CSC 355 Database Systems Lecture 6

Eric J. Schwabe

School of Computing, DePaul University

Spring 2020

Topics:

- SQL queries
  - Review GROUP BY and HAVING
  - Query problems
  - Introduction to joins

Aggregate Functions

- Given an attribute, these functions take the values of that attribute in the set of returned rows and compute a single value from them
  - COUNT(…): Number of non-NULL values
  - SUM(…): Sum of the values
  - AVG(…): Average of the values
  - MIN(…): Smallest of the values
  - MAX(…): Largest of the values

GROUP BY

… **GROUP BY** ***grouping attributes*** **…**

- Combines the rows into sets based on the value(s) of some attribute(s)
  - Can only display the value(s) of this attribute(s) and/or aggregate information for each group
  - If we group rows into sets, we cannot look at the values in the individual rows anymore…

HAVING

… **HAVING** ***condition on groups*** …

- Includes only those groups that satisfy the condition
  - the condition may only involve the grouping attribute(s) and/or aggregate functions
  - can use all the same comparisons and logical operators as WHERE

Query Structure (again)

- General form of a query:

SELECT *list of expressions*
  FROM *set of rows*
  [WHERE *condition on rows*]
  [GROUP BY *grouping attributes*]
  [HAVING *condition on groups*]
  [ORDER BY *ordering attributes*] ;

- Grouping goes after WHERE, before ORDER BY

Writing a Query

1. FROM: What table should I use?
2. WHERE: How do I indicate which rows to include?
3. GROUP BY: What attribute’s values will define the sets? (May have to change SELECT * here…)
4. HAVING: How do I indicate which sets to include?
5. ORDER BY: How should I sort the rows/sets?
6. SELECT: What values do I have to compute and display?

Query Problems

- Find the number of workers in each department
  - (…whose salary is more than $40,000)
- Find the average salary over the entire company
- For each department, find the salary of the highest-paid employee
- List the department names and their total budgets, ordered from the largest total budget to the smallest
- For each student, find the total number of classes they have enrolled in and the most recent year that the student enrolled in a class

Joins

- Data that is distributed among multiple tables can be combined into a single set of rows for use in a query using different types of *joins*:
  - Inner joins (equi-join, natural join)
  - Outer joins (left, right, full)

Cartesian Product

- What if we list two tables in the FROM?
- The rows in the result come from combining all pairs of rows from the two tables – the *Cartesian Product* of the tables
  - (This is sometimes called the “cross join”…)
- This is almost certainly more rows than we want – most combinations are meaningless!

Equi-Join

- An equi-join keeps only those rows where the two combined rows agree on the shared attribute(s):

  …FROM *TABLE1*, *TABLE2*
  WHERE
  TABLE1.Attribute = *TABLE2.Attribute*;

Natural Join

- Like an equi-join, but one of the duplicated columns is removed (the most common join):

  SELECT *all but the duplicated attribute(s)*
  FROM *TABLE1, TABLE2*
  WHERE
      TABLE1.Attribute = TABLE2.Attribute;

Inner Joins

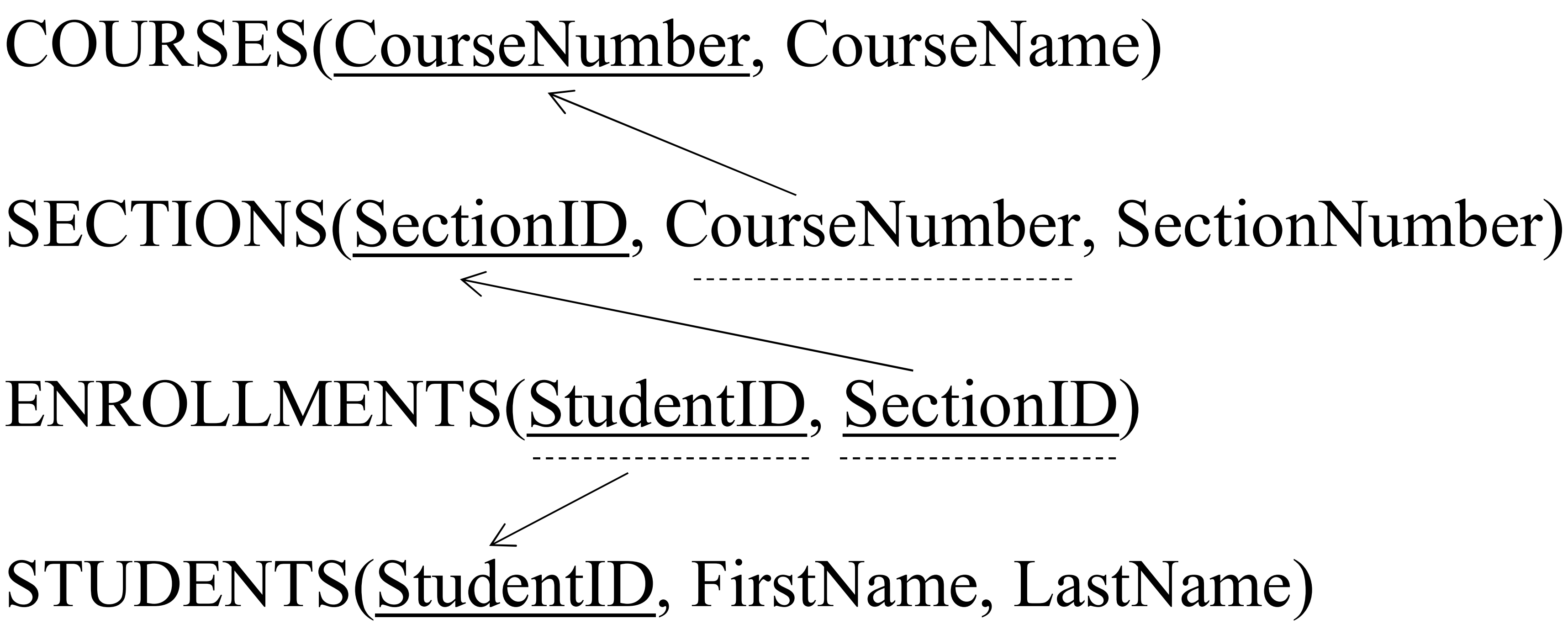
- These are both examples of *inner joins*
- In an inner join, the Cartesian Product is restricted to only include the combined rows that satisfy some condition
  - condition is usually equality in some shared key
  - e.g., equi-joins, natural joins

Inner Joins

- Rather than list of tables in the FROM and a WHERE condition, can use:

  FROM *TABLE1* INNER JOIN *TABLE2*
  ON *condition*

Join Example



COURSES(CourseNumber, CourseName)

SECTIONS(SectionID, CourseNumber, SectionNumber)

ENROLLMENTS(StudentID, SectionID)

STUDENTS(StudentID, FirstName, LastName)

Next:

- More SQL Queries
  - Inner joins
  - Outer joins
  - Query examples
  - Set operations