Introduction to SQL

Peter Scheuermann

Outline

- **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 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

SQL DDL

Remember from last lectures

- Creation of tables (relations):

  ```
  CREATE TABLE name ( list_of_columns )
  ```

  - Create new relation with given *name* and list of *columns*
  - Specify *domain type* for each column
  - Also: Specify ***Integrity Constraints***
    - **PRIMARY KEY** and **FOREIGN KEY REFERENCES** *parent_table*
    - **NULL / NOT NULL** constraints
    - More later on…

- Deletion of tables (relations):

  ```
  DROP TABLE name
  ```

  - the schema information and the tuples are deleted.

SQL DML Statements

- Insertion of new data into a table / relation
  - **Syntax:**
    INSERT INTO *table* [“**(**”*list-of-columns*“**)**”] **VALUES** “**(**“ list-of-*expression* “**)**”
  - Example:

    ```
    INSERT INTO Students (sid, name) VALUES (53688, ‘Smith’)
    ```

- Updating of tuples in a table / relation
  - **Syntax:**
    UPDATE *table* **SET** *column*“**=**“*expression* {“**,**”*column*“**=**“*expression*}
    [**WHERE** *search_condition*]
  - Example:

    ```
    UPDATE students
           SET gpa = gpa - 0.1
     WHERE gpa >= 3.3
    ```

- Deleting of tuples from a table / relation
  - **Syntax:**
    DELETE FROM *table* [**WHERE** *search_condition*]
  - Example:

    ```
    DELETE  FROM Students WHERE name = ‘Smith’
    ```

More details on those in a while...

SELECT Statement

- Used for queries on single or multiple tables

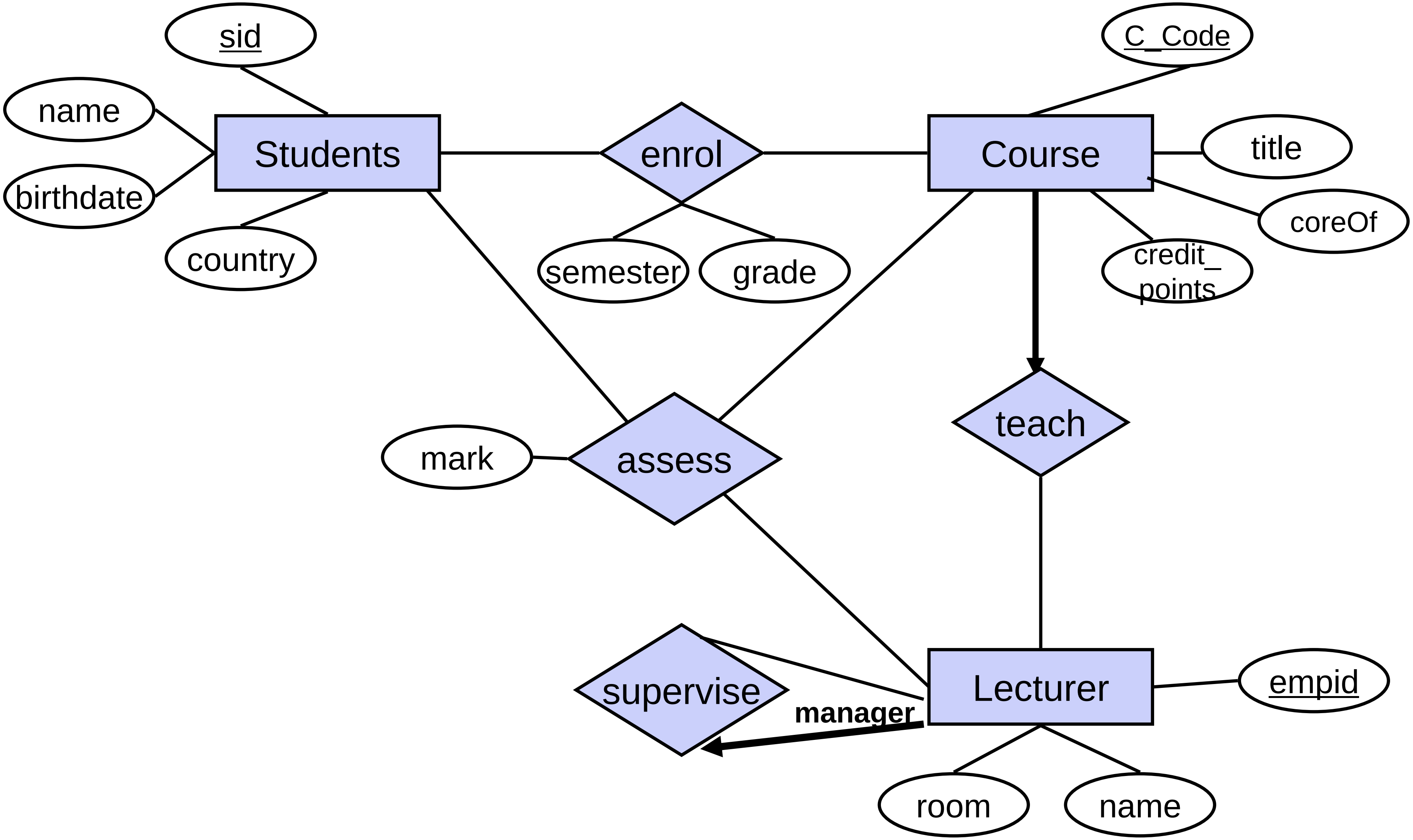
- Clauses of the SELECT statement:
  - **SELECT** Lists the columns (and expressions) that should be returned from the query
  - **FROM** Indicates the table(s) from which data will be obtained
  - **WHERE** Indicates the conditions to include a tuple in the result
  - **GROUP BY** Indicates the grouping of tuples to apply aggregate ops
  - **HAVING** Indicates the conditions to include a group
  - **ORDER BY** Sorts the result according to specified criteria

- The result of an SQL query is a relation

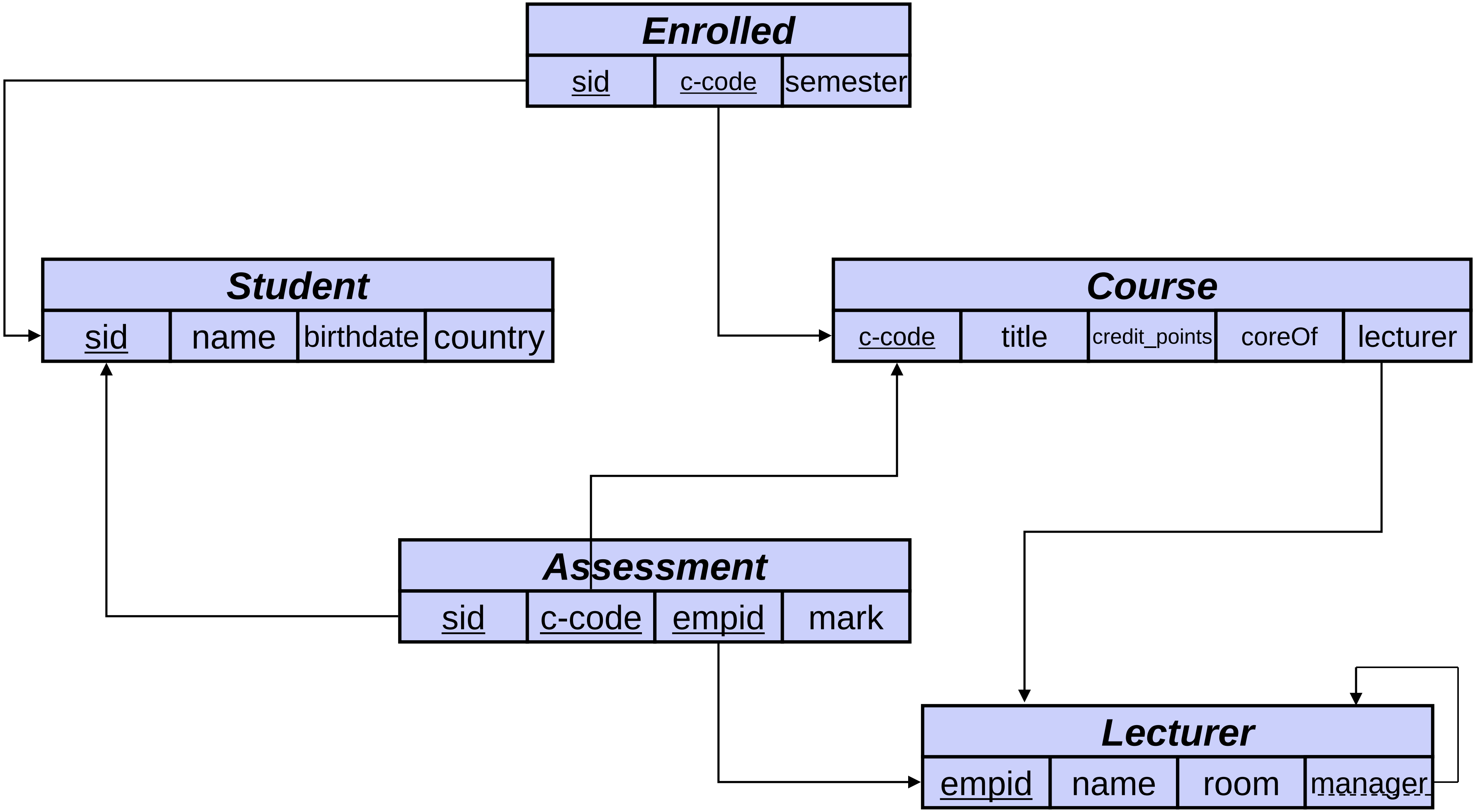
- A SFW-query is equivalent to the relational algebra expression

$$\Pi_{A1,\ A2,\ ...,\ An} \left(\sigma_{condition} \left(R_1 \ \mathrm{x} \ R_2 \ \mathrm{x} \ ... \ \mathrm{x} \ R_m \right) \right)$$

Running Example



Running Example - Database Schema



Enrolled

sid	c-code	semester

Student

sid	name	birthdate	country

Course

c-code	title	credit_points	coreOf	lecturer

Assessment

sid	c-code	empid	mark

Lecturer

empid	name	room	manager

Example: Basic SQL Query

- List the names of all Chinese students.

```
SELECT name FROM Student WHERE country=‘China’
```

- Corresponding relational algebra expression

$$\pi_{\textit{name}} \left(\sigma_{\textit{country}=\text{‘}\textit{China}\text{’}} (\text{Student}) \right)$$

- Note: SQL does not permit the ‘-’ character in names, and SQL names are case insensitive.
- You may wish to use upper case wherever we use bold font.

Example: Order By Clause

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

```
select  name
from  Student
where  country=‘China’
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 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 are 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 contain arithmetic expressions involving the operators +, -, * and /, and operating on constants or attributs of tuples.

- The query:

```
    SELECT c_code, title, credit_points*2, lecturer
      FROM Course
```

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

The Rename Operation

- 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, mark and lecturer of all assessments for PHYS101; rename the column name *empid* as *lecturer.*

    select *sid, empid* **as** *lecturer, mark*

    from *Assessment*

    where *c-code = ‘PHYS101’*

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: **= , > , >= , < , <= , != , <>**

- 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 Course codes for classes taught by employee 1011 that are worth more than one credit points:

```
SELECT c_code
FROM Course
WHERE lecturer = 1011 AND credit_points > 1
```

The WHERE Clause (cont’d)

- SQL includes a Between comparison operator (called “range queries”)
  - Example: Find all students (by SID) who gained high grades in ENG138.

```
SELECT sid
FROM Enrolled
WHERE c_code = ‘ENG138’ AND
     grade BETWEEN 75 AND 100
```

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” courses.

```
select title
  from courses
where c_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.

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 must be explicitly stated in the **where** clause

- Examples:
  - Find the Cartesian product *Student* x *Course*

```
SELECT *
  FROM Student, Course
```

  - Find the student ID, name, and gender of all students enrolled in EECS495:

```
SELECT sid, name, gender
 FROM Student, Enrolled
WHERE Student.sid = Enrolled.sid AND
      c_code = ‘EECS495’
```

Join Example

- Which students did enroll in what semester?

Join involves multiple tables in FROM clause

```
SELECT name,c-code, 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 lecturer, retrieve the lecturer's name, and the name of his or her immediate supervisor.

```
SELECT L.name, M.name
  FROM Lecturer L 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)
  - L and M are also called tuple variables

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 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 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

The common columns in joined tables are usually the primary key of one 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 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 courses they enrolled, with semester added

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

  - Who is teaching “EECS495”?

```
select name
  from Course inner join Lecturer on lecturer=empid
where c_code='EECS495'
```

More 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	sid2	c-code	grade
112	*‘A’*	*01.01.94*	*India*	*112*	*SOFT1*	*P*
200	*‘B’*	*31.5.89*	*China*	*200*	*COMP2*	*C*

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

left outer join *result*

sid	name	birthdate	country	sid2	c- code	grade
112	*‘A’*	*01.01.94*	*India*	*112*	*SOFT1*	*P*
200	*‘B’*	*31.5.89*	*China*	*200*	*COMP2*	*C*
210	*‘C’*	*29.02.90*	*USA*	*null*	*null*	*null*

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

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

Examples for Aggregate Functions

- How many students enrolled?

```
select count(*) from Enrolled
select count(distinct sid) from Enrolled
```

- Which was the best grade for ‘Dmining15’?

```
select max(grade)
  from Enrolled
where c_code = ‘Dmining15’
```

- What was the average mark for Dmining15 ?

```
select avg(grade)
  from Enrolled where c_code=‘Dmining15’
```

Set Operations

- The set operations **union, intersect**, and **except** 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* *(not supported by all dbms)*
  - max(0, *m* – *n)* times in *r* **except all** *s* *(not supported by all dbms)*

Set Operations

- Find all customers who have a loan, an account, or both:

```
(select customer_name from depositor)
   union
(select customer_name from borrower)
```

- Find all customers who have both a loan and an account

```
(select customer_name from depositor)
   intersect
(select customer_name from borrower)
```

- Find all customers who have an account but no loan

```
(select customer_name from depositor)
   except
(select customer_name from borrower)
```

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 which enrolled in a course without a grade so far.

```
SELECT sid
 FROM Enrolled
WHERE grade 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** enrolled **where** grade < 80 or grade >= 80
    ignores all students with null grade

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 Enrolled
```

  - Number of all assignments

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