SQL Primer

Thomas Schwarz, SJ

SQL DDL

- Create a database with CREATE DATABASE

```
CREATE DATABASE IF NOT EXISTS USNavy;
```

SQL DDL

- Data Types
  - Character strings of fixed or varying length
    - CHAR(n) - fixed length string of up to *n* characters
    - VARCHAR(n) - fixed length string of up to *n* characters
      - Uses and endmarker or string-length for storage efficiency
  - Bit strings
    - BIT(n) strings of length exactly *n*
    - BIT VARYING(n) - strings of length up to *n*

SQL DDL

- Data Types:
  - Boolean: BOOLEAN: TRUE, FALSE, UNKNOWN
  - Integers: INT = INTEGER, SHORTINT
  - Floats: FLOAT = REAL, DOUBLE, DECIMAL(n,m)
  - Dates: DATE
    - SQL Standard: ‘1948-05-14’)
  - Times: TIME
    - SQL Standard: 19:20:02.4

SQL DDL

- Data Types:
  - MySQL: ENUM('M', 'F')

SQL DDL

- CREATE TABLE creates a table

```
CREATE TABLE Movies(
   title          CHAR(100),
   year           INT,
   length         INT,
   genre          CHAR(10),
   studioName     CHAR(30),
   producerC#     INT
);
```

SQL DDL

```
CREATE TABLE MovieStar(
   name                CHAR(30),
   address             VARCHAR(255),
   gender              CHAR(1),
   birthday            DATE
);
```

SQL DDL

- Drop Table drops a table

```
DROP TABLE Movies;
```

SQL DDL

- Altering a table with ALTER TABLE
    - with ADD followed by attribute name and data type
    - with DROP followed by attribute name

```
ALTER TABLE MovieStar ADD phone CHAR(16);
```

```
ALTER TABLE MovieStar DROP Birthday;
```

SQL DDL

- Default Values
  - Conventions for unknown data
    - Usually, NULL
  - Can use other values for unknown data

```
CREATE TABLE MovieStar(
   name                CHAR(30),
   address             VARCHAR(255),
   gender              CHAR(1) DEFAULT '?',
   birthday            DATE DEFAULT '0000-00-00'
);
```

SQL DDL

- Declaring Keys

  1. Declare one attribute to be a key

  2. Add one additional declaration:

     - Particular set of attributes is a key

  - Can use

  1. PRIMARY KEY

  2. UNIQUE

SQL DDL

- UNIQUE for a set S:
    - Two tuples cannot agree on all attributes of S unless one of them is NULL
        - Any attempted update that violates this will be rejected
- PRIMARY KEY for a set S:
    - Attributes in S cannot be NULL

SQL DDL

```
CREATE TABLE MovieStar(
   name                CHAR(30) PRIMARY KEY,
   address             VARCHAR(255),
   gender              CHAR(1),
   birthday            DATE
);
```

SQL DDL

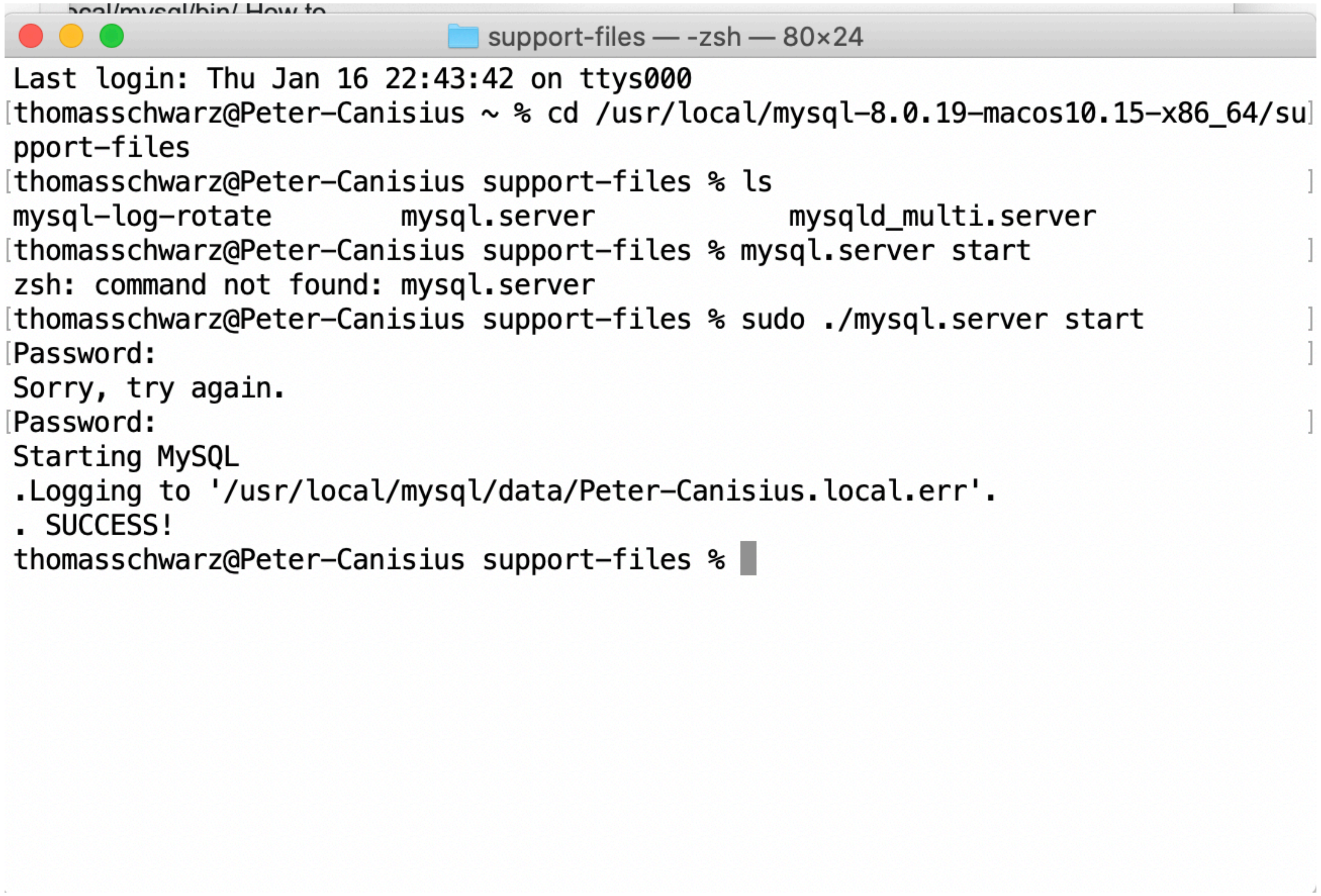
```
CREATE TABLE MovieStar(
   name                CHAR(30),
   address             VARCHAR(255),
   gender              CHAR(1) DEFAULT '?',
   birthday            DATE DEFAULT '0000-00-00',
   PRIMARY KEY (name)
);
```

SQL DDL

```
CREATE TABLE Movies(
   title          CHAR(100),
   year           INT,
   length         INT,
   genre          CHAR(10),
   studioName     CHAR(30),
   producerC#     INT,
   PRIMARY KEY (title, year)
);
```

SQL Work Bench

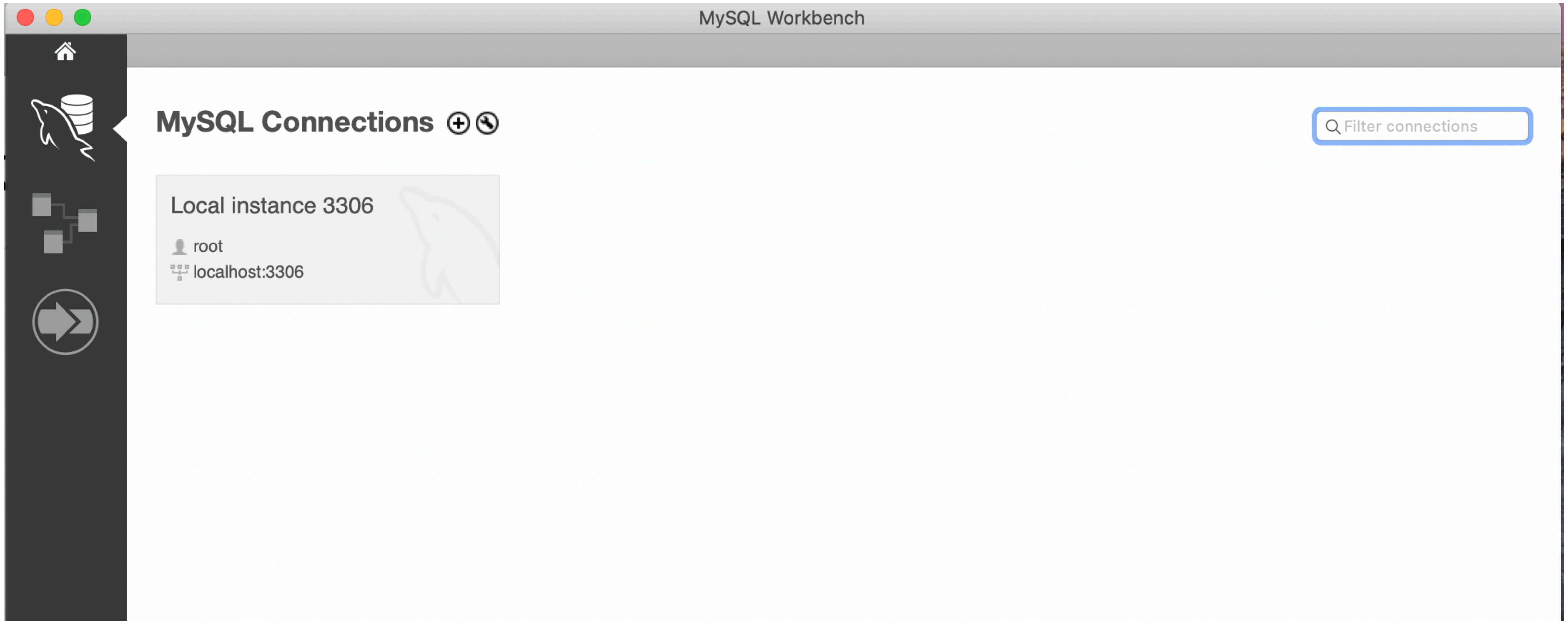
- Starting MySQL server through a terminal
  - Find mysql.server



```
Last login: Thu Jan 16 22:43:42 on ttys000
thomasschwarz@Peter-Canisius ~ % cd /usr/local/mysql-8.0.19-macos10.15-x86_64/su
pport-files
thomasschwarz@Peter-Canisius support-files % ls
mysql-log-rotate        mysql.server            mysqld_multi.server
thomasschwarz@Peter-Canisius support-files % mysql.server start
zsh: command not found: mysql.server
thomasschwarz@Peter-Canisius support-files % sudo ./mysql.server start
Password:
Sorry, try again.
Password:
Starting MySQL
.Logging to '/usr/local/mysql/data/Peter-Canisius.local.err'.
. SUCCESS!
thomasschwarz@Peter-Canisius support-files % 
```

SQL Workbench

- Open up SQL workbench
  - Select the SQL server (should be only one)



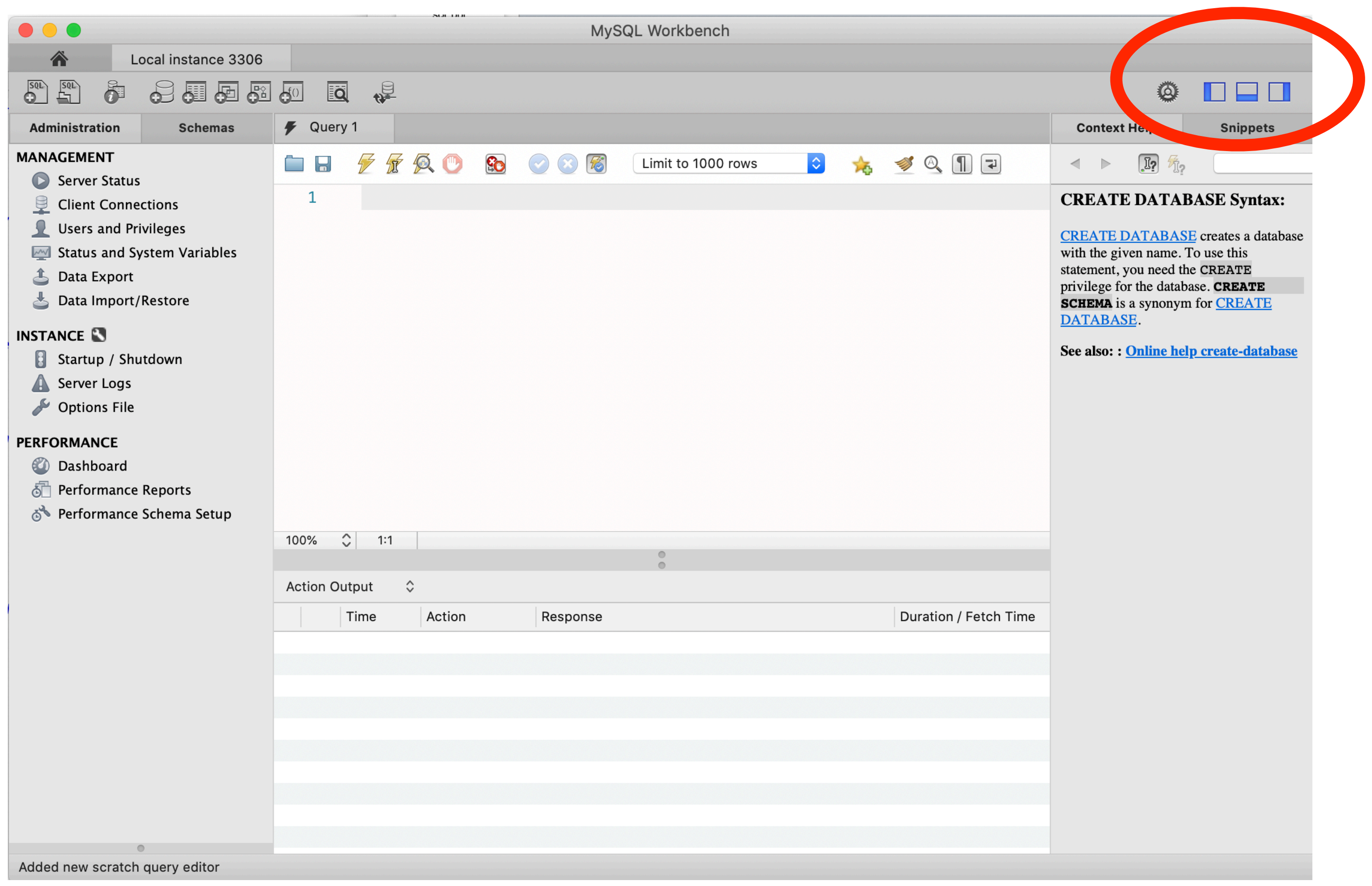
Local instance 3306

root

localhost:3306

SQL Workbench

- Select panels on the right



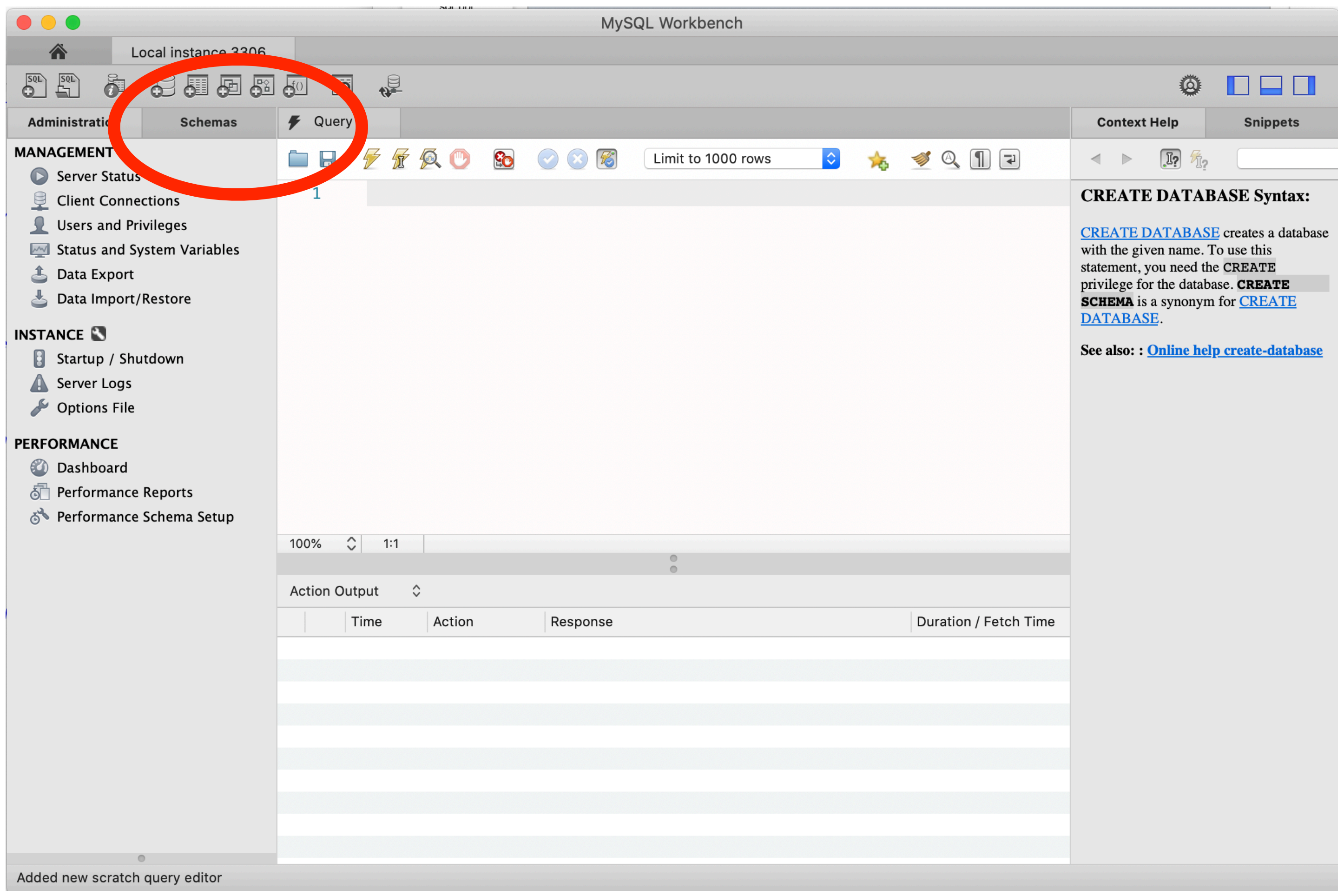
CREATE DATABASE Syntax:

CREATE DATABASE creates a database with the given name. To use this statement, you need the **CREATE** privilege for the database. **CREATE SCHEMA** is a synonym for CREATE DATABASE.

See also: : Online help create-database

SQL Workbench

- Select Schemas
  - Should have at least one master schema called sys



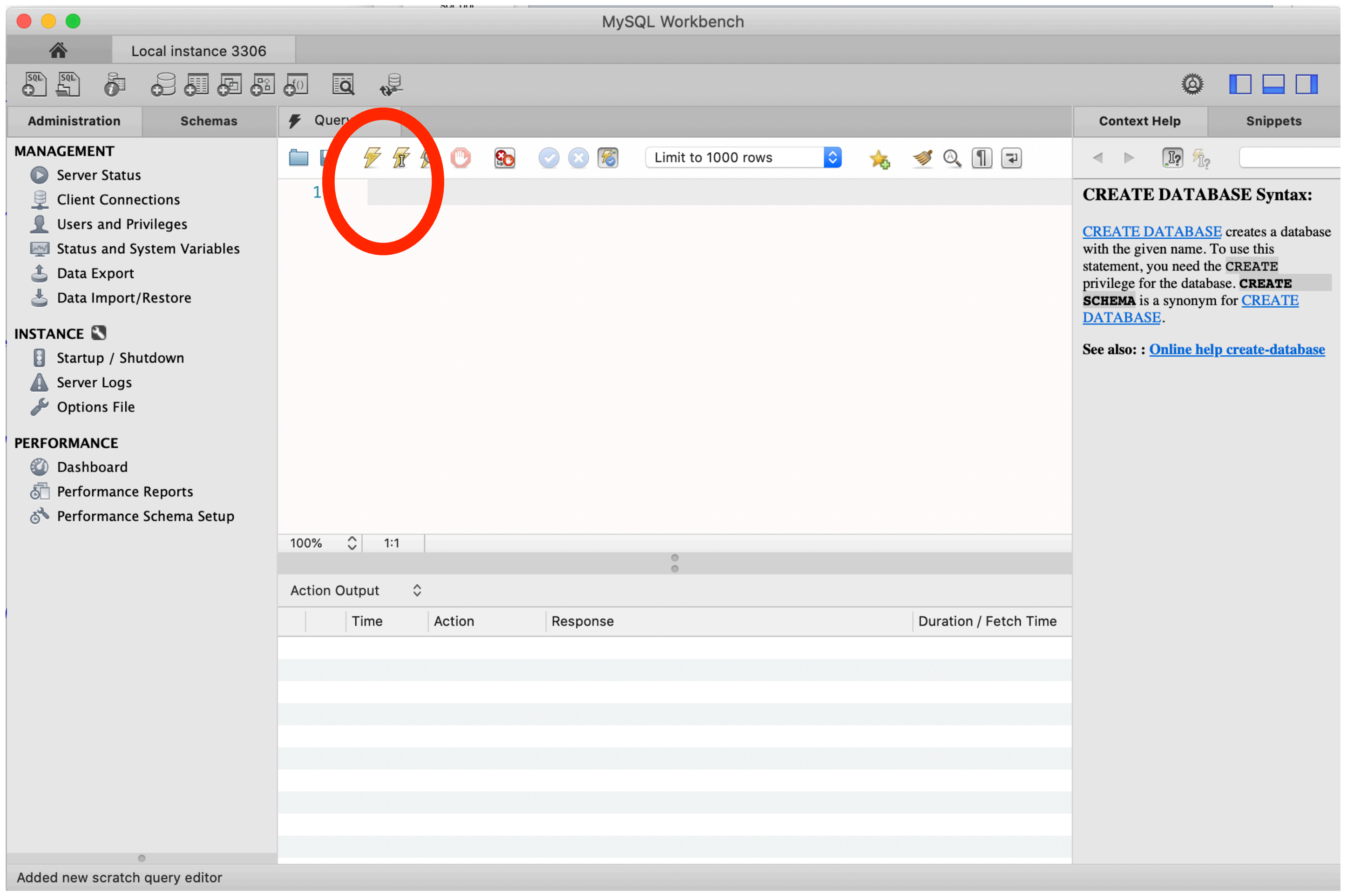
CREATE DATABASE Syntax:

CREATE DATABASE creates a database with the given name. To use this statement, you need the CREATE privilege for the database. CREATE SCHEMA is a synonym for CREATE DATABASE.

See also: : Online help create-database

SQL Workbench

- Write queries in middle panel
- Execute them with the flash symbol
  - CREATE DATABASE IF NOT EXISTS sales;



CREATE DATABASE Syntax:

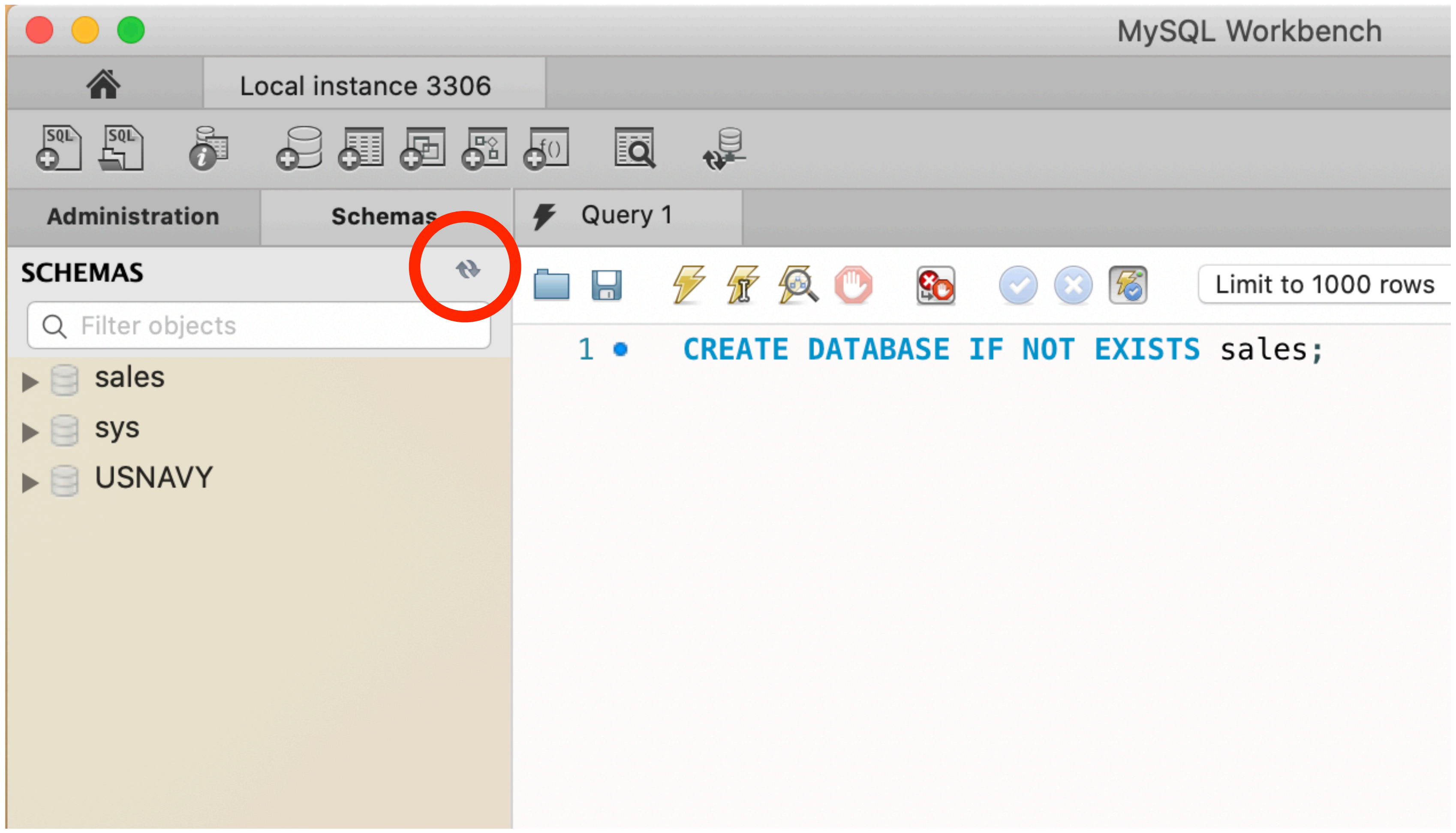
CREATE DATABASE creates a database with the given name. To use this statement, you need the **CREATE** privilege for the database. **CREATE SCHEMA** is a synonym for CREATE DATABASE.

See also: : Online help create-database

Added new scratch query editor

SQL Workbench

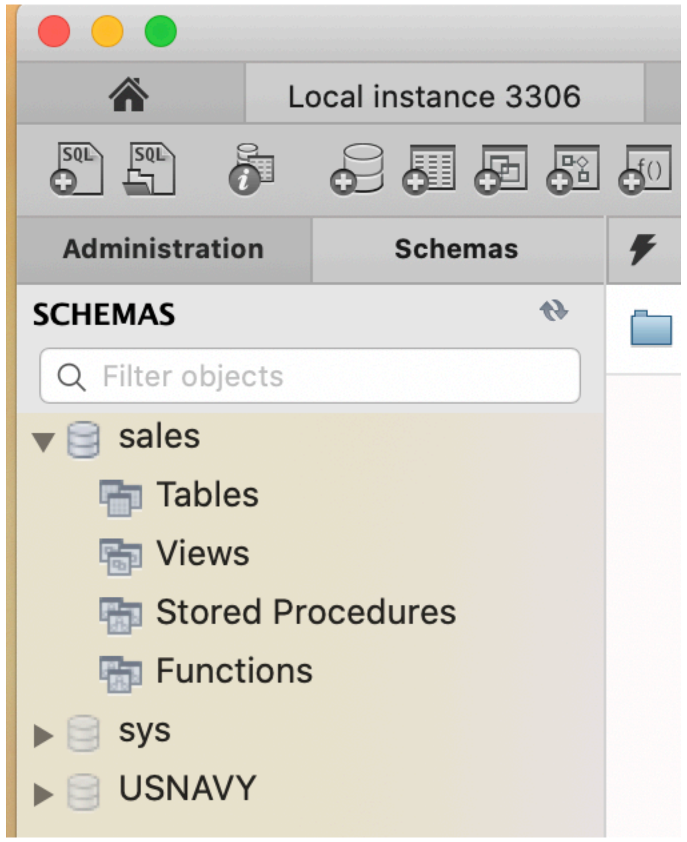
- After creating a database, need to update schemas in the upper right corner



```
CREATE DATABASE IF NOT EXISTS sales;
```

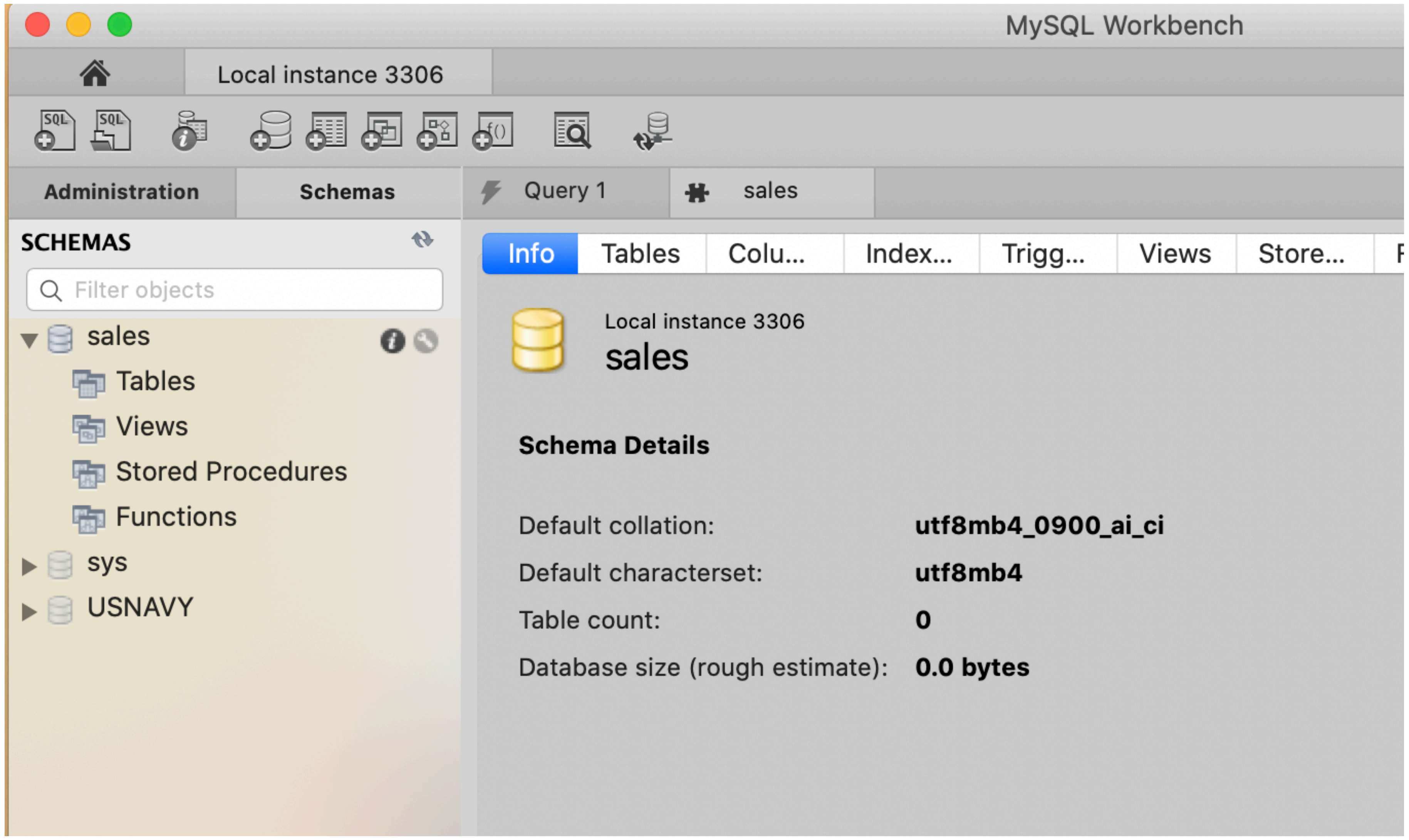
SQL Workbench

- There is more information on the schema



SQL Workbench

- The information symbol (i) has more information



Local instance 3306
sales

Schema Details

Default collation:	utf8mb4_0900_ai_ci
Default characterset:	utf8mb4
Table count:	0
Database size (rough estimate):	0.0 bytes

SQL Workbench

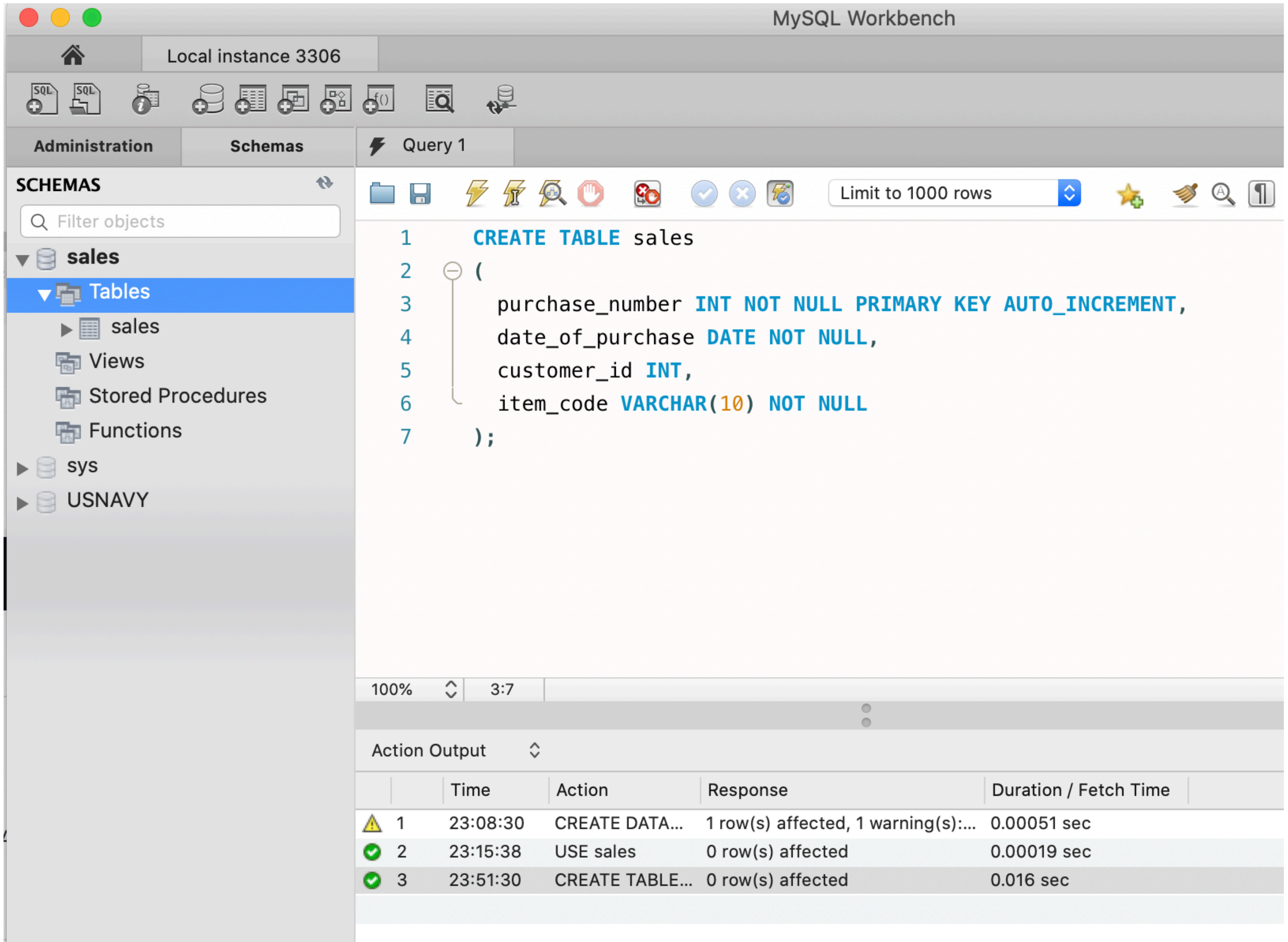
- Execute a query
  - `USE sales;`
- Now we can manipulate and use this database

SQL Workbench

- Use queries to create a table

  -
    ```
    sales(purchase_number:int,
          date_of_purchase:date,
          customer_id:int,
          item_code VARCHAR(10) )
    ```

SQL Workbench



```
CREATE TABLE sales
(
  purchase_number INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
  date_of_purchase DATE NOT NULL,
  customer_id INT,
  item_code VARCHAR(10) NOT NULL
);
```

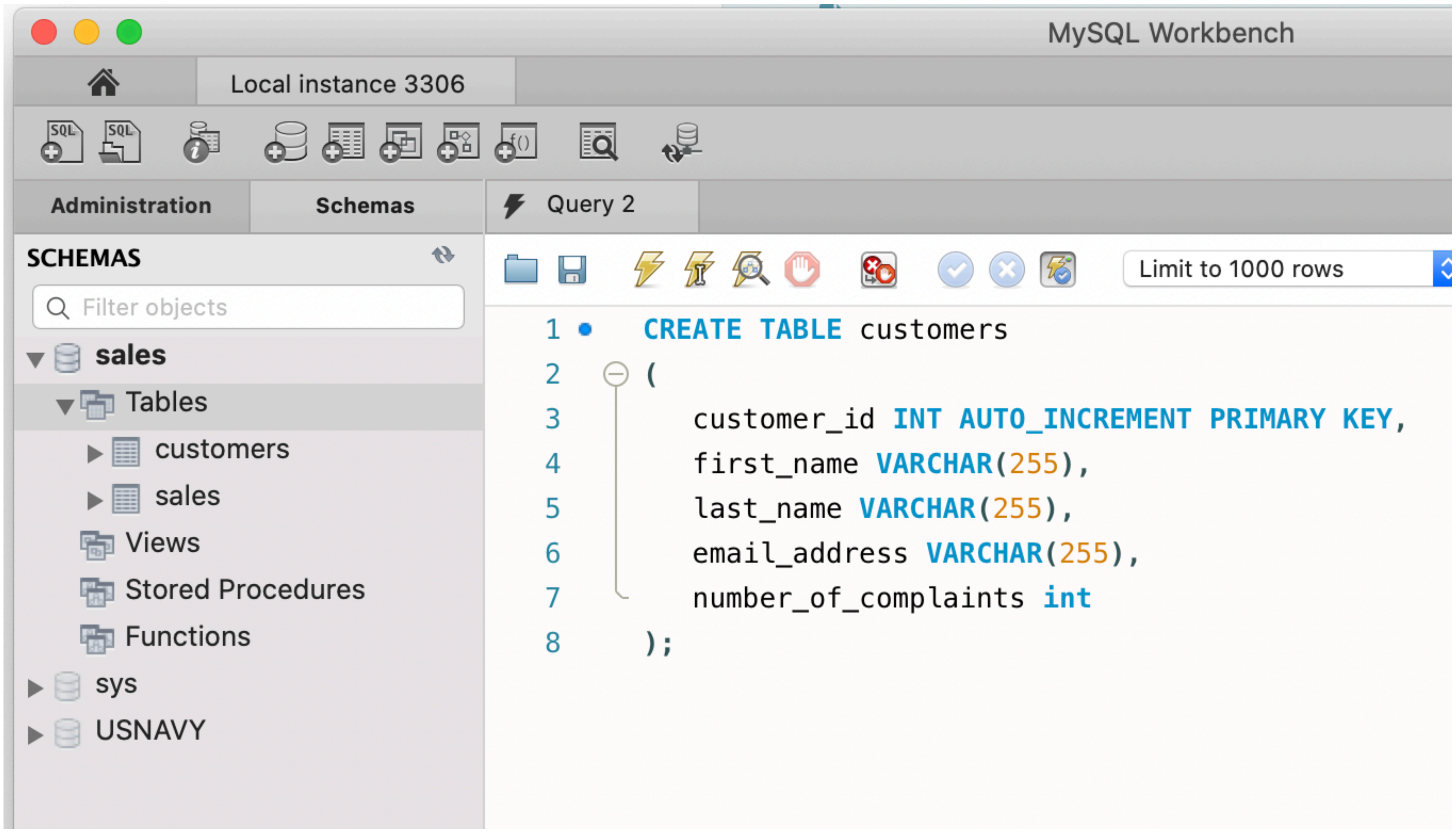
		Time	Action	Response	Duration / Fetch Time
⚠	1	23:08:30	CREATE DATA...	1 row(s) affected, 1 warning(s):...	0.00051 sec
✓	2	23:15:38	USE sales	0 row(s) affected	0.00019 sec
✓	3	23:51:30	CREATE TABLE...	0 row(s) affected	0.016 sec

SQL Workbench

- Create a table

```
customers(customer_id: int,
          first_name: varchar(255),
          last_name: varchar(255),
          email_address: varchar(255),
          number_of_complaints: int)
```

SQL Workbench



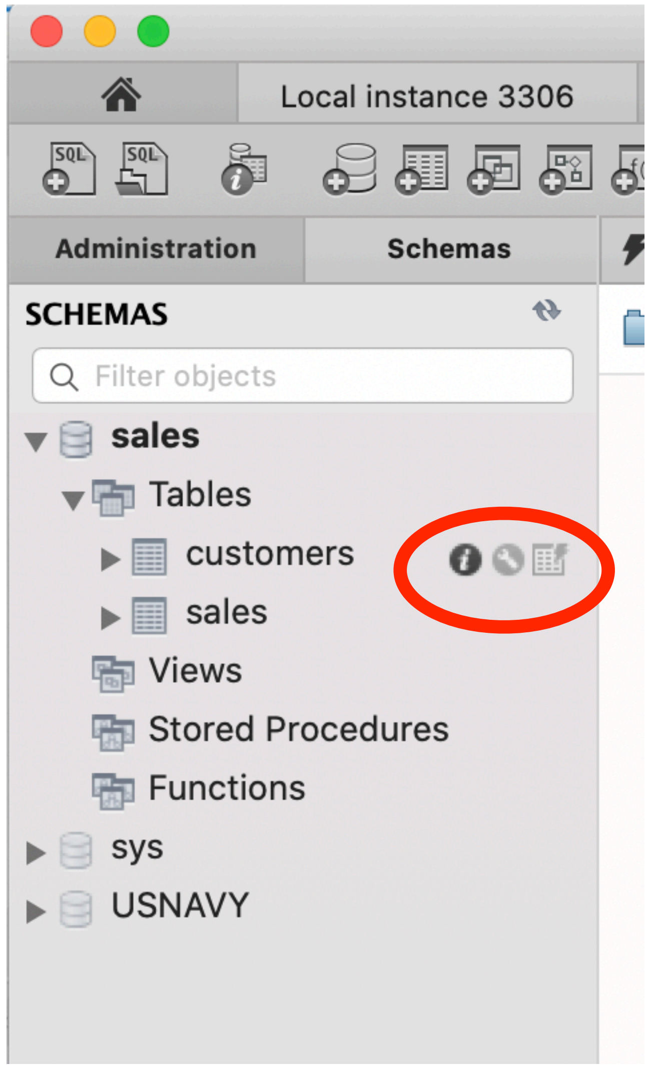
```sql
CREATE TABLE customers
(
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(255),
    last_name VARCHAR(255),
    email_address VARCHAR(255),
    number_of_complaints int
);
```

SQL Workbench

- Referring to MYSQL objects
  - Use a default database
    - `USE sales;`
    - `SELECT * FROM customers;`
  - Use the dot notation to specify database
    - `SELECT * FROM sales.customers;`

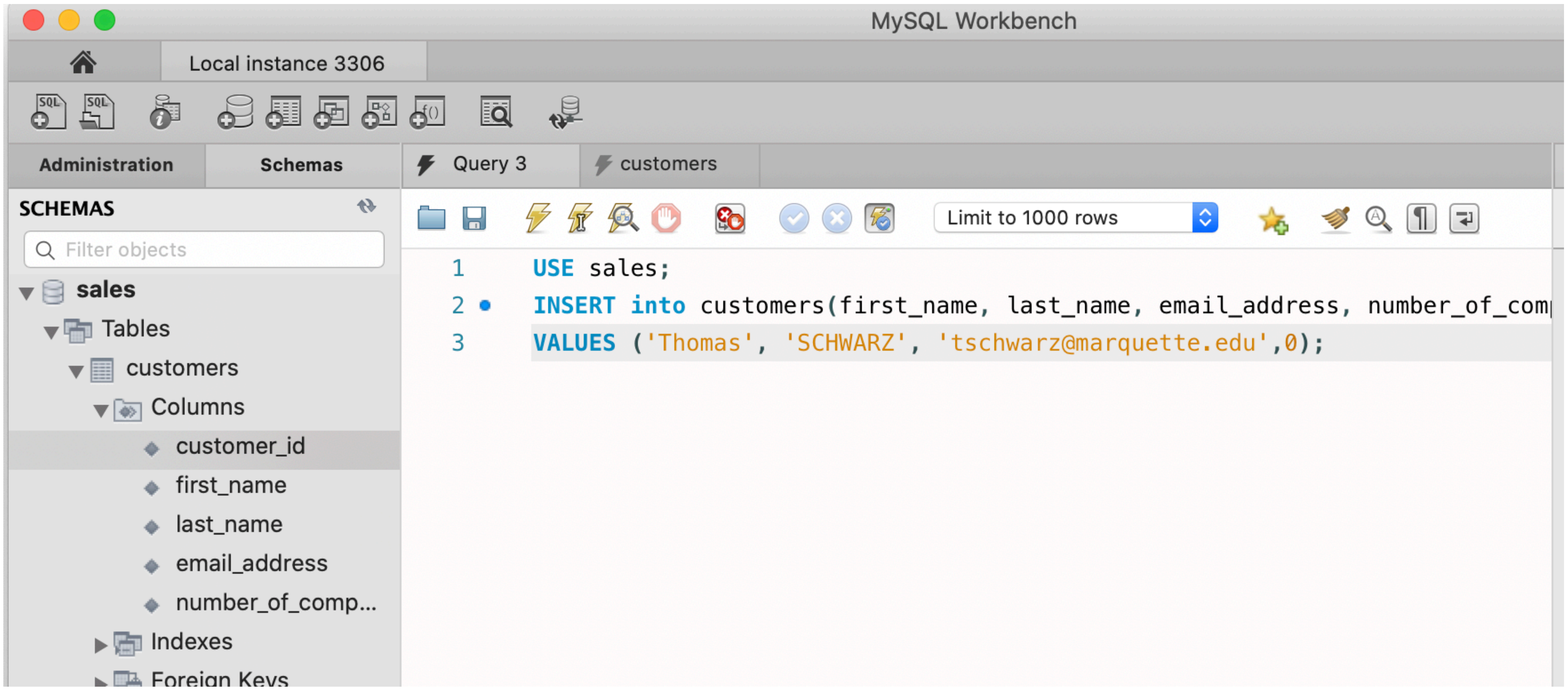
SQL Workbench

- Information on Tables appears next to them in the left panel



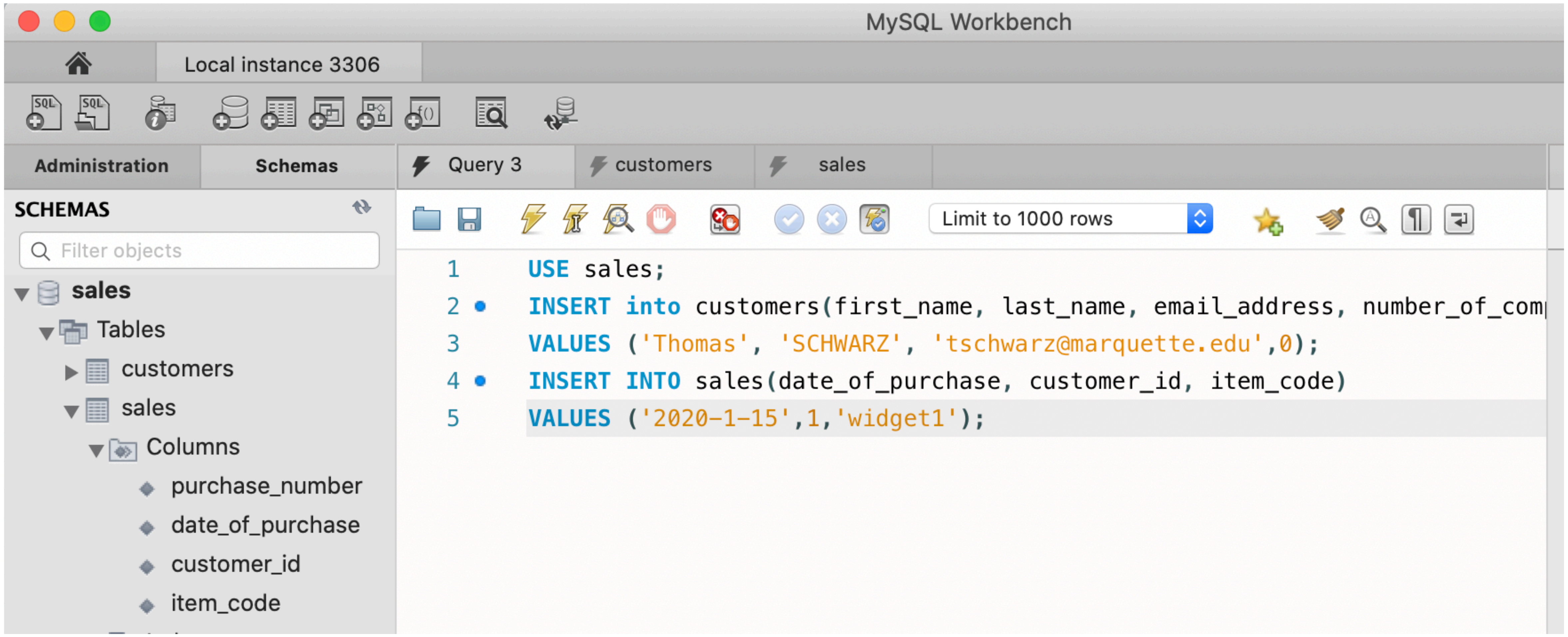
SQL Workbench

- Inserting into a data base:



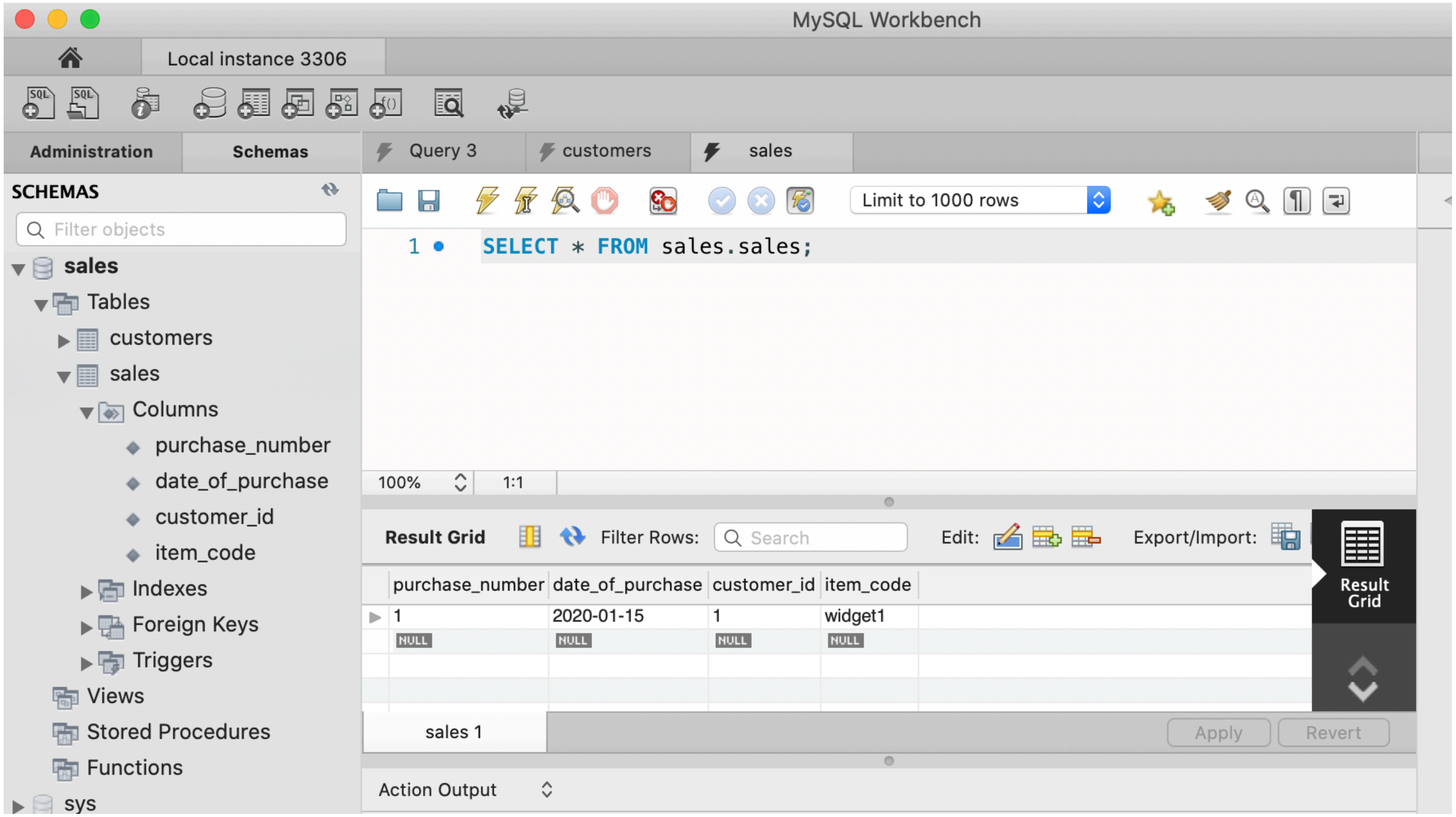
```
USE sales;
INSERT into customers(first_name, last_name, email_address, number_of_com
VALUES ('Thomas', 'SCHWARZ', 'tschwarz@marquette.edu',0);
```

SQL Workbench



```
USE sales;
INSERT into customers(first_name, last_name, email_address, number_of_com
VALUES ('Thomas', 'SCHWARZ', 'tschwarz@marquette.edu',0);
INSERT INTO sales(date_of_purchase, customer_id, item_code)
VALUES ('2020-1-15',1,'widget1');
```

SQL Workbench



```sql
SELECT * FROM sales.sales;
```

purchase_number	date_of_purchase	customer_id	item_code
1	2020-01-15	1	widget1
NULL	NULL	NULL	NULL

SELECT

- SELECT is the most frequent command
  - Basic use:
    - SELECT attribute1, attribute2, … FROM databasetable
    - SELECT * FROM databasetable

SELECT

- SELECT — WHERE clause:
  - Imposes a condition on the results

SELECT

- =   equals (comparison operator)
- AND, OR
- IN, NOT IN
- LIKE, NOT LIKE
- BETWEEN … AND
- EXISTS, NOT EXISTS
- IS NULL, IS NOT NULL
- comparison operators

SELECT

- AND operator
  - Combines two statements (concerning one or more tables)

```
SELECT
   *
FROM
   employees
WHERE
   first_name = 'Denis' and gender = 'M;
```

SELECT

- OR is the Boolean or
- Trick Question: How many records will this query return?

```
SELECT
   *
FROM
   employees
WHERE
   last_name = 'Denis' AND gender = 'M' OR gender = 'F'
```

SELECT

- Operator precedence:
  - AND < OR

```
SELECT
   *
FROM
   employees
WHERE
   last_name = 'Denis' AND (gender = 'M' OR gender = 'F')
```

SELECT

- Quiz:
    - Retrieve all female employees with first name 'Aruna' or 'Kelly'

SELECT

- IN, NOT IN
  - Checks for membership in lists
  - MySQL: faster than equivalent OR formulation

```
SELECT
   *
FROM
   employees
WHERE
   first_name NOT IN ('Elvis','Kevin','Thomas');
```

SELECT

- LIKE
  - Pattern matching
    - Wild cards
      - % means zero or more characters
      - _ means a single letter
      - [ ] means any single character within the bracket
      - ^ means any character not in the bracket
      - - means a range of characters

Like Examples

- WHERE name LIKE 't%'
  - any values that start with 't'
- WHERE name LIKE '%t'
  - any values that end with 't'
- WHERE name LIKE '%t%'
  - any value with a 't' in it
- WHERE name LIKE '_t%'
  - any value with a 't' in second position

Like Examples

- WHERE name LIKE '[ts]%'
  - any values that start with 't' or 's'
- WHERE name LIKE '[t-z]'
  - any values that start with 't', 'u', 'v', 'w', 'x', 'y', 'z'
- WHERE name LIKE '[!ts]%'
  - any value that does not start with a 't' or a 's'
- WHERE name LIKE '_t%'
  - any value with a 't' in second position

SELECT

- BETWEEN … AND …
    - Selects records with a value in the range
        - endpoints included

```
SELECT
   *
FROM
   employees
WHERE
   hire_data between 1990-01-01 and 1999-12-31;
```

SELECT

- SELECT DISTINCT

```
SELECT DISTINCT
     gender
FROM
     employees
```

SELECT

- Aggregate Functions
  - Applied to a row of a result table
    - COUNT
    - SUM
    - MIN
    - MAX
    - AVG

SELECT

- SELECT COUNT

    -
      ```
      SELECT
      	COUNT(emp_no)
      FROM
      	employees
      ```

SELECT

- SELECT COUNT

```
SELECT COUNT(employees.emp_no)
FROM employees
WHERE
     first_name LIKE ('Tom%') or first_name
LIKE('Tho%');
```

SELECT

- Combine COUNT with DISTINCT

```
SELECT
   COUNT(DISTINCT first_name, last_name)
FROM
    employees
```

SELECT

- Combine COUNT with DISTINCT

```
SELECT
   COUNT(DISTINCT emp_no)
FROM
   salaries
WHERE
   salary >=100000;
```

SELECT

- ORDER BY
  - Orders result by default in ascending order
    - ASC ascending
    - DSC descending

```
SELECT
   *
FROM
   employees
WHERE
    hire_date > '2000-01-01'
ORDER BY first_name;
```

SELECT

- GROUP BY
    - Just before ORDER BY in a query
        - Needed with aggregate functions
            - Example: Getting all first names in order

```
SELECT
   first_name
FROM
   employees
GROUP BY first_name;
```

SELECT

- GROUP BY
  - Example: Counting first names in the employee data base
    - Hint: you want to include the attribute on which you group

```
SELECT
   first_name, COUNT(first_name)
FROM
   employees
GROUP BY first_name
ORDER BY first_name;
```

SELECT

- GROUP BY
  - Example: Counting first names in the employee data base
    - To make it look better, add an AS clause

```
SELECT
   first_name, COUNT(first_name)
FROM
   employees
GROUP BY first_name
ORDER BY first_name;
```

Queries with more than one table

- Normally, combine tables by listing them in the FROM clause

```
SELECT name
FROM movies, moviesExec
WHERE title = ‘Star Wars’
        AND movies.producerC# = moviesExec.cert#
```

Queries with more than one table

- Find all movie execs that live with a star

-
  ```
  MovieStar(name, address, gender, birthdate)
  MovieExec(name, address, cert#, netWorth)
  ```

```
SELECT MovieStar.name, MovieExec.name)
FROM MovieStar, MovieExec
WHERE
   MovieStar.address = MovieExec.address
```

Queries with more than one table

- Tuple Variables
  - Sometimes need to combine two tuples in the same table
  - Can extend the FROM clause

```
SELECT Star1.name, Star2.name
FROM MovieStars Star1, MovieStars Star2
WHERE
  Star1.address = Star2.address
  AND  Star1.name < Star2.name
```

Queries with more than one table

- Unions, intersections, excepts
- To execute the corresponding set operations
-

```
(SELECT name, address
 FROM movieStars
 WHERE gender = 'F'
)
 INTERSECT
(SELECT name, address
 FROM movieExecs
 WHERE netWorth > 1000000
)
```

Updates

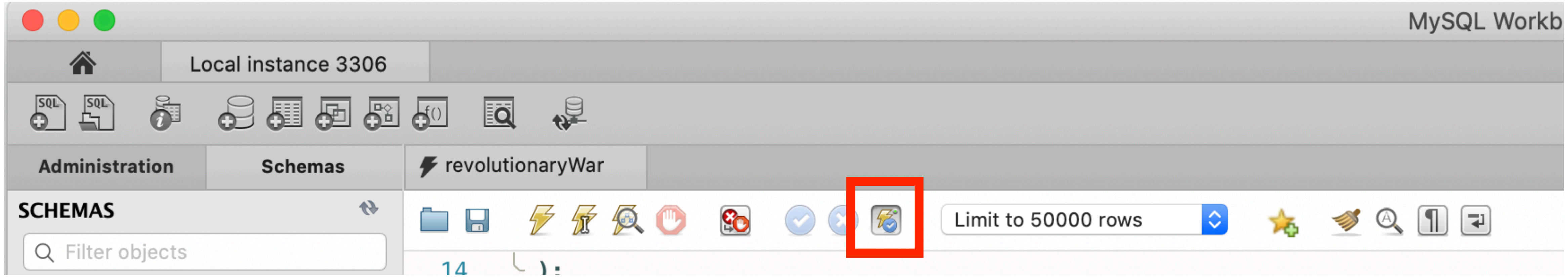
- Changes existing records
- Syntax:

```
UPDATE tablename
SET attr1=val1, attr2=val2, …
WHERE conditions;
```

- Does not need to change <u>all</u> attributes
- If there is no WHERE condition, all records are updated

Commit and Rollback

- A database allows us to rollback to a previous state unless we have committed
- MySQLWorkbench has an auto-commit button



- Rollback puts database into the state of the last commit

Delete

- Just like an update

```
DELETE FROM tablename
WHERE condition
```

- The Where clause is not necessary

Delete, Drop, Truncate

- Drop Table:
  - Definite action: cannot recover with rollback
- Truncate:
  - All records removed
  - Auto-increment values reset
  - Table description stays
- Delete:
  - Delete removes records row by row
  - Auto-increment values remain
  - Slower than truncate

Subqueries

- Subqueries are helper queries

Subqueries

- Subqueries producing a scalar value
  - Example: Producer of Star Wars

```
SELECT name
From movies, movieExec
WHERE title = 'Star Wars'
        AND
      producerC# = cert#;
```

  - Can achieve the same effect by first looking for the producerC#

Subqueries

- Example: Producer of Star Wars

```
SELECT name
FROM movieExec
WHERE cert# =
     (SELECT producerC#
      FROM movies
      WHERE title = 'star wars'
     )
```

- This might be implemented with the same query execution as before

Subqueries

- Subqueries with conditions involving relations
  - We obtain a relation R as a subquery
  - E.g. with subquery (SELECT * FROM foobar)
  - Queries are:
    - EXISTS R
    - *s* IN R   *s* NOT IN R
    - *s* > ALL R   NOT   *s* > ALL R
    - *s* > ANY R   NOT *s* > ANY R

Subqueries

- Subqueries involving tuples
  - Tuple is a list of scalar values
  - Can compare tuples with the same number of components
  - Example:
    - Finding the producers of 'Harrison Ford' movies

Subqueries

```
SELECT name
FROM movieExec
WHERE cert# IN
    (SELECT producerC#
     FROM movies
     WHERE (title, year) IN
        (SELECT movieTitle, movieYear
         FROM StarsIn
         WHERE starName = 'Harrison Ford'
        )
     );
```

Subqueries

- To analyze a query, start with the inmost query

```
SELECT name
FROM movieExec
WHERE cert# IN
     (SELECT producerC#
      FROM movies
      WHERE (title, year) IN
          (SELECT movieTitle, movieYear
           FROM StarsIn
           WHERE starName = 'Harrison Ford'
          )
      );
```

Subqueries

- This query can also be written without nested subqueries

```
SELECT name
FROM movieExec, movies, starsIn
WHERE  cert# = producerC#
     AND starsIn.title = movies.title
     AND starsIn.year = movie.year
     AND starName = 'Harrison Ford'
```

Subqueries

- Correlated subqueries
  - Subquery is evaluated many times
    - Once for each value given
- Example

```
SELECT title
FROM movies Old
WHERE year < ANY (
   SELECT year
   FROM movies
   WHERE title = Old.title
 );
```

Subqueries

- Scoping rules
  - First look for the subquery and tables in that subquery
  - Then go to the nesting subquery
  - etc.

Subqueries

- Subqueries in FROM clauses
  - Here we join on a subquery aliased Prod

```
SELECT name
FROM movieExecs, ( SELECT producerC#
                   FROM movies, starsIn
                   WHERE movies.title = starsIn.title
                      AND movies.year = starsIn.year
                      AND starName = 'Harrison Ford'
                 ) Prod
WHERE cert# = Prod.producerC#
```

Eliminating Duplicates

- Use Distinct

```
SELECT DISTINCT name
FROM movies
```

- Warning: Invoking distinct is costly

Eliminating Duplicates

- Union, intersection, difference usually remove duplicates automatically
- If we do not want this, but bag semantics:
    - Use the keyword all

```
(SELECT title, year
FROM movies)
UNION ALL
(SELECT movieTitle AS title,
        movieYear AS year
 FROM
 starsIn);
```

Aggregate Functions

- COUNT
  - numeric and non-numeric data
  - null values excepted
- SUM, MIN, MAX, AVG - only numeric data

- Exercise: Find the number of different stars in the starsIn table

```
SELECT COUNT(DISTINCT name)
FROM starsIn
```

Aggregate Functions

- Find the combined net-worth of movieExecs

```
SELECT SUM(networth)
FROM movieExecs
```

- Find the average net-worth of movieExecs

```
SELECT ROUND(AVG(networth),2)
FROM movieExecs
```

Aggregate Functions

- Dealing if NULL values
    - IFNULL(EXPR1, EXPR2):
        - Gives EXPR1 if it is not NULL and EXPR2 if not
-
```
SELECT
    name,
    IFNULL(studio, 'not president') AS studio
FROM movieExecs;
```

Aggregate Functions

- COALESCE(EXPR1, EXPR2, EXPR3, … EXPRn)
    - Gives first nonNULL expression

Grouping

- Aggregation happens usually with grouping
  - To group, use GROUP BY followed by a WHERE clause

```
SELECT studioName, SUM(length) AS totalRunTime
FROM movies
GROUP BY studioName;
```

Grouping

- Example
  - Computing the total run time of movies produced by a producer

```
SELECT name, SUM(length) AS totalRunTime
FROM MovieExec, Movies
WHERE producerC# = cert#
GROUP BY name;
```

Grouping

- Aggregation and Nulls
  - NULL does not contribute to a sum, average, or count
- Grouping and Nulls
  - NULL is an ordinary value for grouping purposes
- Aggregation except COUNT over an empty bag gives result NULL