# A quick guide to writing SQL queries

## MAS 201

## Access (Query) & Modification Language: SQL

- SQL
  - used by the database user
  - **declarative**:we only describe **what** we want to retrieve
  - based on tuple relational calculus
- The result of a query is a table (regardless of the query language used)

## SQL Queries: Basic One-table

- Basic form
  **SELECT** $A_1,\dots,A_N$
  **FROM** R
  **WHERE** <condition>
- **WHERE** clause is optional
- Find all tuples of R that satisfy the (boolean) condition and return their attributes $A_1,\dots,A_N$

Find first names and last names of all students

```
SELECT  first_name, last_name
FROM    students;
```

Find all students whose first name is John; project all attributes

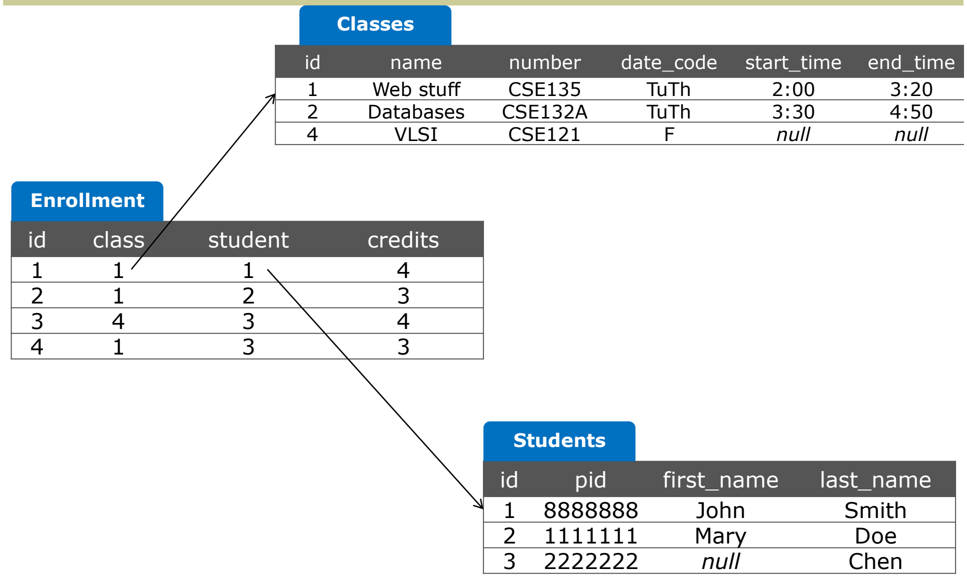
```
SELECT  *
FROM    students
WHERE   first_name = 'John';
```

## SQL Queries: Putting together multiple tables

- Basic form
  **SELECT** $A_1,\dots,A_N$
  **FROM** $R_1,\dots,R_M$
  **WHERE** <condition>
- When more than one relations in the **FROM** clause have an attribute named **A**, we refer to a specific **A** attribute as **<RelationName>.A**
- Hardest to get used to, yet most important feature of SQL

Produce a table that shows the pid, first name and last name of every student enrolled in the class with ID 1, along with the number of credit units in the “class 1” enrollment

# (repeat)



**Classes**

| id | name | number | date_code | start_time | end_time |
|---|---|---|---|---|---|
| 1 | Web stuff | CSE135 | TuTh | 2:00 | 3:20 |
| 2 | Databases | CSE132A | TuTh | 3:30 | 4:50 |
| 4 | VLSI | CSE121 | F | *null* | *null* |

**Enrollment**

| id | class | student | credits |
|---|---|---|---|
| 1 | 1 | 1 | 4 |
| 2 | 1 | 2 | 3 |
| 3 | 4 | 3 | 4 |
| 4 | 1 | 3 | 3 |

**Students**

| id | pid | first_name | last_name |
|---|---|---|---|
| 1 | 8888888 | John | Smith |
| 2 | 1111111 | Mary | Doe |
| 3 | 2222222 | *null* | Chen |

# Take One: Understanding FROM as producing all combinations

```
SELECT  students.pid, students.first_name,
        students.last_name, enrollment.credits
FROM    students, enrollment
WHERE   students.id = enrollment.student
    AND enrollment.class = 1 ;
```

"FROM" produces all 12 tuples made from one "students" tuple and one "enrollment" tuple

← Student part of the tuple → ← Enrollment part of the tuple →

| Students. id | pid | first_name | last_name | Enrollment. id | class | student | credits |
|---|---|---|---|---|---|---|---|
| 1 | 88.. | John | Smith | 1 | 1 | 1 | 4 |
| 1 | 88.. | John | Smith | 2 | 1 | 2 | 3 |
| 1 | 88.. | John | Smith | 3 | 4 | 3 | 4 |
| 1 | 88.. | John | Smith | 4 | 1 | 3 | 3 |
| 2 | 11.. | Mary | Doe | 1 | 1 | 1 | 4 |
| 2 | 11.. | Mary | Doe | 2 | 1 | 2 | 3 |
| 2 | 11.. | Mary | Doe | 3 | 4 | 3 | 4 |
| 2 | 11.. | Mary | Doe | 4 | 1 | 3 | 3 |
| 3 | 22.. | *null* | Chen | 1 | 1 | 1 | 4 |
| 3 | 22.. | *null* | Chen | 2 | 1 | 2 | 3 |
| 3 | 22.. | *null* | Chen | 3 | 4 | 3 | 4 |
| 3 | 22.. | *null* | Chen | 4 | 1 | 3 | 3 |

# Understanding WHERE as qualifying the tuples that satisfy the condition

| Students. id | pid | first_name | last_name | Enrollment. id | class | student | credits |
|---|---|---|---|---|---|---|---|
| 1 | 88.. | John | Smith | 1 | 1 | 1 | 4 |
| ~~1~~ | ~~88..~~ | ~~John~~ | ~~Smith~~ | ~~2~~ | ~~1~~ | ~~2~~ | ~~3~~ |
| ~~1~~ | ~~88..~~ | ~~John~~ | ~~Smith~~ | ~~3~~ | ~~4~~ | ~~3~~ | ~~4~~ |
| ~~1~~ | ~~88..~~ | ~~John~~ | ~~Smith~~ | ~~4~~ | ~~1~~ | ~~3~~ | ~~3~~ |
| ~~2~~ | ~~11..~~ | ~~Mary~~ | ~~Doe~~ | ~~1~~ | ~~1~~ | ~~1~~ | ~~4~~ |
| 2 | 11.. | Mary | Doe | 2 | 1 | 2 | 3 |
| ~~2~~ | ~~11..~~ | ~~Mary~~ | ~~Doe~~ | ~~3~~ | ~~4~~ | ~~3~~ | ~~4~~ |
| ~~2~~ | ~~11..~~ | ~~Mary~~ | ~~Doe~~ | ~~4~~ | ~~1~~ | ~~3~~ | ~~3~~ |
| ~~3~~ | ~~22..~~ | ~~*null*~~ | ~~Chen~~ | ~~1~~ | ~~1~~ | ~~1~~ | ~~4~~ |
| ~~3~~ | ~~22..~~ | ~~*null*~~ | ~~Chen~~ | ~~2~~ | ~~1~~ | ~~2~~ | ~~3~~ |
| ~~3~~ | ~~22..~~ | ~~*null*~~ | ~~Chen~~ | ~~3~~ | ~~4~~ | ~~3~~ | ~~4~~ |
| 3 | 22.. | *null* | Chen | 4 | 1 | 3 | 3 |

# Understanding SELECT as keeping the listed columns (highlighted below)

| Students. id | pid | first_name | last_name | Enrollment. id | class | student | credits |
|---|---|---|---|---|---|---|---|
| 1 | 88.. | John | Smith | 1 | 1 | 1 | 4 |
| ~~1~~ | ~~88..~~ | ~~John~~ | ~~Smith~~ | ~~2~~ | ~~1~~ | ~~2~~ | ~~3~~ |
| ~~1~~ | ~~88..~~ | ~~John~~ | ~~Smith~~ | ~~3~~ | ~~4~~ | ~~3~~ | ~~4~~ |
| ~~1~~ | ~~88..~~ | ~~John~~ | ~~Smith~~ | ~~4~~ | ~~1~~ | ~~3~~ | ~~3~~ |
| ~~2~~ | ~~11..~~ | ~~Mary~~ | ~~Doe~~ | ~~1~~ | ~~1~~ | ~~1~~ | ~~4~~ |
| 2 | 11.. | Mary | Doe | 2 | 1 | 2 | 3 |
| ~~2~~ | ~~11..~~ | ~~Mary~~ | ~~Doe~~ | ~~3~~ | ~~4~~ | ~~3~~ | ~~4~~ |
| ~~2~~ | ~~11..~~ | ~~Mary~~ | ~~Doe~~ | ~~4~~ | ~~1~~ | ~~3~~ | ~~3~~ |
| ~~3~~ | ~~22..~~ | ~~*null*~~ | ~~Chen~~ | ~~1~~ | ~~1~~ | ~~1~~ | ~~4~~ |
| ~~3~~ | ~~22..~~ | ~~*null*~~ | ~~Chen~~ | ~~2~~ | ~~1~~ | ~~2~~ | ~~3~~ |
| ~~3~~ | ~~22..~~ | ~~*null*~~ | ~~Chen~~ | ~~3~~ | ~~4~~ | ~~3~~ | ~~4~~ |
| 3 | 22.. | *null* | Chen | 4 | 1 | 3 | 3 |

Net result of the query is

| Students .pid | Students.first_name | Students.last_name | Enrollment.credits |
|---|---|---|---|
| 88.. | John | Smith | 4 |
| 11.. | Mary | Doe | 3 |
| 22.. | *null* | Chen | 3 |

## Generalize to any number of tables

Produce a table that shows the pid, first name and last name of every student enrolled in the CSE135 class along with the number of credit units in his/her 135 enrollment

HOW TO UNDERSTAND THE FROM AND WHERE
(AT LEAST UNTIL WE TALK ABOUT DUPLICATES):
Find the students,
whose students.id appears in an enrollment tuple
as enrollment.student,
and the enrollment.class of this tuple is the class.id
of a class tuple whose number is CSE135

```
SELECT  students.pid, students.first_name,
        students.last_name, enrollment.credits
FROM    students, enrollment, classes
WHERE   classes.number = 'CSE135'
        AND students.id = enrollment.student
        AND enrollment.class = classes.id ;
```

## You can omit table names in SELECT, WHERE when attribute is unambiguous

```
SELECT  pid, first_name, last_name, credits
FROM    students, enrollment, classes
WHERE   number = 'CSE135'
        AND students.id = student
        AND class = classes.id ;
```

# Take Two on the previous exercises: The algebraic way to express joins

Produce a table that shows the pid, first name and last name of every student enrolled in the class with ID 1, along with the number of credit units in the "class 1" enrollment

```
SELECT  students.pid, students.first_name,
        students.last_name, enrollment.credits
FROM    students JOIN enrollment
                ON students.id = enrollment.student
WHERE enrollment.class = 1 ;
```

# Take two cont'd

FROM clause result

← Student part of the tuple → ← Enrollment part of the tuple →

| Students.id | pid | first_name | last_name | Enrollment.id | class | student | credits |
|---|---|---|---|---|---|---|---|
| 1 | 88.. | John | Smith | 1 | 1 | 1 | 4 |
| 2 | 11.. | Mary | Doe | 2 | 1 | 2 | 3 |
| 3 | 22.. | *null* | Chen | 3 | 4 | 3 | 4 |
| 3 | 22.. | *null* | Chen | 4 | 1 | 3 | 3 |

SELECT clause result

| Students.id | pid | first_name | last_name | Enrollment.id | class | student | credits |
|---|---|---|---|---|---|---|---|
| 1 | 88.. | John | Smith | 1 | 1 | 1 | 4 |
| 2 | 11.. | Mary | Doe | 2 | 1 | 2 | 3 |
| ~~3~~ | ~~22..~~ | ~~*null*~~ | ~~Chen~~ | ~~3~~ | ~~4~~ | ~~3~~ | ~~4~~ |
| 3 | 22.. | *null* | Chen | 4 | 1 | 3 | 3 |

Net result of the query is

| Students.pid | Students.first_name | Students.last_name | Enrollment.credits |
|---|---|---|---|
| 88.. | John | Smith | 4 |
| 11.. | Mary | Doe | 3 |
| 22.. | *null* | Chen | 3 |

## Take two (algebraic approach) Second example

Produce a table that shows the pid, first name and last name of every student enrolled in the CSE135 class along with the number of credit units in his/her 135 enrollment

Take One:

```
SELECT  students.pid, students.first_name,
        students.last_name, enrollment.credits
FROM    students, enrollment, classes
WHERE   classes.number = 'CSE135'
     AND students.id = enrollment.student
     AND enrollment.class = classes.id
```

Take Two:

```
SELECT  students.pid, students.first_name,
        students.last_name, enrollment.credits
FROM    (students JOIN enrollment ON student.id = enrollment.student)
        JOIN classes ON enrollment.class = class.id
WHERE   classes.number = 'CSE135'
```

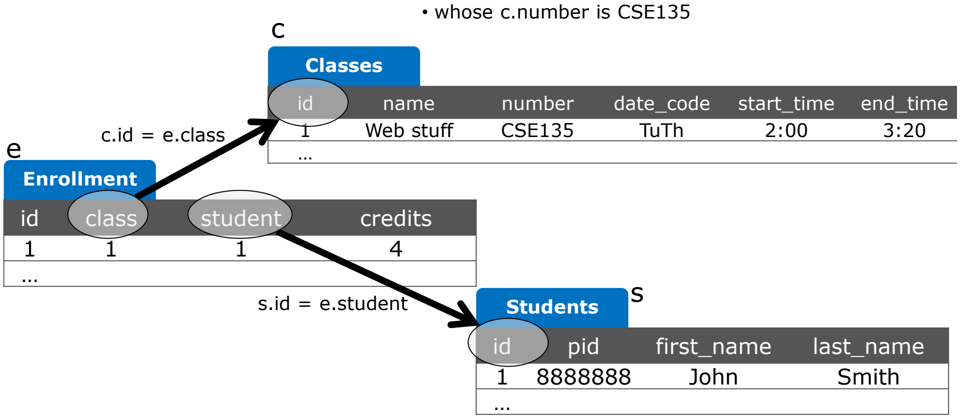
## Heuristics on writing queries

- Have you reached the point where you understand how queries work but have difficulty writing queries yourself?
- The following heuristics will help you translate a requirement expressed in English into a query
  - The key point is to translate informal English into a precise English statement about what tuples you expect to find in the database

## Hints for writing FROM/WHERE: Rephrase the statement, see it as a navigation across primary/foreign keys

Produce a table that shows the pid, first name and last name of every student enrolled in the CSE135 class along with the number of credit units in his/her 135 enrollment

- Find any students tuple s,
- that is connected to an enrollment tuple e
  - i.e., whose s.id appears in an enrollment tuple e as e.student,
- and e is connected to a classes tuple c
  - i.e., the e.class of e appears as c.id of the tuple c,
- whose c.number is CSE135



Classes

| id | name | number | date_code | start_time | end_time |
|---|---|---|---|---|---|
| 1 | Web stuff | CSE135 | TuTh | 2:00 | 3:20 |
| … | | | | | |

Enrollment

| id | class | student | credits |
|---|---|---|---|
| 1 | 1 | 1 | 4 |
| … | | | |

Students

| id | pid | first_name | last_name |
|---|---|---|---|
| 1 | 8888888 | John | Smith |
| … | | | |

- Find any students tuple s,
- that is connected to an enrollment tuple e
  - i.e., whose s.id appears in an enrollment tuple e as e.student,
- and e is connected to a classes tuple c
  - i.e., the e.class of e appears as c.id of the tuple c,
- whose c.number is CSE135

```
FROM students AS s
```

- Find any students tuple s,
- that is connected to an enrollment tuple e
  - i.e., whose s.id appears in an enrollment tuple e as e.student,
- and e is connected to a classes tuple c
  - i.e., the e.class of e appears as c.id of the tuple c,
- whose c.number is CSE135

*Take One: Declarative*

```
FROM students AS s,
        enrollment AS e
WHERE s.id = e.student
```

*Take Two: Algebraic*

```
FROM students AS s
        JOIN enrollment AS e
        ON s.id = e.student
```

- Find any students tuple s,
- that is connected to an enrollment tuple e
  - i.e., whose s.id appears in an enrollment tuple e as e.student,
- and e is connected to a classes tuple c
  - i.e., the e.class of e appears as c.id of the tuple c,
- whose c.number is CSE135

*Take One: Declarative*

```
FROM students AS s,
          enrollment AS e,
          classes AS c
WHERE s.id = e.student
          AND c.id = e.class
```

*Take Two: Algebraic*

```
FROM ( students AS s
          JOIN enrollment AS e
          ON s.id = e.student )
          JOIN classes AS c
          ON c.id = e.class
```

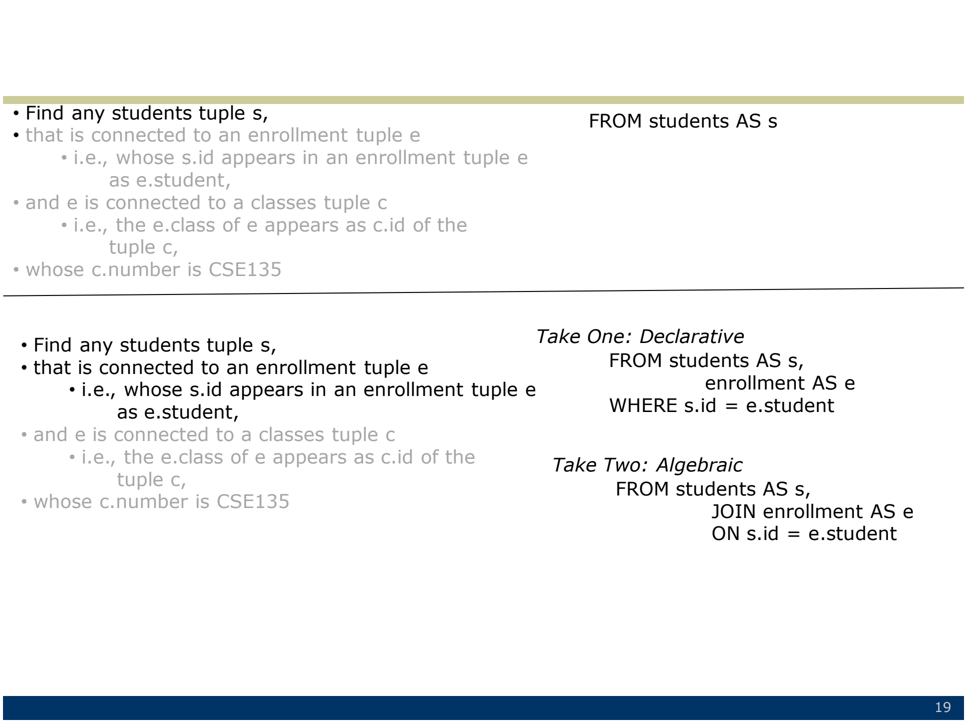
- Find any students tuple s,
- that is connected to an enrollment tuple e
  - i.e., whose s.id appears in an enrollment tuple e as e.student,
- and e is connected to a classes tuple c
  - i.e., the e.class of e appears as c.id of the tuple c,
- whose c.number is CSE135

*Take One: Declarative*

```
FROM students AS s,
          enrollment AS e,
          classes AS c
WHERE s.id = e.student
          AND c.id = e.class
          AND c.number = ‘CSE135’
```

*Take Two: Algebraic*

```
FROM ( students AS s
          JOIN enrollment AS e
          ON s.id = e.student )
          JOIN classes AS c
          ON c.id = e.class
WHERE c.number = ‘CSE135’
```



- Find any students tuple s,
- that is connected to an enrollment tuple e
    - i.e., whose s.id appears in an enrollment tuple e as e.student,
- and e is connected to a classes tuple c
    - i.e., the e.class of e appears as c.id of the tuple c,
- whose c.number is CSE135

```
FROM students AS s
```

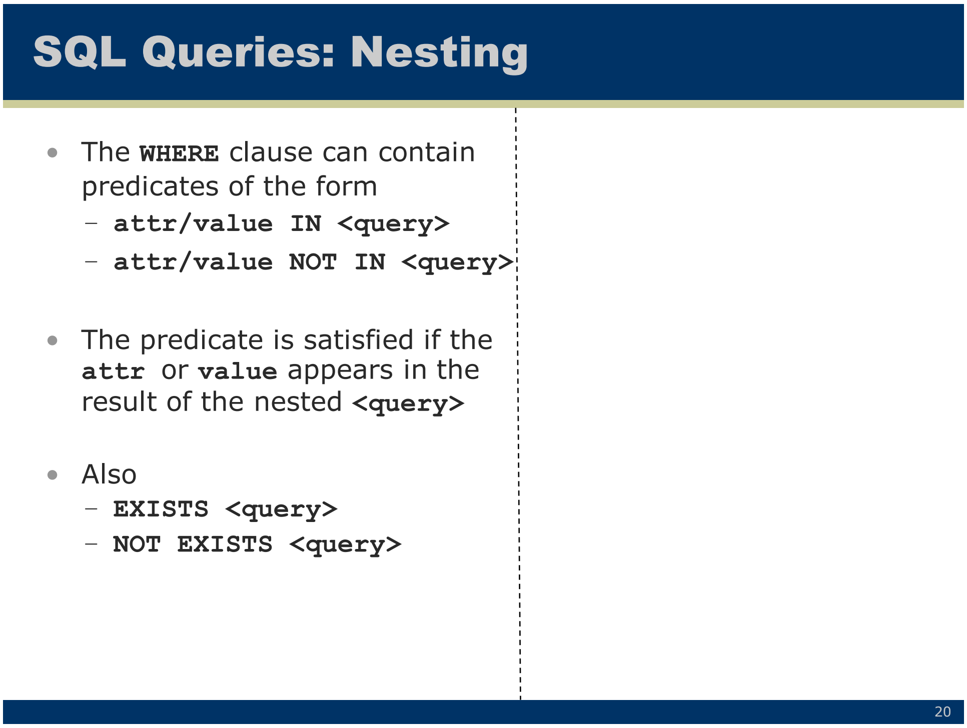
- Find any students tuple s,
- that is connected to an enrollment tuple e
    - i.e., whose s.id appears in an enrollment tuple e as e.student,
- and e is connected to a classes tuple c
    - i.e., the e.class of e appears as c.id of the tuple c,
- whose c.number is CSE135

*Take One: Declarative*

```
FROM students AS s,
              enrollment AS e
WHERE s.id = e.student
```

*Take Two: Algebraic*

```
FROM students AS s,
              JOIN enrollment AS e
              ON s.id = e.student
```



## SQL Queries: Nesting

- The **WHERE** clause can contain predicates of the form
    - **attr/value IN <query>**
    - **attr/value NOT IN <query>**
- The predicate is satisfied if the **attr** or **value** appears in the result of the nested **<query>**
- Also
    - **EXISTS <query>**
    - **NOT EXISTS <query>**

## Nested subquery example (uncorrelated subquery)

Produce a table that shows the pid, first and last name of every student enrolled in CSE135

```
SELECT pid, first_name, last_name
FROM students
WHERE id IN
  ( SELECT student
    FROM enrollment, classes
    WHERE number='CSE135'
       AND class=classes.id
  )
```

"Uncorrelated" in the sense that the nested query could be a standalone query

Nested queries help modularize the task:
Nested query finds the id's of the students who take CSE135.
Then the outer query prints out pid and name for every student whose id appears in the result of the nested query

## Nested subquery example, correlated

```
SELECT pid, first_name, last_name
FROM students
WHERE EXISTS
  ( SELECT *
    FROM enrollment, classes
    WHERE number='CSE135'
       AND class=classes.id
       AND student = students.id
  )
```

Correlation of nested query to outside query. The nested query is not a standalone.

There may be IN queries that are correlated and EXISTS queries that are uncorrelated.

## SQL Queries, advanced: Aliases

- Use the same relation more than once in the **FROM** clause
- Tuple variables
- **Problem:** Find the other classes taken by students who take CSE135
  - First, also showing the students, i.e., produce a table where each row has the name of a 135 student and the name of another class he/she takes

produce a table where each row has the name of a 135 student and the name of another class he/she takes

```
SELECT  c_others.name, first_name, last_name
FROM    classes AS c_135, enrollment AS e_135,
        students,
        enrollment AS e_others, classes AS c_others
WHERE   c_135.number = 'CSE135'
      AND c_135.id = e_135.class
      AND e_135.student = students.id
      AND students.id = e_others.student
      AND e_others.class = c_others.id
      AND NOT (c_others.number = 'CSE135')
```

## Second, show just the other classes. Notice use of DISTINCT

```
SELECT  DISTINCT c_others.name
FROM    classes AS c_135, enrollment AS e_135,
        enrollment AS e_others, classes AS c_others
WHERE   c_135.number = 'CSE135'
        AND c_135.id = e_135.class
        AND e_135.student = e_others.student
        AND e_others.class = c_others.id
        AND NOT (c_others.number = 'CSE135')
```

## Use of nested subqueries may reduce need for aliases => easier to write, read

Find the CSE135 students who take a Friday 11:00 am class

```
SELECT  first_name, last_name
FROM    students, enrollment, classes
WHERE   students.id = student
        AND class = classes.id
        AND number = 'CSE135'
        AND students.id IN
        (
            SELECT  student
            FROM    enrollment, classes
            WHERE   classes.id = class
            AND     date_code = 'F'
            AND     start_time = '11:00'
        )
```

Nested query computes the id's of students enrolled in Friday 11:00AM classes

# SQL Queries: Aggregation & Grouping

- Aggregate functions: **SUM**,**AVG**, **COUNT**, **MIN**, **MAX**, and recently user defined functions as well
- **GROUP BY**

**Employee**

| Name | Dept | Salary |
|---|---|---|
| Joe | Toys | 45 |
| Nick | PCs | 50 |
| Jim | Toys | 35 |
| Jack | PCs | 40 |

**Example**: Find the average salary of all employees:

```
SELECT AVG(Salary) AS AvgSal
FROM Employee
```

| AvgSal |
|---|
| 42.5 |

**Example**: Find the average salary for each department:

```
SELECT Dept, AVG(Salary) AS AvgSal
FROM Employee
GROUP BY Dept
```

| Dept | AvgSal |
|---|---|
| Toys | 40 |
| PCs | 45 |

# SQL Grouping: Conditions that Apply on Groups

- **HAVING <condition>** may follow a **GROUP BY**clause
- If so, the condition applies to each group, and groups not satisfying the condition are eliminated

- **Example**: Find the average salary in each department that has more than 1 employee:

```
SELECT Dept,AVG(Salary) AS AvgSal
FROM Employee
GROUP BY Dept
HAVING COUNT(Name) >1
```

# Let’s mix features we’ve seen: Aggregation after joining tables

- **Problem:** List all enrolled students and the number of total credits for which they have registered

```
SELECT   students.id, first_name, last_name, SUM(credits)
FROM     students, enrollment
WHERE    students.id = enrollment.student
GROUP BY students.id, first_name, last_name
```

# The outerjoin operator

- New construct in FROM clause

**R**

| RJ | RV |
|---|---|
| 1 | RV1 |
| 2 | RV2 |

**S**

| SJ | SV |
|---|---|
| 1 | SV1 |
| 3 | SV3 |

- R LEFT OUTER JOIN S ON R.<attr of R>=S.<attr of J>

```
SELECT *
FROM R LEFT OUTERJOIN S ON R.RJ=S.SJ
```

| RJ | RV | SJ | SV |
|---|---|---|---|
| 1 | RV1 | 1 | SV1 |
| 2 | RV2 | Null | Null |

- R FULL OUTER JOIN S ON R.<attr of R>=S.<attr of J>

```
SELECT *
FROM R FULL OUTERJOIN S ON R.RJ=S.SJ
```

| RJ | RV | SJ | SV |
|---|---|---|---|
| 1 | RV1 | 1 | SV1 |
| 2 | RV2 | Null | Null |
| Null | Null | 3 | SV3 |

## An application of outerjoin

- **Problem:** List all students and the number of total credits for which they have registered
  - Notice that you must also list non-enrolled students

```
SELECT   students.id, first_name, last_name, SUM(credits)
FROM     students LEFT OUTER JOIN enrollment ON
   students.id = enrollment.student
GROUP BY students.id, first_name, last_name
```

## SQL: More Bells and Whistles …

- Pattern matching conditions
  - `<attr> LIKE <pattern>`

Retrieve all students whose name contains “Sm”

```
SELECT *
FROM Students
WHERE name LIKE ‘%Sm%’
```

## …and a Few “Dirty” Points

- **Null values**
  - All comparisons involving NULL are **false** by definition
  - All aggregation operations, except `COUNT(*)`, ignore NULL values

## Null Values and Aggregates

- Example:

R

| a | b |
|---|---|
| x | 1 |
| x | 2 |
| x | null |
| null | null |
| null | null |

```
SELECT COUNT(a),COUNT(b),AVG(b), COUNT(*)
FROM R
GROUP BY a
```

| count(a) | count(b) | avg(b) | count(*) |
|---|---|---|---|
| 3 | 2 | 1.5 | 3 |
| 0 | 0 | null | 2 |

## Universal Quantification by Negation (difficult)

Problem:

- Find the students that take **every** class ‘John Smith’ takes

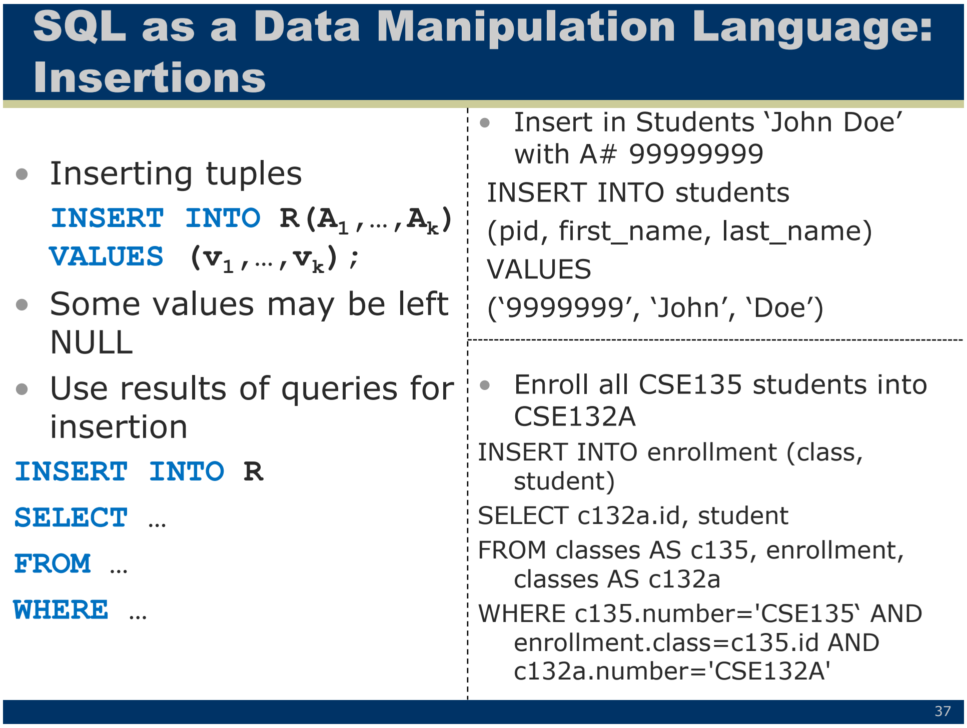
Rephrase:

- Find the students such that there is no class that ‘John Smith’ takes and they do not take

## Discussed in class and discussion section

How to solve in easy steps the following complex query:

Create a table that shows all time slots (date, start time, end time)
when students of CSE135 attend a lecture of another class;
Show also how many students attend a class at each time slot.



# SQL as a Data Manipulation Language: Insertions

- Inserting tuples

  **INSERT INTO** R($A_1$,…,$A_k$)
  **VALUES** ($v_1$,…,$v_k$);
- Some values may be left NULL
- Use results of queries for insertion

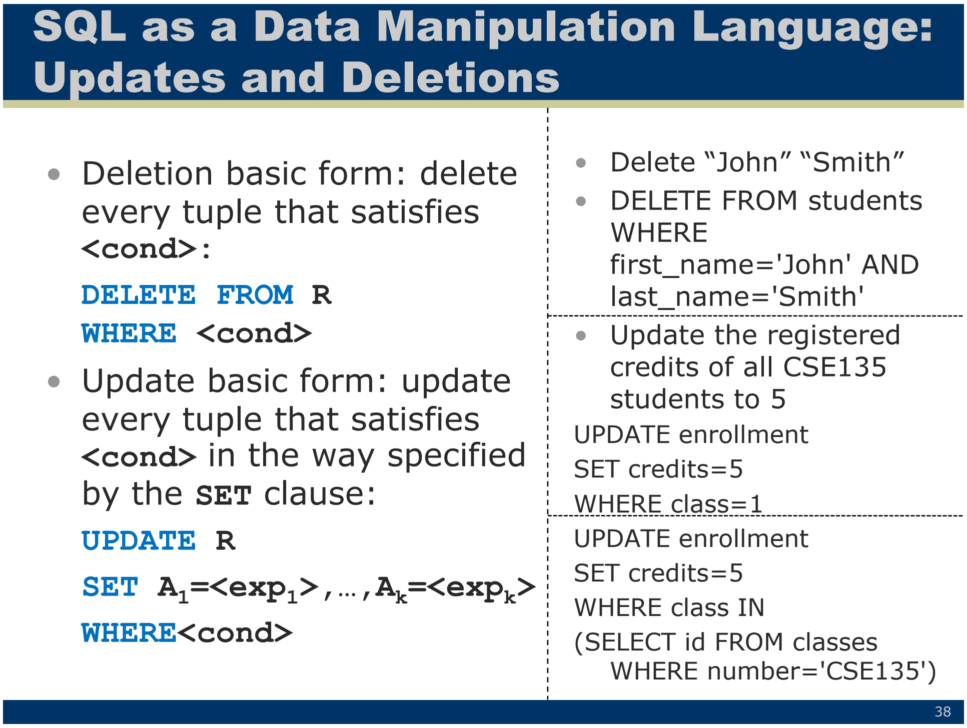
**INSERT INTO** R
**SELECT** …
**FROM** …
**WHERE** …

- Insert in Students 'John Doe' with A# 99999999

INSERT INTO students
(pid, first_name, last_name)
VALUES
('9999999', 'John', 'Doe')

- Enroll all CSE135 students into CSE132A

INSERT INTO enrollment (class, student)
SELECT c132a.id, student
FROM classes AS c135, enrollment, classes AS c132a
WHERE c135.number='CSE135` AND enrollment.class=c135.id AND c132a.number='CSE132A'



# SQL as a Data Manipulation Language: Updates and Deletions

- Deletion basic form: delete every tuple that satisfies **<cond>**:

  **DELETE FROM** R
  **WHERE** <cond>
- Update basic form: update every tuple that satisfies **<cond>** in the way specified by the **SET** clause:

  **UPDATE** R
  **SET** $A_1$=<$exp_1$>,…,$A_k$=<$exp_k$>
  **WHERE**<cond>

- Delete "John" "Smith"
- DELETE FROM students WHERE first_name='John' AND last_name='Smith'

- Update the registered credits of all CSE135 students to 5

UPDATE enrollment
SET credits=5
WHERE class=1

UPDATE enrollment
SET credits=5
WHERE class IN
(SELECT id FROM classes
WHERE number='CSE135')