Roots: SQL for Data Scientists (lab/recitation)

Rob Carter, Duke OIT

rob@duke.edu

git clone https://github.com/carte018/mids706w4.git

Duke

Decisions

- Option 1: Continue with deeper performance and security topics from original syllabus (30 slides, ~30-45 minutes)
- Option 2: Build out demo environments to work lab exercises (~20-30 minutes, if you haven’t already done)
- Option 3: Dive into self-paced lab work and field questions (on labs, or slides, or anything related)
- Option 4: Walk through lab work on-screen as a group

Duke

Continuing the Syllabus

Duke

Ex: # 2020/2021 Psych Majors (Parentheses)

- AND has precedence over OR
- First form: all 2020 students + all 2021 Psych majors
- Second form: (correct) all Psych majors in class 2020 or 2021
- Parentheses matter

```
MariaDB [normalized]> select count(student.userid) from
    -> (student inner join majors using(userid))
    -> where student.class = 2020 or student.class = 2021 and
    -> majors.major = 'Psychology';
+----------------------+
| count(student.userid) |
+----------------------+
|                 4508 |
+----------------------+
1 row in set (0.03 sec)

MariaDB [normalized]> select count(student.userid) from
    -> (student inner join majors using(userid))
    -> where (student.class = 2020 or student.class = 2021)
    -> and majors.major = 'Psychology';
+----------------------+
| count(student.userid) |
+----------------------+
|                 1082 |
+----------------------+
1 row in set (0.02 sec)
```

Duke

Ex 2: # Civil Engineers on Campus

- Implicit join -> cartesian disaster
- “Fix" with "distinct" **works** but **slow**
- Explicit (inner) joins **much** cheaper (130x)
- Meets or exceeds performance of star schema simple query (I/O constraints)

```
MariaDB [normalized]> select count(student.userid) from student,majors,
    -> room_assignment where student.userid = room_assignment.userid and
    -> student.userid in (select userid from majors where majors.major like
    -> 'Civil%');
+----------------------+
| count(student.userid) |
+----------------------+
|              17136623 |
+----------------------+
1 row in set (0.96 sec)

MariaDB [normalized]> select count(distinct student.userid) from student,majors,
    -> room_assignment where student.userid = room_assignment.userid and
    -> student.userid in (select distinct userid from majors where majors.major
    -> like 'Civil%');
+------------------------------+
| count(distinct student.userid) |
+------------------------------+
|                         1063 |
+------------------------------+
1 row in set (2.62 sec)

MariaDB [normalized]> select count(student.userid) from (student inner join majors
    -> using(userid)) inner join room_assignment using(userid) where
    -> majors.major like 'Civil%';
+----------------------+
| count(student.userid) |
+----------------------+
|                 1063 |
+----------------------+
1 row in set (0.02 sec)

MariaDB [normalized]> select count(userid) from unnormalized
    -> where dorm is not null and room_number is not null
    -> and major like 'Civil%';
+---------------+
| count(userid) |
+---------------+
|          1063 |
+---------------+
1 row in set (0.03 sec)

MariaDB [normalized]>
```

Ex 3: Indexing FTW (careful with LIKE)

- 32 million record cross-join (for demonstration)
- Without index on “major”, query takes 1.8 sec.
- With index on “major", query takes 1.0 sec. (nearly 50%)
- “foo%” uses index efficiently; "%foo" does not !!

```
MariaDB [normalized]> select count(student.class) from
    -> (student cross join majors)
    -> where majors.major like 'Psych%';
+----------------------+
| count(student.class) |
+----------------------+
|             32385000 |
+----------------------+
1 row in set (1.83 sec)

MariaDB [normalized]> alter table majors add index maj_index (major);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [normalized]> select count(student.class) from
    -> (student cross join majors)
    -> where majors.major like 'Psych%';
+----------------------+
| count(student.class) |
+----------------------+
|             32385000 |
+----------------------+
1 row in set (1.02 sec)

MariaDB [normalized]> select count(student.class) from
    -> (student cross join majors)
    -> where majors.major like '%hology';
+----------------------+
| count(student.class) |
+----------------------+
|             32385000 |
+----------------------+
1 row in set (2.16 sec)

MariaDB [normalized]> alter table majors drop index maj_index;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [normalized]> select count(student.class) from
    -> (student cross join majors)
    -> where majors.major like '%hology';
+----------------------+
| count(student.class) |
+----------------------+
|             32385000 |
+----------------------+
1 row in set (2.16 sec)
```

Explain: Performance Insight

Duke

EXPLAIN

- Every SQL engine has some form of “EXPLAIN”
- EXPLAIN exposes the internal “query plan"
- Details differ, but usually includes insight in tables, row counts, and index usage at a minimum
- Can be run before a new query to verify how it might behave or after a slow/failed query to see why it did what it did

Duke

Explain Ex 2

```
MariaDB [normalized]> explain select count(distinct student.userid) from
    -> student,majors,room_assignment where student.userid =
    -> room_assignment.userid and student.userid in
    -> (select distinct userid from majors where majors.major like
    -> 'Civil%');
+------+-------------+-----------------+--------+---------------+-------------+---------+---------------------------------+-------+
-----------------------------------------+
| id   | select_type | table           | type   | possible_keys | key         | key_len | ref                             | rows  |
 Extra                                    |
+------+-------------+-----------------+--------+---------------+-------------+---------+---------------------------------+-------+
-----------------------------------------+
|    1 | PRIMARY     | room_assignment | index  | userid        | userid      | 4       | NULL                            |  7818 |
 Using index                              |
|    1 | PRIMARY     | student         | eq_ref | PRIMARY       | PRIMARY     | 4       | normalized.room_assignment.userid |     1 |
 Using index                              |
|    1 | PRIMARY     | <subquery2>     | eq_ref | distinct_key  | distinct_key | 4       | func                            |     1 |
                                          |
|    1 | PRIMARY     | majors          | index  | NULL          | userid      | 4       | NULL                            | 16357 |
 Using index; Using join buffer (flat, BNL join) |
|    2 | MATERIALIZED | majors          | ALL    | userid        | NULL        | NULL    | NULL                            | 16357 |
 Using where                              |
+------+-------------+-----------------+--------+---------------+-------------+---------+---------------------------------+-------+
-----------------------------------------+
5 rows in set (0.01 sec)

MariaDB [normalized]> explain select count(userid) from
    -> (student inner join majors using(userid))
    -> inner join room_assignment using(userid)
    -> where majors.major like 'Civil%';
+------+-------------+-----------------+--------+---------------+---------+---------+---------------------------------+------+-------------+
| id   | select_type | table           | type   | possible_keys | key     | key_len | ref                             | rows | Extra       |
+------+-------------+-----------------+--------+---------------+---------+---------+---------------------------------+------+-------------+
|    1 | SIMPLE      | room_assignment | index  | userid        | userid  | 4       | NULL                            | 7818 | Using index |
|    1 | SIMPLE      | student         | eq_ref | PRIMARY       | PRIMARY | 4       | normalized.room_assignment.userid |    1 | Using index |
|    1 | SIMPLE      | majors          | ref    | userid        | userid  | 4       | normalized.room_assignment.userid |    1 | Using where |
+------+-------------+-----------------+--------+---------------+---------+---------+---------------------------------+------+-------------+
3 rows in set (0.00 sec)
```

Duke

Explain Ex 3

```
MariaDB [normalized]> explain select count(student.class) from
    -> (student cross join majors)
    -> where majors.major like 'Psych%';
+------+-------------+---------+-------+---------------+-----------+---------+------+-------+
----------------------------------+
| id   | select_type | table   | type  | possible_keys | key       | key_len | ref  | rows  |
 Extra                            |
+------+-------------+---------+-------+---------------+-----------+---------+------+-------+
----------------------------------+
|    1 | SIMPLE      | majors  | range | maj_index     | maj_index | 258     | NULL |  2158 |
 Using where; Using index         |
|    1 | SIMPLE      | student | ALL   | NULL          | NULL      | NULL    | NULL | 15282 |
 Using join buffer (flat, BNL join) |
+------+-------------+---------+-------+---------------+-----------+---------+------+-------+
----------------------------------+
2 rows in set (0.00 sec)

MariaDB [normalized]> explain select count(student.class) from
    -> (student cross join majors)
    -> where majors.major like '%hology';
+------+-------------+---------+-------+---------------+-----------+---------+------+-------+
-------------------------------------------------------+
| id   | select_type | table   | type  | possible_keys | key       | key_len | ref  | rows  |
 Extra                                                 |
+------+-------------+---------+-------+---------------+-----------+---------+------+-------+
-------------------------------------------------------+
|    1 | SIMPLE      | student | ALL   | NULL          | NULL      | NULL    | NULL | 15282 |
                                                       |
|    1 | SIMPLE      | majors  | index | NULL          | maj_index | 258     | NULL | 16357 |
 Using where; Using index; Using join buffer (flat, BNL join) |
+------+-------------+---------+-------+---------------+-----------+---------+------+-------+
-------------------------------------------------------+
2 rows in set (0.00 sec)
```

Duke

Miscellany

Duke

Architecture and DBA stuff

Duke

Redundancy

- Some vendors offer redundant DB server options
  - Eg. “RAC” by Oracle
- Additional SQL query engine horsepower/load balancing
- Protection during upgrades, single point failures
- Single back-end data store
  - No data protection, some risk of lock contention

Duke

Replication

- Some vendors provide replication mechanisms
  - MySQL replication, MS-SQL replication
- Multiple engines, each with its own data store
- Updates synchronized (usually via “log shipping”)
- Typically single-write/multi-read
- Better data protection, less write throughput gain, possible update lag

Duke

Clustering

- Some vendors provide full clustering for databases
  - MySQL Galera, Amazon Aurora
- Multiple engines with local stores
- Usually multi-write, with inter-server locking and synchronous or near-real-time asynchronous updates
- Provide both performance scalability and data redundancy
- Highly complex, difficult to manage, quirky

Duke

Performance (Again)

- You may not know (nor want to know) architectural details
- Architectural details can affect performance, availability
- Also cost
- Good DBAs can help you navigate the options, support you when things go wrong

Duke

Four things DBAs (seem to) hate

- Overly expensive queries, esp. on shared Database engines
- Excessive data bloat (intermediate tables, materialized views, unnormalized table expansion)
- Unsubstantiated architectural demands
- Overly expensive queries, esp. on shared Database engines

Duke

Security: An Afterthought

Duke

Three Key Goals

- Data Integrity: Making sure the DB's data remains intact (or can be recovered if it doesn’t)

- Access Management: Making sure the right people have the right access

- Privacy/Data Minimization: Limiting access to "need to know”

Duke

Data Integrity

- Referential Integrity (already discussed)
- Constraints (already discussed)
- Replication (already discussed)
- Backups…

Duke

A digression about Backups

- Another reason to keep your DBAs close
- Databases are treacherous to back up
  - Data are constantly in flux
  - Most updates linger in RAM before being committed to disk
  - Backing up an inconsistent database **will** miss data
- "Cold" backups and/or log-archiving are effective options

Duke

Access Management

- Typically hinges on the use of GRANT/REVOKE statements
- GRANT <privilege> ON <object> TO <user/role> [WITH GRANT OPTION]
- REVOKE <privilege> ON <object> FROM <user/role>
- System privs: Create <object>
- Object privs: INSERT, SELECT, UPDATE, EXECUTE
- Objects: typically tables or views, may be procedures or other objects

Duke

Access Management Tips

- Always aim for least access
    - If the client is read-only, limit to SELECT rights
    - If the client needs only certain tables, avoid granting rights to “db.*”
    - If the client does not need insert/delete rights, grant only {SELECT,UPDATE}
    - Be aware that some operations may require more rights than you'd expect, and restricting some rights may limit options for the SQL query optimizer

Duke

Privacy and Data Minimization

- Most SQL access controls are non-granular
  - Table-level security
- Some vendors offer more granular row-level security options
- Column-level security is a different story…

Duke

Privacy and Data Minimization

- Scenario: The Chair of the Psychology department has a need to see COVID-19 testing results for Psychology majors in order to plan class adjustments for upper division Psych classes. She does not need to see their room assignments, and does not need to see info about non-Psych majors at all

- Multiple possible approaches to solving this scenario

Duke

Solution 1: normalized view

- Create "psych_covid" view
  - Psych students with non-null test information (524 total)
- Grant “psych1" user SELECT on the new view ONLY
- ("identified by" is a MySQL-ism for creating a user during a grant operation)

```
MariaDB [normalized]> create view psych_covid as
    -> select student.userid as userid,
    -> student.firstname as firstname,
    -> student.lastname as lastname,
    -> student.class as class,
    -> tests.test_result as test_result,
    -> tests.test_date as test_date,
    -> tests.test_returned as test_returned
    -> from (student inner join tests using(userid))
    -> inner join majors using(userid) where
    -> majors.major = 'Psychology';
Query OK, 0 rows affected (0.00 sec)

MariaDB [normalized]> select count(*) from psych_covid;
+----------+
| count(*) |
+----------+
|      524 |
+----------+
1 row in set (0.01 sec)

MariaDB [normalized]> grant select on normalized.psych_covid
    -> to ‘psych1’@‘localhost’ identified by 'P@55word';
Query OK, 0 rows affected (0.00 sec)
```

Duke

Solution 1: normalized view

- psych1 user can only see the view (not the component tables)
- As far as psych1 is concerned, the entire database consists of test results for Psych majors.

```
root@rlyeh-02  /home/rob $ mysql -u psych1 -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 67
Server version: 5.5.65-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use normalized;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [normalized]> show tables;
+----------------------+
| Tables_in_normalized |
+----------------------+
| psych_covid          |
+----------------------+
1 row in set (0.00 sec)

MariaDB [normalized]> select * from psych_covid where
    -> class = 2023 limit 10;
+---------+------------+----------+-------+-------------+------------+---------------+
| userid  | firstname  | lastname | class | test_result | test_date  | test_returned |
+---------+------------+----------+-------+-------------+------------+---------------+
| 4201288 | Pahvie     | Blanco   | 2023  |           1 | 2020-07-23 | 2020-07-24    |
| 4201437 | Maria-Rosa | Knowles  | 2023  |           1 | 2020-05-05 | 2020-05-08    |
| 4201463 | Marian     | Gollon   | 2023  |           0 | 2020-04-22 | 2020-04-25    |
| 4201520 | Pooja      | Kamau    | 2023  |           0 | 2020-03-15 | 2020-03-18    |
| 4201682 | Mehreen    | Mohl     | 2023  |           1 | 2020-05-09 | 2020-05-12    |
| 4201879 | Wrenn      | Figuei   | 2023  |           0 | 2020-05-24 | 2020-05-27    |
| 4201887 | Carsietta  | Schafer  | 2023  |           1 | 2020-06-27 | 2020-06-30    |
| 4202068 | Haoyue     | Linhardt | 2023  |        NULL | 2020-03-16 | NULL          |
| 4202120 | Anshuman   | Cash     | 2023  |        NULL | 2020-07-13 | NULL          |
| 4202171 | Twan       | Mihaich  | 2023  |        NULL | 2020-03-27 | NULL          |
+---------+------------+----------+-------+-------------+------------+---------------+
10 rows in set (0.00 sec)

MariaDB [normalized]> select count(*) from psych_covid;
+----------+
| count(*) |
+----------+
|      524 |
+----------+
1 row in set (0.01 sec)
```

Other potential solutions

- Create a view of just psychology majors, grant psych1 SELECT on both that view and the full tests table.
    - psych1 can only get personalization data for psych majors, but can see all test results (with opaque identifiers)
- If DB supports row-level security, re-normalize to colocate major and test information and use “where major=“ constraint for row-level security (MySQL can’t do this)
- Start from * schema and write stored procedure to restrict results based on CURRENT_USER value and returned data, attach AFTER SELECT trigger to call it on results
- Write an application to access the database and give Chair access to the app

Duke

SQL Injection: A Parable

- Say you develop an app that allows psych1 to:
  - enter a set of criteria for a SELECT statement
  - runs a query with those criteria AND LIMITS RESULTS TO PSYCH MAJORS ONLY
- Imagine that psych1 wants to try to be naughty…

Duke

SQL Injection: A Parable

- Imagine your app takes “where" clauses as input and constructs SELECT statements as:
  - SELECT * from (student inner join tests using(userid)) join majors using(userid) where $input AND majors.major = ‘Psychology’;

- What happens if psych1 enters:
  - $input = “student.class = 2023”?
  - $input = "student.class = 2023; select * from student where userid is not null”?
  - $input = "student.class = 2023; drop table tests; drop table majors; drop table student;”?

Duke

Moral

- Limiting access goes for application users as well as real users
- CLEANSE YOUR INPUT! (most languages make this easy)
- Remember that the best security is applied as close as possible to what you are protecting
- Little Johnnie Drop Tables, like the wolf, is always at the door

Duke

Preparing Your Environment

Duke

Build the Machine

Duke

Stuff you'll need

- A client machine (Linux, Windows, Mac) with a working SSH client and network access to:

- A server machine to run MySQL with sufficient storage to hold a modest-sized database (VCM VMs are fine for this, but may require using VPN)

- A running MySQL server on your server machine, and a GIT client (to retrieve the lab exercises and database) and zip tool (for unpacking the database install data)

- Concepts we discussed in the first section last week :-)

Duke

Reserving a VCM VM

- https://vcm.duke.edu
- “Reserve a VM”
- “Lamp Stack”
- Select authN mechanism (SSH keys if you have them registered, otherwise choose less secure password auth)
- You will log in as “vcm” using the password hidden in the UI under “View Password”

Duke

Installing MySQL

- SSH into your VCM machine as “vcm” and:
- apt install -y mysql-server (to install MySQL server)
- mysql -u root (note: no initial password for MySQL)
- Optional (but recommended): Set root password in MySQL:
  - update mysql.user set plugin=‘mysql_native_password’ where user=‘root’ and host=‘localhost’;
  - flush privileges;
  - set password for ‘root’@‘localhost’ = ‘YOUR_FAVORITE_MYSQL_PASSWORD’;
- Henceforth, you’ll use “mysql -u root -p” (and enter YOUR_FAVORITE_MYSQL_PASSWORD) to get in

Duke

Install Lab

Database(s)

Duke

Getting the Materials

- ssh into your VM as “vcm” and run:
  - git clone https://github.com/carte018/mids706w4.git
  - cd mids706w4
  - unzip class.zip (apt install -y zip [if unzip not found])

Duke

Install Lab Data

- mysql -u root -p -e ‘create database running_totals’
- mysql -u root -p running_totals < class/running_totals.sql
- mysql -u root -p -e ‘create database dewey_cheatham_howe
- mysql -u root -p dewey_cheatham_howe < class/dewey_cheatham_howe.sql
- mysql -u root -p -e ‘create database orders’
- mysql -u root -p orders < class/orders.sql
- mysql -u root -p -e ‘create database normalized’
- mysql -u root -p normalized < class/normalized.sql

Duke

Done!

Duke

Lab Exercises

Duke

Lab Notes

- Lab notes are in the PDF on your VCM (or other) MySQL server:
  - Lab notes for MIDS 706 week 4.pdf
- You can also retrieve that file from:
  - https://github.com/carte018/mids706w4
- and view it in the browser or in a PDF viewer
- **NB:** Some of the lab exercises built on each other, so you need to work through them sequentially — earlier exercises may prepare the data for later exercises

Duke

Exercises 1-7 (modeling)

- Recall ER and Relational models
- Look around the schema (using ‘show’ and ‘describe’) in your MySQL databases
- Think about what the real-world data are and how they’re modeled in the relational tables provided

Duke

Normal Forms (in more detail)

- 1NF: Columns are singular (eg: “major”, not “major1”,”major2”…), at most one value per column, and no fully duplicated rows

- 2NF: 1NF **plus** every table has a primary key (which may be compound) and no columns are **functionally dependent** on anything but the primary key column(s) (eg. if primary key is “last name”, having both “first name” and “full name” columns would violate functional dependence)

- 3NF: 2NF **plus** no columns have transitive functional dependencies (eg.: a table mapping id [PK] to zip, city, and state could be in 2NF but not 3NF because zip -> city+state)

Duke

Exercises 8-10 (constraints)

- MySQL ALTER TABLE syntax:
  - ALTER TABLE <table> ADD CONSTRAINT <title> UNIQUE(col[,…]);
  - ALTER TABLE <table> MODIFY col_name type_def col_constraint(s);
    - eg: ALTER TABLE foo MODIFY fav_song VARCHAR(255) NOT NULL;
- MySQL “show create <table_name>” for extended info
- LIMIT clause: SELECT … LIMIT n; (to limit to first “n” results)
- SELECT <fields> FROM table(s) ==> SELECT * FROM table(s)
- INSERT INTO <table> (field[,…]) VALUES(value[,…]);

Duke

Exercises 11-18 (subqueries)

- IN clauses allow WHERE to use lists:
  - SELECT x FROM t WHERE f IN (val1, val2, val3…);
- Subqueries can replace lists or tables in other queries
  - NB: when subqueries replace tables, they need aliases (“AS” clauses)
    - SELECT x from (SELECT y from t where…) as ss where ss.field = value;
- Subqueries can operate on tables (or even other subqueries) distinct from their wrapping queries
- SELECT count(*) returns # rows in the result. SELECT count(distinct(x)) returns number of *different values* of x in the table.

Duke

Exercises 19-21 (joins)

- Inner join syntax:
  - general: <t1> INNER JOIN <t2> ON t1.field1 = t2.field2
  - simplified: <t1> INNER JOIN <t2> USING(field)
- Outer join syntax:
  - <t1> LEFT JOIN <t2> ON t1.field1 = t2.field2
  - <t1> RIGHT JOIN <t2> ON t1.field1 = t2.field2
  - MySQL does not support full outer joins
- Cross join syntax (WATCH OUT!):
  - <t1> CROSS JOIN <t2> or (sometimes) <t1>,<t2>

Duke

Exercises 19-21 (joins)

- GROUP BY clauses:
  - allow queries that “roll up” results into “groups”
  - SELECT x,count(*) FROM t1 GROUP BY x;
    - returns one row per distinct value of “x” with value and count of rows in the group — how many rows have each value of “x”
- GROUP BY … HAVING clauses:
  - allow restricting returned groups by some constraint
    - SELECT x,count(*) from t1 GROUP BY x HAVING count(*) > 200;
      - same as above, but only return groups where count(*) > 200

Duke

Exercises 22-24 (views)

- Any SELECT statement can be memorialized as a view
  - CREATE VIEW <v_name> AS <select statement>;
  - MySQL does not support materialized views, but "CREATE MATERIALIZED VIEW" in other DBMS
- Views are interchangeable with tables in both GRANTs and SELECTS.
- Views are by definition read-only (although some DBMS may support special "write-thru" views)

Duke

Exercises 26-28 (performance)

- Improper or “implicit” joins cause many (most?) performance issues for queries
- Boolean algebra can get complex - parentheses are explicit
- Indexes are fast; leading globs in LIKE queries bypass them
- Never SELECT x from (SELECT x from t) — simpler is better
- Often “unique” or “distinct” is shorthand for “…and then remove all the dreck from the unintentional cross join” :-)

Duke