CMPT 354: Database System I

Lecture 3. SQL Basics

Announcements!

- About Piazza
  - 115 enrolled (as of today)
  - Posts are anonymous to classmates
- You should have started doing A1
  - Please come to office hours if you need any help

SQL Motivation

- Dark times in 2000s
  - Are relational databases dead?

- Now, as before: everyone sells SQL
  - Pig, Hive, Impala
  - SparkSQL

- NoSQL
  - “Non SQL”
  - “Not-Only-SQL”
  - “Not-Yet-SQL”

SQL: Introduction

- “S.Q.L.” or “sequel”
- Supported by all major commercial database systems
- Standardized – many new features over time
- Declarative language

SQL is a…

- Data Definition Language (DDL)
  - Define relational *schema*
  - Create/alter/delete tables and their attributes

- Data Manipulation Language (DML)
  - Insert/delete/modify tuples in tables
  - Query one or more tables – discussed next!

Outline

- **Single-table Queries**
  - The SFW query
  - Useful operators: DISTINCT, ORDER BY, LIKE
  - Handle missing values: NULLs

- **Multiple-table Queries**
  - Foreign key constraints
  - Joins: basics
  - Joins: SQL semantics

The SFW Query

```
SELECT <columns>
FROM   <table name>
WHERE  <conditions>
```

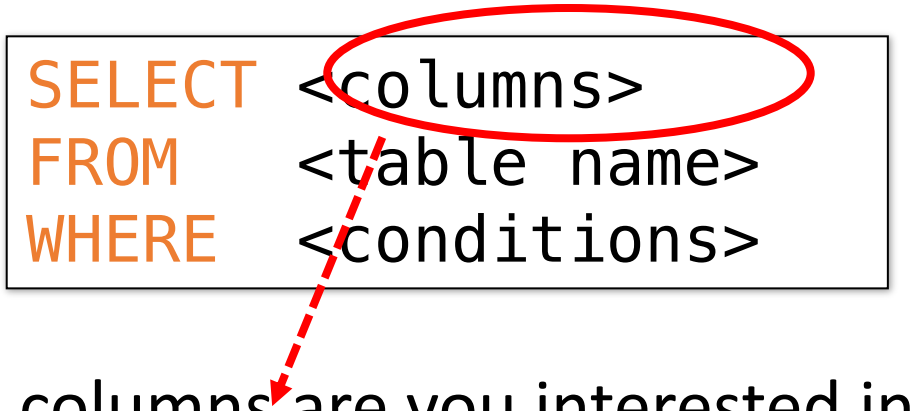
- To write the query, ask yourself three questions:
  - Which **table** are you interested in?
  - Which **rows** are you interested in?
  - Which **columns** are you interested in?

Conditions

```
SELECT <columns>
FROM   <table_name>
WHERE  <conditions>
```

- Which rows are you interested in?
  - WHERE gpa > 3.5
  - WHERE school = ‘SFU’ AND gpa > 3.5
  - WHERE (school = ‘SFU’ OR school = ‘UBC’) AND gpa > 3.5
  - WHERE age * 365 > 7500

Columns



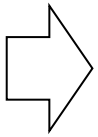
- Which columns are you interested in?
  - SELECT *
  - SELECT name, age
  - SELECT name as studentName, age
  - SELECT name, age * 365 as ageDay

A Few Details

- SQL **commands** are case insensitive:
  - Same: SELECT, Select, select
  - Same: Student, student
  - Same: gpa, GPA
- **Values** are **not:**
  - Different: 'SFU', 'sfu'
- SQL strings are enclosed in **single quotes**
  - e.g. name = 'Mike’
  - Single quotes in a string can be specified using an initial single quote character as an escape
    - author = 'Shaq O''Neal'
- Strings can be compared **alphabetically** with the comparison operators
  - e.g. 'fodder' < 'foo' is TRUE

DISTINCT: Eliminating Duplicates

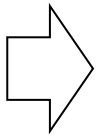
```
SELECT School
FROM   Students
```



School
SFU
SFU
UBC
UT
UT

Versus

```
SELECT DISTINCT School
FROM   Students
```



School
SFU
UBC
UT

ORDER BY: Sorting the Results

```
SELECT    name, gpa, age
FROM      Students
WHERE     school = 'SFU'
ORDER BY  gpa DESC, age ASC
```

- The output of an SQL query can be ordered
  - By any number of attributes, and
  - In either ascending or descending order
- The default is to use ascending order, the keywords ***ASC*** and ***DESC***, following the column name, sets the order

LIKE: Simple String Pattern Matching

```
SELECT *
FROM   Students
WHERE  name LIKE 'Sm_t%'
```

SQL provides pattern matching support with the **LIKE** operator and two symbols

- The **%** symbol stands for zero or more arbitrary characters
- The _ symbol stands for exactly one arbitrary character
- The **%** and _ characters can be escaped with ****
  - E.g., name **LIKE** 'Michael_Jordan'

Exercise - 1

- Which names will be returned?

```
SELECT *
FROM   Students
WHERE  name LIKE 'Sm_t%'
```

1. Smit
2. SMIT
3. Smart
4. Smith
5. Smythe
6. Smut
7. Smeath
8. Smt

Exercise - 1

- Which names will be returned?

```
SELECT *
FROM   Students
WHERE  name LIKE 'Sm_t%'
```

1. Smit
2. SMIT
3. Smart
4. Smith
5. Smythesfddfesd
6. Smut
7. Smeath
8. Smt

1, 4, 5, 6

NULLS in SQL

- Whenever we don't have a value, we can put a NULL
- Can mean many things:
  - Value does not exists
  - Value exists but is unknown
  - Value not applicable
  - Etc.
- NULL constraints

```
CREATE TABLE Students (
        name CHAR(20) NOT NULL,
        age CHAR(20) NOT NULL,
        gpa FLOAT
)
```

What will happen?

name	age	gpa
Mike	20	4.0
Joe	18	NULL
Alice	21	3.8

1. SELECT gpa*100 FROM students
2. SELECT name FROM students WHERE gpa > 3.5
3. SELECT name FROM students WHERE age > 15 OR gpa > 3.5

Two Important Rules

- Arithmetic operations (+, -, *, /) on nulls return **NULL**
  - NULL * 100
    - NULL
  - NULL * 0
    - NULL

1. SELECT gpa*100 FROM students

2. SELECT gpa*0 FROM students

- Comparisons with nulls evaluate to **UNKNOWN**
  - NULL > 3.5
    - UNKNOWN
  - NULL = NULL
    - UNKNOWN

3. SELECT name FROM students WHERE gpa > 3.5

4. SELECT name FROM students WHERE gpa = NULL

Combinations of true, false, unknown

- Truth values for *unknown* results
  - *true* **OR** *unknown* = *true*,
  - *false* **OR** *unknown* = *unknown,*
  - *unknown* **OR** *unknown* = *unknown* ,
  - *true* **AND** *unknown* = *unknown*,
  - *false* **AND** *unknown* = *false*,
  - *unknown* **AND** *unknown* = *unknown*

```
SELECT * FROM students WHERE
age > 15 OR gpa > 3.5
```

```
SELECT * FROM students WHERE
age > 15 AND gpa > 3.5
```

- The result of a **WHERE** clause is treated as *false* if it evaluates to *unknown*
  - *WHERE unknown* → *false*

What will happen?

name	age	gpa
Mike	20	4.0
Joe	18	NULL
Alice	21	3.8

1. SELECT gpa*100 FROM students
2. SELECT name FROM students WHERE gpa > 3.5
3. SELECT name FROM students WHERE gpa > 3.5

gpa
400
NULL
380

name
Mike
Alice

name
Mike
Joe
Alice

Exercise - 2

- Will it return all students?

```
SELECT *
FROM   Students
WHERE  age < 25 OR age >= 25
```

Exercise - 2

- Will it return all students?

```
SELECT *
FROM   Students
WHERE  age < 25 OR age >= 25
   OR age is NULL
```

There are special operators to test for null values

- **IS NULL** tests for the presence of nulls and
- **IS NOT NULL** tests for the absence of nulls

Outline

- Single-table Queries
  - The SFW query
  - Other useful operators: DISTINCT, LIKE, ORDER BY
  - NULLs
- **Multiple-table Queries**
  - **Foreign key constraints**
  - **Joins: basics**
  - **Joins: SQL semantics**

Foreign Key constraints

- Foreign-key constraint:
  - **student_id** references **sid**

Students

sid	name	gpa
101	Bob	3.2
123	Mary	3.8

Enrolled

student_id	cid	grade
123	354	A
123	454	A+
156	354	A

Foreign Key constraints

- Foreign-key constraint:
  - **student_id** references **sid**

Students

sid	name	gpa
101	Bob	3.2
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A
123	454	A+
156	354	A

Declaring Foreign Keys

student_id	cid	grade
123	354	A
123	454	A+
156	354	A

```
CREATE TABLE Enrolled(
      student_id CHAR(20),
      cid        CHAR(20),
      grade      CHAR(10),
      PRIMARY KEY (student_id, cid),
      FOREIGN KEY (student_id) REFERENCES Students(sid)
)
```

Insert operations

- What if we insert a tuple into Enrolled, but no corresponding student?
  - INSERT is rejected

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A
123	454	A+
156	354	A
190	354	A

Delete operations

- What if we delete a student, who has enrolled courses?
  - Disallow the delete *(ON DELETE RESTRICT)*

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A
123	454	A+
156	354	A

ON DELETE RESTRICT

student_id	cid	grade
123	354	A
123	454	A+
156	354	A

```
CREATE TABLE Enrolled(
      student_id CHAR(20),
      cid            CHAR(20),
      grade  CHAR(10),
      PRIMARY KEY (student_id, cid),
      FOREIGN KEY (student_id) REFERENCES Students(sid)
      ON DELETE RESTRICT
)
```

Delete operations

- What if we delete a student, who has enrolled courses?
  - Remove all of the courses for that student *(ON DELETE CASCADE)*

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A
123	454	A+
156	354	A

ON DELETE CASCADE

student_id	cid	grade
123	354	A
123	454	A+
156	354	A

```
CREATE TABLE Enrolled(
      student_id CHAR(20),
      cid            CHAR(20),
      grade  CHAR(10),
      PRIMARY KEY (student_id, cid),
      FOREIGN KEY (student_id) REFERENCES Students(sid)
      ON DELETE CASCADE
)
```

Delete operations

- What if we delete a student, who has enrolled courses?
  - *Set Foreign Key to NULL (ON DELETE SET NULL)*

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A
123	454	A+
NULL	354	A

Interestingly, although it satisfies the foreign-key constraint, it violates the primary-key constraint, thus the deletion operation is disallowed.

ON DELETE SET NULL

student_id	cid	grade
123	354	A
123	454	A+
156	354	A

```
CREATE TABLE Enrolled(
      student_id CHAR(20),
      cid            CHAR(20),
      grade  CHAR(10),
      PRIMARY KEY (student_id, cid),
      FOREIGN KEY (student_id) REFERENCES Students(sid)
      ON DELETE SET NULL
)
```

Outline

- Single-table Queries
  - The SFW query
  - Other useful operators: DISTINCT, LIKE, ORDER BY
  - NULLs
- Multiple-table Queries
  - Foreign key constraints
  - **Joins: basics**
  - Joins: SQL semantics
  - Set Operators

Why do we have multiple tables?

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A
123	454	A+
156	354	A

VS.

EnrolledStudents

student_id	name	gpa	cid	grade
123	Mary	3.8	354	A
123	Mary	3.8	454	A+
156	Mike	3.7	354	A

Store data into multiple tables vs. single table

- Multiple tables
  - Data updating is easier (e.g., update Mary's gpa to 3.9)
  - Querying each individual table is faster (e.g., retrieve Mary's gpa)

- A single table
  - Data exchange is easier (e.g., share your data with others)
  - Avoid the cost of joining multiple tables (e.g., retrieval all the courses that Mary has taken)

Joins

The SFW query over a single table

```
SELECT <columns>
FROM   <table name>
WHERE  <conditions>
```

Which rows are you interested in?

The SFW query over multiple tables

```
SELECT <columns>
FROM   <table names>
WHERE  <conditions>
```

Which rows are you interested in?

How to join the multiple tables?

Joins: Example

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A+
123	454	A+
156	354	A

Find all student who have got an A+ in 354; return their names and gpas

```
SELECT name, gpa
FROM   Students, Enrolled
WHERE  sid = student_id AND
       cid = 354 AND grad = 'A+'
```

How to join the two tables?

Which rows are you interested in?

Other ways to write joins

```
SELECT name
FROM   Students, Enrolled
WHERE  sid = student_id AND
       cid = 354 AND grad = 'A+'
```

```
SELECT name
FROM   Students
JOIN   Enrolled ON sid = student_id
WHERE cid = 354 AND grad = 'A+'
```

```
SELECT name
FROM   Students
JOIN   Enrolled ON sid = student_id
       AND cid = 354 AND grad = 'A+'
```

The Need for Tuple Variable

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	name	grade
123	354	DB I	A+
123	454	DB II	A+
156	354	DB I	A

```
SELECT name
FROM   Students, Enrolled
WHERE  sid = student_id
```

Tuple Variable

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	name	grade
123	354	DB I	A+
123	454	DB II	A+
156	354	DB I	A

```
SELECT Students.name
FROM   Students, Enrolled
WHERE  sid = student_id
```

```
SELECT S.name
FROM   Students S, Enrolled
WHERE  sid = student_id
```

Outline

- Single-table Queries
  - The SFW query
  - Other useful operators: DISTINCT, LIKE, ORDER BY
  - NULLs
- Multiple-table Queries
  - Foreign key constraints
  - Joins: basics
  - **Joins: SQL semantics**
  - Set Operators

Meaning (Semantics) of Join Queries

```
SELECT x1.a1, x1.a2, …, xn.ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions(x1,…, xn)
```

```
Answer = {}
for x1 in R1 do
    for x2 in R2 do
        …..
            for xn in Rn do
                if Conditions(x1,…, xn)
                    then Answer = Answer ∪ {(x1.a1, x1.a2, …, xn.ak)}
return Answer
```

This is called **nested loop semantics** since we are interpreting what a join means using a nested loop

Note: this is a *multiset* union

Three steps

```
SELECT x1.a1, x1.a2, …, xn.ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions(x1,…, xn)
```

1. **Take cross product**
   - $R_1 \times R_2 \times \ldots \times R_n$
2. **Apply conditions**
   - `Conditions(x1,…, xn)`
3. **Apply projections**
   - `x1.a1, x1.a2, …, xn.ak`

Note: This is NOT how the DBMS executes the query.

Exercise

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A+
123	454	A+
156	354	A

```
SELECT name
FROM   Students, Enrolled
WHERE  sid = student_id AND grade >= 'A'
```

Which one(s) are correct?

name
Mary

(A)

name
Mary
Mike

(B)

name
Mary
Mike
Mary

(C)

name
Mary
Mary
Mike

(D)

Outline

- Single-table Queries
  - The SFW query
  - Other useful operators: DISTINCT, LIKE, ORDER BY
  - NULLs
- Multiple-table Queries
  - Foreign key constraints
  - Joins: basics
  - Joins: SQL semantics
  - **Set Operators**

Set Operations

- SQL supports union, intersection and set difference operations
  - Called **UNION**, **INTERSECT**, and **EXCEPT**
  - These operations must be performed on *union compatible* tables
- Although these operations are supported in the SQL standard, implementations may vary
  - **EXCEPT** may not be implemented
    - When it is, it is sometimes called **MINUS**

One of Two Courses

- Find all students who have taken either 354 or 454

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A+
123	454	A+
156	354	A

```
SELECT name
FROM   Students, Enrolled
WHERE  sid = student_id AND (cid = 354 OR cid = 454)
```

One of Two Courses - UNION

- Find all students who have taken either 354 or 454

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A+
123	454	A+
156	354	A

```
SELECT name
FROM   Students, Enrolled
WHERE  sid = student_id AND cid = 354
UNION
SELECT name
FROM   Students, Enrolled
WHERE  sid = student_id AND cid = 454
```

Both Courses

- Find all students who have taken both 354 and 454

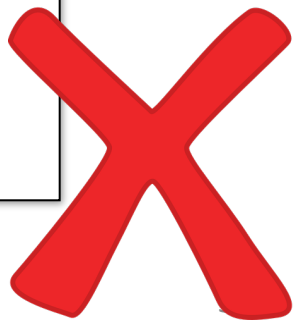
Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A+
123	454	A+
156	354	A

```
SELECT name
FROM   Students S, Enrolled E
WHERE  sid = student_id AND
       (E.cid = 354 AND E.cid = 454)
```



Both Courses Again

- Find all students who have taken both 354 and 454

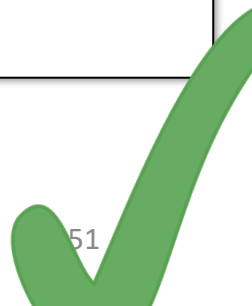
Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A+
123	454	A+
156	354	A

```
SELECT name
FROM   Students S, Enrolled E1, Enrolled E2
WHERE  S.sid = E1.student_id AND S.sid = E2.student_id
       AND (E1.cid = 354 AND E2.cid = 454)
```



Both Courses - **INTERSECT**

- Find all students who have taken both 354 and 454

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A+
123	454	A+
156	354	A

```
SELECT name
FROM   Students, Enrolled
WHERE  sid = student_id AND cid = 354
INTERSECT
SELECT name
FROM   Students, Enrolled
WHERE  sid = student_id AND cid = 454
```

One Course But Not The Other

- Find all students who have taken 354 but not 454

Students

sid	name	gpa
123	Mary	3.8
156	Mike	3.7

Enrolled

student_id	cid	grade
123	354	A+
123	454	A+
156	354	A

```
SELECT name
FROM   Students, Enrolled
WHERE  sid = student_id AND cid = 354
EXCEPT
SELECT name
FROM   Students, Enrolled
WHERE  sid = student_id AND cid = 454
```

Set Operations and Duplicates

- Unlike other SQL operations, **UNION**, **INTERSECT**, and **EXCEPT** queries eliminate duplicates by default
- SQL allows duplicates to be *retained* in these three operations using the **ALL** keyword (i.e., multi-set operations)

```
SELECT name
FROM   Students, Enrolled
WHERE  sid = student_id AND cid = 354
INTERSECT ALL
SELECT name
FROM   Students, Enrolled
WHERE  sid = student_id AND cid = 454
```

Acknowledge

- Some lecture slides were copied from or inspired by the following course materials
  - “W4111: Introduction to databases” by Eugene Wu at Columbia University
  - “CSE344: Introduction to Data Management” by Dan Suciu at University of Washington
  - “CMPT354: Database System I” by John Edgar at Simon Fraser University
  - “CS186: Introduction to Database Systems” by Joe Hellerstein at UC Berkeley
  - “CS145: Introduction to Databases” by Peter Bailis at Stanford
  - “CS 348: Introduction to Database Management” by Grant Weddell at University of Waterloo