MIDS 706 Week 4 Roots: SQL for Data Scientists (part ii)

Rob Carter, Duke OIT

Duke

Today's Tour

- A bit of theory: Models, Normalization
- Constraints, keys, and referential integrity
- Advanced features and RDBMS peculiarities
- Performance: 'cause we all want some
- Architectural issues and DBAs
- Security: a perennial afterthought

Duke

Theory: Data Modeling (“Lite”)

Duke

Data Models

- Data of consequence represent something else (real)
- **Model**: Abstract way of organizing data to reflect that representation
- Two Broad (Mostly interchangeable) Categories: ER and Relational

Duke

ER Model

- Cf: Chen, Peter (1976)
- Entities - Intern (2), Employee (1), Business (1)
- Relations
    - **John** mentors (Lisa, Anthony), works for Circle-J
- **Keys** are unique identifiers
- Relation **Cardinality** (1-1, 1-many, many-1)

Duke

ER Diagram

(strong entity)

BUSINESS

ID (304A-10)
name (Circle-J)
address (Rt. 12, Box 3)
vertical (farming[wheat])

1-many

(weak entity)

EMPLOYEE

ssn (xxx-yy-zzzz)
name (John W.)
businessid (304A-10)

1-0 or more

(weak entity)

INTERN

mentorssn(xxx-yy-zzzz)
name (Anthony,Lisa)
studentID (40014,41213)

Duke

Relational Models

- Cf. Edgar Codd (1970)
- Describe **Tables** that represent
- **Relations** between
- **Tuples (rows)** representing
- **Entities**
- No explicit concept of cardinality
- Directly correlate with RDB tables

Duke

Relational Model

Business

ID (pk)	Name	Address	Vertical
304A-10	Circle-J	Rt. 12, Box 3	farming
108B-21	Data Assoc	101 Main St	IT

Employees

ssn (pk)	Name	BusinessID (fk)
xxx-yy-zzzz	John W	304A-10
aaa-bb-cccc	Gina T	304A-10

Interns

StudentID (pk)	Name	MentorID (fk)
40014	Anthony	xxx-yy-zzzz
41213	Lisa	xxx-yy-zzzz

Duke

Sorta Theory: Schemas and Normalization

Duke

Schemas (schemata)

- Database Schema: Abstract organization of a database
- Often refer to “table schema”
    - Table schema represents columns and constraints
- Table schemata include data typing, indexing, and keying
- We will use some totally fictional data for demonstration

Duke

Table Schema

Student COVID Testing

Field (column)	Type	Null	Key	Default	Extra
userid	int(11)	NO		NULL	
firstname	varchar(255)	NO		NULL	
lastname	varchar(255)	NO		NULL	
dorm	varchar(255)	YES		NULL	
major	varchar(255)	YES		NULL	
class	int(11)	YES		NULL	
room_number	int(11)	YES		NULL	
test_date	date	YES		NULL	
test_returned	date	YES		NULL	
test_result	tinyint(1)	YES		NULL	

Duke

Star Schema

- “Star schema” - everything in one table
- Reasonable for many simple cases
- Problematic for high cardinality relations, security
- Often the first schema style we reach for

Duke

Star Schema Data

Unnormalized

userid	firstname	lastname	dorm	major	class	room_number	test_date	test_returned	test_result
4201200	Kaylene	Nakazibwe	G-A	Sociology	2022	918	NULL	NULL	NULL
4201201	Bilyana	McMahon	Epworth	Sociology	2021	544	NULL	NULL	NULL
4201202	Yuri	Crossey	Alspaugh	Psychology	2020	433	2020-05-26	2020-05-29	1
4201203	Joie	Knuffman	House CC	ECE	2020	1266	NULL	NULL	NULL
4201204	Evangaline	Spiritos	NULL	Psychology	2022	NULL	NULL	NULL	NULL

…and so on for 15,000 rows

Duke

Star Schema Fail (multiple majors!)

userid	firstname	lastname	dorm	major	class	room_number	test_date	test_returned	test_result
4201200	Kaylene	Nakazibwe	G-A	Sociology	2022	918	NULL	NULL	NULL
4201200	Kayle	Nakazibwe	G-A	Psychology	2022	918	NULL	NULL	NULL
4201201	Bilyana	McMahon	Epworth	Sociology	2021	544	NULL	NULL	NULL
4201202	Yuri	Crossey	Alspaugh	Psychology	2020	433	2020-05-26	2020-05-29	1
4201202	Yuri	Crossey	Alspaugh	Sociology	2020	433	2020-05-26	2020-05-29	0
4201203	Joie	Knuffman	House CC	ECE	2020	1266	NULL	NULL	NULL
4201204	Evangaline	Spiritos	NULL	Psychology	2022	NULL	NULL	NULL	NULL
4201204	Angie	Spiritos	NULL	English	2022	NULL	NULL	NULL	NULL
4201204	Angie	Spiritos	NULL	Sociology	2022	NULL	NULL	NULL	NULL

Duke

Star Schema Fail
(just add more columns!)

userid	firstname	lastname	dorm	major1	major2	class	room_number	test_date	test_returned	test_result
4201200	Kaylene	Nakazibwe	G-A	Sociology	Psychology	2022	918	NULL	NULL	NULL
4201201	Bilyana	McMahon	Epworth	Sociology	NULL	2021	544	NULL	NULL	NULL
4201202	Yuri	Crossey	Alspaugh	Psychology	Sociology	2020	433	2020-05-26	2020-05-29	1
4201203	Joie	Knuffman	House CC	ECE	NULL	2020	1266	NULL	NULL	NULL
4201204	Evangaline	Spiritos	NULL	Psychology	English	2022	NULL	NULL	NULL	NULL
4201204	Angie	Spiritos	NULL	Sociology		2022	NULL	NULL	NULL	NULL

Duke

Normalization

- More from our friend Codd
- Restructuring database to reduce redundancy and eliminate classic “anomalies"
  - Update - data replicated across rows can drift
  - Insertion - rows require data that may not exist
  - Deletion - removing rows may have unintended results

Duke

Normalization

- Can be specified formally (mathematically) using ~10 “forms" and language of relational algebra
- For our purposes, normalization is about reducing/removing redundancy and formalizing keys - we’ll stick to 1st-3rd normal form (1NF/2NF/3NF)
- Typically involves breaking “star schema” into multiple tables
- At any normalization level, there may be multiple “valid” normalized representations
    - Choices focus on how the data are to be used

Duke

Normalizing the example

- Consider entities (students, tests) as candidate tables
- Consider relations and their cardinality (1 student -> many majors) (1-many relations should span tables)
- Consider the types of queries, types of updates that will be performed

Duke

Normalized (somewhat)

- Reduces redundancy
  - Student with > 1 test, > 1 major
- No need to store empty values
  - students don’t have to have tests
  - students don’t have to have rooms

```
MariaDB [normalized]> describe student;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| userid    | int(11)     | NO   |     | NULL    |       |
| firstname | varchar(255) | NO   |     | NULL    |       |
| lastname  | varchar(255) | NO   |     | NULL    |       |
| class     | int(11)     | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

MariaDB [normalized]> describe tests;
+--------------+-----------+------+-----+---------+-------+
| Field        | Type      | Null | Key | Default | Extra |
+--------------+-----------+------+-----+---------+-------+
| userid       | int(11)   | NO   |     | NULL    |       |
| test_date    | date      | YES  |     | NULL    |       |
| test_returned | date      | YES  |     | NULL    |       |
| test_result  | tinyint(1) | YES  |     | NULL    |       |
+--------------+-----------+------+-----+---------+-------+
4 rows in set (0.01 sec)

MariaDB [normalized]> describe majors;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| userid | int(11)     | NO   |     | NULL    |       |
| major  | varchar(255) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

MariaDB [normalized]> describe room_assignment;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| userid      | int(11)     | NO   |     | NULL    |       |
| dorm        | varchar(255) | YES  |     | NULL    |       |
| room_number | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Duke

Normalized (data)

- 15,000 students
- 16,121 majors
  - now some students have > 1 major
- 7615 room_assigments
  - some students aren't on-campus
- Tests are also one-to-many

```
MariaDB [normalized]> select count(*) from student;
+----------+
| count(*) |
+----------+
|    15000 |
+----------+
1 row in set (0.00 sec)

MariaDB [normalized]> select count(*) from room_assignment;
+----------+
| count(*) |
+----------+
|     7615 |
+----------+
1 row in set (0.00 sec)

MariaDB [normalized]> select count(*) from majors;
+----------+
| count(*) |
+----------+
|    16121 |
+----------+
1 row in set (0.01 sec)

MariaDB [normalized]> select count(*) from tests;
+----------+
| count(*) |
+----------+
|     3499 |
+----------+
1 row in set (0.01 sec)
```

Duke

Practicum

Duke

Constraints, Keys, and Referential Integrity

Duke

SQL Constraints

- Establish inviolable rules for data in a DB
  - Note: Enforcement may vary from RDBM to RDBM
- Can be used to ensure accuracy, preserve consistency, increase performance, or enforce business/data rules

Duke

Basic Constraints

- Column-level
  - NOT NULL - the column must have a non-null value
  - UNIQUE - the value in the column(s) must be different in every row (may be multi-column “unique(x,y)”)
  - DEFAULT - set a default value for INSERT operations

Duke

Basic Constraints

- Table-level
  - PRIMARY KEY - NOT NULL + UNIQUE (poss. index)
  - FOREIGN KEY - ties two tables together (more later)
  - CHECK - arbitrary constraint in the table (eg. test_date <= test_returned) (MySQL > 8.0.16*)

Duke

At Table Creation

```
MariaDB [test3]> create table student (
    -> userid int(11) PRIMARY KEY,
    -> firstname varchar(255) NOT NULL,
    -> lastname varchar(255) NOT NULL,
    -> class int(11) NOT NULL);
Query OK, 0 rows affected (0.00 sec)

MariaDB [test3]> describe student;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| userid    | int(11)      | NO   | PRI | NULL    |       |
| firstname | varchar(255) | NO   |     | NULL    |       |
| lastname  | varchar(255) | NO   |     | NULL    |       |
| class     | int(11)      | NO   |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Duke

After the fact (ALTER TABLE)

```
MariaDB [test3]> create table student (
    -> userid int(11),
    -> firstname varchar(255),
    -> lastname varchar(255),
    -> class int(11));
Query OK, 0 rows affected (0.01 sec)

MariaDB [test3]> describe student;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| userid    | int(11)      | YES  |     | NULL    |       |
| firstname | varchar(255) | YES  |     | NULL    |       |
| lastname  | varchar(255) | YES  |     | NULL    |       |
| class     | int(11)      | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

MariaDB [test3]> alter table student add PRIMARY KEY (userid),
    -> modify firstname varchar(255) NOT NULL,
    -> modify lastname varchar(255) NOT NULL,
    -> modify class int(11) NOT NULL;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [test3]> describe student;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| userid    | int(11)      | NO   | PRI | 0       |       |
| firstname | varchar(255) | NO   |     | NULL    |       |
| lastname  | varchar(255) | NO   |     | NULL    |       |
| class     | int(11)      | NO   |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Duke

Constraint Enforcement

- INSERT, UPDATE, DELETE, DROP operations are not allowed to violate constraints

```
MariaDB [test3]> describe student;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| userid    | int(11)      | NO   | PRI | 0       |       |
| firstname | varchar(255) | NO   |     | NULL    |       |
| lastname  | varchar(255) | NO   |     | NULL    |       |
| class     | int(11)      | NO   |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

MariaDB [test3]> insert into student (userid,firstname,lastname,class) values(444000,'Louise','Murcheson',2022);
Query OK, 1 row affected (0.00 sec)

MariaDB [test3]> insert into student (userid,firstname,lastname,class) values(444000,'Larry','Mendehlson',NULL);
ERROR 1048 (23000): Column 'class' cannot be null
MariaDB [test3]> insert into student (userid,firstname,lastname,class) values(444000,'Larry','Mendehlson',2023);
ERROR 1062 (23000): Duplicate entry '444000' for key 'PRIMARY'
MariaDB [test3]> insert into student (userid,firstname,lastname,class) values(444001,'Larry','Mendehlson',2023);
Query OK, 1 row affected (0.01 sec)

MariaDB [test3]> update student set userid = 444000 where userid = 444001;
ERROR 1062 (23000): Duplicate entry '444000' for key 'PRIMARY'
MariaDB [test3]> update student set userid = 444002 where userid = 444001;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Duke

Constraining the example data, defining keys

- If fields should not be null, make them NOT NULL
- Add or designate a primary key for every table (2NF, RI)
- Designate foreign keys as such (more on why later) (2NF, RI)
- Where appropriate, set reasonable defaults, unique constraints

Duke

Constraining the example data, defining keys

```
MariaDB [normalized]> alter table student add PRIMARY KEY (userid), modify firstname varchar(255) NOT NULL,
   -> modify lastname varchar(255) NOT NULL, modify class varchar(255) DEFAULT 2024 NOT NULL;
Query OK, 15000 rows affected (0.09 sec)
Records: 15000  Duplicates: 0  Warnings: 0

MariaDB [normalized]> alter table majors add column declarationid int(11) primary key AUTO_INCREMENT FIRST;
Query OK, 16121 rows affected (0.12 sec)
Records: 16121  Duplicates: 0  Warnings: 0

MariaDB [normalized]> alter table room_assignment add column assignmentid int(11) primary key AUTO_INCREMENT FIRST;
Query OK, 7615 rows affected (0.05 sec)
Records: 7615  Duplicates: 0  Warnings: 0

MariaDB [normalized]> alter table tests add column testid int(11) primary key AUTO_INCREMENT FIRST;
Query OK, 3499 rows affected (0.04 sec)
Records: 3499  Duplicates: 0  Warnings: 0

MariaDB [normalized]> alter table majors add FOREIGN KEY (userid) REFERENCES student(userid);
Query OK, 16121 rows affected (0.18 sec)
Records: 16121  Duplicates: 0  Warnings: 0

MariaDB [normalized]> alter table room_assignment add FOREIGN KEY (userid) REFERENCES student(userid);
Query OK, 7615 rows affected (0.10 sec)
Records: 7615  Duplicates: 0  Warnings: 0

MariaDB [normalized]> alter table tests add FOREIGN KEY (userid) REFERENCES student(userid);
Query OK, 3499 rows affected (0.06 sec)
Records: 3499  Duplicates: 0  Warnings: 0
```

Duke

Constraining the example data — uniqueness

```
MariaDB [normalized]> ALTER TABLE tests ADD CONSTRAINT one_test_per_day UNIQUE(userid,test_date);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [normalized]> SHOW CREATE TABLE tests;
+-------------------------------+
| Table | Create Table
+-------------------------------+
| tests | CREATE TABLE `tests` (
  `testid` int(11) NOT NULL AUTO_INCREMENT,
  `userid` int(11) NOT NULL,
  `test_date` date DEFAULT NULL,
  `test_returned` date DEFAULT NULL,
  `test_result` tinyint(1) DEFAULT NULL,
  PRIMARY KEY (`testid`),
  UNIQUE KEY `one_test_per_day` (`userid`,`test_date`),
  CONSTRAINT `tests_ibfk_1` FOREIGN KEY (`userid`) REFERENCES `student` (`userid`)
) ENGINE=InnoDB AUTO_INCREMENT=3564 DEFAULT CHARSET=latin1 |
+-------------------------------+
1 row in set (0.00 sec)

MariaDB [normalized]> select * from tests limit 2;
+--------+---------+------------+---------------+-------------+
| testid | userid  | test_date  | test_returned | test_result |
+--------+---------+------------+---------------+-------------+
|      1 | 4201202 | 2020-05-26 | 2020-05-29    |           1 |
|      2 | 4201209 | 2020-06-21 | NULL          |        NULL |
+--------+---------+------------+---------------+-------------+
2 rows in set (0.00 sec)

MariaDB [normalized]> insert into tests (userid,test_date,test_returned,test_result)
values(4201209,DATE('2020-06-21'),DATE('2020-06-27'),0);
ERROR 1062 (23000): Duplicate entry '4201209-2020-06-21' for key 'one_test_per_day'
```

Duke

Normalized, Constrained

```
MariaDB [normalized]> describe student;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| userid    | int(11)      | NO   | PRI | NULL    |       |
| firstname | varchar(255) | NO   |     | NULL    |       |
| lastname  | varchar(255) | NO   |     | NULL    |       |
| class     | varchar(255) | NO   |     | 2024    |       |
+-----------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

MariaDB [normalized]> describe majors;
+---------------+-------------+------+-----+---------+----------------+
| Field         | Type        | Null | Key | Default | Extra          |
+---------------+-------------+------+-----+---------+----------------+
| declarationid | int(11)     | NO   | PRI | NULL    | auto_increment |
| userid        | int(11)     | NO   | MUL | NULL    |                |
| major         | varchar(255)| YES  |     | NULL    |                |
+---------------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

MariaDB [normalized]> describe room_assignment;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| assignmentid | int(11)      | NO   | PRI | NULL    | auto_increment |
| userid       | int(11)      | NO   | MUL | NULL    |                |
| dorm         | varchar(255) | YES  |     | NULL    |                |
| room_number  | int(11)      | YES  |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

MariaDB [normalized]> describe tests;
+---------------+------------+------+-----+---------+----------------+
| Field         | Type       | Null | Key | Default | Extra          |
+---------------+------------+------+-----+---------+----------------+
| testid        | int(11)    | NO   | PRI | NULL    | auto_increment |
| userid        | int(11)    | NO   | MUL | NULL    |                |
| test_date     | date       | YES  |     | NULL    |                |
| test_returned | date       | YES  |     | NULL    |                |
| test_result   | tinyint(1) | YES  |     | NULL    |                |
+---------------+------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

MariaDB [normalized]>
```

Duke

Referential Integrity

- Referential integrity: Ensuring that all links between entities/ tables are “valid” -- all references exist
- When one table **refers** to another, implicit RI constraints arise:
  - On INSERT/UPDATE: new references must be valid
  - On DELETE/DROP: resultant state must be valid across tables
- Defining FOREIGN KEY constraint in a table enables RI
  - Constraint is in the 'child' table; 'parent table’ fulfills reference

Duke

RI Enforcement

- Wild West case (no foreign key defs)
- Inserting an order referencing
  - non-existent productid
  - non-existent account
- Perfectly fine without RI enforcement (no foreign key constraints)

```
MariaDB [test4]> select * from accounts;
+-----------+--------------+------------------+
| accountid | name         | email            |
+-----------+--------------+------------------+
| A-101-11  | Linda Lu     | llu@gmail.com    |
| A-112-21  | Felix Franks | ffmeow@gmail.com |
| B-212-03  | Ronald Jones | rj2013@gmail.com |
+-----------+--------------+------------------+
3 rows in set (0.00 sec)

MariaDB [test4]> select * from products;
+--------------+----------------------------+-------+
| productid    | description                | price |
+--------------+----------------------------+-------+
| ISBN-2012193 | The Wealth of Nations      |    24 |
| ISBN-3038277 | The Communist Manifesto    |    22 |
| ISBN-4818121 | Getting Ahead in the Market |    28 |
+--------------+----------------------------+-------+
3 rows in set (0.00 sec)

MariaDB [test4]> select * from orders;
+---------+--------------+------------------+
| orderid | productid    | ordering_account |
+---------+--------------+------------------+
|       1 | ISBN-2012193 | A-112-21         |
+---------+--------------+------------------+
1 row in set (0.00 sec)

MariaDB [test4]> insert into orders (productid,ordering_account)
    -> values('PART-3014','C-111-38');
Query OK, 1 row affected (0.00 sec)

MariaDB [test4]> select * from orders;
+---------+--------------+------------------+
| orderid | productid    | ordering_account |
+---------+--------------+------------------+
|       1 | ISBN-2012193 | A-112-21         |
|       2 | PART-3014    | C-111-38         |
+---------+--------------+------------------+
2 rows in set (0.00 sec)
```

RI Enforcement

- Add foreign key designators
  - DB can now identify RI violations
  - Inserts fail on violation

```
MariaDB [test4]> delete from orders where orderid = 2;
Query OK, 1 row affected (0.00 sec)

MariaDB [test4]> alter table orders add foreign key (productid)
    -> references products(productid);
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

MariaDB [test4]> alter table orders add foreign key (ordering_account)
    -> references accounts(accountid);
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

MariaDB [test4]> describe orders;
+-----------------+-------------+------+-----+---------+----------------+
| Field           | Type        | Null | Key | Default | Extra          |
+-----------------+-------------+------+-----+---------+----------------+
| orderid         | int(11)     | NO   | PRI | NULL    | auto_increment |
| productid       | varchar(255)| NO   | MUL | NULL    |                |
| ordering_account| varchar(255)| NO   | MUL | NULL    |                |
+-----------------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

MariaDB [test4]> insert into orders (productid,ordering_account)
    -> values('PART-3014','C-111-38');
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
(`test4`.`orders`, CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`productid`)
REFERENCES `products` (`productid`))
```

Duke

RI Enforcement

- Designating foreign keys also affects delete/update elsewhere
  - RI Enforcement may block operations (mySQL)
  - RI Enforcement may propagate changes across referring tables
  - Depends on RDBMS, config

```
MariaDB [test4]> insert into orders(productid,ordering_account)
    -> values('ISBN-4818121','B-212-03');
Query OK, 1 row affected (0.00 sec)

MariaDB [test4]> select * from orders;
+---------+--------------+------------------+
| orderid | productid    | ordering_account |
+---------+--------------+------------------+
|       1 | ISBN-2012193 | A-112-21         |
|       4 | ISBN-4818121 | B-212-03         |
+---------+--------------+------------------+
2 rows in set (0.00 sec)

MariaDB [test4]> update products set productid = 'ISBN-4818131'
    -> where productid='ISBN-4818121';
ERROR 1451 (23000): Cannot delete or update a parent row: a
foreign key constraint fails (`test4`.`orders`,
CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`productid`)
REFERENCES `products` (`productid`))
```

Duke

RI for Student COVID-19

- Designating “userid" a foreign key, referencing student table
  - Ensures RI enforcement in other tables
  - No test can refer to a non-existent student
  - No student can be removed with an outstanding test, room assignment, etc.

```
MariaDB [normalized]> describe student;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| userid    | int(11)      | NO   | PRI | NULL    |       |
| firstname | varchar(255) | NO   |     | NULL    |       |
| lastname  | varchar(255) | NO   |     | NULL    |       |
| class     | varchar(255) | NO   |     | 2024    |       |
+-----------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

MariaDB [normalized]> describe majors;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| declarationid | int(11)      | NO   | PRI | NULL    | auto_increment |
| userid       | int(11)      | NO   | MUL | NULL    |                |
| major        | varchar(255) | YES  |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

MariaDB [normalized]> describe room_assignment;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| assignmentid | int(11)      | NO   | PRI | NULL    | auto_increment |
| userid       | int(11)      | NO   | MUL | NULL    |                |
| dorm         | varchar(255) | YES  |     | NULL    |                |
| room_number  | int(11)      | YES  |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

MariaDB [normalized]> describe tests;
+--------------+-----------+------+-----+---------+----------------+
| Field        | Type      | Null | Key | Default | Extra          |
+--------------+-----------+------+-----+---------+----------------+
| testid       | int(11)   | NO   | PRI | NULL    | auto_increment |
| userid       | int(11)   | NO   | MUL | NULL    |                |
| test_date    | date      | YES  |     | NULL    |                |
| test_returned | date      | YES  |     | NULL    |                |
| test_result  | tinyint(1) | YES  |     | NULL    |                |
+--------------+-----------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

MariaDB [normalized]>
```

Transactions

Duke

Transactions

- Group SQL commands together for execution
- Transaction is **atomic** - all succeeds or all fails
- **COMMIT** attempts to execute entire transaction
- **ROLLBACK** returns state to pre-transaction state
- MySQL defaults to “autocommit” - some DBMS default to transactions (explicit “COMMIT")

Duke

Law Office Example

- Simple Law Office billable hours database
- billable_hours has two foreign keys (case_id and assoc_id)
- cases has two foreign keys (client and associate)
- client and associate IDs are assigned automatically (auto_increment)

```
MariaDB [test5]> select * from clients;
+-----------+-----------------+-----------------+
| client_id | name            | payment_account |
+-----------+-----------------+-----------------+
|         1 | Roger Zelazny   | 10151-11-20188  |
|         2 | Dan Simmons     | 20810-21-19384  |
|         3 | Terry Pratchett | 11111-22-91842  |
+-----------+-----------------+-----------------+
3 rows in set (0.00 sec)

MariaDB [test5]> select * from associates;
+----------+--------------+-------------+
| assoc_id | name         | hourly_rate |
+----------+--------------+-------------+
|        1 | John Dewey   |         250 |
|        2 | Eli Cheatham |         280 |
|        3 | Reginald Howe |         390 |
+----------+--------------+-------------+
3 rows in set (0.00 sec)

MariaDB [test5]> select * from cases;
+---------+-----------------------------------------------+--------+-----------+
| case_id | name                                          | client | associate |
+---------+-----------------------------------------------+--------+-----------+
|   10100 | Prince Corwin v. rebma.gov                    |      1 |         1 |
|   12013 | United Federation v. Rollins Group, LLC       |      2 |         3 |
|   91108 | Rincewind Wizard v. Unseen University, Inc.   |      3 |         2 |
+---------+-----------------------------------------------+--------+-----------+
3 rows in set (0.00 sec)

MariaDB [test5]> describe billable_hours;
+-----------+---------+------+-----+---------+----------------+
| Field     | Type    | Null | Key | Default | Extra          |
+-----------+---------+------+-----+---------+----------------+
| work_unit | int(11) | NO   | PRI | NULL    | auto_increment |
| case_id   | int(11) | YES  | MUL | NULL    |                |
| assoc_id  | int(11) | YES  | MUL | NULL    |                |
| hours     | int(11) | YES  |     | NULL    |                |
+-----------+---------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Duke

Law Office Example

- Imagine that Howe takes a new case from a new client and wants to enter initial (minimal) 4 billable hours into the database.
- This is tricky because of foreign key constraints — multiple updates need to **all** occur together or not at all.

```
MariaDB [test5]> select * from clients;
+-----------+-----------------+-----------------+
| client_id | name            | payment_account |
+-----------+-----------------+-----------------+
|         1 | Roger Zelazny   | 10151-11-20188  |
|         2 | Dan Simmons     | 20810-21-19384  |
|         3 | Terry Pratchett | 11111-22-91842  |
+-----------+-----------------+-----------------+
3 rows in set (0.00 sec)

MariaDB [test5]> select * from associates;
+----------+--------------+-------------+
| assoc_id | name         | hourly_rate |
+----------+--------------+-------------+
|        1 | John Dewey   |         250 |
|        2 | Eli Cheatham |         280 |
|        3 | Reginald Howe |         390 |
+----------+--------------+-------------+
3 rows in set (0.00 sec)

MariaDB [test5]> select * from cases;
+---------+-----------------------------------------------+--------+-----------+
| case_id | name                                          | client | associate |
+---------+-----------------------------------------------+--------+-----------+
|   10100 | Prince Corwin v. rebma.gov                    |      1 |         1 |
|   12013 | United Federation v. Rollins Group, LLC       |      2 |         3 |
|   91108 | Rincewind Wizard v. Unseen University, Inc.   |      3 |         2 |
+---------+-----------------------------------------------+--------+-----------+
3 rows in set (0.00 sec)

MariaDB [test5]> describe billable_hours;
+-----------+---------+------+-----+---------+----------------+
| Field     | Type    | Null | Key | Default | Extra          |
+-----------+---------+------+-----+---------+----------------+
| work_unit | int(11) | NO   | PRI | NULL    | auto_increment |
| case_id   | int(11) | YES  | MUL | NULL    |                |
| assoc_id  | int(11) | YES  | MUL | NULL    |                |
| hours     | int(11) | YES  |     | NULL    |                |
+-----------+---------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Duke

Law Office Example

```
MariaDB [test5]> start transaction;
Query OK, 0 rows affected (0.00 sec)

MariaDB [test5]> insert into clients (name,payment_account) values ('Neil Gaiman','66600-66-00666');
Query OK, 1 row affected (0.00 sec)

MariaDB [test5]> insert into cases (case_id,name,client,associate) values
  -> (‘60606','Crowley v. Anathema Device',(SELECT client_id from clients where clients.name = 'Neil Gaiman'),3);
Query OK, 1 row affected (0.00 sec)

MariaDB [test5]> insert into billable_hours (case_id,assoc_id,hours) values ('60606',3,4);
Query OK, 1 row affected (0.00 sec)

MariaDB [test5]> COMMIT;
```

Duke

Rollback

- Cheatham starts a transaction, then thinks better of it
- ROLLBACK discards transaction
- Howe's transaction applies, Cheathams does not

```
MariaDB [test5]> start transaction;
Query OK, 0 rows affected (0.00 sec)

MariaDB [test5]> insert into clients (name,payment_account) values ('Douglas Adams','40401-22-21324');
Query OK, 1 row affected (0.00 sec)

MariaDB [test5]> insert into cases (case_id, name, client, associate) values
  -> ('41101','Dent v. Vogun Empire',(SELECT client_id from clients where clients.name = 'Doug Adams'),2);
Query OK, 1 row affected (0.00 sec)

MariaDB [test5]> insert into billable_hours (case_id,assoc_id,hours) values ('41101',2,4);
Query OK, 1 row affected (0.00 sec)

MariaDB [test5]> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

MariaDB [test5]> select * from billable_hours;
+-----------+---------+----------+-------+
| work_unit | case_id | assoc_id | hours |
+-----------+---------+----------+-------+
|         1 |   60606 |        3 |     4 |
+-----------+---------+----------+-------+
1 row in set (0.00 sec)

MariaDB [test5]> select * from clients;
+-----------+-----------------+------------------+
| client_id | name            | payment_account  |
+-----------+-----------------+------------------+
|         1 | Roger Zelazny   | 10151-11-20188   |
|         2 | Dan Simmons     | 20810-21-19384   |
|         3 | Terry Pratchett | 11111-22-91842   |
|         4 | Neil Gaiman     | 66600-66-00666   |
+-----------+-----------------+------------------+
4 rows in set (0.00 sec)

MariaDB [test5]> select * from cases;
+---------+-----------------------------------------+--------+-----------+
| case_id | name                                    | client | associate |
+---------+-----------------------------------------+--------+-----------+
|   10100 | Prince Corwin v. rebma.gov              |      1 |         1 |
|   12013 | United Federation v. Rollins Group, LLC |      2 |         3 |
|   60606 | Crowley v. Anathema Device              |      4 |         3 |
|   91108 | Rincewind Wizard v. Unseen University, Inc. |      3 |         2 |
+---------+-----------------------------------------+--------+-----------+
4 rows in set (0.00 sec)
```

Advanced Features/ Refresher

Duke

Advanced Feature:

SQL Subqueries

Duke

Very Brief Overview

- Subqueries turn query results into table equivalents or lists
- Used in SELECT statements, often rely on “IN”:
  - SELECT * FROM student WHERE student.userid IN (SELECT DISTINCT userid FROM tests WHERE test_result = TRUE);
- May be used with FROM (requires aliasing derived table)
  - SELECT userid,firstname FROM (select * from student) AS s1
- Also for INSERT, etc.
  - INSERT (a,b) VALUES (x,(SELECT y FROM c WHERE i = 1))

Duke

Advanced Feature:

JOIN

Duke

Very Brief Overview

- JOINs allow combining data from multiple tables based on some correlating field or fields
- Different scopes
  - Inner - just the intersection of tables (Venn overlap)
  - Outer - “for every X, add matching Y (or NULLs)
    - X left join Y = "for every X…”, X right join Y = "for every Y…”
  - Cross - cartesian product of tables (usually wrong)
- Result can be aliased, used as a virtual table, **joined again**

Duke

Very Simple Examples

- Inner join returns 2 rows (1 for each order with associated account info)
- Left outer join returns 3 rows (one for each account, with order(s) attached)

```
MariaDB [test4]> select * from accounts;
+-----------+--------------+------------------+
| accountid | name         | email            |
+-----------+--------------+------------------+
| A-101-11  | Linda Lu     | llu@gmail.com    |
| A-112-21  | Felix Franks | ffmeow@gmail.com |
| B-212-03  | Ronald Jones | rj2013@gmail.com |
+-----------+--------------+------------------+
3 rows in set (0.00 sec)

MariaDB [test4]> select * from orders;
+---------+-------------+-----------------+
| orderid | productid   | ordering_account |
+---------+-------------+-----------------+
|       1 | ISBN-2012193 | A-112-21        |
|       4 | ISBN-4818121 | B-212-03        |
+---------+-------------+-----------------+
2 rows in set (0.00 sec)

MariaDB [test4]> select * from products;
+-------------+---------------------------+-------+
| productid   | description               | price |
+-------------+---------------------------+-------+
| ISBN-2012193 | The Wealth of Nations     |    24 |
| ISBN-3038277 | The Communist Manifesto   |    22 |
| ISBN-4818121 | Getting Ahead in the Market |    28 |
+-------------+---------------------------+-------+
3 rows in set (0.00 sec)

MariaDB [test4]> select * from accounts as a inner join orders as o on a.accountid = o.ordering_account;
+-----------+--------------+------------------+---------+-------------+------------------+
| accountid | name         | email            | orderid | productid   | ordering_account |
+-----------+--------------+------------------+---------+-------------+------------------+
| A-112-21  | Felix Franks | ffmeow@gmail.com |       1 | ISBN-2012193 | A-112-21         |
| B-212-03  | Ronald Jones | rj2013@gmail.com |       4 | ISBN-4818121 | B-212-03         |
+-----------+--------------+------------------+---------+-------------+------------------+
2 rows in set (0.00 sec)

MariaDB [test4]> select * from accounts as a left join orders as o on a.accountid = o.ordering_account;
+-----------+--------------+------------------+---------+-------------+------------------+
| accountid | name         | email            | orderid | productid   | ordering_account |
+-----------+--------------+------------------+---------+-------------+------------------+
| A-112-21  | Felix Franks | ffmeow@gmail.com |       1 | ISBN-2012193 | A-112-21         |
| B-212-03  | Ronald Jones | rj2013@gmail.com |       4 | ISBN-4818121 | B-212-03         |
| A-101-11  | Linda Lu     | llu@gmail.com    |    NULL | NULL        | NULL             |
+-----------+--------------+------------------+---------+-------------+------------------+
3 rows in set (0.00 sec)
```

Duke

Advanced Feature: Views

Duke

Views

- View =~ result of a stored query, presented like a table
- Rather than store the result, the view stores the query
- Can be used to hide query complexity (extensive joins, complex subqueries)
- Can provide limited ”query-level” security - view can be separately authorized from other tables
- Some RDBMS (Oracle, Postgresql) support **materialized** views, which are more like snapshots, updated periodically

Duke

A Simple View: Psych Majors

```
MariaDB [normalized]> create view psychology_majors as
    -> select student.*,majors.major from
    -> student inner join majors on
    -> student.userid = majors.userid where
    -> majors.major = 'Psychology';
Query OK, 0 rows affected (0.00 sec)

MariaDB [normalized]> select count(*) from psychology_majors;
+----------+
| count(*) |
+----------+
|     2159 |
+----------+
1 row in set (0.01 sec)

MariaDB [normalized]> select * from psychology_majors limit 10;
+---------+-----------+----------+-------+------------+
| userid  | firstname | lastname | class | major      |
+---------+-----------+----------+-------+------------+
| 4201202 | Yuri      | Crossey  | 2020  | Psychology |
| 4201204 | Evangaline | Spiritos | 2022 | Psychology |
| 4201214 | Niesha    | Spathis  | 2022  | Psychology |
| 4201217 | Emmett    | Gorbunova | 2022 | Psychology |
| 4201233 | Gabe      | Enloe    | 2022  | Psychology |
| 4201238 | Archana   | Lalo     | 2020  | Psychology |
| 4201249 | Regan     | Guth     | 2023  | Psychology |
| 4201268 | Tamakia   | Ruis     | 2021  | Psychology |
| 4201288 | Pahvie    | Blanco   | 2023  | Psychology |
| 4201290 | Boluwatife | Durand  | 2023  | Psychology |
+---------+-----------+----------+-------+------------+
10 rows in set (0.00 sec)
```

Duke

More Complex View: Junior Psych Majors In Single Rooms

- Combine student table cols with concatenated room identifier (dorm+room)
- from inner join between student, room_assignment, and majors tables
- where major is Psychology and (depending on current month) class is this or next year
- and use GROUP BY room identifier HAVING count(*) = 1 (in rooms with only one student)
- Spot my logic error? :-)

```
MariaDB [normalized]> CREATE VIEW junior_psych_in_singles AS
    -> SELECT student.*,CONCAT(room_assignment.dorm," ",room_assignment.room_number)
    -> AS room from ((student INNER JOIN room_assignment ON student.userid =
    -> room_assignment.userid) INNER JOIN majors ON student.userid = majors.userid)
    -> WHERE majors.major = 'Psychology' AND (((MONTH(CURDATE()) >= 5) AND
    -> student.class = (YEAR(CURDATE()) + 1)) OR ((MONTH(CURDATE()) < 5) AND
    -> student.class = (YEAR(CURDATE())))) GROUP BY room HAVING count(*) = 1;
Query OK, 0 rows affected (0.01 sec)

MariaDB [normalized]> select count(*) from junior_psych_in_singles;
+----------+
| count(*) |
+----------+
|      270 |
+----------+
1 row in set (0.02 sec)

MariaDB [normalized]> select * from junior_psych_in_singles limit 10;
+---------+-----------+--------------+-------+---------------+
| userid  | firstname | lastname     | class | room          |
+---------+-----------+--------------+-------+---------------+
| 4204750 | Zean      | Voegele      | 2021  | Alspaugh 1016 |
| 4205909 | Vani      | Zucco        | 2021  | Alspaugh 1032 |
| 4208293 | Dionta    | Loksztejn    | 2021  | Alspaugh 1143 |
| 4212334 | Marki     | Blanks       | 2021  | Alspaugh 1190 |
| 4204140 | Darasjot  | Apokremiotis | 2021  | Alspaugh 1206 |
| 4215609 | Rickey    | Marner       | 2021  | Alspaugh 122  |
| 4213988 | Heng      | Steimel      | 2021  | Alspaugh 1250 |
| 4204869 | Jamiyla   | Maule        | 2021  | Alspaugh 1322 |
| 4204790 | Lesli     | Duphiney     | 2021  | Alspaugh 1331 |
| 4204158 | Janghoon  | Sam          | 2021  | Alspaugh 1348 |
+---------+-----------+--------------+-------+---------------+
10 rows in set (0.01 sec)
```

Duke

Advanced Feature: Triggers, Stored Procedures

Duke

Triggers

- Triggers are “tripwires" that execute DB operations
- May be bound to rows or statements
  - row level triggers fire once for every row in an event
  - statement level triggers fire once for entire statement
- Typically only attached to INSERT,UPDATE,DELETE
- May be set to run before or after event

Duke

Triggers

- Triggers can be useful for
  - executing business logic
  - performing validation (ala "check" constraints)
  - auditing, etc.
- Can harm performance
- May be **extremely** opaque and RDBMS-specific

Duke

Stored Procedures

- Sequences of DB operations stored as DB functions
- Stored Procedures can be called directly or used in triggers
- “Good" for “hiding" business logic in a DB, minimizing network traffic
- Can get exorbitantly complex (loops, if/then), expensive
- Depend sensitively on RDBMS implementation - NOT PORTABLE

Duke

Example

- Two tables “entries" has a list of numbers and “running_total" has timestamped totals of the “entries"
- update_running(int) procedure adds a row to running_total with current date and new total of entries in entries table
- after_entries_insert trigger calls update_running(inserted value) for every row inserted in entries table

```
MariaDB [test6]> select * from entries;
+--------+
| number |
+--------+
|     25 |
+--------+
1 row in set (0.00 sec)

MariaDB [test6]> select * from running_total;
+------------+-------+
| add_time   | total |
+------------+-------+
| 2020-09-01 |    25 |
+------------+-------+
1 row in set (0.00 sec)

MariaDB [test6]> delimiter ##
MariaDB [test6]> create procedure update_running (IN count int(11))
    -> begin
    ->   select max(total) into @oldmax from running_total;
    ->   insert into running_total values(CURRENT_DATE(),@oldmax + count);
    -> end##
Query OK, 0 rows affected (0.00 sec)

MariaDB [test6]> create trigger after_entries_insert
    -> after insert
    -> on entries for each row
    -> begin
    ->   call update_running(NEW.number);
    -> end##
Query OK, 0 rows affected (0.00 sec)

MariaDB [test6]> delimiter ;
MariaDB [test6]> insert into entries values(38);
Query OK, 1 row affected (0.00 sec)

MariaDB [test6]> insert into entries values(21);
Query OK, 1 row affected (0.00 sec)

MariaDB [test6]> select * from entries;
+--------+
| number |
+--------+
|     25 |
|     38 |
|     21 |
+--------+
3 rows in set (0.00 sec)

MariaDB [test6]> select * from running_total;
+------------+-------+
| add_time   | total |
+------------+-------+
| 2020-09-01 |    25 |
| 2020-09-07 |    63 |
| 2020-09-07 |    84 |
+------------+-------+
3 rows in set (0.00 sec)
```

SP + Trigger Advantages

- Trigger is “set-and-forget” - set once and run as needed
- SP enshrines business logic in the DB itself; no reliance on user memory or consistency
- Great for managing audit trails, guaranteeing consistency of data
- Stored procedure can be changed over time as business logic changes

Duke

SP and Trigger Disadvantages

- Portability (there often is none) between RDBMS
- Logic is “hidden” — "spooky action at a distance”
- Triggers run synchronously -- complex trigger can drag down performance
- Where feasible, use other options (CHECK constraints, write-thru views); otherwise, DOCUMENT!

Duke

Other Useful Dangers

Duke

Neat Features (non-portable)

- Special “LIKE" syntax
  - Most RDBMS support globbing; some support regex’s
    - Watch out for performance impacts!
- Full-text indexing plugins
  - Solr, Elastic Search
    - Super fast, super-dependent on specific versions - Watch out for "vendor lock-in”

Duke

Advice about Portability

- Prefer using “ANSI Standard” SQL where possible
- Consider trade-off between external code and using non-portable SQL statements
- For one-offs and cases where portability is irrelevant, feel more free to use extensions
- To paraphrase Larry Wall (co-author of Perl):

Duke

A SQL solution is correct if it gets the job done (properly) before your boss fires you.

— apologies to Larry Wall

Duke

Performance

Duke

The Good News

- Modern RDMBS engines are **staggeringly** fast at most things
- RDBMS comprises two main parts:
  - Storage Engine: Moves data in and out of RAM, manages || transfers, maintains indexing, etc.
  - Query Processor: Executes queries after first optimization and analysis
- SQL optimization is a very well-studied science with 30 years of development behind it

Duke

The Good News

- Typical query sequence:
  - Parse SQL statement(s)
  - Bind to data objects (tables, columns)
  - Formulate execution plan(s)
  - Estimate cardinality (row-count), cost(time), cost(memory)
  - Pass “best fit” plan to execution engine

Duke

The Bad News

- SQL query optimization can't do everything
- There are usually multiple ways to structure a query, not all of equal cost/optimizability
- Some simple errors in SQL statement construction can not only destroy performance but also produce errant results
- Some really extreme cases can lead to calls from your DBAs

Duke

Performance Tips

- Avoid a few common mistakes:
  - Always use explicit joins, and explicitly bound them (inner, left, right). Avoid accidental “cross join” (BE CAREFUL).
  - Be aware of when parentheses are your friends (or are required).
    - A and B or C — did you instead mean A and (B or C)
  - Use indexes (**); avoid using leading wildcards in LIKE matches
  - Avoid unnecessarily complicated queries - simple optimizes better
  - Think carefully when you start to use "distinct" or “unique"

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

Thanks for your time and attention!

MIDS students: we'll revisit this material in next week's lab session and in more depth in Mary Claire's sections coming up next!

Duke