Georgia Tech

Lecture 3: Advanced SQL

CREATING THE NEXT®

Today's Agenda

- Recap
- String and Date/Time Functions
- Output Control
- Nested Queries
- Window Functions
- Common Table Expressions
- Joins

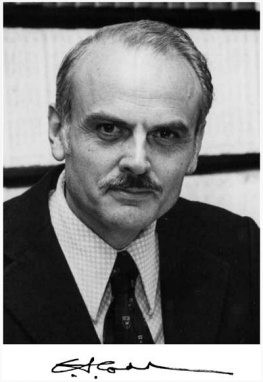
Administrivia

- Office hours
- Get started with the first assignment
- BuzzDB snippets
- Visual Code setup, ZSH shell
- SQLFiddle

Recap

Relational Model

Proposed in 1970 by Ted Codd (IBM Almaden).
Data model to avoid this maintenance.



- Store database in simple data structures
- Access data through high-level language
- Physical storage left up to implementation

Core Operators

- These operators take in **relations** (*i.e.*, tables) as input and return a relation as output.
- We can "chain" operators together to create more complex operations.

- Selection (σ)
- Projection (Π)
- Union ($\cup$)
- Intersection ($\cap$)
- Difference ($-$)
- Product ($\times$)
- Join ($\bowtie$)

List of SQL Features

- Aggregations + Group By
- String / Date / Time Operations
- Output Control + Redirection
- Nested Queries
- Join
- Common Table Expressions
- Window Functions

String and Date/Time Functions

String Operations

	String Case	String Quotes
SQL-92	Sensitive	Single Only
Postgres	Sensitive	Single Only
MySQL	Insensitive	Single/Double
SQLite	Sensitive	Single/Double
DB2	Sensitive	Single Only
Oracle	Sensitive	Single Only

```
WHERE UPPER(name) = UPPER('MaRiA') // SQL-92
WHERE name = 'MaRiA' // MySQL
```

String Operations

- LIKE is used for string matching.
- String-matching operators
  - % : Matches any substring (including empty strings).
  - _ : Match any one character

```
SELECT * FROM students AS s
  WHERE s.login LIKE '%@%'
SELECT * FROM students AS s
  WHERE s.login LIKE '%@c_'
```

String Operations

- SQL-92 defines string functions.
  - Many DBMSs also have their own unique functions
- These functions can be used in any expression (projection, predicates, *e.t.c.*)

```
SELECT SUBSTRING(name,0,5) AS abbrv_name
  FROM students WHERE sid = 1

SELECT * FROM students AS s
  WHERE UPPER(s.name) LIKE 'M%'
```

String Operations

- SQL standard says to use || operator to concatenate two or more strings together.

SQL-92

```
SELECT name FROM students WHERE login = LOWER(name) || '@cs'
```

MSSQL

```
SELECT name FROM students WHERE login = LOWER(name) + '@cs'
```

MySQL

```
SELECT name FROM students WHERE login = CONCAT(LOWER(name), '@cs')
```

Date/Time Operations

- Operations to manipulate and modify DATE/TIME attributes.
- Can be used in any expression.
- Support/syntax varies wildly!
- **Task:** Get the number of days since 2000.

PostgreSQL

```
SELECT (now()::date - '2000-01-01'::date) AS days;
```

MySQL

```
SELECT DATEDIFF(CURDATE(), '2000-01-01') AS days;
```

SQL Server

```
SELECT DATEDIFF(day, '2000/01/01', GETDATE()) AS days;
```

Output Control

Output Redirection

- Store query results in another table:
  - Table must not already be defined.
  - Table will have the same number of columns with the same types as the input.

SQL-92

```
SELECT DISTINCT cid INTO CourseIds
  FROM enrolled;
```

MySQL

```
CREATE TABLE CourseIds (
  SELECT DISTINCT cid FROM enrolled
);
```

Output Redirection

- Insert tuples from query into another table:
  - Inner SELECT must generate the same columns as the target table.
  - DBMSs have different options/syntax on what to do with duplicates.

SQL-92

```
INSERT INTO CourseIds
  (SELECT DISTINCT cid FROM enrolled);
```

Output Control

- ORDER BY <column*> [ASC|DESC]
  - Order the output tuples by the values in one or more of their columns.

```
SELECT sid, grade FROM enrolled
  WHERE cid = 2
  ORDER BY grade DESC
```

```
SELECT sid, grade FROM enrolled
  WHERE cid = 2
  ORDER BY grade DESC, sid ASC
```

sid	grade
1	4
4	2

Output Control

- LIMIT <count> [offset]
  - Limit the number of tuples returned in output.
  - Can set an offset to return a "range"

```
SELECT sid, name FROM students
  WHERE login LIKE '%@cs'
  LIMIT 10
```

```
SELECT sid, name FROM students
  WHERE login LIKE '%@cs'
  LIMIT 20 OFFSET 10
```

Nested Queries

Nested Queries

- Queries containing other queries.
- They are often difficult to optimize.
- **Inner queries** can appear (almost) anywhere in query.

```
SELECT name FROM students --- Outer Query
  WHERE sid IN
        (SELECT sid FROM enrolled) --- Inner Query
```

Nested Queries

- **Task:** Get the names of students in course 2

```
SELECT name FROM students
  WHERE ...
```

Nested Queries

- **Task:** Get the names of students in course 2

```
SELECT name FROM students
  WHERE ...
   SELECT sid FROM enrolled
    WHERE cid = 2
```

Nested Queries

- **Task:** Get the names of students in course 2

```
SELECT name FROM students
  WHERE sid IN (
   SELECT sid FROM enrolled
     WHERE cid = 2
 )
```

name
Maria
Peter

Nested Queries

- ALL ⟶ Must satisfy expression for all rows in sub-query
- ANY ⟶ Must satisfy expression for at least one row in sub-query.
- IN ⟶ Equivalent to '=ANY()'.
- EXISTS ⟶ Returns true if the subquery returns one or more records.

Nested Queries

- **Task:** Get the names of students in course 2

```
SELECT name FROM students
  WHERE sid = ANY (
   SELECT sid FROM enrolled
     WHERE cid = 2
  )
```

Nested Queries

- **Task:** Get the names of students in course 2

```
SELECT name FROM students AS s
  WHERE EXISTS (                    --- EXISTS operator
    SELECT sid FROM enrolled AS e
      WHERE cid = 2 and s.sid = e.sid
  )
```

Nested Queries

- **Task:** Get the names of students in course 2

```
SELECT (SELECT s.name              --- Inner query in projection expression
        FROM students AS s
        WHERE s.sid = e.sid) AS sname
FROM enrolled AS e
WHERE cid = 2
```

Nested Queries

- **Task:** Get the names of students **not** in course 2

```
SELECT name FROM students
  WHERE sid ...
```

Nested Queries

- **Task:** Get the names of students **not** in course 2

```
SELECT name FROM students
  WHERE sid != ALL (
   SELECT sid FROM enrolled
    WHERE cid = 2
)
```

name
Rahul
Shiyi

Nested Queries

- **Task:** Find students record with the highest id that is enrolled in at least one course.

```
--- Won't work in SQL-92
SELECT MAX(e.sid), s.name
  FROM enrolled AS e, students AS s
  WHERE e.sid = s.sid;
```

Nested Queries

- **Task:** Find students record with the highest id that is enrolled in at least one course.

```
--- "Is greater than every other sid"
SELECT sid, name
  FROM students
  WHERE ...
--- "Is greater than every other sid"
SELECT sid, name
  FROM students
  WHERE sid >= ALL(
    SELECT sid FROM enrolled
)
```

sid	name
4	Peter

Nested Queries

- **Task:** Find students record with the highest id that is enrolled in at least one course.

```
SELECT sid, name FROM students
  FROM students
  WHERE sid IN (
    SELECT MAX(sid) FROM enrolled
  )
SELECT sid, name FROM students
  WHERE sid IN (
    SELECT sid FROM enrolled
    ORDER BY sid DESC LIMIT 1
)
```

Nested Queries

- **Task:** Find all courses that has no students enrolled in it.

```
SELECT * FROM courses
  WHERE ...
  --- ”with no tuples in the 'enrolled' table”
```

Nested Queries

- **Task:** Find all courses that has no students enrolled in it.

```
SELECT * FROM courses
  WHERE NOT EXISTS(
    SELECT * FROM enrolled
     WHERE courses.id = enrolled.cid
  )
```

id	name
4	Programming Languages

Window Functions

Window Functions

- Performs a "sliding" calculation across a set of **related tuples**.
- Unlike GROUP BY, tuples do not collapse into a group
- So needed if must refer back to individual tuples

```
SELECT ... FUNC-NAME(...) --- Special Window Functions, Aggregation Functions
  OVER(...) --- How to slice up data? Can also sort.
  FROM tableName
```

Window Functions

- Special window functions:
  - ROW_NUMBER() ⟶ Number of the current row
  - RANK() ⟶ Order position of the current row.
- Aggregation functions:
  - All the functions that we discussed earlier (*e.g.*, MIN, MAX, AVG)

```
SELECT *, ROW_NUMBER()
  OVER () AS row_num
  FROM enrolled
```

sid	cid	grade	row_num
1	1	B	1
1	2	A	2
2	3	B	3
4	2	A	4

Window Functions

- The OVER keyword specifies how to **group** together tuples when computing the window function.
- Use PARTITION BY to specify group.

```
SELECT cid, sid, ROW_NUMBER()
  OVER (PARTITION BY cid)      --- Note the row numbering
  FROM enrolled
  ORDER BY cid
```

cid	sid	row_number
1	1	1
2	1	1
2	4	2
3	2	1

Window Functions

- You can also include an ORDER BY in the window grouping to sort entries in each group.

```
SELECT cid, sid, ROW_NUMBER()
  OVER (ORDER BY cid)          --- Note the row numbering
  FROM enrolled
  ORDER BY cid
```

cid	sid	row_number
1	1	1
2	1	2
2	4	3
3	2	4

Window Functions

- **Task:** Find the students with the highest grade for each course.

```
SELECT cid, sid, grade, rank FROM (
  SELECT *, RANK()  -- Group tuples by cid and then sort by grade
    OVER (PARTITION BY cid ORDER BY grade ASC) AS rank
    FROM enrolled
  ) AS ranking
  WHERE ranking.rank = 1
```

cid	sid	grade	rank
1	1	B	1
2	1	A	1
3	2	B	1

Window Functions

- **Task:** Get the name of the students with the second highest grade for each course.

```
SELECT cid, sid, grade, rank FROM (
  SELECT *, RANK()
    OVER (PARTITION BY cid ORDER BY grade ASC) AS rank
    FROM enrolled
  ) AS ranking
  WHERE ranking.rank = 2  --- Update rank
```

cid	sid	grade	rank
2	4	C	2

Window Functions

- **Task:** Get the name of the students with the second highest grade for each course.

```
SELECT * FROM (
  SELECT C.name, S.name, E.grade, RANK()
    OVER (PARTITION BY E.cid ORDER BY E.grade ASC) AS grade_rank
    FROM students S, courses C, enrolled E
    WHERE S.sid = E.sid AND C.cid = E.cid   --- Connect with students
  ) AS ranking
  WHERE ranking.grade_rank = 2
```

name	name	grade	rank
Machine Learning	Peter	C	2

Common Table Expressions

Common Table Expressions

- Provides a way to write auxiliary statements for use in a larger query.
  - Think of it like a temp table just for one query.
- Alternative to nested queries and **materialized views**.

```
WITH cteName AS (
 SELECT 1
)
SELECT * FROM cteName
```

column
1

Common Table Expressions

- You can bind output columns to names before the AS keyword.

```
WITH cteName (col1, col2) AS (
 SELECT 1, 2
)
SELECT col1 + col2 FROM cteName
```

column
3

Common Table Expressions

- **Task:** Find students record with the highest id that is enrolled in at least one course.

```
WITH cteSource (maxId) AS (
 SELECT MAX(sid) FROM enrolled
)
SELECT name FROM students, cteSource
 WHERE students.sid = cteSource.maxId
```

Common Table Expressions – Recursion

- **Task:** Print the sequence of numbers from 1 to 10.

```
WITH RECURSIVE cteSource (counter) AS (
  (SELECT 1)
    UNION ALL
  (SELECT counter + 1 FROM cteSource WHERE counter < 10)
)
SELECT * FROM cteSource
```

Joins

Types of Join

- Types of Join
  - (INNER) JOIN ($\bowtie$) $\longrightarrow$ Returns records that have matching values in both tables
  - LEFT OUTER JOIN ($⟕$) $\longrightarrow$ Returns all records from the left table, and the matched records from the right table
  - RIGHT OUTER JOIN ($⟖$) $\longrightarrow$ Returns all records from the right table, and the matched records from the left table
  - FULL OUTER JOIN ($⟗$) $\longrightarrow$ Returns all records when there is a match in either left or right table

Example Database

SQL Fiddle: Link

students

sid	name
1	Maria
2	Rahul
3	Shiyi
4	Peter

hobbies

id	hobby
1	Stars
1	Climbing
2	Coding
5	Rugby

Types of Join: Inner Join

- **Task:** List the hobbies of students.

```
SELECT name, hobby
  FROM students JOIN hobbies
  ON students.id = hobbies.user_id;
```

name	grade
Maria	Stars
Maria	Climbing
Rahul	Coding

Types of Join: Left Outer Join

- **Task:** List the hobbies of **all** students.

```
SELECT name, hobby
  FROM students LEFT OUTER JOIN hobbies
  ON students.id = hobbies.user_id;
```

name	grade
Maria	Stars
Maria	Climbing
Rahul	Coding
Peter	NULL
Shiyi	NULL

Types of Join: Right Outer Join

- **Task:** List **all** the hobbies of students.

```
SELECT name, hobby
  FROM students RIGHT OUTER JOIN hobbies
  ON students.id = hobbies.user_id;
```

name	grade
Maria	Stars
Maria	Climbing
Rahul	Coding
NULL	Rugby

Types of Join: Full Outer Join

- **Task:** List **all** the hobbies of **all** students.

```
SELECT name, hobby
  FROM students FULL OUTER JOIN hobbies
  ON students.id = hobbies.user_id;
```

name	grade
Maria	Stars
Maria	Climbing
Rahul	Coding
NULL	Rugby
Peter	NULL
Shiyi	NULL

More Types of Join

- SEMI JOIN (⋉)
  - Returns record from the left table if there is **a** matching record in the right table
  - Unlike regular JOIN, only returns columns from the left table and no duplicates.
  - We do not care about the values of other columns in the right table's record
  - Used to execute queries with IN or EXISTS operators
- ANTI JOIN (▷)
  - Opposite of a SEMI JOIN
  - Returns record from the left table if there is **no** matching record in the right table
  - Used to execute queries with NOT IN or NOT EXISTS operators
- LATERAL JOIN (⋈) (*a.k.a.*, Dependent Join, CROSS APPLY)
  - Subqueries appearing in FROM clause can be preceded by the key word LATERAL
  - Table functions appearing in FROM clause can also be preceded by the key word LATERAL

Types of Join: Semi Join

- **Task:** List the names of students **with** hobbies (not their hobbies).

```
SELECT name
  FROM students
  WHERE id IN
    (SELECT id
     FROM hobbies);
```

name
Maria
Rahul
Peter

Types of Join: Anti Join

- **Task:** List the names of students **without** hobbies.

```
SELECT name
  FROM students
  WHERE id NOT IN
    (SELECT id
     FROM hobbies);
```

name
Shiyi

Types of Join: Lateral Join

- **Task:** List the names of students **with** hobbies (get student name once for each occurence of their hobby).

```
SELECT name
  FROM students, LATERAL (SELECT id FROM hobbies
                          WHERE students.id = hobbies.id) ss;
```

name
Maria
Maria
Rahul
Peter

Conclusion

- SQL is not a dead language.
- You should (almost) always strive to compute your answer as a single SQL statement.

Next Class

- Storage Management