressions involving the operations +, -, * and /, and operating on constants or attributs of tuples.

- The query:

```
SELECT uos_code, title, credit_points*2, coreOf, lecturer
FROM UnitOfStudy
```

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

The Rename Operation

- SQL allows renaming relation 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.
- can also assign names to relations as shown in example below

- Example:
  - Find the student id, employee id and lecturer of all assessments for PHYS1001; rename the column name *empid* as *lecturer.*

```
select a.sid, a.empid as lecturer, a.mark
 from Assessment a
where a.uos = 'PHYS1001'
```

The WHERE Clause

- The where clause specifies conditions that the result must satisfy
  - ‘corresponds’ to the selection predicate of the relational algebra.
- 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 taught by lecturer 1011 that are worth more than four credit points:

```
SELECT uos_code
 FROM UnitOfStudy
WHERE lecturer = 1011 AND credit_points > 4
```

The FROM Clause

- The **from** clause lists the relations involved in the query
  - corresponds to the Cartesian product operation of the relational algebra.
  - 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 INFO2120:

```
SELECT sid, name, gender
  FROM Student, Enrolled
 WHERE Student.sid = Enrolled.sid AND
       uos_code = 'INFO2120'
```

Join Example

› 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

Aliases

› Some queries need to refer to the same relation twice

› In this case, aliases are given to the relation name

- Example: For each academic, retrieve the academic's name, and the name of his or 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 **Lecturer**; **L** represents lecturers in role of supervisees and **M** represents lecturers in role of supervisors (managers)

Agenda

- Overview
- Basic SQL Queries
- **Join Queries**
- Aggregate Functions and Set Operations

More on Joins

- **Join** – a relational operation that causes two or more tables to be combined into a single table
- **Theta join** – a join in which tuples are combined if they meet some condition (denoted θ)
- **Equi-join** – a simple case of a theta join in which the joining condition is based on equality between values in the common columns (columns with same name/domain) – common columns appear only once (Ramakrishnan & Gherke)
- **Natural join** – an equi-join in which we consider all common columns (columns with same name/domain) – common columns appear only once
- ***Left / Right Outer join*** *– a join in which rows that do not have matching values in common columns are nonetheless also included (exactly once) in the result table (either rows from the relation to the left or right of the join expression that do not have matching values in common columns are also included – depending on whether we have a left or right outer join.)*
  - *(as opposed to inner join, in which rows must have matching values in order to appear in the result table)*
- ***Full outer join*** *– includes rows that would be found in an equi-join as well as rows from the left relation that don't have matches, and rows from the right relation that don't have matches*

The common columns in joined tables are usually the primary key of the dominant table and the foreign key of the dependent table in 1:M relationships

SQL Join Operators

- SQL offers join operators to directly formulate the natural join, equi-join, and the theta join RA operations.
  - R **natural join** S
  - R **inner join** S **on** <join condition>
  - R **inner join** S **using** (<list of attributes>)
- These additional operations are typically used as subquery expressions in the from clause
  - List all students and in which courses they enrolled.

```
select name, uos_code, semester
  from Student natural join Enrolled
```

  - Who is teaching “INFO2120”?

```
select name
  from UnitOfStudy inner join Lecturer on lecturer=empid
 where uos_code='INFO2120'
```

Join Operators

- Available join types:
  - **inner join**
  - **left outer join**
  - **right outer join**
  - **full outer join**
- Join Conditions:
  - **natural**
  - **on** <*join condition*>
  - **using** <*attribute list*>

e.g: *Student* **inner join** *Enrolled* **using** *(sid)*

inner join result					
sid	name	birthdate	country	uos_code	semester
112	*'A'*	*01.01.84*	*India*	*SOFT1*	*1*
200	*'B'*	*31.5.79*	*China*	*COMP2*	*2*

e.g : *Student* **left outer join** *Enrolled* **using** *(sid)*

left outer join *result*					
sid	name	birthdate	country	uos_code	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*

Agenda

- Overview
- Basic SQL Queries
- Join Queries
- **Aggregate Functions and Set Operations**

Aggregate Operators/Functions

- These functions operate on the *multiset* of values of a column of a relation, and return a value

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- Must use **distinct** in addition to aggregate over *sets*

Note: with aggregate functions you can't have single-valued columns included in the **select** clause

Examples of Aggregate Operators/Functions

- How many students enrolled?

```
select count(*) from Enrolled

select count(distinct sid) from Enrolled
```

- Which was the best mark for 'SOFT2007'?

```
select max(mark)
  from Assessment
where uos_code = 'SOFT2007'
```

- What was the average mark for SOFT2007?

```
select avg(mark)
 from Assessment where uos_code='SOFT2007'
```

NULL Values

- It is possible for tuples to have a null value, denoted by `null`, for some of their attributes
  - Integral part of SQL to handle missing / unknown information
  - **null** signifies that a value *does not exist*, it does *not mean "0" or "blank"*!
- The predicate `is null` can be used to check for null values
  - e.g. Find students who don't have a mark for an assessment yet.

```
SELECT sid
  FROM Assessment
 WHERE mark IS NULL
```

- Consequence: Three-valued logic
- The result of any arithmetic expression involving null is null
  - e.g. 5 + null returns null
- However, (most) aggregate functions simply ignore nulls

NULL Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
  - e.g. 5 < *null* or *null* <> *null* or *null* = *null*

- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*, (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: (*true* **and** *unknown*) = *unknown*, (*false* **and** *unknown*) = *false*, (*unknown* **and** *unknown*) = *unknown*
  - NOT: (**not** *unknown*) = *unknown*

- Result of **where** clause predicate is treated as false if it evaluates to unknown
  - e.g: **select** sid **from** assessment **where** mark < 50
    ignores all students without a mark so far

NULL Values and Aggregation

- Aggregate functions except **count(*)** ignore null values on the aggregated attributes
  - result is null if there is no non-null amount

- Examples:
  - Average mark of all assignments

```
SELECT AVG (mark)        -- ignores tuples with nulls
FROM Assessment
```

  - Number of all assignments

```
SELECT COUNT (*)         -- counts all tuples (only with *)
FROM Assessment
```

THE UNIVERSITY OF SYDNEY

Set Operations

› The set operations **union, intersect,** and **except** (Oracle: **minus**) operate on relations and correspond to the relational algebra operations $\cup$, $\cap$, $-$.

› Each of the above operations automatically eliminates duplicates; 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* *(*)*

(*) not supported by Oracle

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)
```

- 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)
```

Examples for Set Operations

- List all names in the database.

```
select name from Student
union
select name from Lecturer
```

- Which students did not enroll in any units?

```
select sid from Student
minus
select sid from Enrolled
```

- Find students who enrolled for ‘COMP5138’ and ‘COMP5318’.

```
select sid from Enrolled where uoscode=‘COMP5138’
intersect
select sid from Enrolled where uoscode=‘COMP5318’
```

Summary

- **Foundations of SQL**
  - Relational Algebra:
    Projection, Selection, Rename, Set Operations, Cross Product,

    Joins (equi-join, theta-join, natural join)
  - how it relates to SQL queries
- **Introduction to SQL**
  - SELECT … FROM … WHERE … ORDER BY …
  - **Joins** in SQL
  - **NULL values and semantic**
- **Aggregate Functions and Set Operations**
  - Count, Sum, Min, Max, Avg, …
  - Union, Intersect, and Except

References

Each database textbook has a pretty standard chapter on SQL that covers all commands that we discussed in this lecture:

- 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
  - Triggers
- 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; only brief on triggers*
  - 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.*

The BETWEEN Operator

- SQL includes a between comparison operator (called “range queries”)

  - Example: Find all students (by SID) who gained marks in the distinction and high-distinction range in COMP5138.

```
SELECT sid
 FROM Assessment
WHERE uos_code = 'COMP5138'   AND
      mark BETWEEN 75 AND 100
```

Example Queries II

› Find the students with marks between 0 and 10.

select *sid*
from *Assessment*
where *mark* **between** 0 **and** 10

› Who is teaching "INFO2120"?

select *name*
from *UnitsOfStudy*, *Lecturer*
where *uos_code*='INFO2120' **and** *lecturer*=*empid*

› List students who are enrolled in INFO2120 in either semester 1 or 2

select *sid* **from** *Enrolled*
where *uos_code* ='INFO2120' **and** *semester* **in** (1, 2)

String Operations

- SQL includes a string-matching operator for comparisons on character strings.
  - **LIKE** is used for string matching
- Patterns are described using two special characters (“wildcards”):
  - percent (%). The % character matches any substring.
  - underscore (_). The _ character matches any character.
- List the titles of all “COMP” unit of studies.

```
select title
 from UnitOfStudy
where uos_code like 'COMP%'
```

- 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.

Regular Expression Matches

- New since SQL:2003: regular expression string matching
  - typically implemented as set of SQL functions, e.g. **regexp_like(…)**
- What are regular expressions?
  - Pattern consisting of *character literals* and/or *metacharacters*
  - *metacharacters* specify how to process a regular expression
    - ( ) grouping
    - | alternative
    - [ ] character list
    - . matches any character
    - * repeat preceeding pattern zero, one, or more times
    - + repeat preceeding pattern one or more times
    - ^ start of a line
    - $ end of line
- Example:

```
  select title
   from UnitOfStudy
  where regexp_like(uos_code,'^COMP[:digit:]{4}')
```

Date and Time in SQL

SQL Type	Example	Accuracy	Description
DATE	‘2012-03-26’	1 day	a date (some systems incl. time)
TIME	‘16:12:05’	ca. 1 ms	a time, often down to nanoseconds
TIMESTAMP	‘2012-03-26 16:12:05’	ca. 1 sec	Time at a certain date: SQL Server: DATETIME
INTERVAL	‘5 DAY’	years - ms	a time duration

- Comparisons
  - Normal time-order comparisons with ‘=‘, ‘>’, <‘, ‘<=‘, ‘>=‘, …
- Constants
  - CURRENT_DATE db system’s current date
  - CURRENT_TIME db system’s current timestamp
- Example: Which students enrolled before today?

Date and Time in SQL (cont'd)

- Database systems support a variety of date/time related ops
  - Unfortunately not very standardized – a lot of slight differences
- Main Operations
  - **EXTRACT(** *component* **FROM** *date* **)**
    - e.g. EXTRACT(year FROM enrolmentDate)
  - **DATE** *string* (Oracle syntax: TO_DATE(*string*,*template*))
    - e.g. DATE '2012-03-01'
    - Some systems allow templates on how to interpret *string*
    - Oracle syntax: TO_DATE('01-03-2012', 'DD-Mon-YYYY')
  - **+/- INTERVAL**:
    - e.g. '2012-04-01' + INTERVAL '36 HOUR'
- Many more -> check database system's manual


TO BE CONTINUED....