Relational Model of Data

Thomas Schwarz, SJ

Data Model

- Notation for describing data
  1. Structure of the Data
     - Conceptual level, not binary
  2. Operations on Data
     - Limits on operations are useful!
  3. Constraints on Data

Data Model

- In this class:
  - Relational Data Model
  - Semi-structured Data Model

Relational Data Model

- Relational Model is based on Tables

Title	Year	Length	Genre
Gone with the wind	1939	231	drama
Star wars	1977	124	scifi
Wayne’s world	1992	95	comedy

- The tables are conceptual
  - Implementations will differ

Relational Data Model

- Operations are part of Relational Algebra
- Constraints
    - On individual attributes
        - Non-null constraints
        - Unique constraints
    - On tuples

Semi-Structured Data Model

- Data is presented (in general) using trees or graphs
  - Popular formats
    - XML
    - JSON
  - Describing metadata stored with data

```json
{
  "hollywood": {
    "movies": [
      {
        "title": "Gone with the wind",
        "year": "1939",
        "length": "231",
        "genre": "drama"
      },
      {
        "title": "Star wars",
        "year": "1977",
        "length": "124",
        "genre": "scifi"
      },
      {
        "title": "Gone with the wind",
        "year": "1992",
        "length": "95",
        "genre": "comedy"
      }
    ]
  }
}
```

Object Oriented Data Model

- Values can have structure, not just primitive types
- Relations can have associated methods

- No consensus on how OO DBMS should look like
- Industry implemented Object-Relational DBMS
  - Values no longer need to be primitive

Relational Data Model

- Attributes:
    - Columns of a table are named by ***attributes***
- Schemas:
    - Name of a relation and the set of attributes
        - `Movies(title, year, length, genre)`
- Tuples:
    - Rows of a table (other than header row)
        - `(Gone with the wind, 1939, 231, drama)`

Relational Data Model

- Domains
  - All components of a tuple are *atomic*
  - All components of a tuple must be in *domain*

    -
      ```
      Movies(title:string, year:integer,
             length:integer, genre:string)
      ```

Relational Data Model

- Equivalent representation of a relation
  - Tables are sets of tuples
  - Attributes form a set
- Can reorder rows and columns, but obtain the same table

Year	Genre	Title	Length
1977	scifi	Star wars	124
1992	comedy	Wayne's world	95
1939	drama	Gone with the wind	231

Relational Data Model

- Keys:
    - A set of attributes (always non-null):
        - Two tuples cannot share the same values in this set
    - Example:
        - Movies Table:
            - title and year form a key
                - No two movies in the same year have both the same title and the same year
- `Movies(title, year, length, genre)`

Relational Data Model

- Keys
    - Can be single attribute:
        - All persons working in the US (should) have a Social Security Number (SSN)
        - SSN are unique
    - Can be artificial tuple ids
    - Are defined locally (Marquette ID numbers)

Relational Data Model

- Example

```
Movies(title: string,
       year: string,
       length: integer,
       genre: string,
       studioName: string,
       producerC#: integer)
```

Relational Data Model

- Example

```
MovieStar(name: string,
          year: string,
          address: string,
          gender: char,
          birthdate: date)
```

Relational Data Model

- Example

```
StarsIn(movieTitle: string,
        movieYear: int,
        starName: string)
```

Relational Data Model

- Example

```
MovieExec( name: string,
           address: string,
           cert#: integer,
           netWorth: integer)
```

Relational Data Model

- Example

```
Studio(name: string,
       address: string,
       presC#: integer)
```

SQL DDL

- Create a database with CREATE DATABASE

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

SQL DDL

- Three type of tables in SQL
  - Stored Relations, called tables
  - Views: relations calculated by computation
  - Temporary tables: created during query execution

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

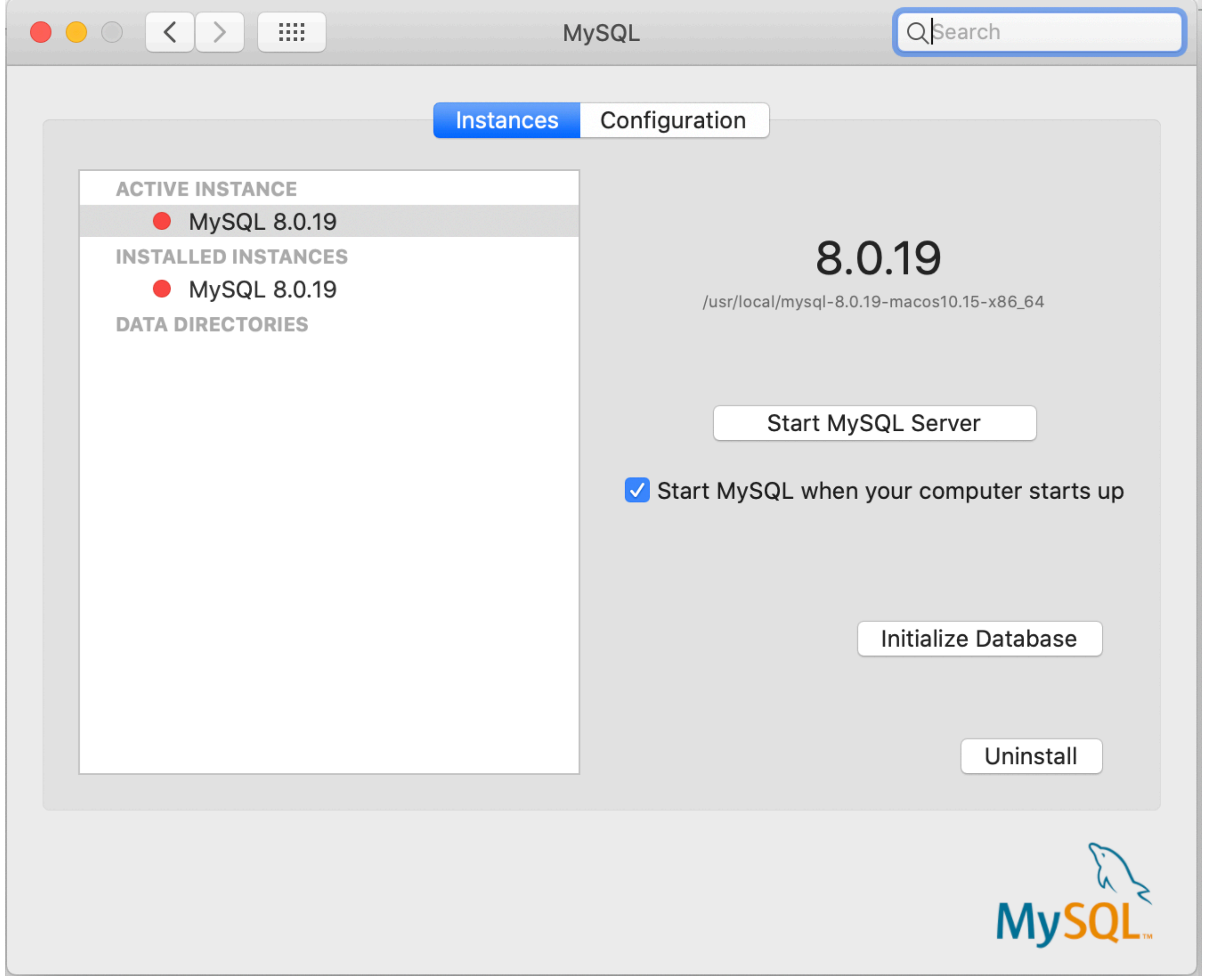
```
CREATE TABLE MovieStar(
   name            CHAR(30),
   address         VARCHAR(255),
   gender          CHAR(1) DEFAULT '?',
   birthday        DATE DEFAULT '0000-00-00',
   PRIMARY KEY (name)
);
```

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

- Insure that your mysql server is running
  - MAC : System Preferences —> MySQL



MySQL

ACTIVE INSTANCE

● MySQL 8.0.19

INSTALLED INSTANCES

● MySQL 8.0.19

DATA DIRECTORIES

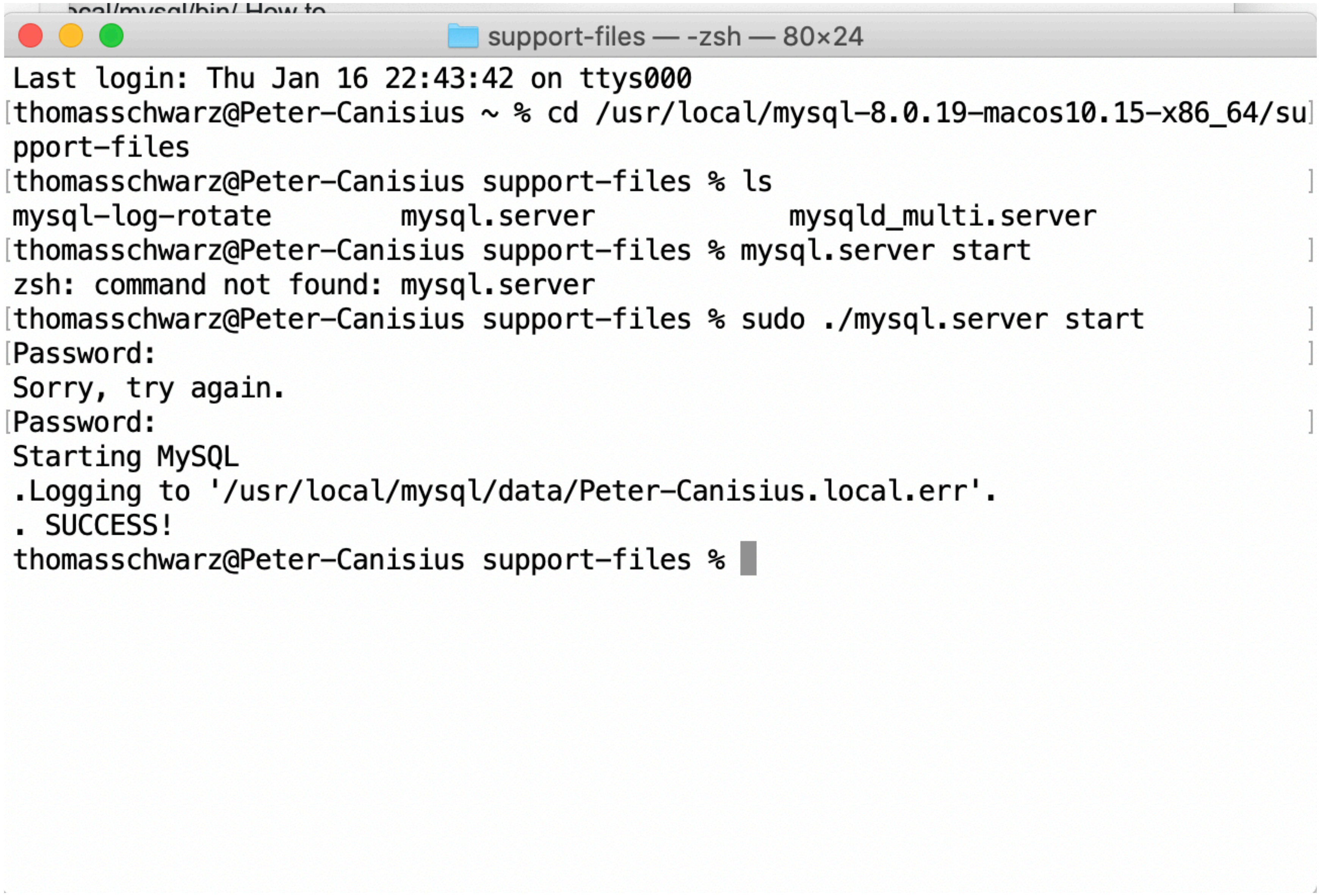
8.0.19

/usr/local/mysql-8.0.19-macos10.15-x86_64

☑ Start MySQL when your computer starts up

SQL Work Bench

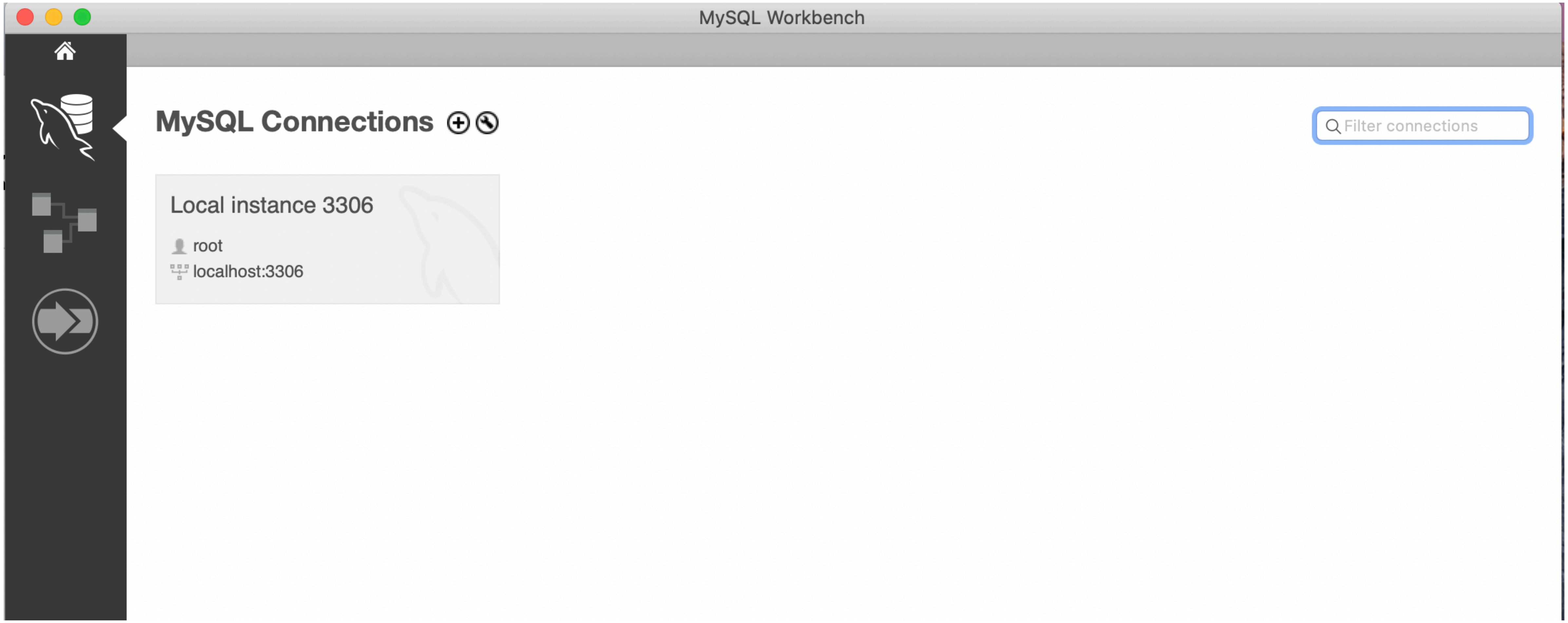
- Starting MySQL server through a terminal
    - Find mysql.server



```
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)



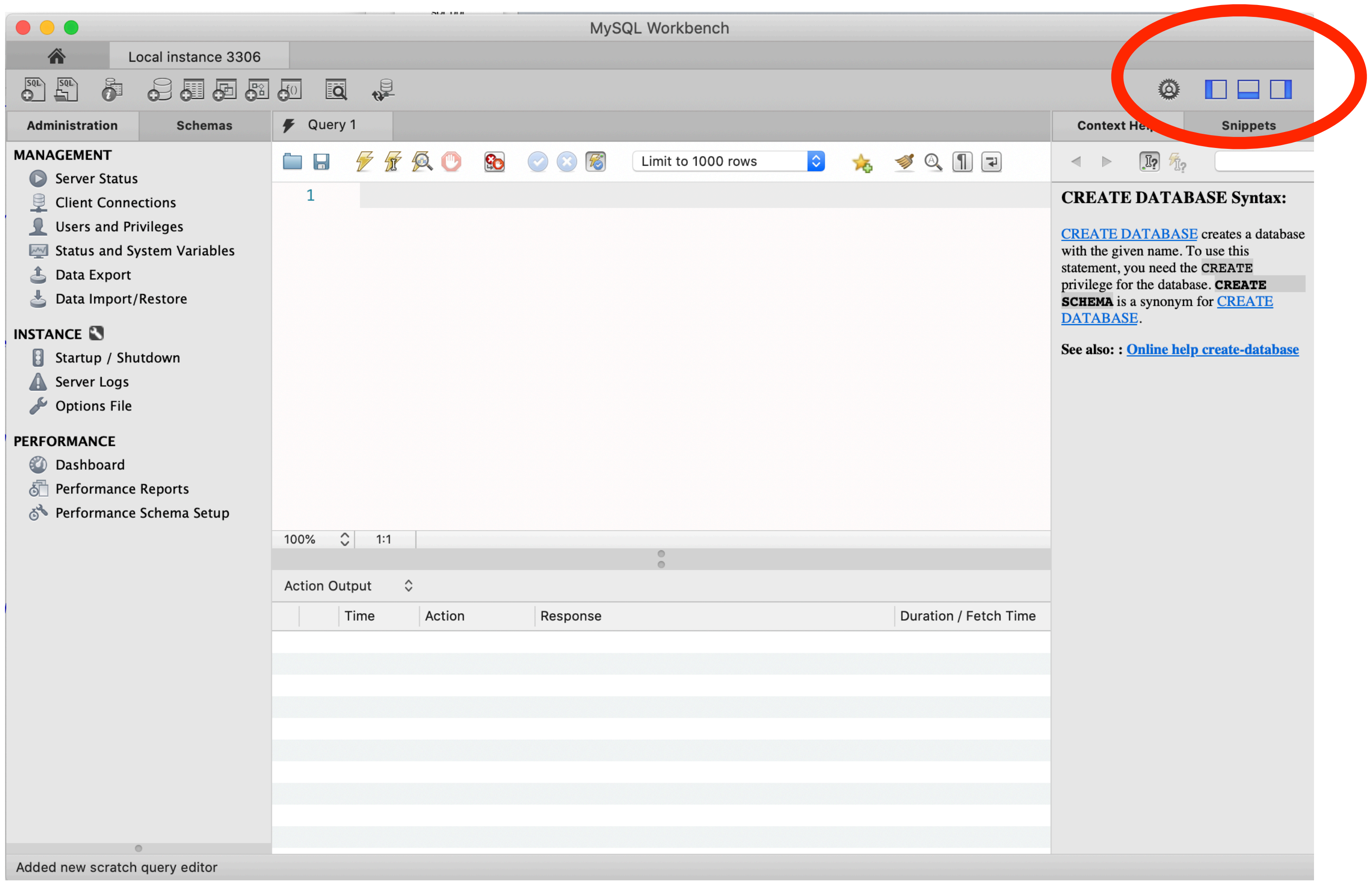
Local instance 3306

root

localhost:3306

SQL Workbench

- Select panels on the right



CREATE DATABASE Syntax:

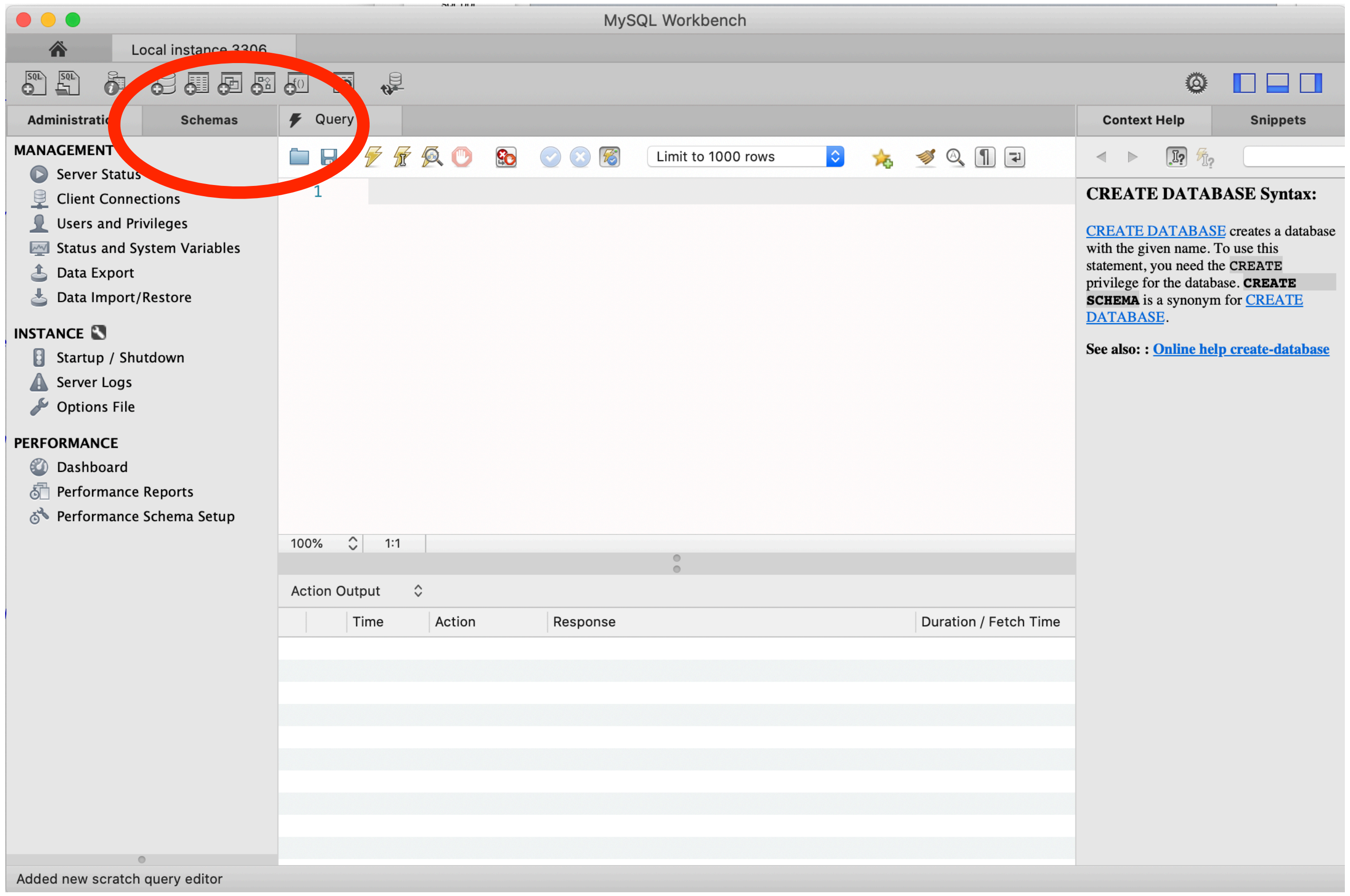
CREATE DATABASE creates a database with the given name. To use this statement, you need the `CREATE` privilege for the database. `CREATE SCHEMA` is a synonym for CREATE DATABASE.

See also: : Online help create-database

Added new scratch query editor

SQL Workbench

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



CREATE DATABASE Syntax:

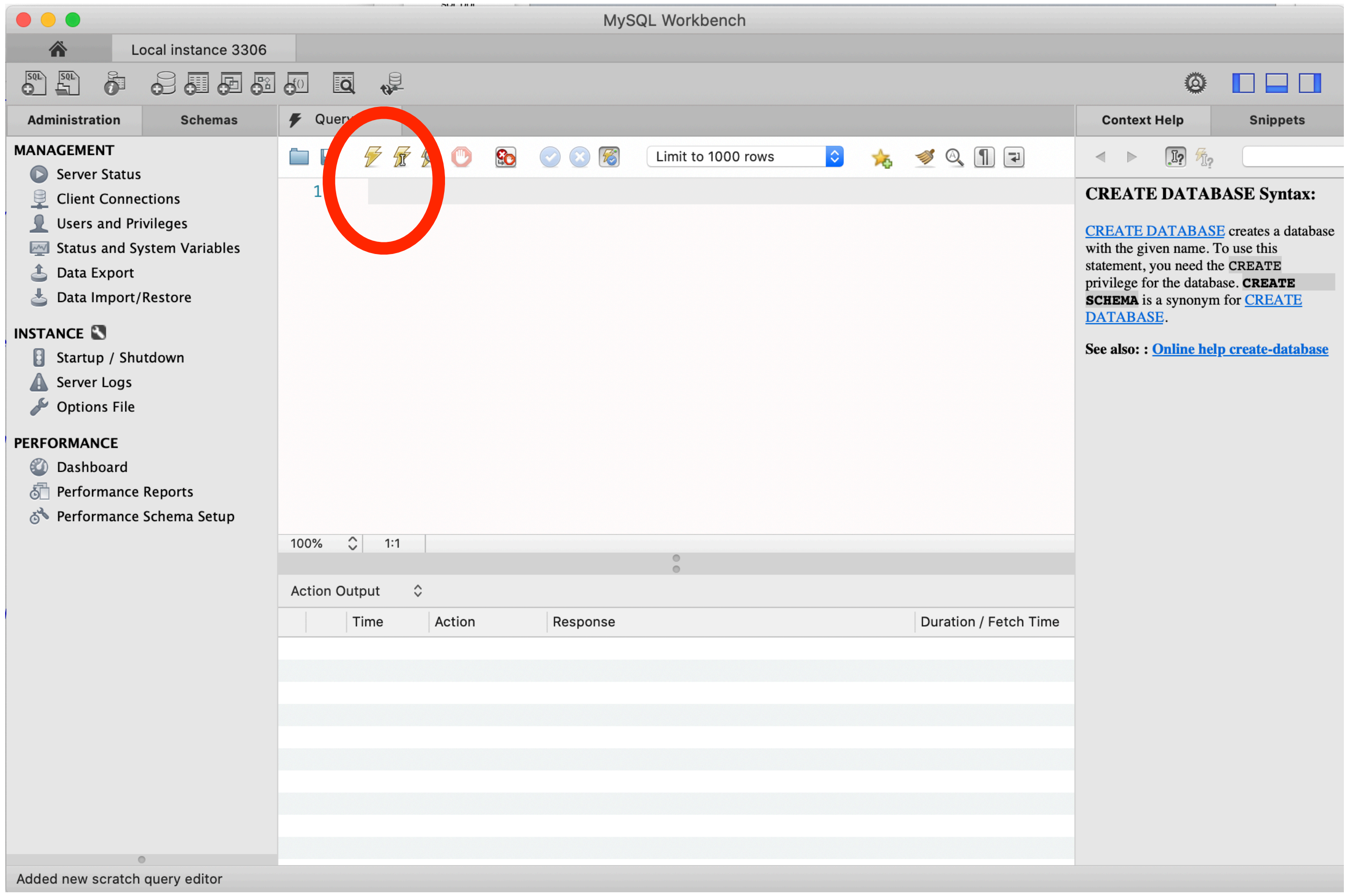
CREATE DATABASE creates a database with the given name. To use this statement, you need the `CREATE` privilege for the database. **`CREATE SCHEMA`** is a synonym for CREATE DATABASE.

See also: : Online help create-database

Added new scratch query editor

SQL Workbench

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



CREATE DATABASE Syntax:

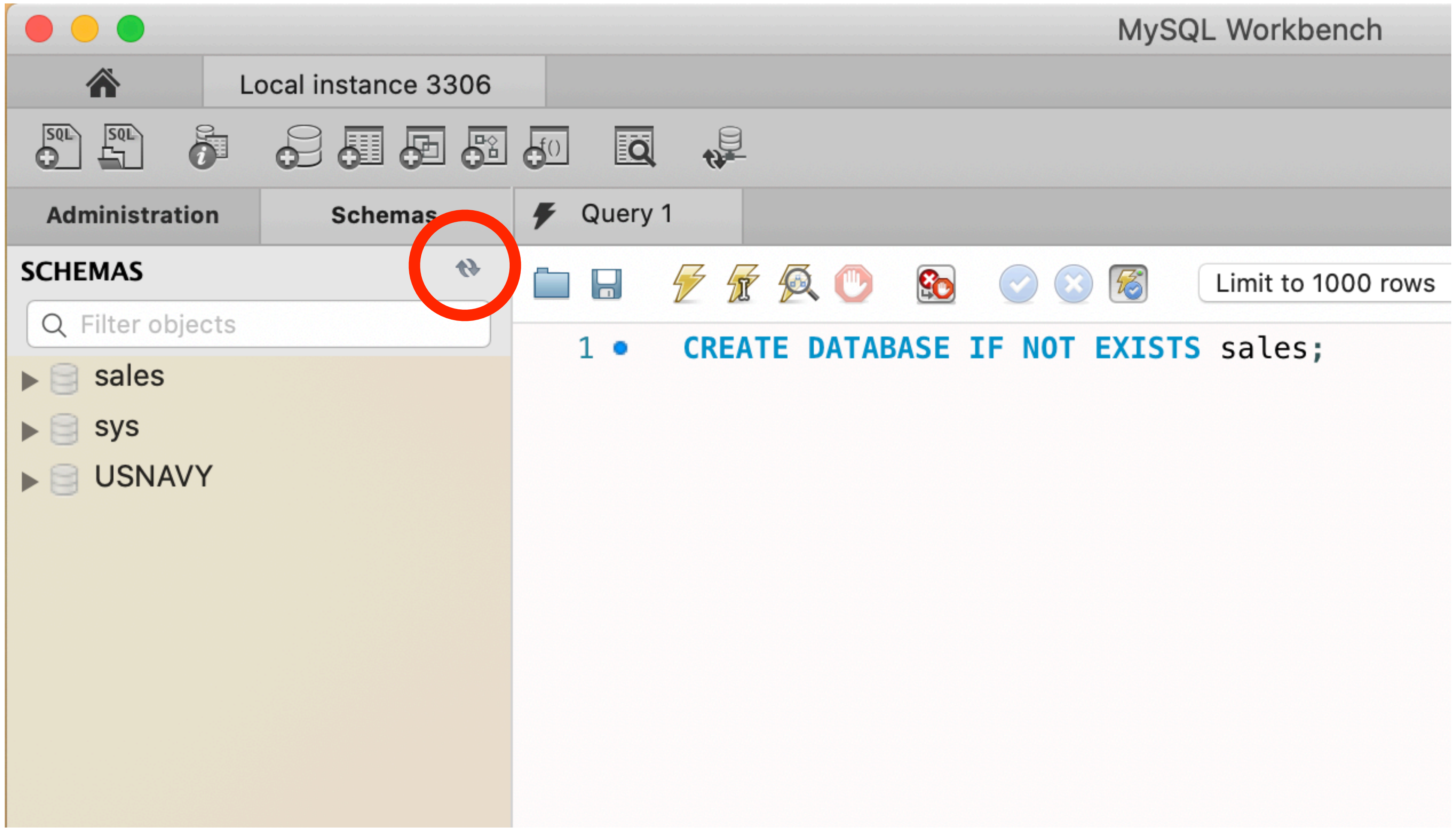
CREATE DATABASE creates a database with the given name. To use this statement, you need the CREATE privilege for the database. **CREATE SCHEMA** is a synonym for CREATE DATABASE.

See also: : Online help create-database

Added new scratch query editor

SQL Workbench

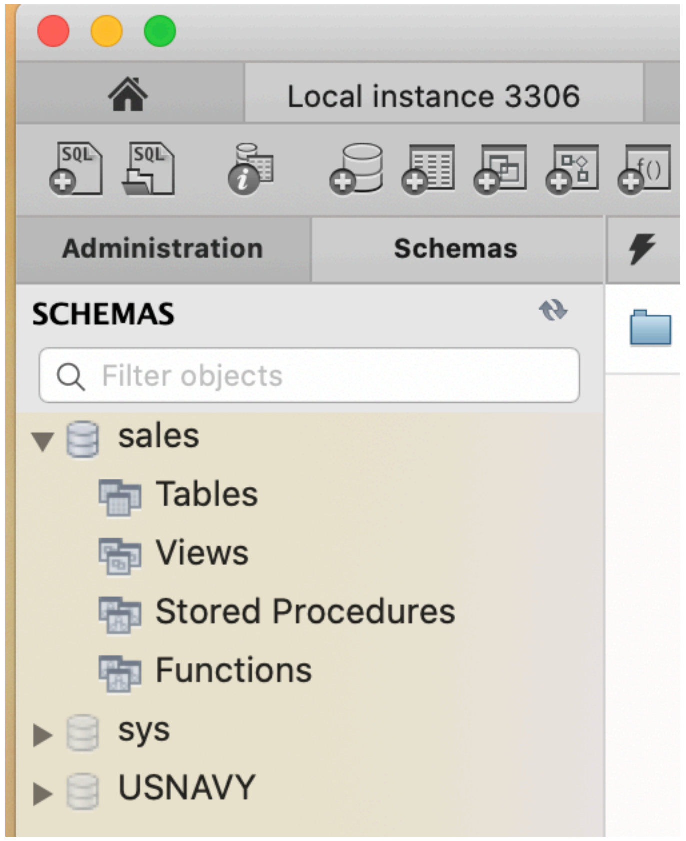
- After creating a database, need to update schemas in the upper right corner



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

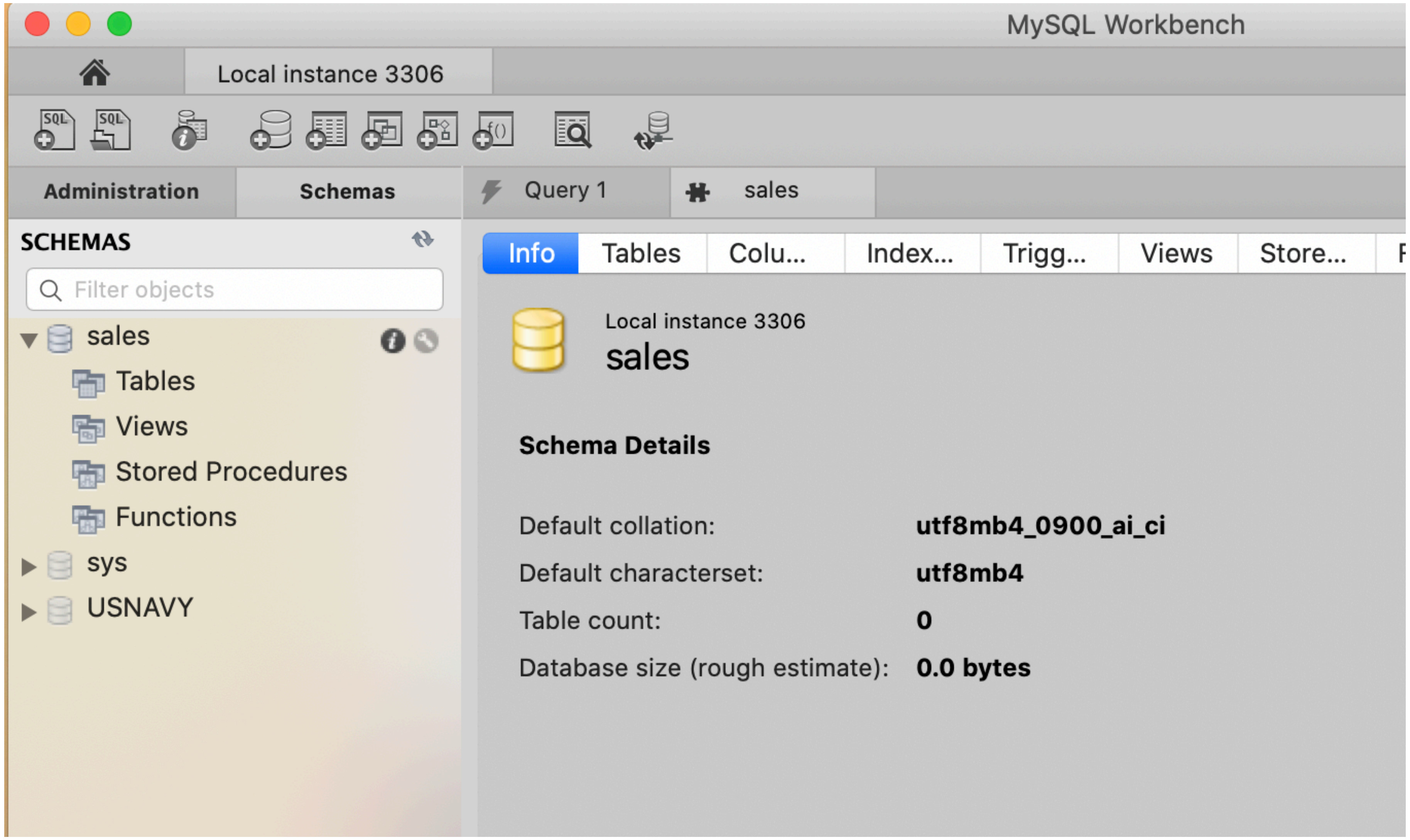
SQL Workbench

- 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

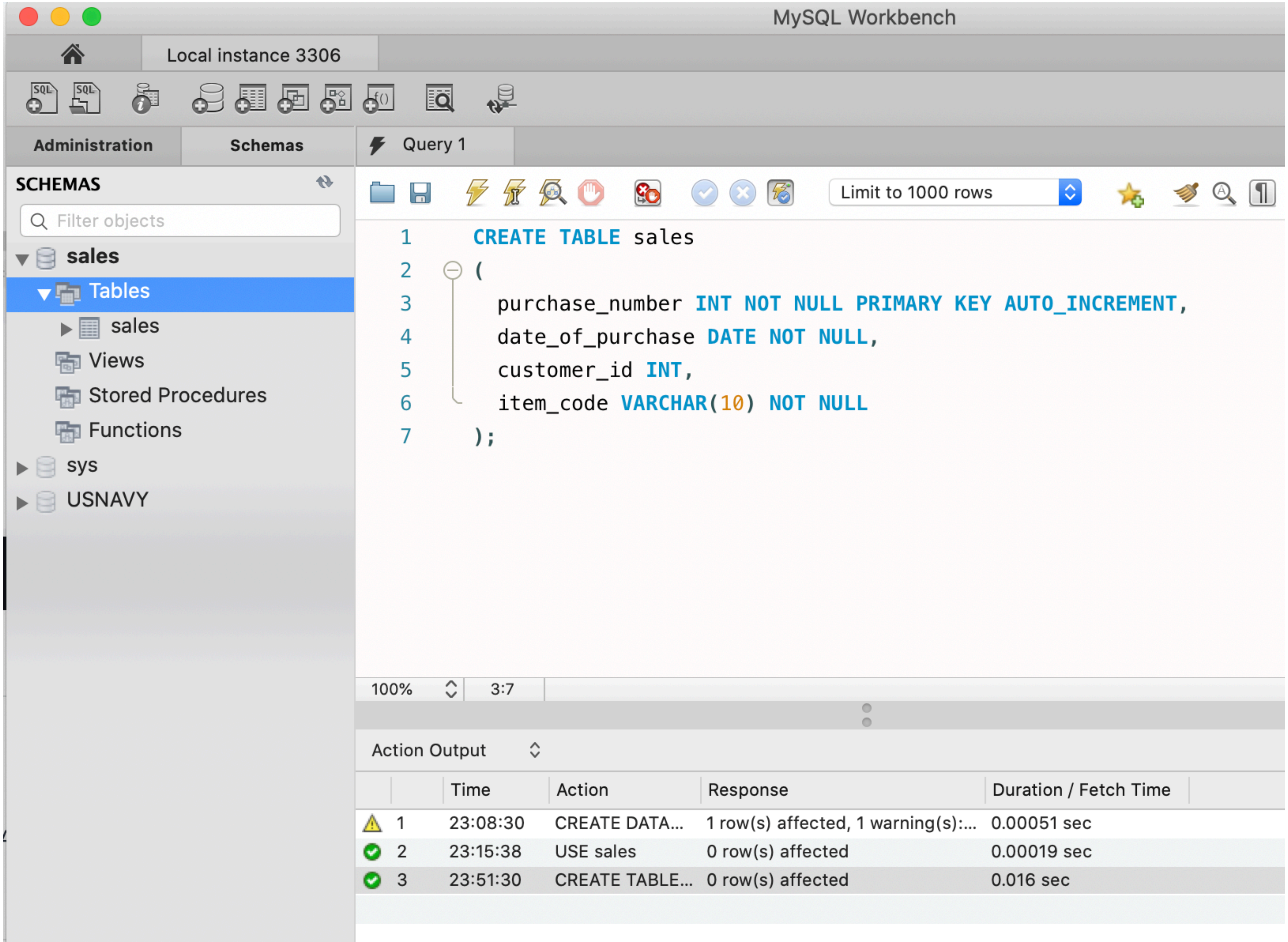
- Execute a query
    - `USE sales;`
- Now we can manipulate and use this database

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



```sql
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
);
```

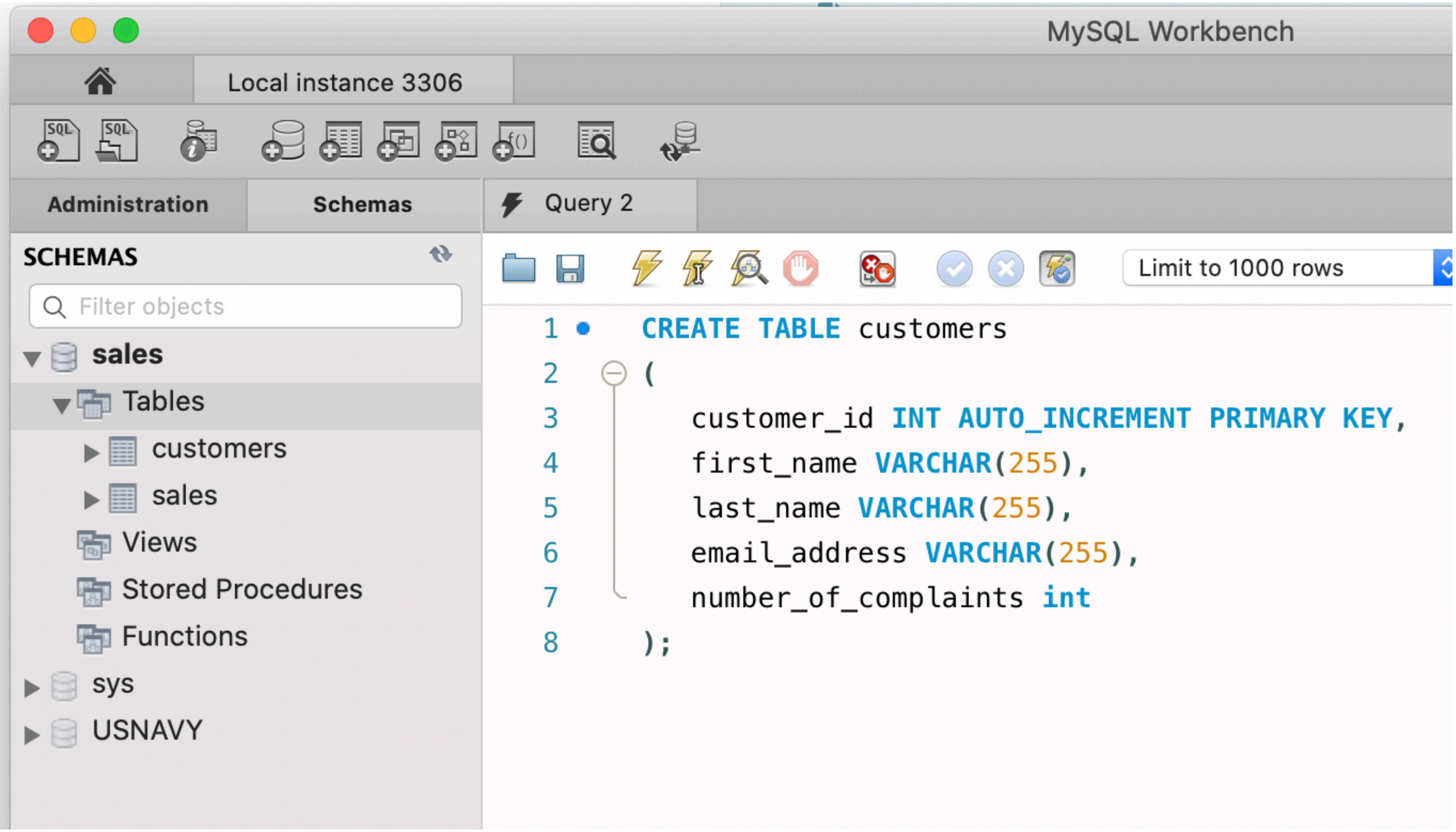
		Time	Action	Response	Duration / Fetch Time
⚠	1	23:08:30	CREATE DATA...	1 row(s) affected, 1 warning(s):...	0.00051 sec
✓	2	23:15:38	USE sales	0 row(s) affected	0.00019 sec
✓	3	23:51:30	CREATE TABLE...	0 row(s) affected	0.016 sec

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



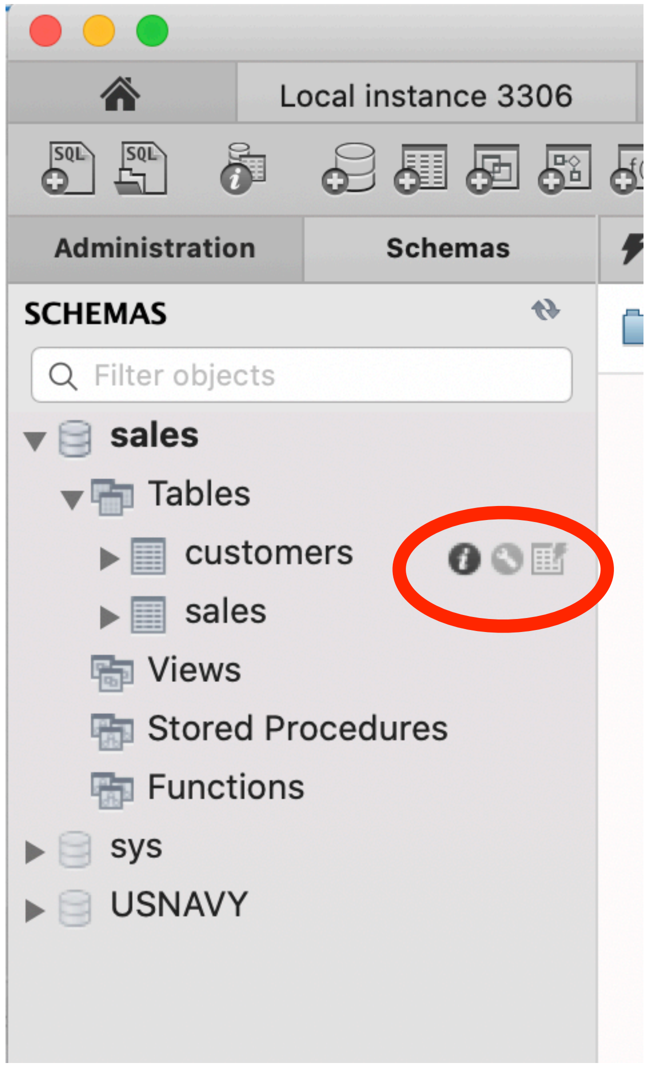
```sql
CREATE TABLE customers
(
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(255),
    last_name VARCHAR(255),
    email_address VARCHAR(255),
    number_of_complaints int
);
```

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;`

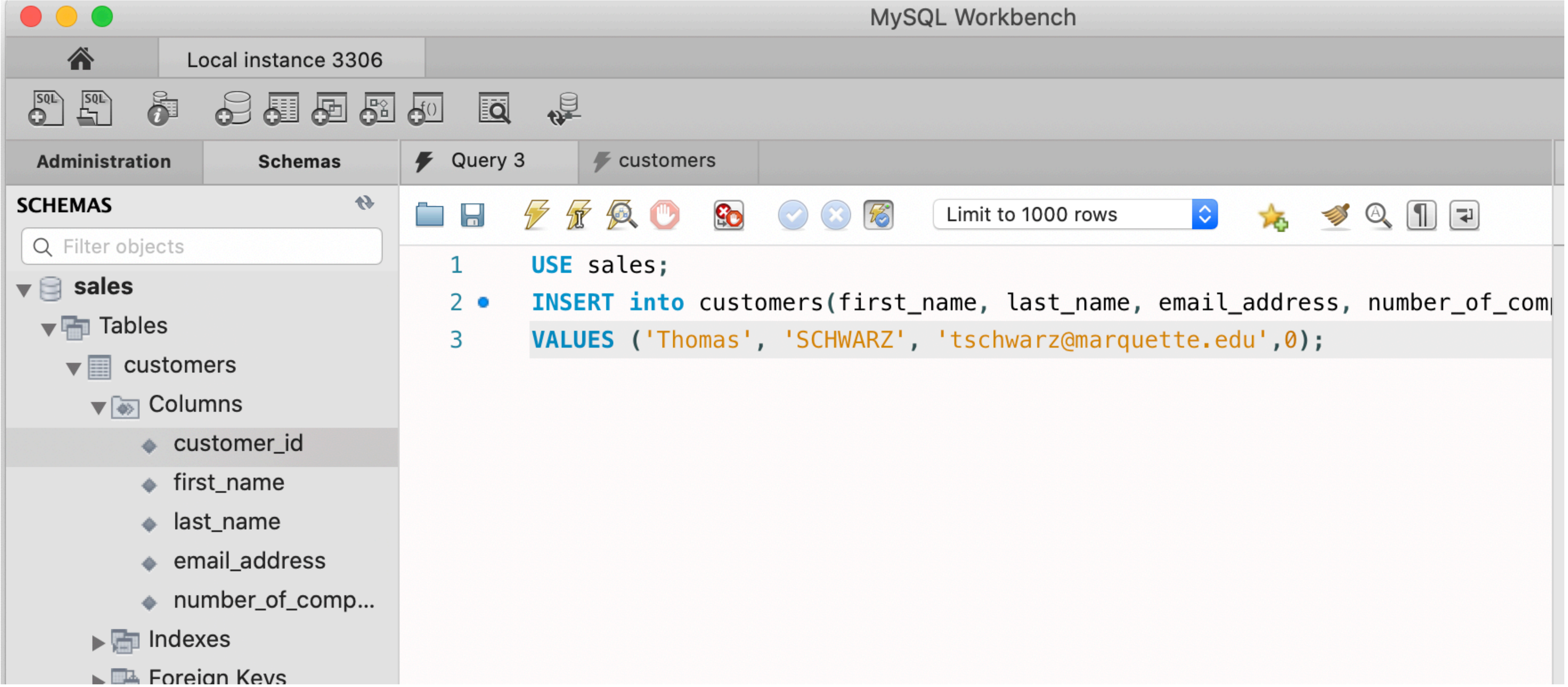
SQL Workbench

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



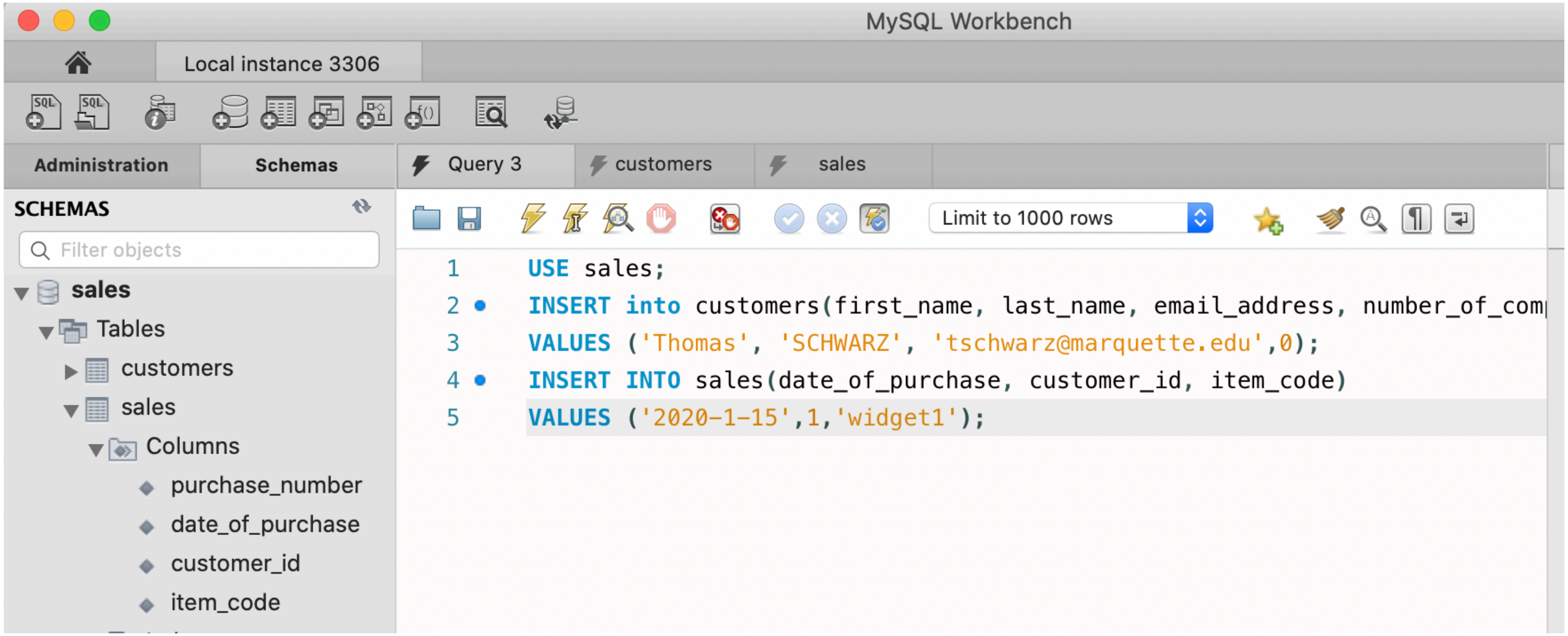
SQL Workbench

- Inserting into a data base:



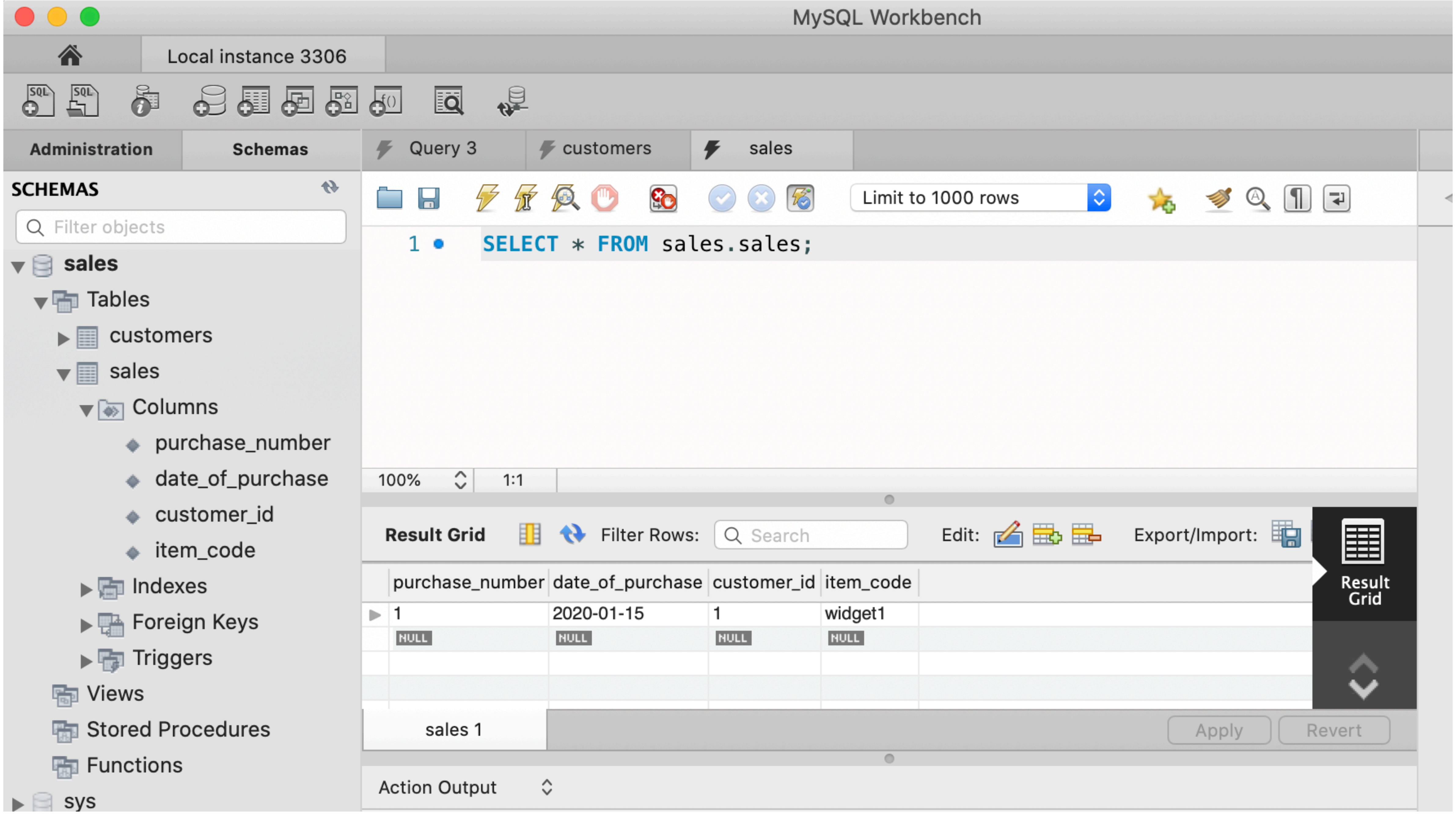
```
USE sales;
INSERT into customers(first_name, last_name, email_address, number_of_com
VALUES ('Thomas', 'SCHWARZ', 'tschwarz@marquette.edu',0);
```

SQL Workbench



```sql
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



Limit to 1000 rows

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

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

An Algebraic Query Language

- Set operations
- Selection (removes rows) and Projection (removes columns)
- Combination operations: Cartesian products, joins
- Renaming

An Algebraic Query Language

- Set operations for relations
  - Assume R and S are relations with
    - identical sets of attributes
    - ordered in the same way
  - $R \cup S$ Union
  - $R \cap S$ Intersection
  - $R - S$ Difference

An Algebraic Query Language

- Projection
  - Creates a new relation with a subset of the attributes
  - $\pi_{A_1,A_2,\ldots,A_n}(R)$ is the relation with values from R but only attributes $A_1, A_2, \ldots, A_n$

An Algebraic Query Language

- Selection
  - $\sigma_{\text{cond}}(R)$ is a relation with the same attributes, but only tupes from R that satisfy the condition

An Algebraic Query Language

- Cartesian product $R \times S$
  - Assumes that set of attributes are disjoint
  - The same as for sets
    - $R \times S = \{(r, s) \mid r \in R, s \in S\}$.

An Algebraic Query Language

- Natural join $R \bowtie S$
  - New relation with attributes that are attributes in one or the other relation
  - All combinations of tuples in R and S that agree on all common attributes.

An Algebraic Query Language

- Example:

R	A	B
	1	2
	3	4
	5	6

S	A	C
	1	4
	5	6

- Cartesian Product
  - Needs to make attributes disjoint

R x S	R.A	B	S.A	C
	1	2	1	4
	1	2	5	6
	3	4	1	4
	3	4	5	6
	5	6	1	4
	5	6	5	6

An Algebraic Query Language

- Example:

R

A	B
1	2
3	4
5	6

S

A	C
1	4
5	6

- Natural join: Common attribute is A
  - Look at common values

R ⋈ **S**

A	B	C
1	2	4
5	6	6

An Algebraic Query Language

- Example:

R	A	B	C
	1	1	2
	1	1	3
	1	2	4

S	A	B	D
	1	1	2
	1	1	5
	2	1	4
	2	2	1

- Natural join: All combinations of tuples with equal values for A and B:

$R \bowtie S$

A	B	C	D
1	1	2	2
1	1	2	5
1	1	3	2
1	1	3	5

An Algebraic Query Language

- Θ - Joins (Theta Joins)
  - Natural joins compares sub-tuples based on equality
  - Theta joins use arbitrary boolean conditions on attribute values
  - The condition is indicated as a subscript under the bowtie, usually named θ.

An Algebraic Query Language

- Example

R	A	B	C
	1	2	1
	0	1	2
	3	3	3

S	A	B	D
	1	1	5
	1	2	6
	4	1	2

- $R \bowtie_\theta S$ with $\theta = R.A \geq S.A$

$R \bowtie_\theta S$	A	R.B	S.B	C	D
	1	2	1	1	5
	1	2	2	2	6
	3	3	1	3	5
	3	3	2	3	6

An Algebraic Query Language

- Example

R	A	B	C
	1	2	1
	0	1	2
	3	3	3

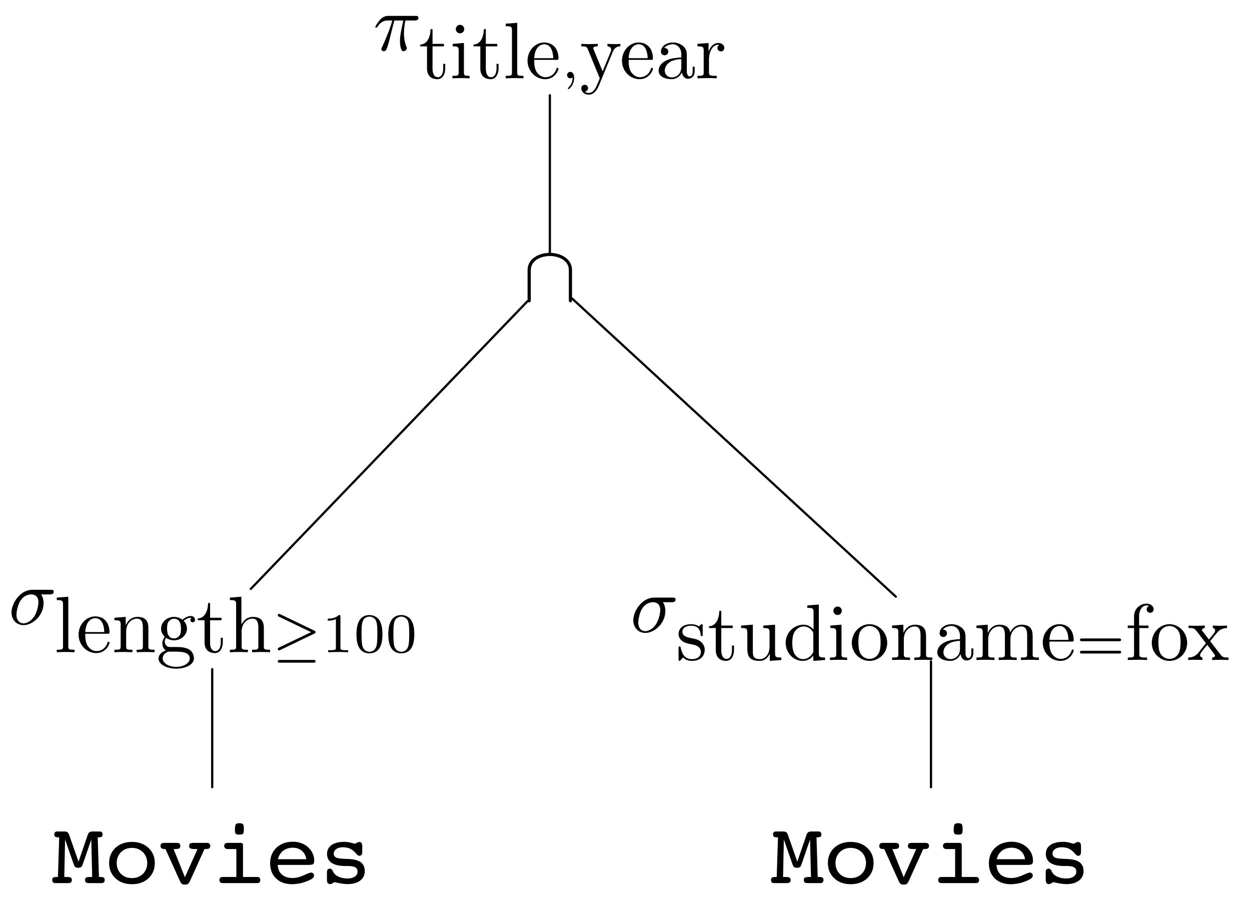
S	A	B	D
	1	1	5
	1	2	6
	4	1	2

- $R \bowtie_{\theta} S$ with $\theta = (R.A \neq S.A \text{ AND } R.B = S.B)$

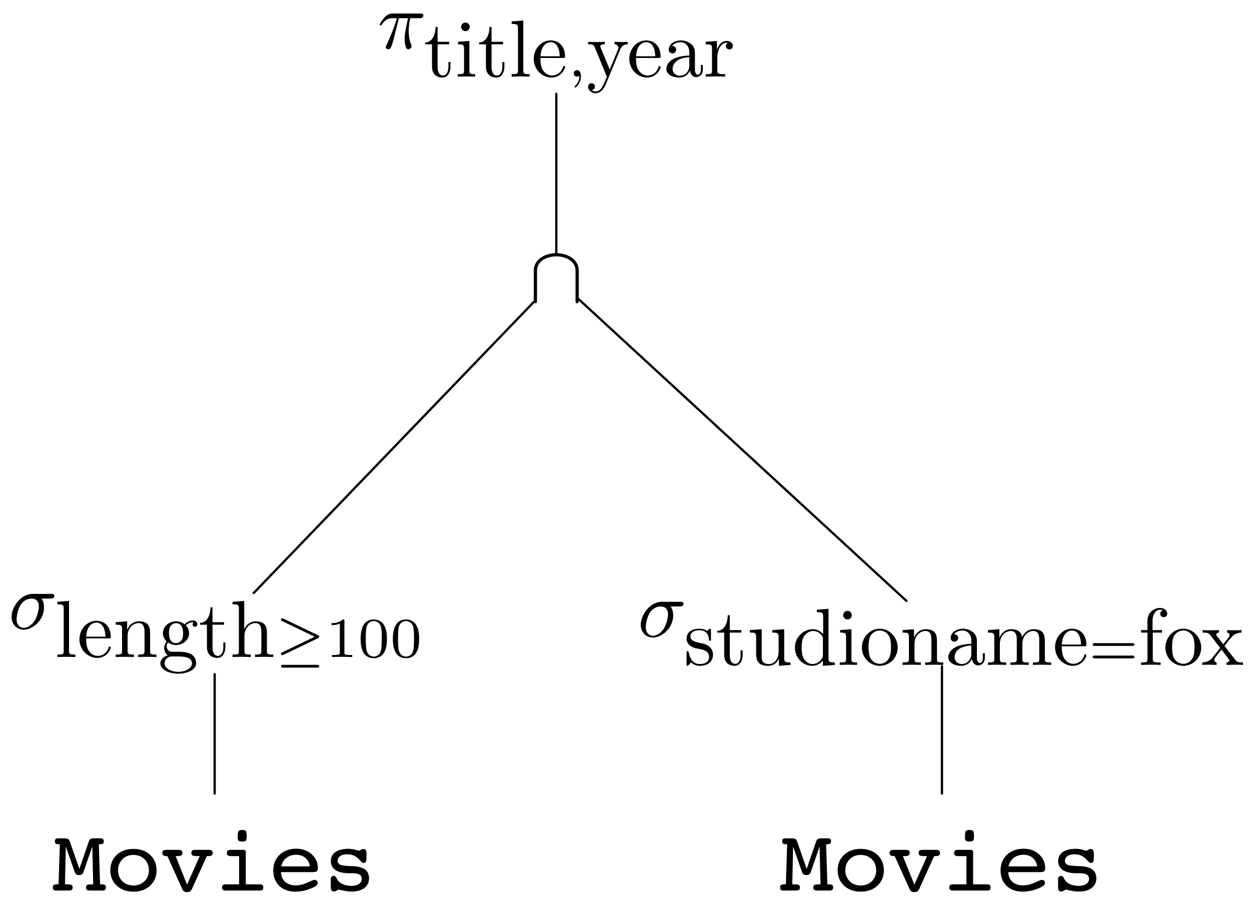
$R \bowtie_{\theta} S$	R.A	S.A	B	C	D
	0	1	2	1	6
	0	4	1	2	2

An Algebraic Query Language

- Combining operations to form queries
  - Example: What are the titles and years of movies made by Fox that are at least 100 minutes long


$\pi_{\text{title,year}}$
$\cap$
$\sigma_{\text{length}\geq 100}$
$\sigma_{\text{studioname}=\text{fox}}$
Movies
Movies

An Algebraic Query Language



$\pi_{\text{title,year}}$

$\cap$

$\sigma_{\text{length}\geq 100}$

$\sigma_{\text{studioname}=\text{fox}}$

Movies

Movies

$\pi_{\text{title,year}}(\sigma_{\text{length}\geq 100}(\text{movies}) \cap \sigma_{\text{studioname}=\text{fox}}(\text{movies}))$

An Algebraic Query Language

- Query Optimizer
    - First translate query to an expression tree
    - Then apply transformation rules to generate equivalent expression trees
        - with lower associated run-times

An Algebraic Query Language

- Naming and Renaming
  - Instead of following a convention, we use an explicit rename operator
  - $\rho_{S(A_1, A_2, \ldots, A_n)}(R)$ yields
    - A relation called S
      - with attributes called $A_1, A_2, \ldots, A_n$

An Algebraic Query Language

- Relationships among operations
    - There are some algebraic identities
        - $R \cap S = R - (R - S)$
        - $R \bowtie_C S = \sigma_C(R \times S)$
- This means that we can only use operations
    - Selection
    - Projection
    - Product
    - Renaming
    - Union
    - Difference

Constraints on Relations

- Constraints restrict the ability to insert data into relations
  - Necessary for maintaining data integrity

Constraints

- Expression in Relational Algebra
  - $R = \emptyset$ for any relational algebra expression R
  - $R \subseteq S$ for any relational algebra expressions R, S

Referential Integrity Constraint

- Referential Integrity constraints
  - An attribute value appearing in one relation should also be in another relation
    - Example: A star should be the star of at least one movie
  - $\pi_A(R) \subseteq \pi_A(S)$

Referential Integrity Constraint

- Selftest:
  - Movies(title, year, length genre, studioName, producerC#)
  - MovieExec(name, address, cert#, netWorth)
  - How do we insure that all producers appear in the MovieExec table?

Referential Integrity Constraint

- Answer
  - $\pi_{\text{producerC\#}}(\text{Movies}) \subseteq \pi_{\text{cert\#}}(\text{MovieExec})$

Referential Integrity Constraint

- Selftest
  - Any movie in
    - StarsIn(movieTitle, movieYear, starName)
  - needs to appear in
    - Movies(title, year, length genre, studioName, producerC#)

Referential Integrity Constraint

- Solution
  - $\pi_{\text{movieTitle, movieYear}}(\text{StarsIn}) \subseteq \pi_{\text{title, year}}(\text{Movies})$

Key Constraints

- Use Relational Algebra to express that an attribute is a key?
    - Any two tuples with the same value in the key must be the same
    - Create a Cartesian Product to get all pairs of tuples
        - Need to rename the copies for clarity
    - Then use a Select on the product

Key Constraints

- Example:
  - MovieStar(name, address, gender, birthday)

1. Create two copies
   - $\rho_{\text{MS1(name, address, gender, birthday}}(\text{MovieStar})$
   - $\rho_{\text{MS2(name, address, gender, birthday}}(\text{MovieStar})$
2. Make sure that name determines address

- $\sigma_{\text{MS1.name=MS2.name AND MS1.address}\neq\text{MS2.address}}(\text{MS1} \times \text{MS2}) = \emptyset$

3. Continue: name determines Gender
4. Continue: name determines birthday

Constraints

- Quiz:
  - Value constraint:
    - Gender in Moviestar can be only 'M', 'F', 'NB'

Constraints

- Quiz:
  - Given:
    - MovieExec(name, address, cert#, netWorth)
    - Studio(name, address, presC#)
  - Express the property rule that one can only be the president of a studio if the net worth exceeds US$10,000,000.00

Solution