Georgia Tech

Lecture 3: Advanced SQL

2

- office hours
- Assignment 1 → sheet 1
- C++ Familiarity
- Video readings

CREATING THE NEXT®

Today's Agenda

Advanced SQL

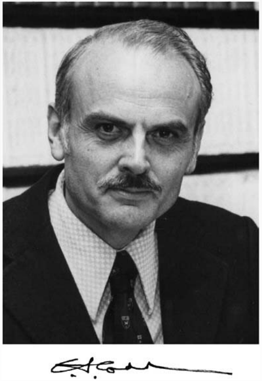
1.1 Recap
1.2 Relational Language
1.3 Aggregates
1.4 Grouping
1.5 String and Date/Time Functions
1.6 Output Control
1.7 Nested Queries
1.8 Window Functions
1.9 Common Table Expressions
1.10 Joins

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 (∪)
- Intersection (∩)
- Difference (−)
- Product (×)
- Join (⋈)

Set-oriented processing

GPUs

SIMD

Relational Language

Relational Language

- User only needs to specify the answer that they want, not how to compute it.
- The DBMS is responsible for efficient evaluation of the query.
  - Query optimizer: re-orders operations and generates query plan

SQL

Execution Engine

SQL History

- Originally “SEQUEL" from IBM’s **System R** prototype.
  - Structured English Query Language
  - Adopted by Oracle in the 1970s.
  - IBM releases DB2 in 1983.
  - ANSI Standard in 1986. ISO in 1987
  - Structured Query Language

vendr lock-in

SQL History

- Current standard is SQL:2016
  - SQL:2016 ⟶ JSON, Polymorphic tables
  - SQL:2011 ⟶ Temporal DBs, Pipelined DML
  - SQL:2008 ⟶ TRUNCATE, Fancy sorting
  - SQL:2003 ⟶ XML, windows, sequences, auto-gen IDs.
  - SQL:1999 ⟶ Regex, triggers, OO
- Most DBMSs at least support SQL-92
- Comparison of different SQL implementations

Standards

D_1, D_2

SQL Dialect

type safety

"PostgreSQL"

Relational Language

- Data Manipulation Language (**DML**)
- Data Definition Language (**DDL**)
- Data Control Language (DCL)
- Also includes:
  - View definition
  - Integrity & Referential Constraints
  - Transactions
- Important: SQL is based on bag semantics (duplicates) not set semantics (no duplicates).

List of SQL Features

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

sqlfiddle

Example Database

students

sid	name	login	age	gpa
1	Maria	maria@cs	19	3.8
2	Rahul	rahul@cs	22	3.5
3	Shiyi	shiyi@cs	26	3.7
4	Peter	peter@ece	35	3.8

enrolled

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

courses

cid	name
1	Computer Architecture
2	Machine Learning
3	Database Systems
4	Programming Languages

Aggregates

Aggregates

- Functions that return a single value from a bag of tuples:
  - AVG(col) ⟶ Return the average col value.
  - MIN(col) ⟶ Return minimum col value.
  - MAX(col) ⟶ Return maximum col value.
  - SUM(col) ⟶ Return sum of values in col.
  - COUNT(col) ⟶ Return number of values for col.

User-Defined Function — Python, Java ...

Aggregates

- Aggregate functions can only be used in the SELECT output list.
- **Task:** Get number of students with a "@cs" login:

```
SELECT COUNT(login) AS cnt
  FROM students WHERE login LIKE '%@cs'
SELECT COUNT(*) AS cnt
  FROM students WHERE login LIKE '%@cs'
SELECT COUNT(1) AS cnt
  FROM students WHERE login LIKE '%@cs'
```

%@cs

CNT
3

Multiple Aggregates

- **Task:** Get the number of students and their average GPA that have a "@cs" login.

```
SELECT AVG(gpa), COUNT(sid)
      FROM students WHERE login LIKE '%@cs'
```

AVG	CNT
3.6666	3

Distinct Aggregates

- COUNT, SUM, AVG support DISTINCT
- **Task:** Get the number of unique students that have an "@cs" login.

```
SELECT COUNT(DISTINCT login)
      FROM students WHERE login LIKE '%@cs'
```

COUNT
3

Aggregates

- Output of columns outside of an aggregate.
- **Task:** Get the average GPA of students enrolled in each course.

```
SELECT AVG(s.gpa), e.cid
  FROM enrolled AS e, students AS s
  WHERE e.sid = s.sid
```

AVG	e.cid
3.5	???

Aggregates

- Output of columns outside of an aggregate.
- **Task:** Get the average GPA of students enrolled in each course.

```
SELECT AVG(s.gpa), e.cid
  FROM enrolled AS e, students AS s
  WHERE e.sid = s.sid
```

AVG	e.cid
3.5	???

- column "e.cid" must appear in the GROUP BY clause or be used in an aggregate function

Grouping

Group By

- Project tuples into subsets and calculate aggregates of each subset.
- **Task:** Get the average GPA of students enrolled in each course.

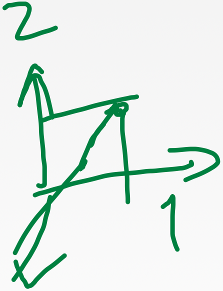
```
SELECT e.cid, AVG(s.gpa)
  FROM enrolled AS e, students AS s
  WHERE e.sid = s.sid
  GROUP BY e.cid
```

e.cid	AVG
1	3.8
3	3.5
2	3.8

1
2
3

→ Selection

→ Projection



X

Group By

- Non-aggregated values in SELECT output clause must appear in GROUP BY clause.

```
SELECT e.cid, AVG(s.gpa), s.name
  FROM enrolled AS e, students AS s
  WHERE e.sid = s.sid
  GROUP BY e.cid
```

```
SELECT e.cid, AVG(s.gpa), s.name
  FROM enrolled AS e, students AS s
  WHERE e.sid = s.sid
  GROUP BY e.cid, s.name
```

Having

102

- Filters results based on aggregate value.
- Predicate defined over a group (WHERE clause for a GROUP BY)

```
SELECT AVG(s.gpa) AS avg_gpa, e.cid
  FROM enrolled AS e, students AS s
  WHERE e.sid = s.sid AND avg_gpa > 3.9
  GROUP BY e.cid
```

```
SELECT AVG(s.gpa) AS avg_gpa, e.cid
  FROM enrolled AS e, students AS s
  WHERE e.sid = s.sid
  GROUP BY e.cid
  HAVING avg_gpa > 3.9
```

SQL → AST

parse tree

rename

query parser

binder

Having

- Filters results based on aggregate value.
- Predicate defined over a group (WHERE clause for a GROUP BY)

```
SELECT AVG(s.gpa) AS avg_gpa, e.cid
  FROM enrolled AS e, students AS s
  WHERE e.sid = s.sid
  GROUP BY e.cid
  HAVING AVG(s.gpa) > 3.9
```

e.cid	AVG
1	3.8
2	3.8

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
```

– MaRiA

– MARIA

String Operations

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

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

```
SELECT * FROM student 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(e.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')
```

DB Migration

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.
- **Demo Time!**

'abc' :: date

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

Snowflake

Q*

- R1
- X
- a2
- Y
- App

- 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);
```

Compute cost

Storage cost

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
```

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

sid	grade
1	A
4	A

Output Control

Varnish

1 . . . 1B

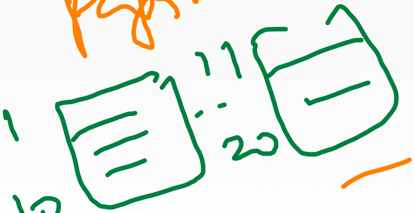
10 Q1

100 Q2

- 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
```

Pagination



Webserver

page

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

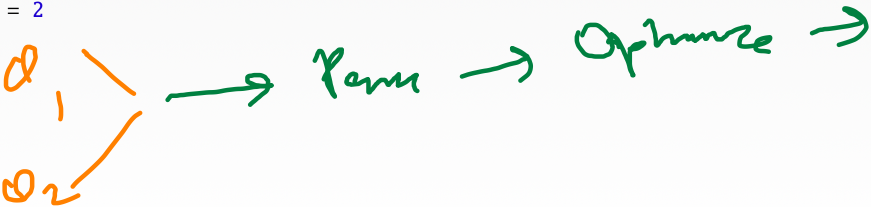
- **Task:** Get the names of students in course 2

```
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

>= All

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
      )
```

plan tree

EXPLAIN Query

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 course.cid = enrolled.cid
  )
```

cid	name
4	Peter

Window Functions

Window Functions

Time series

- 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)

```sql
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

Group By

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
```

1 2 3

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

CTE

Postgresql

- 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

readability

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

c1	c2
1	2

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
```

Plan
Tree

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 (⋈) ⟶ Returns records that have matching values in both tables
  - LEFT OUTER JOIN (⟕) ⟶ Returns all records from the left table, and the matched records from the right table
  - RIGHT OUTER JOIN (⟖) ⟶ Returns all records from the right table, and the matched records from the left table
  - FULL OUTER JOIN (⟗) ⟶ 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

sid	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 EXISTS or IN 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 EXISTS or NOT IN 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.

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

name
Maria
Rahul

Types of Join: Anti Join

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

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

name
Shiyi
Peter

ANSI Join

Types of Join: Lateral Join

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

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

name
Maria
Maria
Rahul

Conclusion

SQL-92

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

Next Class

- Storage Management