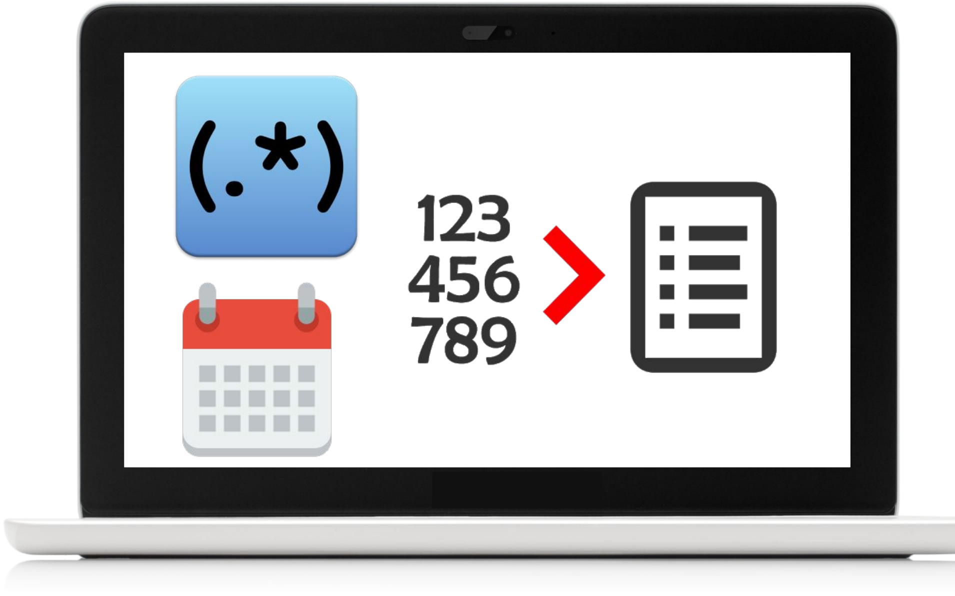
DCL, TCL, Functions

Please go through material for these topics. Complete the reading, exercises, and any videos linked. If the instructions ask to turn in any exercises, please do so through slack to your instructor.



Cognixia™

Outline

1. Data Control Language (DCL)
   - Users
   - Grant vs Revoke
2. Transactional Control Language (TCL)
   - Commit & Rollback
   - Savepoint
3. Built-In Functions
   - String Functions
   - Number Functions
   - Date Functions
4. Open Book Quiz

ddl.sql dml.sql dcl.sql

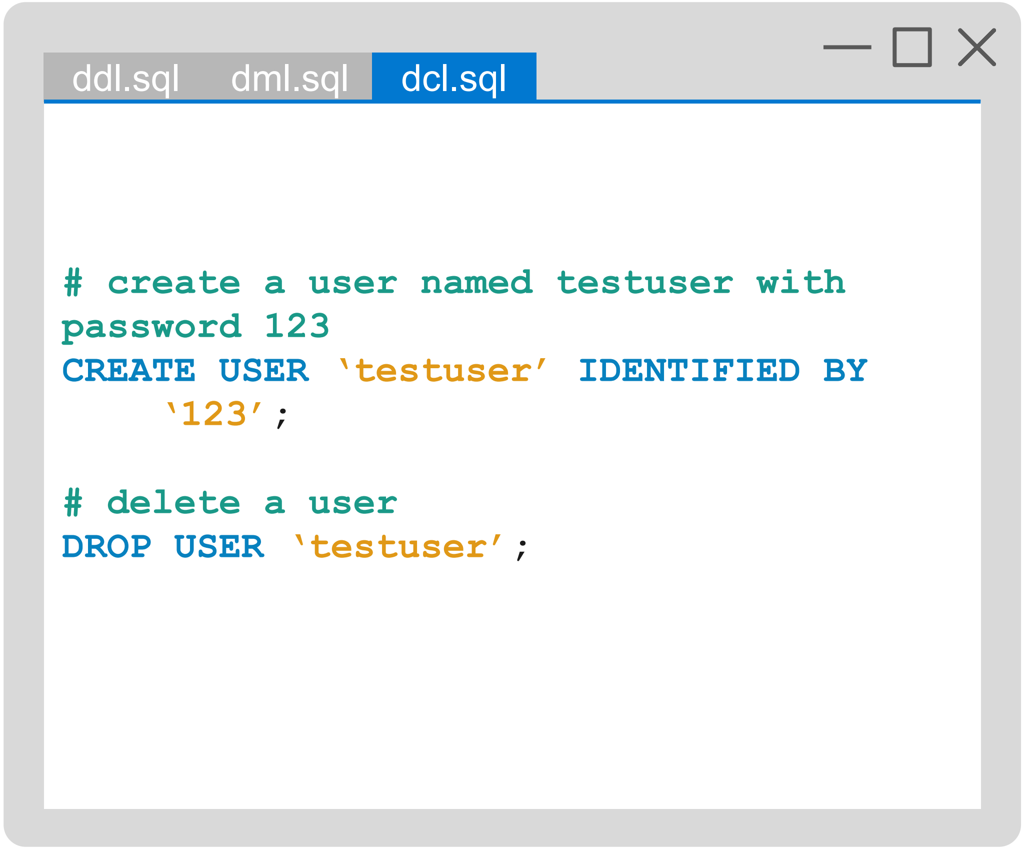
Data Control Language

DCL is used to create roles, permissions, referential integrity, and control access to a database by securing it.

→ **GRANT**
→ **REVOKE**

USER

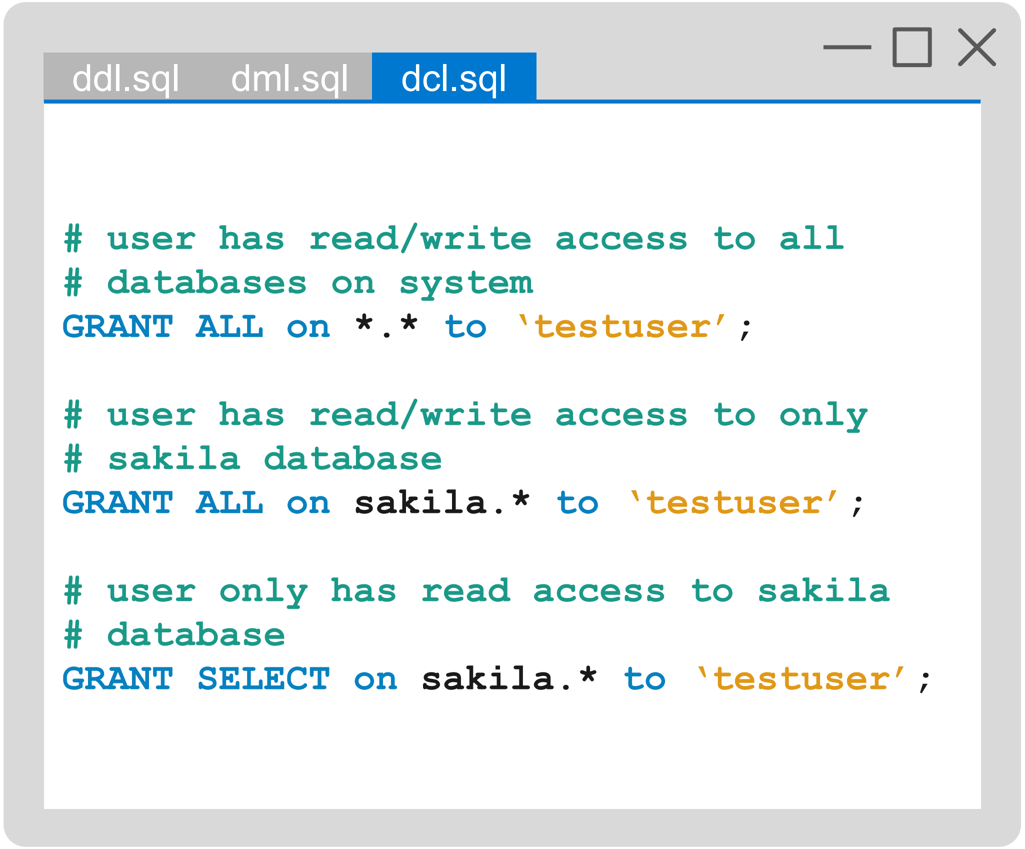
- ➔ Different users can have access to your database
- ➔ With DCL you can specify what rights these users have to read/write to the database



```
# create a user named testuser with
password 123
CREATE USER 'testuser' IDENTIFIED BY
     '123';

# delete a user
DROP USER 'testuser';
```

GRANT

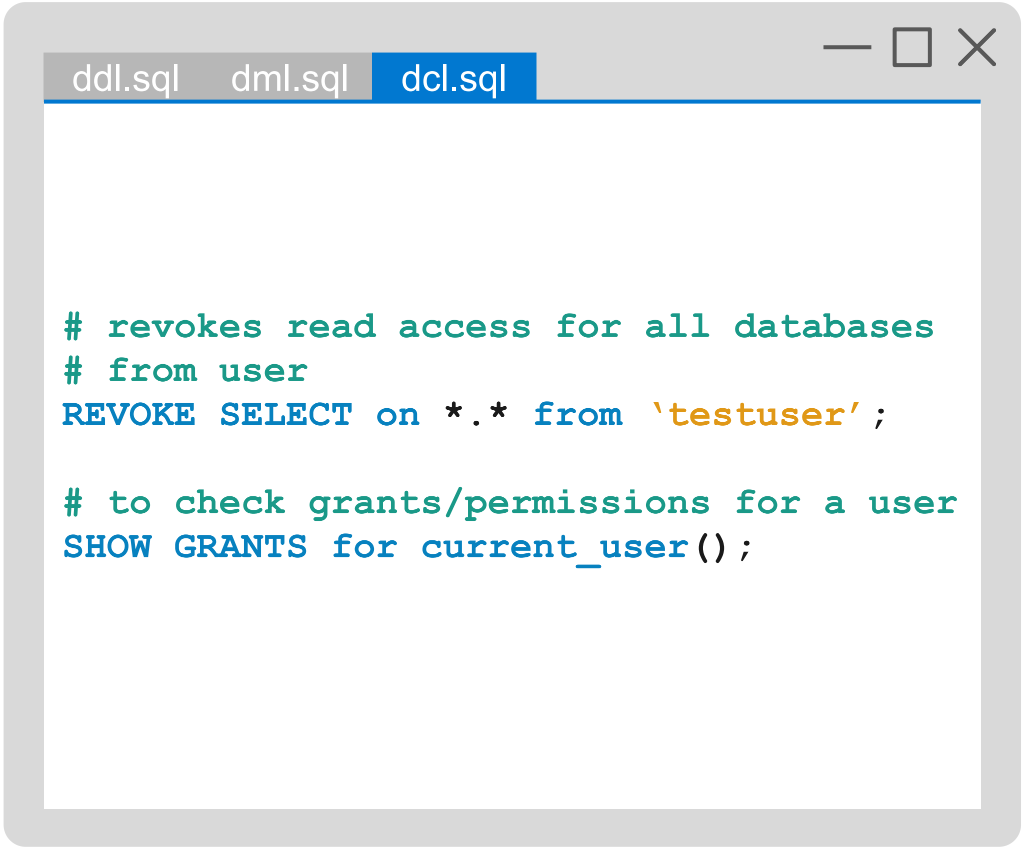
- ➔ GRANT is used to give permission to users on a database
- ➔ Can be given permission to:
  - ◆ **Read** - access to data
  - ◆ **Write** - manipulate the data


ddl.sql dml.sql dcl.sql

```
# user has read/write access to all
# databases on system
GRANT ALL on *.* to 'testuser';

# user has read/write access to only
# sakila database
GRANT ALL on sakila.* to 'testuser';

# user only has read access to sakila
# database
GRANT SELECT on sakila.* to 'testuser';
```

REVOKE

- ➔ REVOKE is used to remove permission from users on a database
- ➔ Can check grants for user with **SHOW GRANTS**

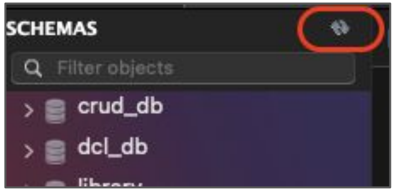


```
# revokes read access for all databases
# from user
REVOKE SELECT on *.* from 'testuser';

# to check grants/permissions for a user
SHOW GRANTS for current_user();
```

Running DCL on Workbench

Follow instructions to create a user and set permissions for that user through SQL Workbench.

- ➔ Open workbench and run the following command to create a database called ***dcl_db***: `CREATE DATABASE dcl_db;`
- ➔ Click on refresh icon at the top left to make sure the database is created
- ➔ Run the following command to create a user named user1: `CREATE USER 'user1' IDENTIFIED BY '123';`



Running DCL on Workbench Cont.

➔ Run the following queries to use our new database and create a simple table:

```
USE dcl_db;
CREATE TABLE temp (
    num INT,
    letter CHAR(1)
);
```

➔ Then run these two insert statements to put data in this table

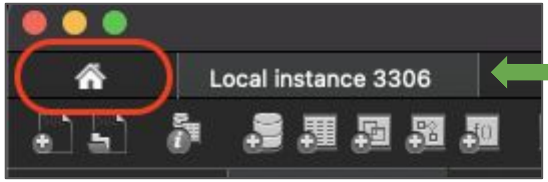
```
INSERT INTO temp VALUES(1, 'A');
INSERT INTO temp VALUES(2, 'B');
```

Running DCL on Workbench Cont.

➔ Give ***user1*** read permissions to the ***dcl_db*** database:

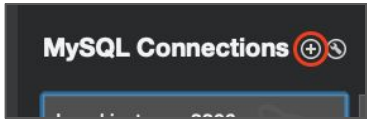
```
GRANT SELECT on dcl_db.* to 'user1';
```

➔ Click on the home tab (top left corner), as highlighted in red in the picture below, so we can create another connection, ***DO NOT CLOSE YOUR CURRENT CONNECTION***



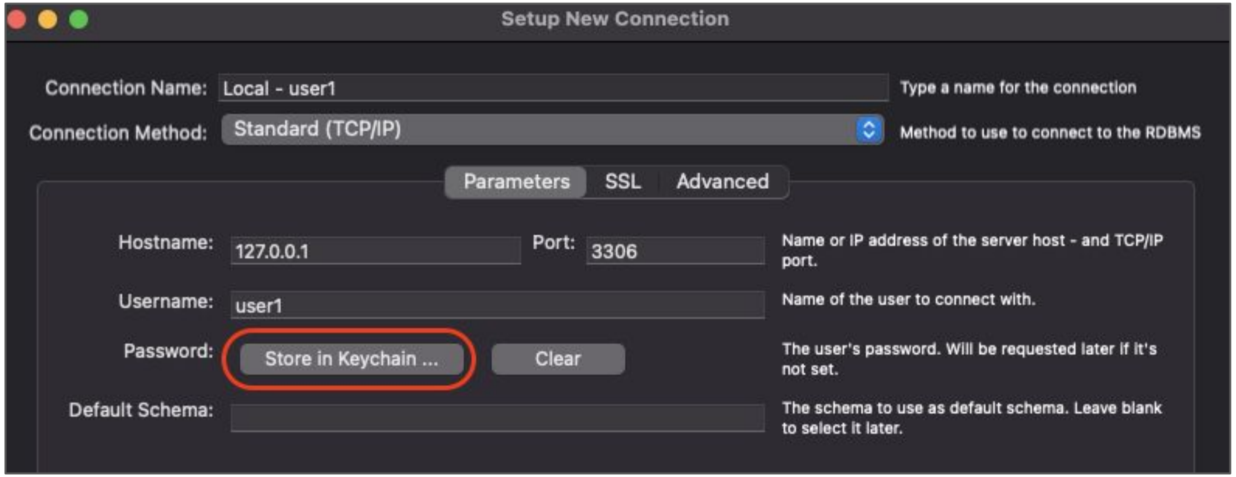
Leave this tab open

➔ Click on the plus icon to create a new connection



Running DCL on Workbench Cont.

➔ Name the connection ***Local - user1*** and a Username of ***user1***, then click on the **Store in Keychain** button


Setup New Connection

Connection Name: Local - user1 — Type a name for the connection

Connection Method: Standard (TCP/IP) — Method to use to connect to the RDBMS

Parameters | SSL | Advanced

Hostname: 127.0.0.1 Port: 3306 — Name or IP address of the server host - and TCP/IP port.

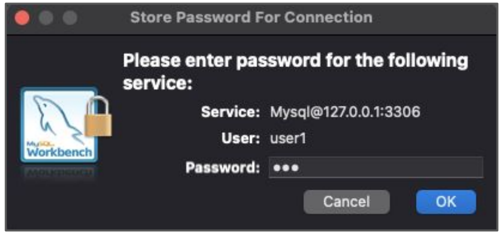
Username: user1 — Name of the user to connect with.

Password: Store in Keychain ... Clear — The user's password. Will be requested later if it's not set.

Default Schema: — The schema to use as default schema. Leave blank to select it later.

Running DCL on Workbench Cont.

➔ Enter the password for the user, which is ***123***, then click **OK**, then click **OK** again to save the connection


Store Password For Connection

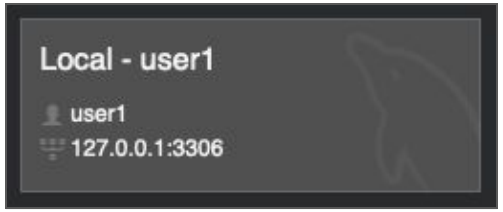
Please enter password for the following service:

Service: Mysql@127.0.0.1:3306

User: user1

Password: •••

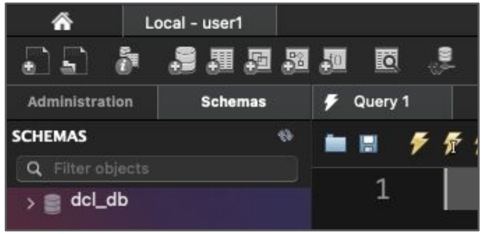
➔ New Connection should be created, click on it to login


Local - user1

user1

127.0.0.1:3306

➔ When the connection opens, you'll see that this user only has access to the ***dcl_db***



Running DCL on Workbench Cont.

- ➔ Try selecting data from the temp table, run the following commands

```
USE dcl_db;
SELECT * FROM temp;
```

- ➔ Try to update a value in the table, it will not let you since this user only has read permissions to this database

```
UPDATE temp SET letter = 'C' WHERE num = 1;
```

- ➔ You can close out of this connection now if you wish ddl.sql | dml.sql | dcl.sql | tcl.sql

Transactional Control Language

TCL is used to control transactions made to the database by executing DML statements.

→ **COMMIT**
→ **SAVEPOINT**
→ **ROLLBACK**

Transactions & TCL Video

- ➔ Please review the following video on enums: https://youtu.be/9K3OaldcmH4
- ➔ Feel free to review attached code used in video:
  - ◆ `3-TCL.sql`
- ➔ Following slides in this section can be used as additional reference

First Transaction

```
INSERT INTO chef(chef_id, name, best_dish, rest_id)
    VALUES(null, 'Chef Ramsey', 'Lamb', 1);

INSERT INTO chef(chef_id, name, best_dish, rest_id)
    VALUES(null, 'Chef Remy, 'Ratatouille', 1);

INSERT INTO chef(chef_id, name, best_dish, rest_id)
    VALUES(null, 'Swedish Chef', 'Swedish Fööd', 2);
```

*** COMMIT TRANSACTION ***

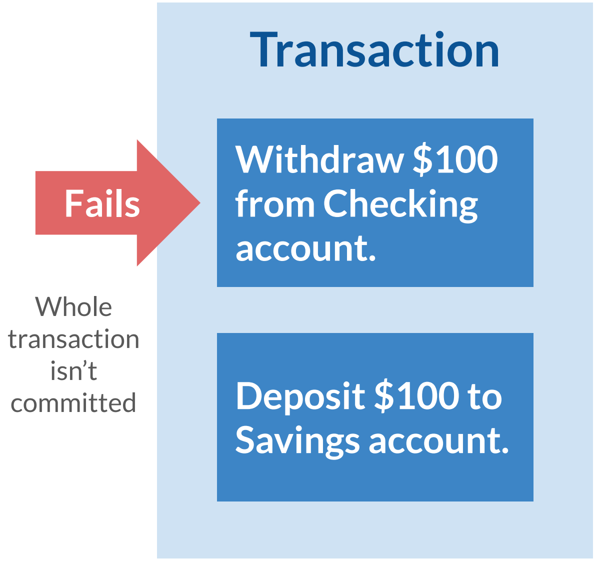
Second Transaction

```
UPDATE chef SET best_dish = 'pizza' WHERE rest_id = 1;

DELETE FROM chef WHERE best_dish = 'steak';
```

*** COMMIT TRANSACTION ***

ACID Properties

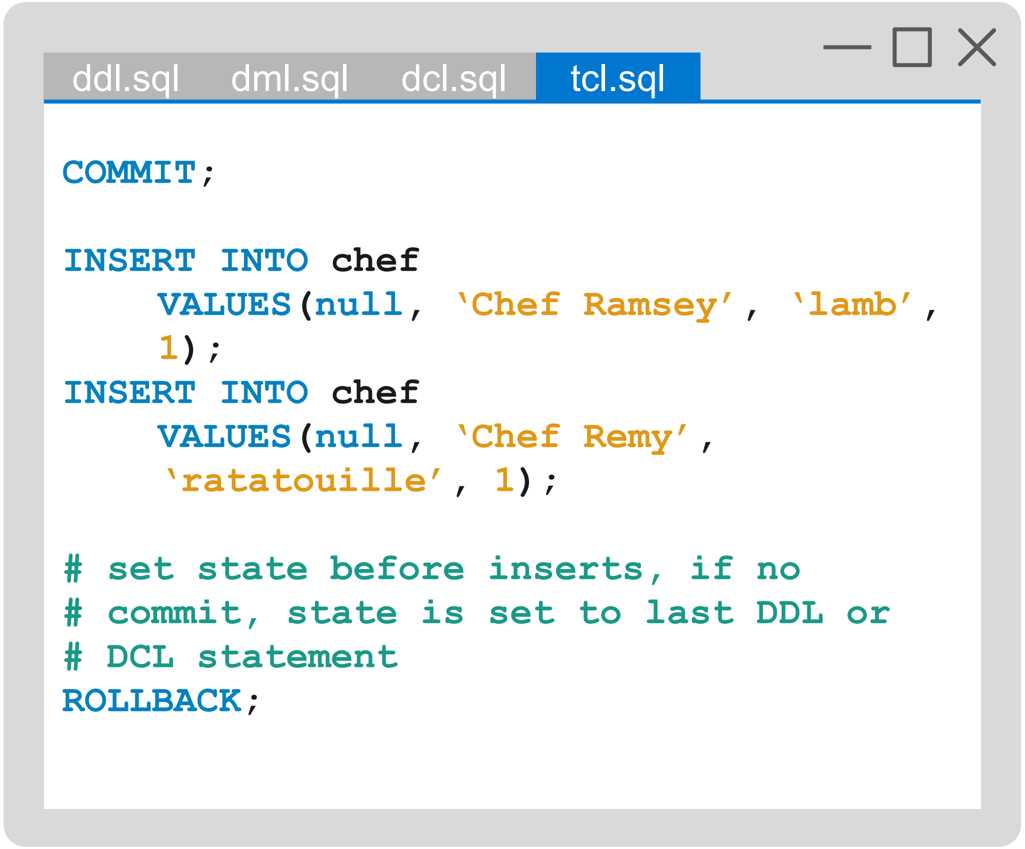
1. **Atomicity** - All statements in a transaction go through or none of them happen.
2. **Consistency** - Transactions are permanent once committed. If transaction fails, database remains in state before transaction attempted.
3. **Isolation** - Each transaction is independent and won't interfere with another transaction.
4. **Durability** - System failures or restarts don't affect committed transactions, committed transactions are never "lost".


Transaction
Fails
Withdraw $100 from Checking account.
Whole transaction isn't committed
Deposit $100 to Savings account.

Transaction - Made up of one or more SQL statements (DML ones). Can be committed manually to database or commit whenever a DDL or DCL command is run.

COMMIT and ROLLBACK

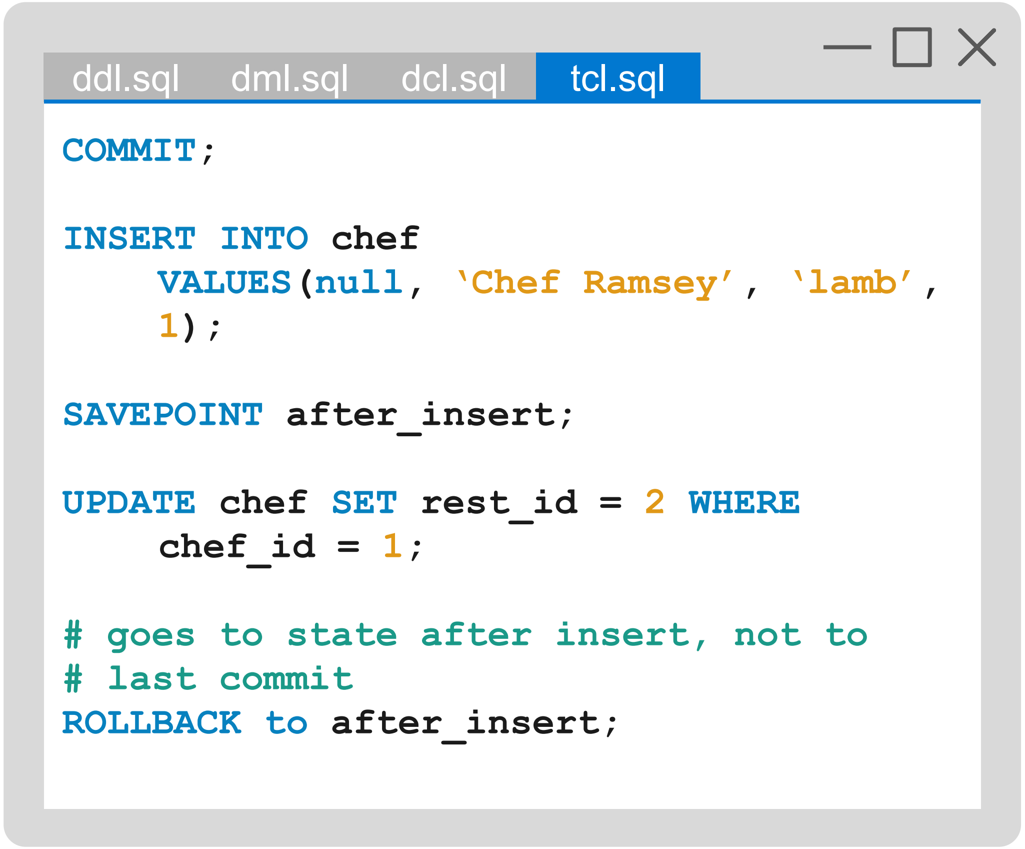
- ➔ A **transaction** is a set of operations performed on a database
- ➔ COMMIT permanently saves these transactions
- ➔ ROLLBACK allows us to restore the database to our last commit state



```
COMMIT;

INSERT INTO chef
     VALUES(null, 'Chef Ramsey', 'lamb',
     1);
INSERT INTO chef
     VALUES(null, 'Chef Remy',
     'ratatouille', 1);

# set state before inserts, if no
# commit, state is set to last DDL or
# DCL statement
ROLLBACK;
```

SAVEPOINT and ROLLBACK

- ➔ SAVEPOINT temporarily saves a transaction so that you may rollback to it when needed
- ➔ Can't return to savepoint once you return to an older state than that of your savepoint



```
COMMIT;

INSERT INTO chef
     VALUES(null, 'Chef Ramsey', 'lamb',
     1);

SAVEPOINT after_insert;

UPDATE chef SET rest_id = 2 WHERE
     chef_id = 1;

# goes to state after insert, not to
# last commit
ROLLBACK to after_insert;
```

Built-In Functions Example Code

➔ Please review the following files:

- `7-String_Functions.sql`
- `8-Number_Functions.sql`
- `9-Date_Functions.sql`

➔ Run each of the commands and make sure you understand what each is doing ddl.sql | dml.sql | dcl.sql | tcl.sql | dql_drl.sql | agg_func.sql | str_func.sql

String Functions

String Functions are used for string manipulation.

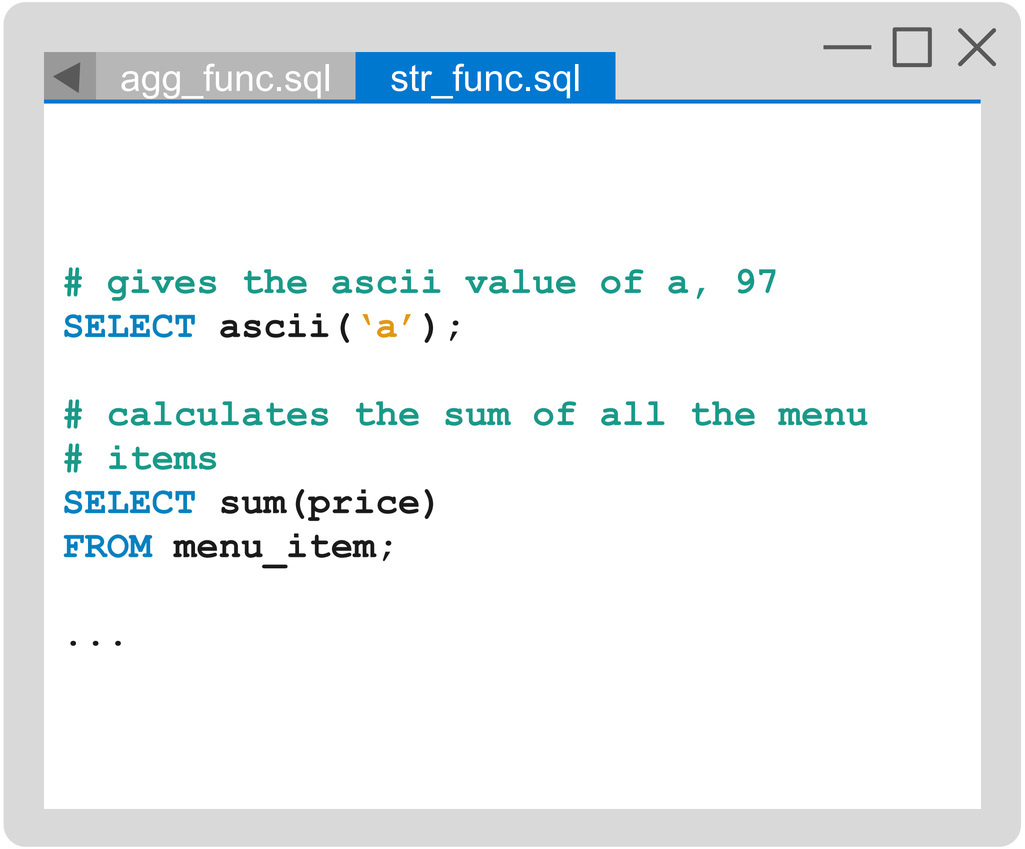
→ **ASCII**
→ **CONCAT**
→ **LENGTH**
→ **UPPER**
→ **SUBSTR**

String Functions

➔ **ASCII** is to convert a character to its numeric ASCII value

➔ **CONCAT** puts together two or more strings

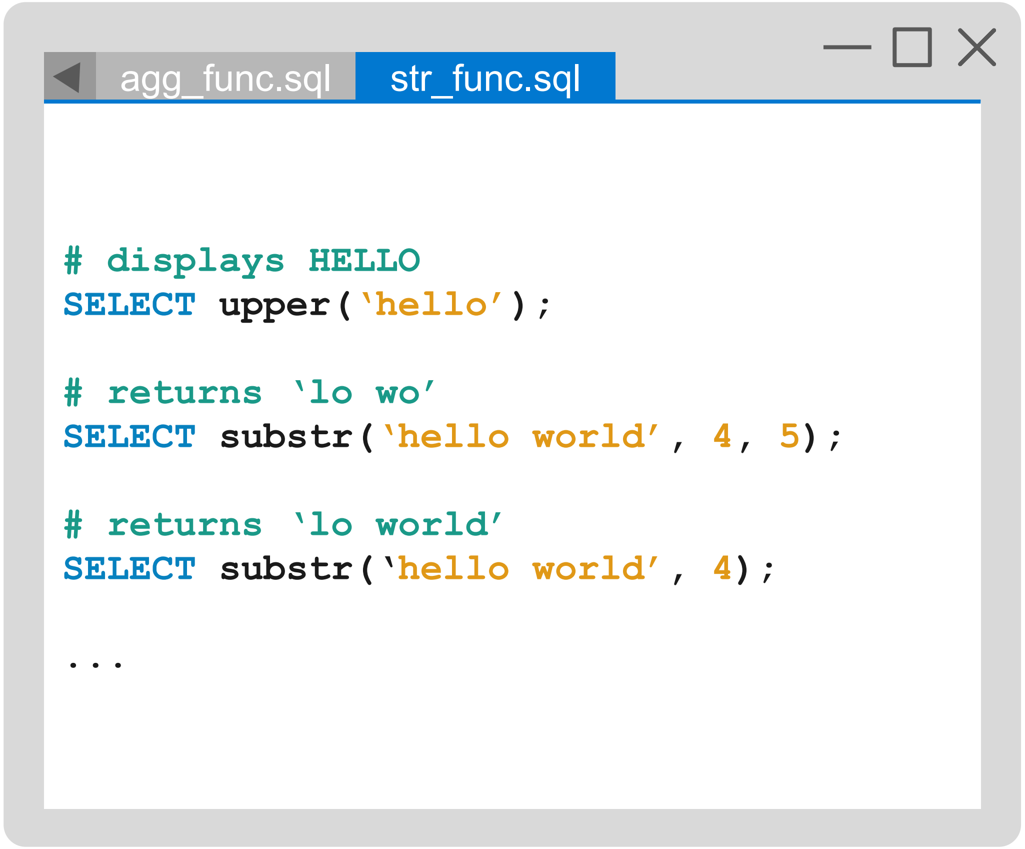
➔ **LENGTH** counts the length of a string



```
# gives the ascii value of a, 97
SELECT ascii('a');

# calculates the sum of all the menu
# items
SELECT sum(price)
FROM menu_item;

...
```

String Functions

- ➔ UPPER changes the string to uppercase
- ➔ SUBSTR selects the substring of a given string



```
# displays HELLO
SELECT upper('hello');

# returns 'lo wo'
SELECT substr('hello world', 4, 5);

# returns 'lo world'
SELECT substr('hello world', 4);

...
```

dcl.sql | tcl.sql | dql_drl.sql | agg_func.sql | str_func.sql | num_func.sql

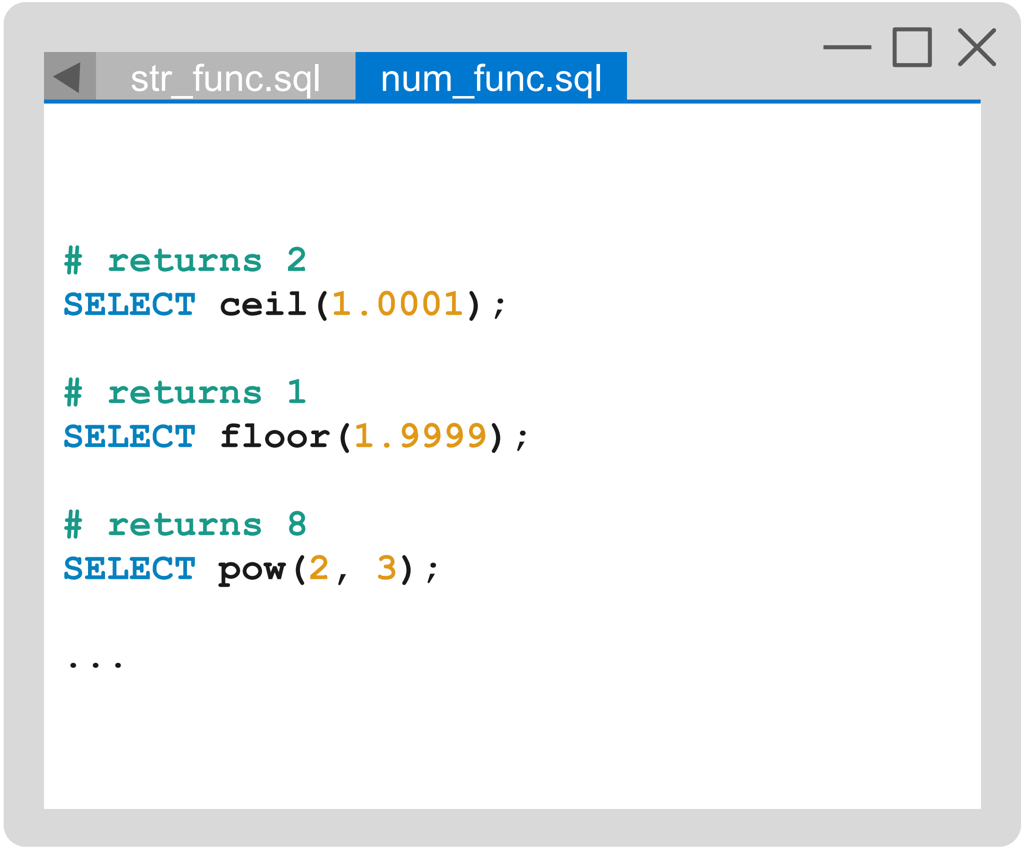
Numeric Functions

Numeric Functions are used for numeric manipulation or calculation.

→ **CEIL**
→ **FLOOR**
→ **POW**
→ **GREATEST**

Numeric Functions

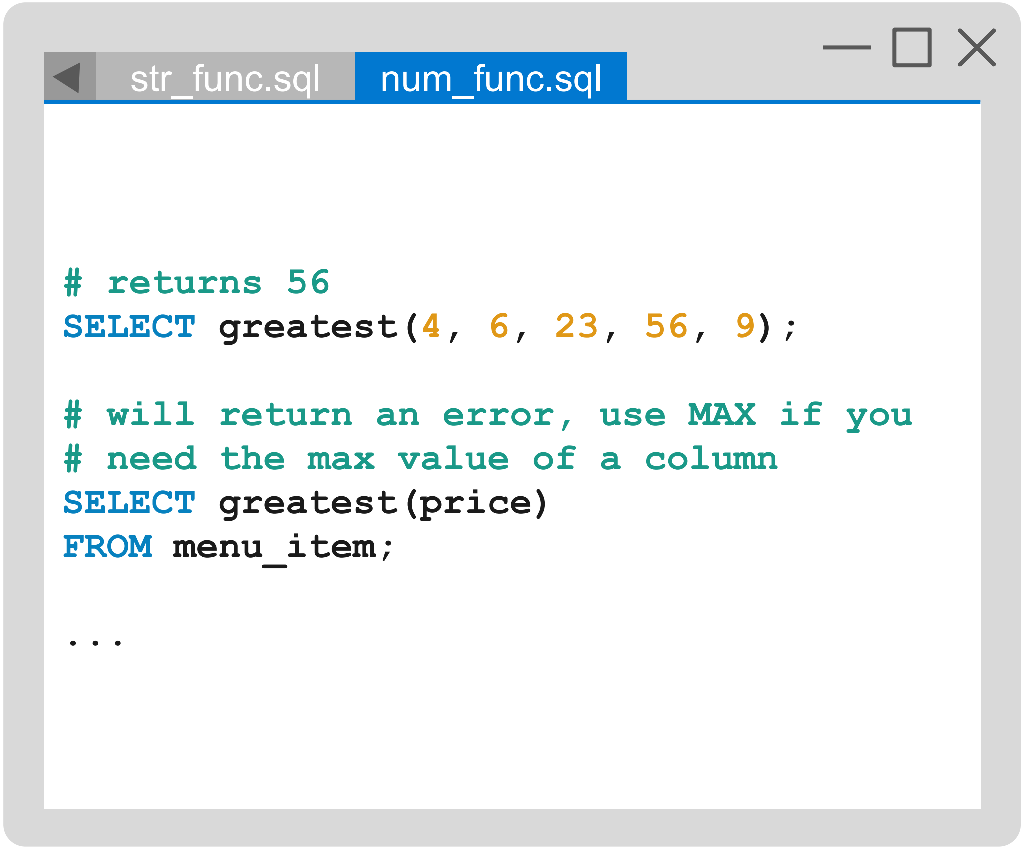
- ➔ **CEIL** rounds number up to next whole number
- ➔ **FLOOR** rounds number down to next whole number
- ➔ **POW** gets the power of a number



```sql
# returns 2
SELECT ceil(1.0001);

# returns 1
SELECT floor(1.9999);

# returns 8
SELECT pow(2, 3);

...
```

Numeric Functions

➔ **GREATEST** gets the largest value in a list
  - Unlike **MAX**, it cannot take a column as a parameter



```
# returns 56
SELECT greatest(4, 6, 23, 56, 9);

# will return an error, use MAX if you
# need the max value of a column
SELECT greatest(price)
FROM menu_item;

...
```

◀ dql_drl.sql agg_func.sql str_func.sql num_func.sql **date_func.sql**

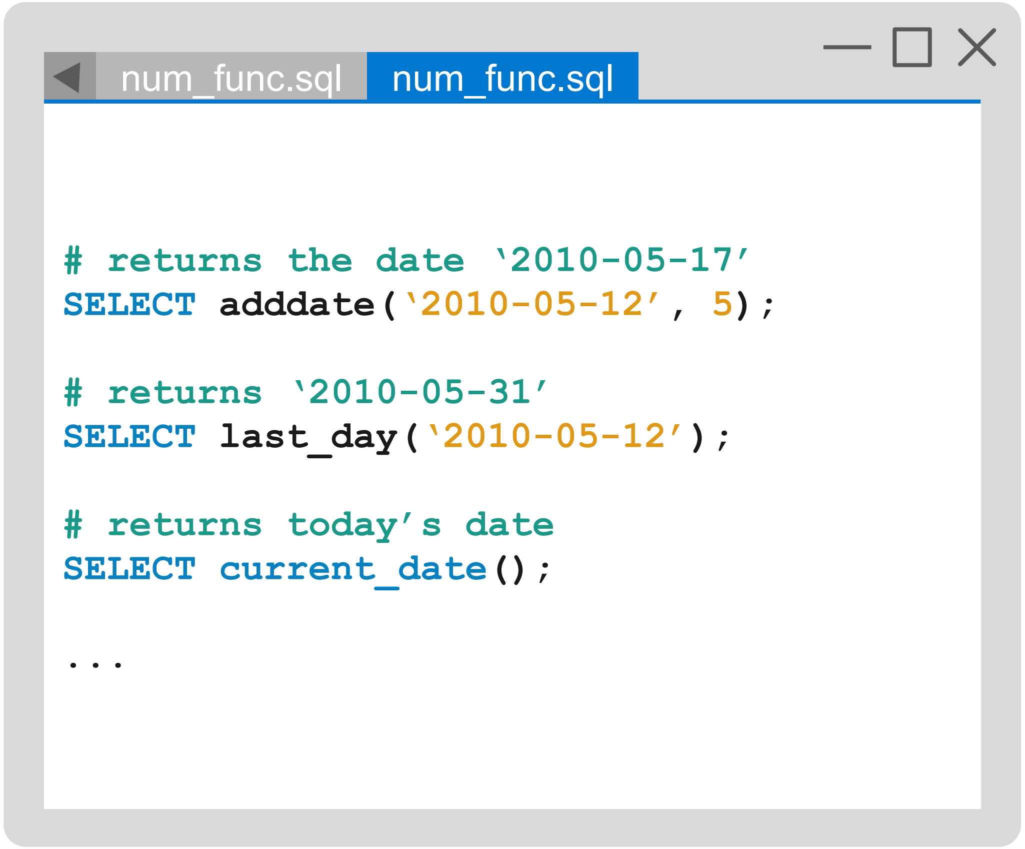
Date Functions

Date Functions are used for manipulation of dates.

→ **ADDDATE**
→ **LAST_DAY**
→ **CURRENT_DATE**
→ **EXTRACT**

Date Functions

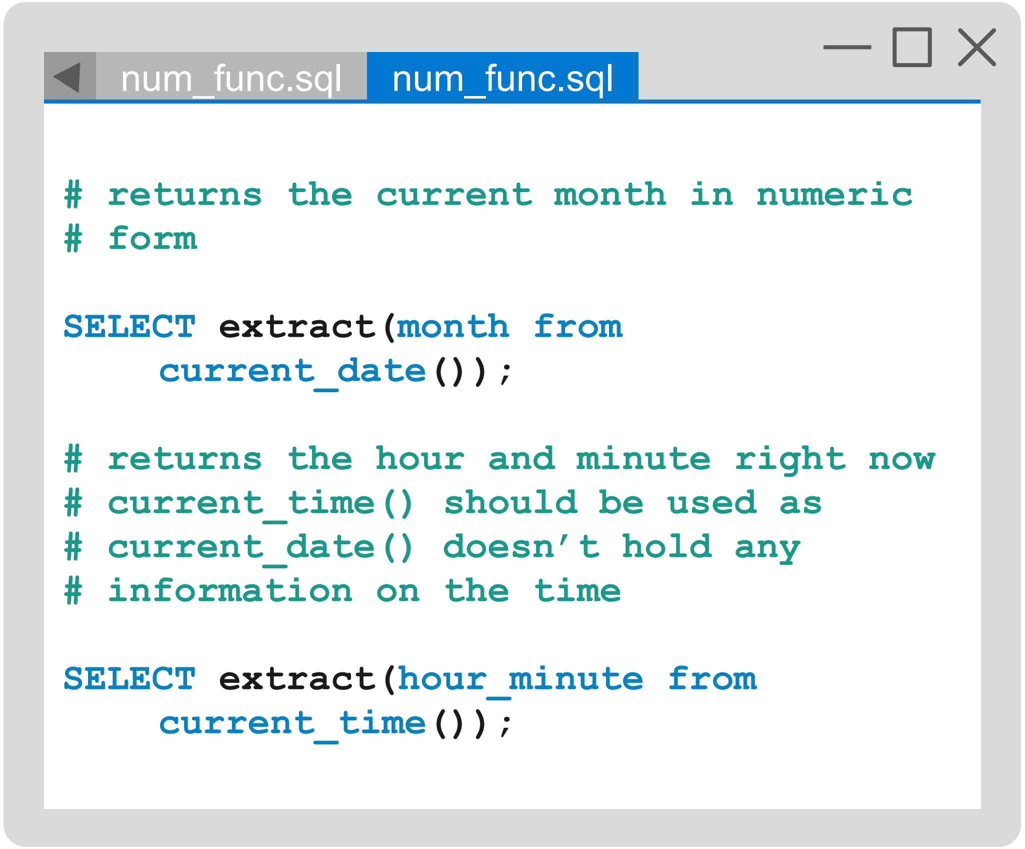
- ➔ ADDDATE takes in a number of days and adds it to the date given
- ➔ LAST_DAY returns the last day of the month for the date given
- ➔ CURRENT_DATE gets today's date



```
# returns the date '2010-05-17'
SELECT adddate('2010-05-12', 5);

# returns '2010-05-31'
SELECT last_day('2010-05-12');

# returns today's date
SELECT current_date();

...
```

Date Functions

→ **EXTRACT** can extract certain parts of a date
  - ◆ Can get things like month, minute, etc.



```
# returns the current month in numeric
# form

SELECT extract(month from
    current_date());

# returns the hour and minute right now
# current_time() should be used as
# current_date() doesn't hold any
# information on the time

SELECT extract(hour_minute from
    current_time());
```

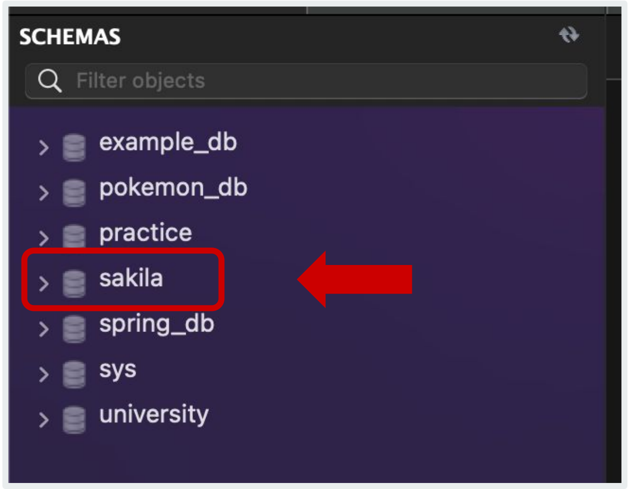
dql_drl.sql agg_func.sql str_func.sql num_func.sql er.sql

ER Generation in MySQL

While we can create ER Diagrams ourselves, it is possible to ***generate ER Diagrams if you already have an existing database*** within MySQL

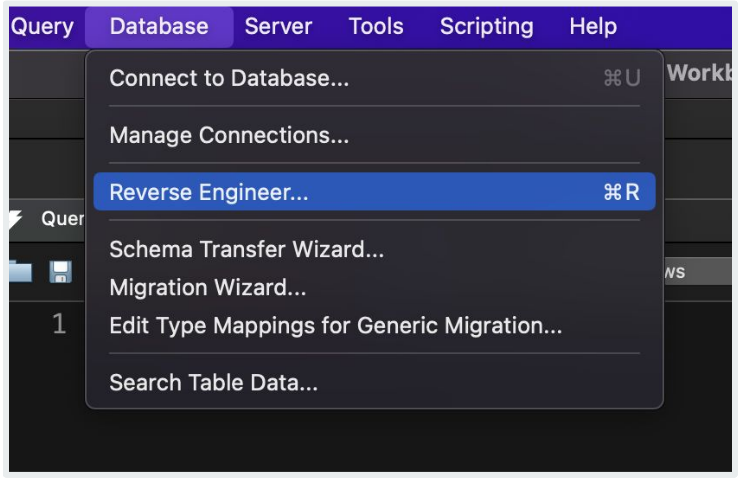
Generate ER Diagram in MySQL

- → Start your MySQL server and open up MySQL Workbench
- → Pick one of the databases under your schemas panel on the left for this demo
- → Most of you should have the ***sakila database*** and that is the schema that will be used for the demo



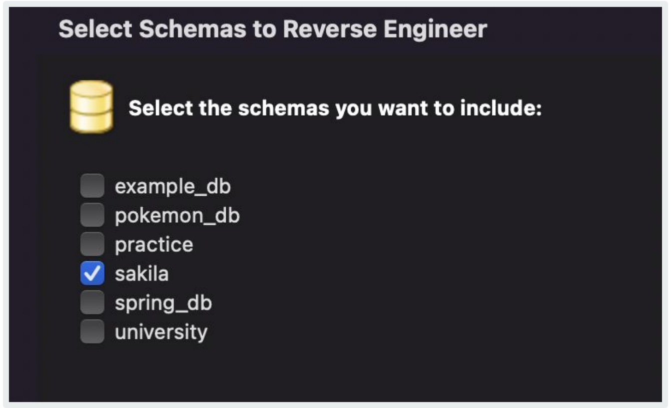
Generate ER Diagram in MySQL

- ➔ Go to the top of the window and select **Database** → **Reverse Engineer…**
- ➔ This will bring up a new window, proceed to the next to the next slide
- ➔ Click on the **Continue** button
- ➔ It will connect to your local database connection and load all your databases



Generate ER Diagram in MySQL

- ➔ Once again click on the **Continue** button
- ➔ You'll get to the next window where it will ask you to select one of your databases, here sakila is checked (pick just one for now)
- ➔ Then click **Continue** once again
- ➔ Move to next window, click **Continue** once database is loaded



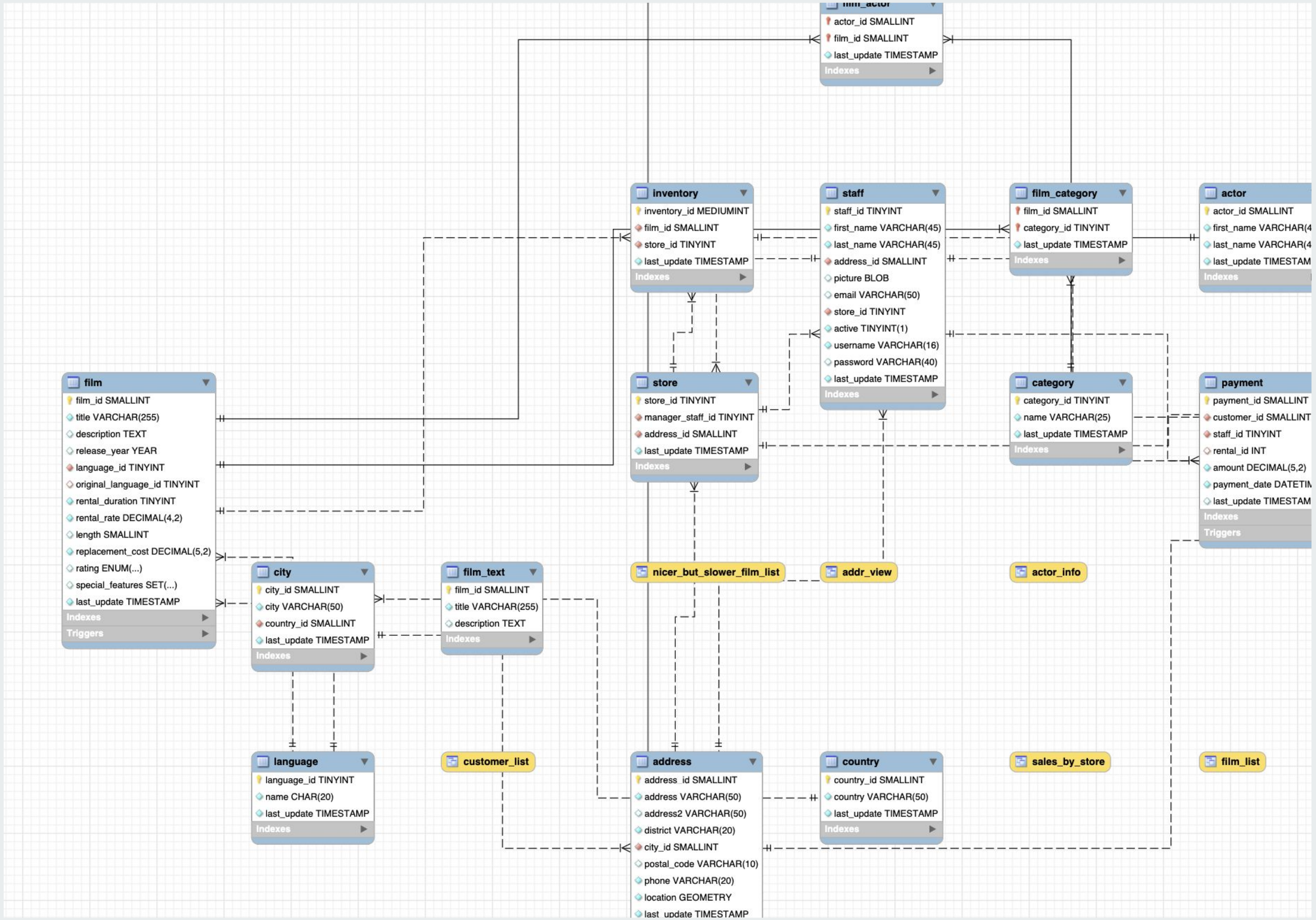
Select Schemas to Reverse Engineer

Select the schemas you want to include:

- [ ] example_db
- [ ] pokemon_db
- [ ] practice
- [✓] sakila
- [ ] spring_db
- [ ] university

Generate ER Diagram in MySQL

- ➔ It will ask what to import from the database, everything will be selected (leave it like that)
- ➔ Then click **Execute** to create the ER Diagram
- ➔ Finally, click **Continue** then **Close** and you should now see the diagram



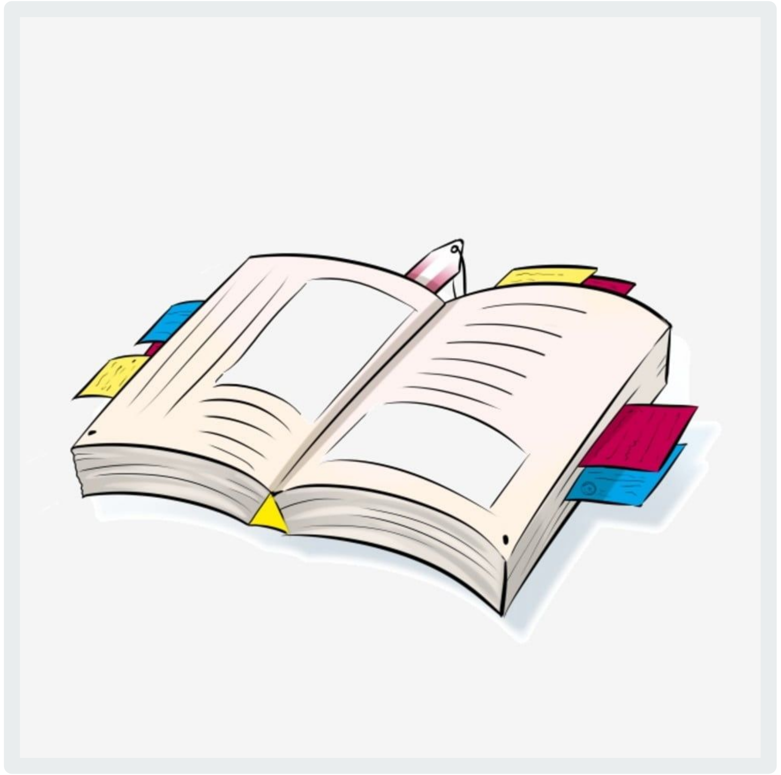
The diagram may not look well organized (like you see on the left), but you can drag around the pieces to make it more well organized. However, you can start seeing what your tables are and which tables contain foreign keys for other tables.

Exercise: Generate an ER Diagram

- ➔ If you haven't already, make sure you have the **University database**
  - ◆ Download and run the `University_Database.sql` script
- ➔ Follow the steps from the earlier instructions and generate an ER Diagram for University
- ➔ ***Clean up the diagram and move things around so it is more organized*** and you can more clearly see the relationship between entities
- ➔ Take a screenshot of the diagram and turn it into **Github Classroom**

Open Book Quiz on DCL, TCL, Functions

- ➔ Check README.md for quiz link
- ➔ This is an **open note**, multiple choice quiz
- ➔ Have it completed by the **start of class tomorrow at 10AM EST**
- ➔ If there are ***any questions, ask your instructor during this time or during office hours***, as they may not able available after hours



FIN