ISYS2120: Data & Information Management

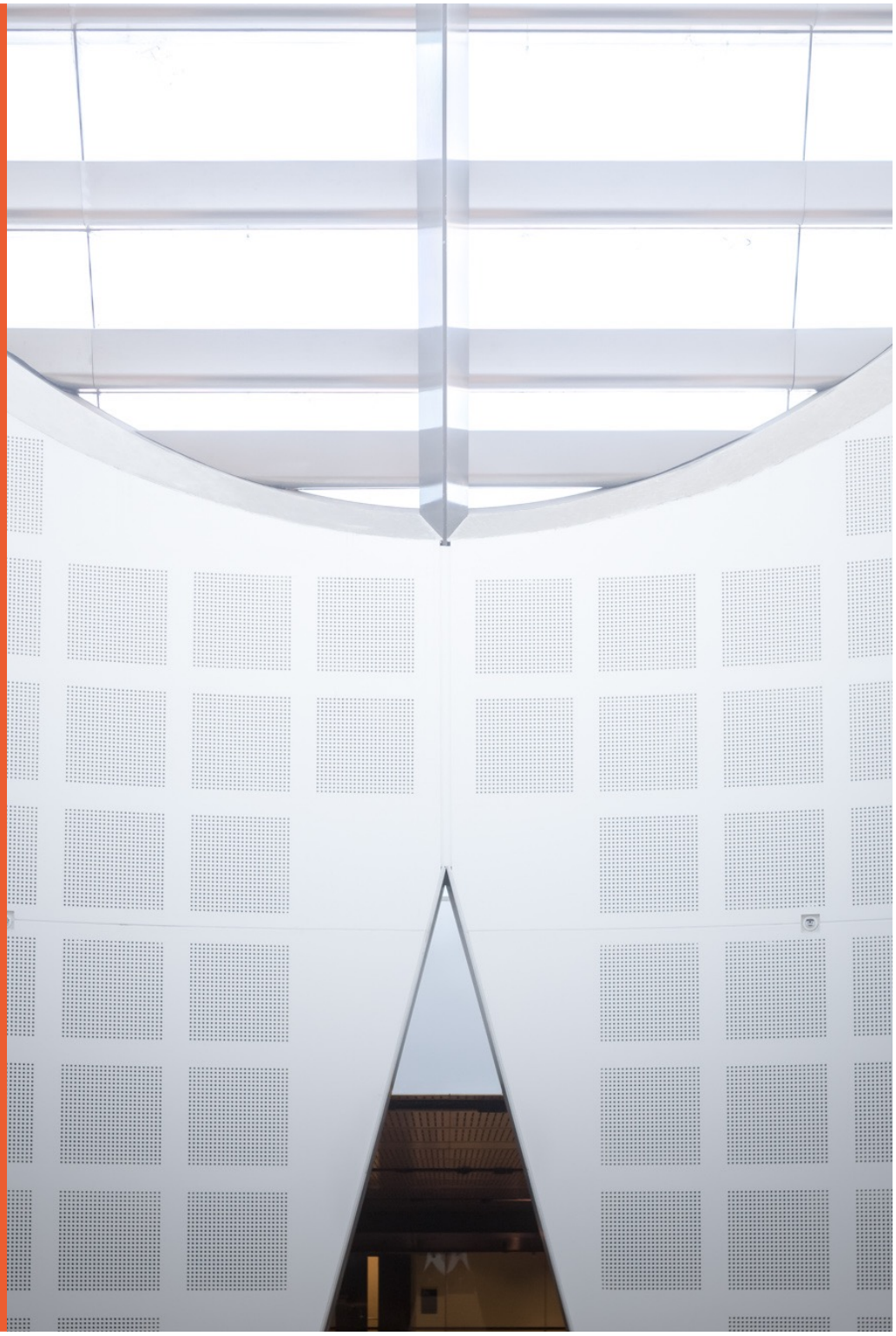
Week 2B: SQL Concepts

Presented by

Alan Fekete

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

THE UNIVERSITY OF SYDNEY



History of SQL (Structured Query Language)

Not examinable

- 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
- SQL:2016
- SQL:2019

SQL Overview

- **DDL** (Data Definition Language) – in lecture 3
  - Create, drop, or alter the relation schema
- **DML** (Data Manipulation Language) –today + lectures 3, 4
  - 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) – in later lectures
  - Commands that control a database, including administering privileges and users

SQL DML

- Commands that retrieve information from the database (SELECT) – today and lecture 4
- Commands that modify the contents of the database (INSERT, DELETE, UPDATE) – lecture 3

Learning SQL

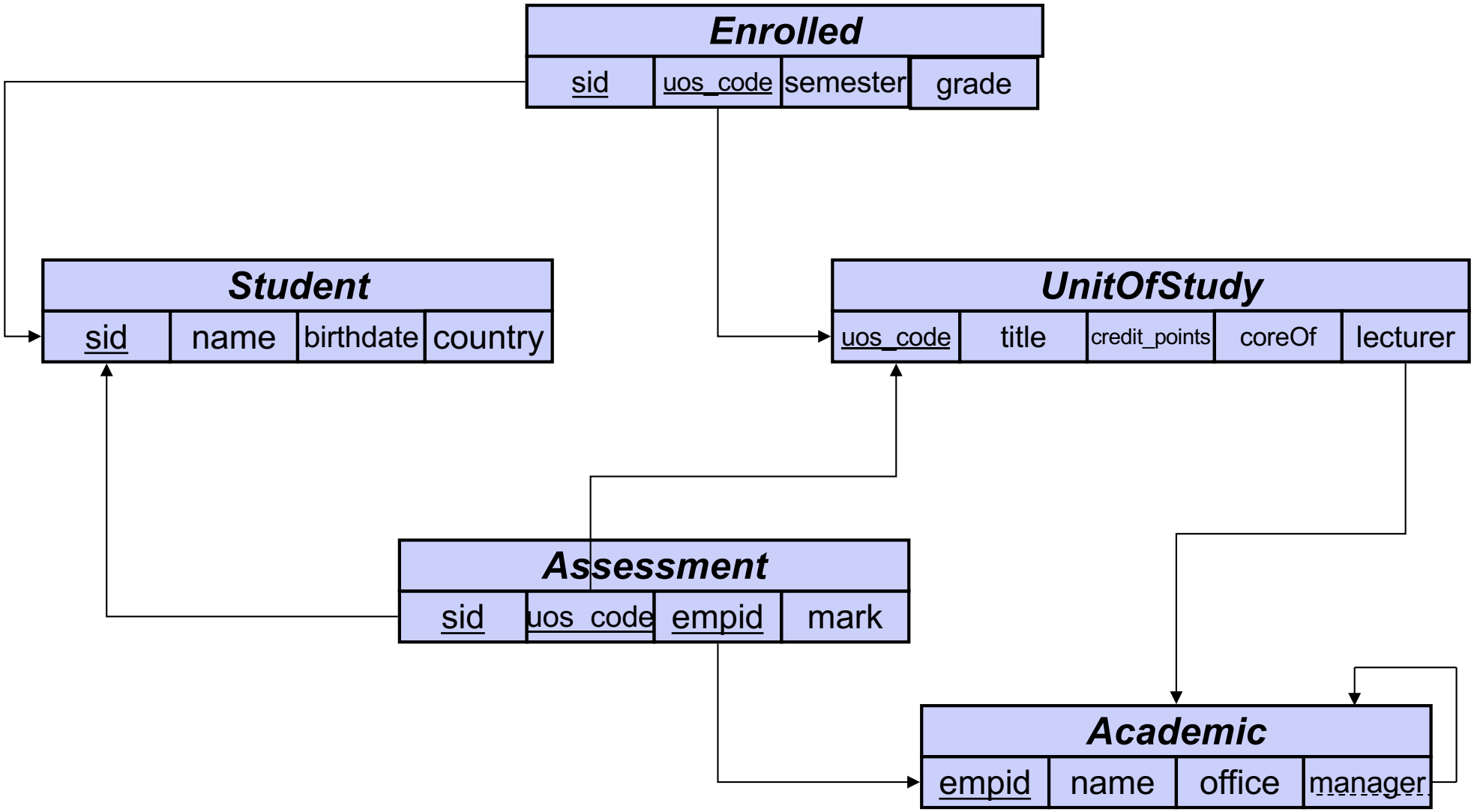
- Essential concepts
  - Understand what result a query will have, when performed on a given state of the database (a “notional machine” for the language)
    - This comes from knowing a way the query might be calculated (it is not necessarily actually performed like that, for improved efficiency)
    - The various language features impact on the calculation done
- Given a query, describe in English what information it ius producing
- Given a statement in English of what a query should do, produce a query that calculates this
  - This is like other programming: find ways to combine the features to have desired effect
  - Often done by finding a somewhat-similar known task/query pair, find what aspects are different, and then adjusting query to suit new task

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 SELECT command is a (single) relation

GROUP BY, HAVING in week 4

Running Example - Database Schema



Example: one-table query

- List the names of all Australian students.

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

- Note: SQL does not permit the '-' character in identifiers (eg table or column names), and
  - SQL identifiers and keywords are case insensitive, i.e. you can use capitals (upper-case) or lower-case letters.
  - You may wish to use upper case where-ever we use bold font on the slides.

Example: order-by

- 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**

Duplicates!

- In contrast to the theory of relational model, SQL allows duplicates in tables as well as in query results.
- To force the elimination of duplicates when calculating a query result, 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, lecturer
  FROM UnitOfStudy
```

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

Renaming

- SQL allows renaming relations and 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.

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

```
select sid, empid as lecturer, grade
 from Assessment
where uos_code = 'PHYS1001'
```

WHERE clause

- The where clause is a logical expression which specifies the condition that rows must satisfy, to be part of the result
  - Similar to Boolean logic in discrete maths, and to Boolean conditions in programming
- Comparison operators in SQL (between values, either constant, or coming from columns of the database): **=** , **>** , **>=** , **<** , **<=** , **!=** , **<>**
- 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 employee 1011 that are worth more than four credit points:

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

BETWEEN in WHERE clause

- SQL includes 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
```

Comments

- A SQL query, or a file with SQL queries, can contain comments (text that is not treated as part of the query, by the database query processor; instead it is there for human readers to understand better)
- One-line comment: from -- till end of line
- Arbitrary comment: from /* till */ and can go over several lines

```
SELECT uos_code
  FROM UnitOfStudy  -- using UnitofStudy table
 WHERE lecturer = 1011
       /* we also must check
      that the unit is not too small in value */
        AND credit_points > 4
```

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.

String constants

- When writing a string constant in a query, be careful about the quotation mark
- SQL uses a single quotation, and not double quotation nor inverted commas
- Many text editors and word processors automatically do inverted commas
  - There may be errors in lecture slides or lab instructions, due to this! Be wary of cut/paste from text into Grok window
- The string whose contents are the four characters Fred is 'Fred' not ‘Fred’ or “Fred”

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:

```
SELECT name
  FROM Student
 WHERE enrolmentDate < CURRENT_DATE
```

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

Multiple tables in FROM Clause

- The **from** clause lists the relations involved in the query
  - If more than one table here, this corresponds to the Cartesian product (that is, all pairs, one coming from each table) of discrete maths.
  - Find the Cartesian product (that is, all pairs, one from each table) *Student* x *UnitOfStudy*

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

Note: this is very rarely what a user wants; the different parts of a row in the result, coming from different tables, are not connected to one another

Connecting Multiple tables

- To limit the results to pairs which match up properly, you can explicitly put a matching condition (join-predicate) in the **where** clause
  - If the same column name appears in more than one of the tables used, one needs to qualify it in a WHERE condition, or in the SELECT list, by preceding with the relation name then fullstop (eg Student.sid is the sid value from the Student table)
  - Eg Find the student ID, name, and gender of all students enrolled in ISYS2120:

```
SELECT Student.sid, name, gender
  FROM Student, Enrolled
 WHERE Student.sid = Enrolled.sid AND
       uos_code = 'ISYS2120'
```

Aliases

- aliases can be given to the relation name in the FROM clause, alias follows the relation name

```
SELECT S.sid, S.name, S.gender
  FROM Student S, Enrolled
 WHERE S.sid = Enrolled.sid AND
       uos_code = 'ISYS2120'
```

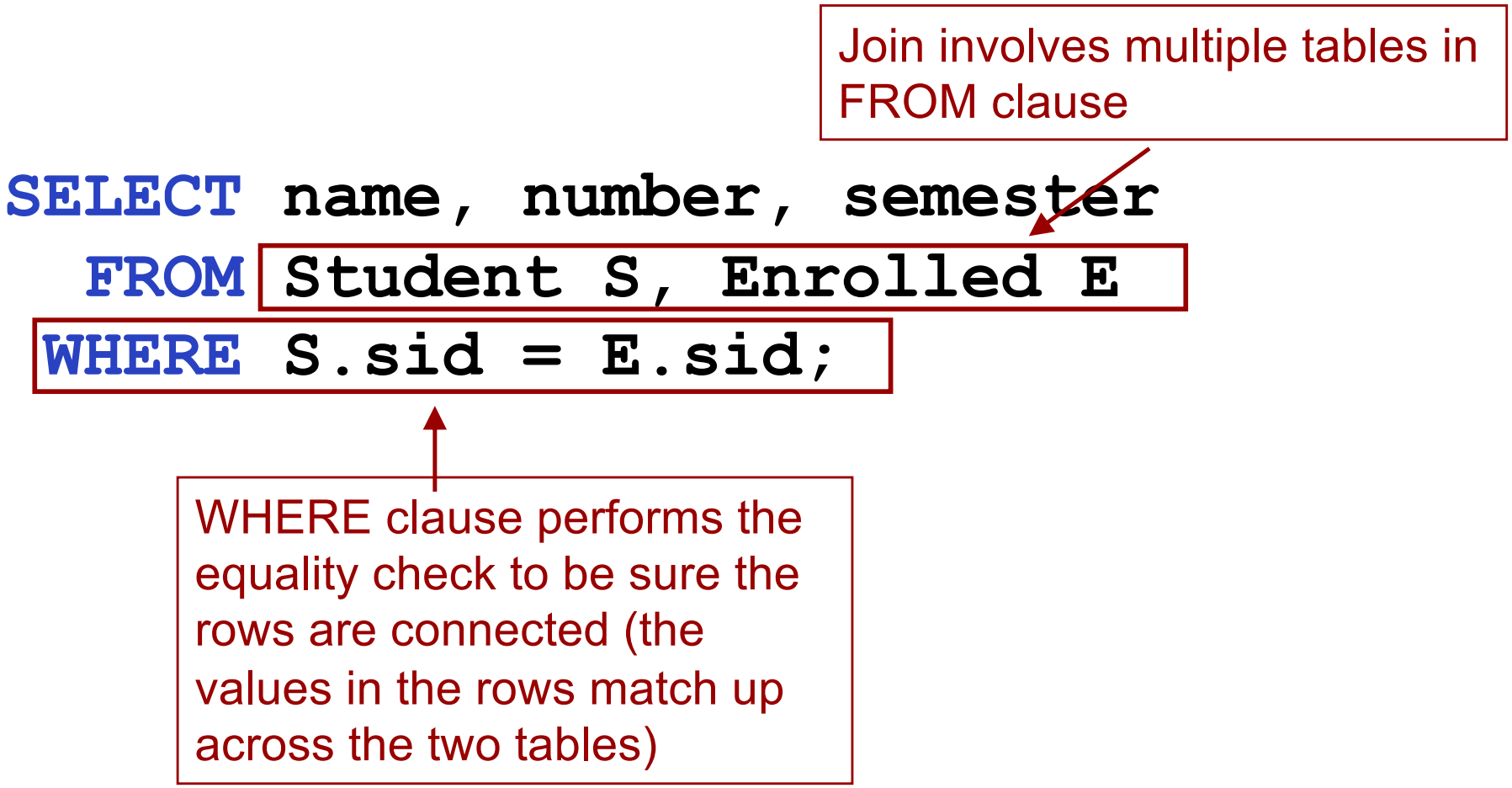
- Why? Some queries need to refer to the same relation twice
- Or, one wants to make reading (or typing) easier, by having a short form of the table name when qualifying an attribute
- Example with same table used twice: For each academic, retrieve the academic's name, and the name of his or her immediate supervisor.

```
SELECT      A.name, M.name
  FROM      Academic A, Academic M
 WHERE      A.manager = M.empid
```

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

Join Example

- Which students were enrolled in what semester?



More on Joins

- **Join** – a relational operation that causes two or more tables with a common domain to be combined into a single table or view
- **Equi-join** – a join in which the joining condition is based on equality between values in the common columns; common columns appear redundantly in the result table
- **Natural join** – an equi-join in which one of the duplicate columns is eliminated in the result table
- **Outer join** – a join in which rows that do not have any matching values in common columns are nonetheless included in the result table (as opposed to inner join, in which rows must have matching values in order to appear in the result table)
- **Union join** – includes all columns from each table in the join, and an instance for each row of each table

Outer join, union join, in week 4

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 do joining without putting a WHERE clause to express the match-up of values.
  - R **natural join** S
    - Put together rows, one from each table, in which the same-named columns have same values (each same-named attribute appears once only)
  - R **inner join** S **on** <join condition>
  - R **inner join** S **using** (<list of attributes>)
- These additional operations are typically used as 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 “ISYS2120”?

```
select name
  from UnitOfStudy inner join Academic on lecturer=empid
 where uos_code='ISYS2120'
```

Revision: Example Queries

- List all units by title
- List all units of study .
- List in alphabetic order the details of all units of study
- Find the students with marks between 0 and 10.
- Who is teaching “ISYS2120”?
- List students who got a distinction or high distinction in ISYS2120.
  - **Tip:** use **in** (not between) comparison operator for sets

Revision: Answers I

- List all units by title

  select *title* **from** *UnitOfStudy*

- List all units of study .

  select * **from** *UnitOfStudy*

- List in alphabetic order the details of all units of study

  select * **from** *UnitOfStudy* **order by** *title*

Revision Answers II

- Find the students with marks between 0 and 10.

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

- Who is teaching “ISYS2120”?

```
select name
from UnitsOfStudy, Academic
where uos_code= 'ISYS2120' and lecturer=empid
```

- List students who got a distinction or high distinction in ISYS2120.

```
select sid
  from Enrolled
 where uos_code = 'ISYS2120' and grade in ('DI', 'HD')
```

An alternative **WHERE** clause **could be**

```
where uos_code = 'ISYS2120' and (grade = 'DI’ or grade = 'HD'
```

Aggregate 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*
- *Be careful: these combine data from many rows, to a single row*

Examples for Aggregate 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'
```

Warning

- Think carefully about the structure of the result calculated
- Why is the following not valid?

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

- *Answer: sid would have a value for each row in Assessment, that meets the condition; but max gives a single value across all the rows*
  - *Not appropriate table structure, for a result where one column has many rows and another column has one row!*

More Exercise Queries

- List all courses by title
- List all information about courses worth 6 crpts.
- In what room is the person teaching ISYS2120
- Find the students with assessments whose mark is less than 10.
- List students who got a distinction or better as their grade in ISYS2120.
- List in alphabetic order the names of all students
- Find the average mark scored in assessments in ISYS2120
- How many students got a grade of 'HD' in a course taught by 'Alan Fekete'.

References

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

- Kifer/Bernstein/Lewis (2nd edition – 2006)
  - Chapter 5
    includes some helpful visualisations on how complex SQL is evaluated
- Ramakrishnan/Gehrke (3rd edition - the 'Cow' book (2003))
  - Chapter 5
    uses the famous 'Sailor-database' as examples
- 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