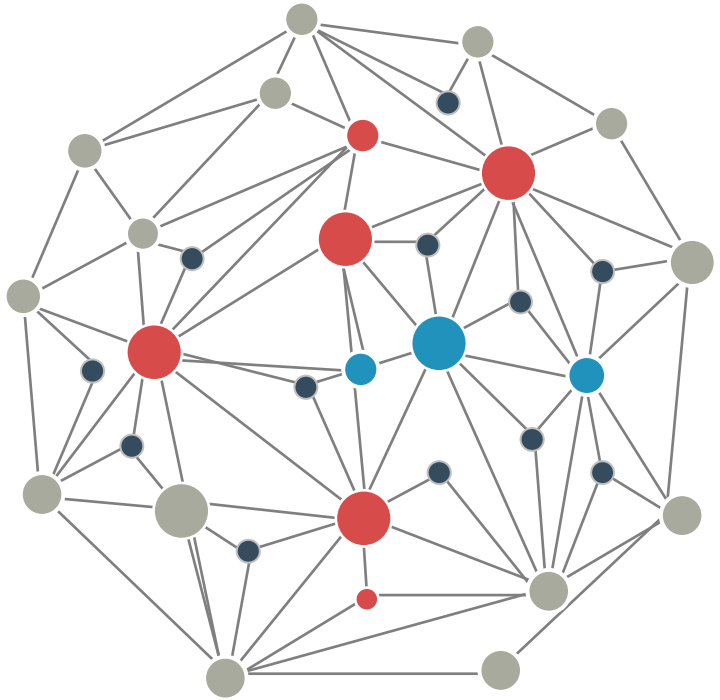
DST2 – Week 9

Structured Query Language (SQL)

Zhaoyuan Fang

zhaoyuanfang@intl.zju.edu.cn



2021/11/23

What we have learned

- Install course set-up environment (MySQL)
  - Take a look at **today's pre-lecture slides**

- The rationale for using database
- Differentiate data vs information vs knowledge
- Database management system (DBMS): history, structure
- Relation, tuples, attributes, keys
- Relational Algebra

Relational Algebra

- Unary operations
  - Select: σ
  - Project: $\prod$
  - Rename: ρ
- Set operations
  - Union: $\cup$; Intersection: $\cap$; Difference: –
  - Cartesian product: x
- Join operations
  - Natural join、Outer join
- Aggregate functions
  - Min, Max, Count, Sum, Avg

From Relational Algebra (RA) to SQL/MySQL

1. RA is mathematically strict, but SQL is NOT so strict!
   e.g. No duplicates in RA, but allowed in SQL (unless using the distinct keyword).

2. RA to SQL correspondences:
   1. *Select* --- where
   2. *Project* --- select
   3. *Rename* --- as

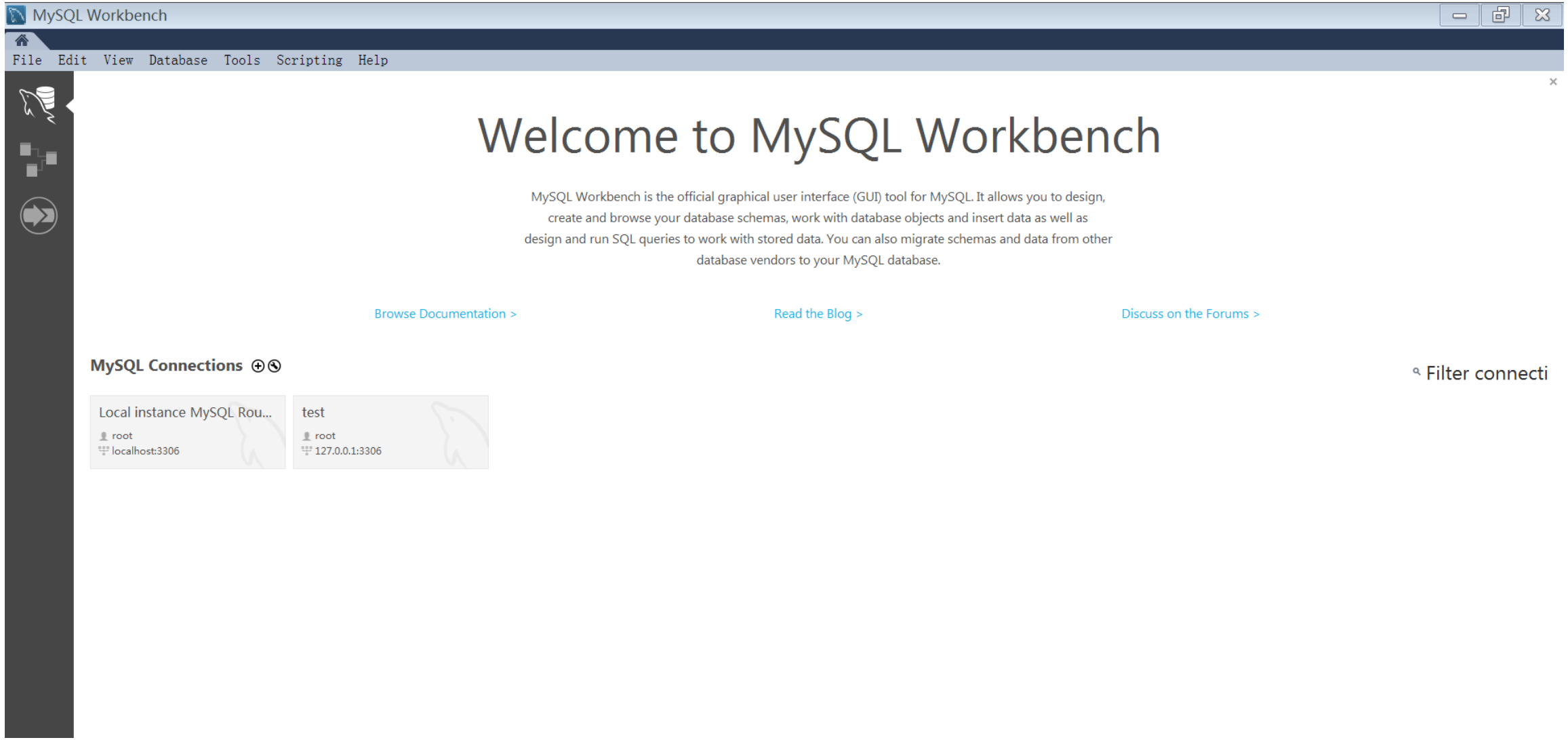
   4. *Natural join* --- natural join i.e. on identical attribute names
      (*Inner join* --- inner join) i.e. on different attribute names
   5. *Left/Right outer join* --- left/right (outer) join

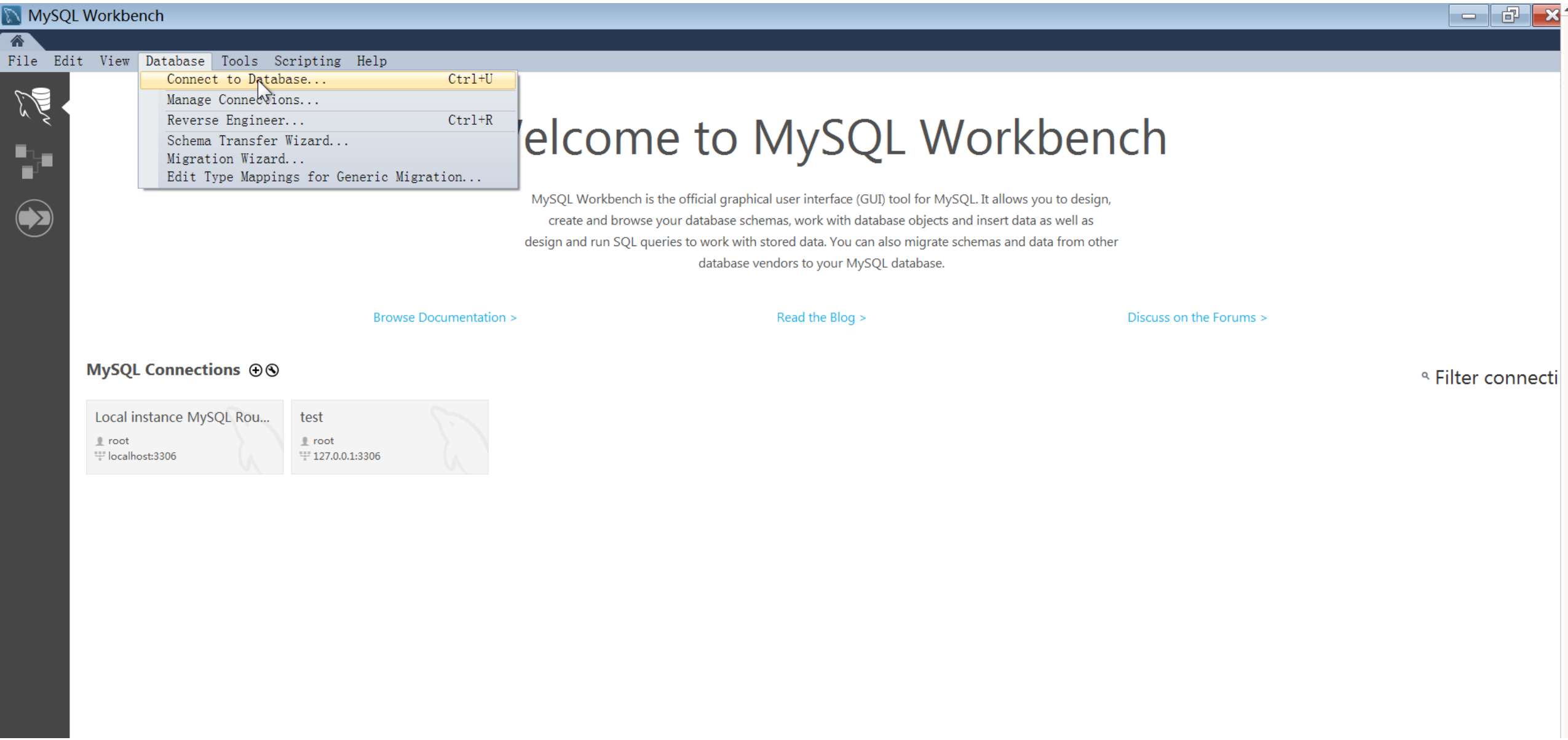
   6. *Union* --- union
   7. *Set intersection* --- where xxx in yyy *
   8. *Set difference* --- where xxx not in yyy *
   9. *Cartesian product* --- from relation1,relation2

   10. Aggregate functions --- min, max, count, sum, avg

* In MySQL, no intersect/except, so a bit different

MySQL Workbench 8.0 CE




MySQL Workbench
File Edit View Database Tools Scripting Help
Connect to Database... Ctrl+U
Manage Connections...
Reverse Engineer... Ctrl+R
Schema Transfer Wizard...
Migration Wizard...
Edit Type Mappings for Generic Migration...
elcome to MySQL Workbench
MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.
Browse Documentation >
Read the Blog >
Discuss on the Forums >
MySQL Connections
Filter connecti
Local instance MySQL Rou...
root
localhost:3306
test
root
127.0.0.1:3306

MySQL Workbench

File Edit View Database Tools Scripting Help

Welcome to MySQL Workbench

Connect to Database

Stored Connection: — Select from saved connection settings

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: root — Name of the user to connect with.

Password: Store in Vault ... 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.

OK Cancel

design,
l as
m other

Browse Documentatio

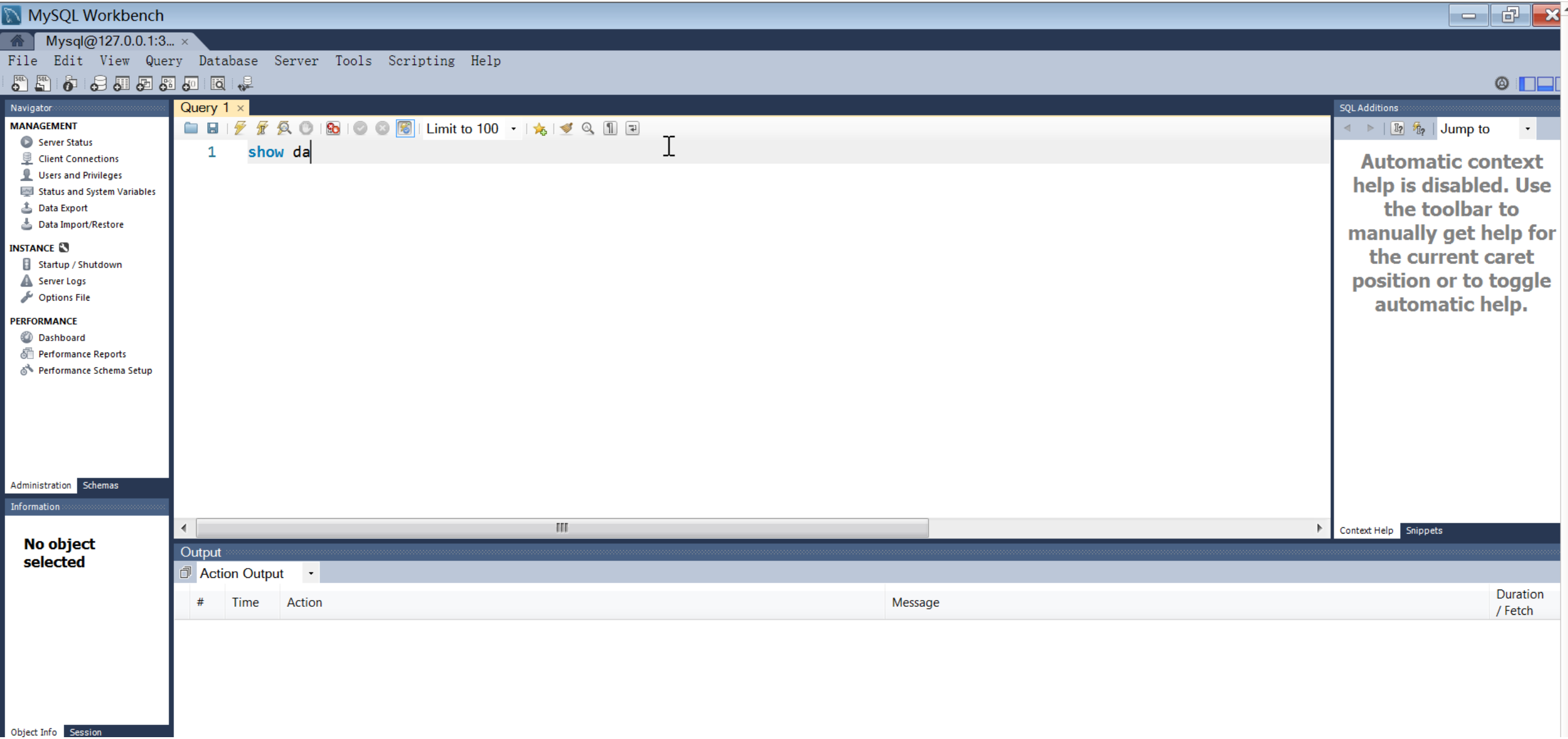
Discuss on the Forums >

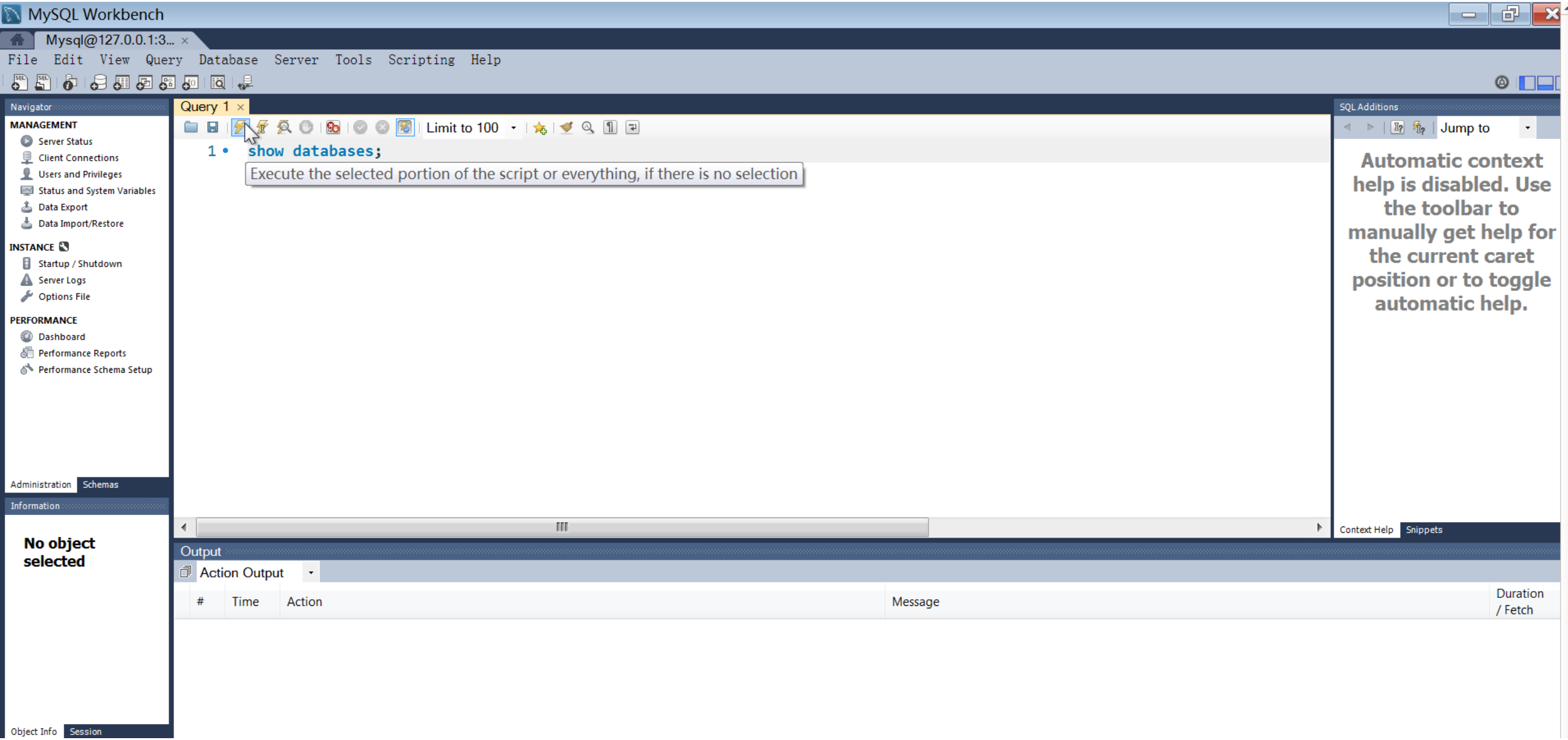
MySQL Connections

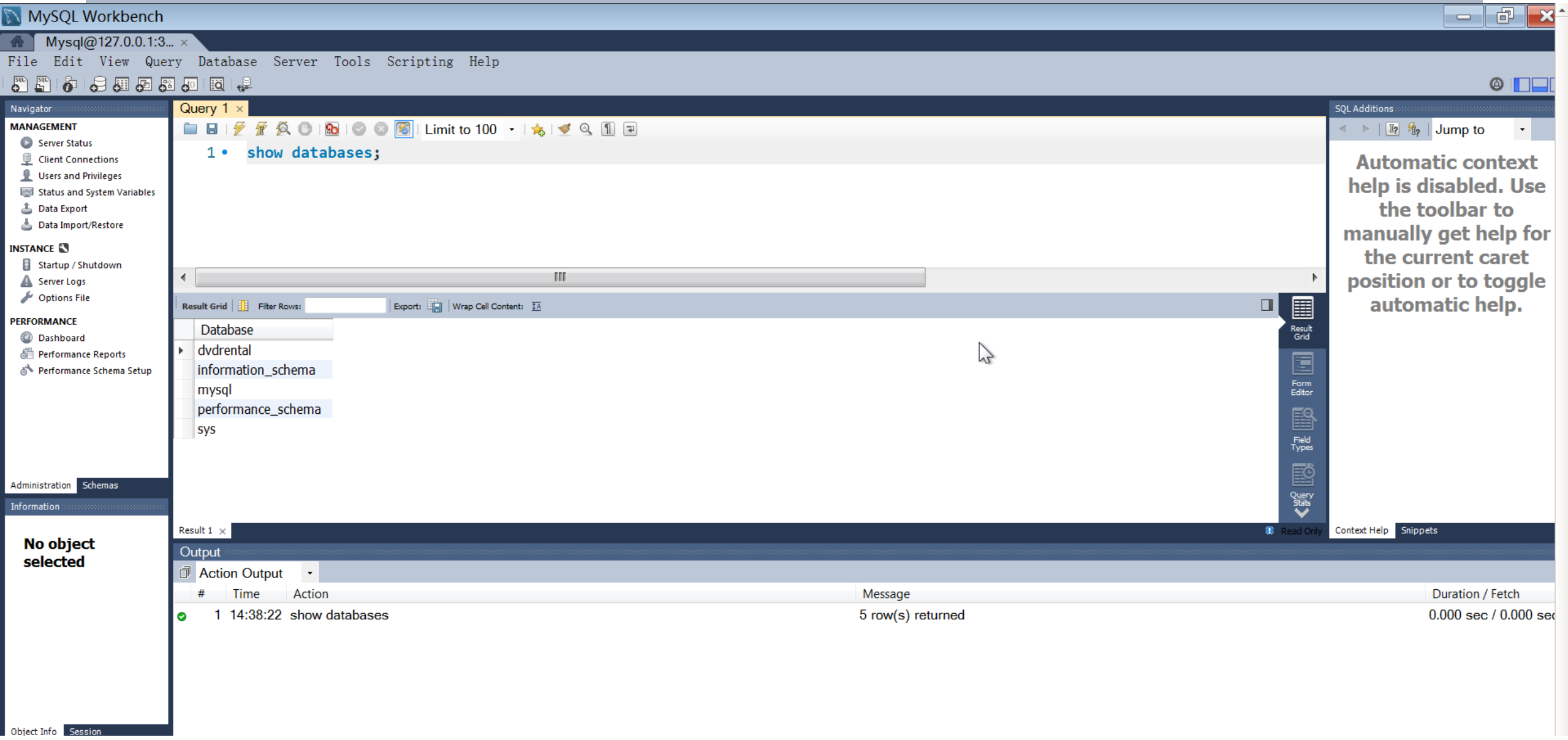
Filter connecti

Local instance MySQL Rou...
root
localhost:3306

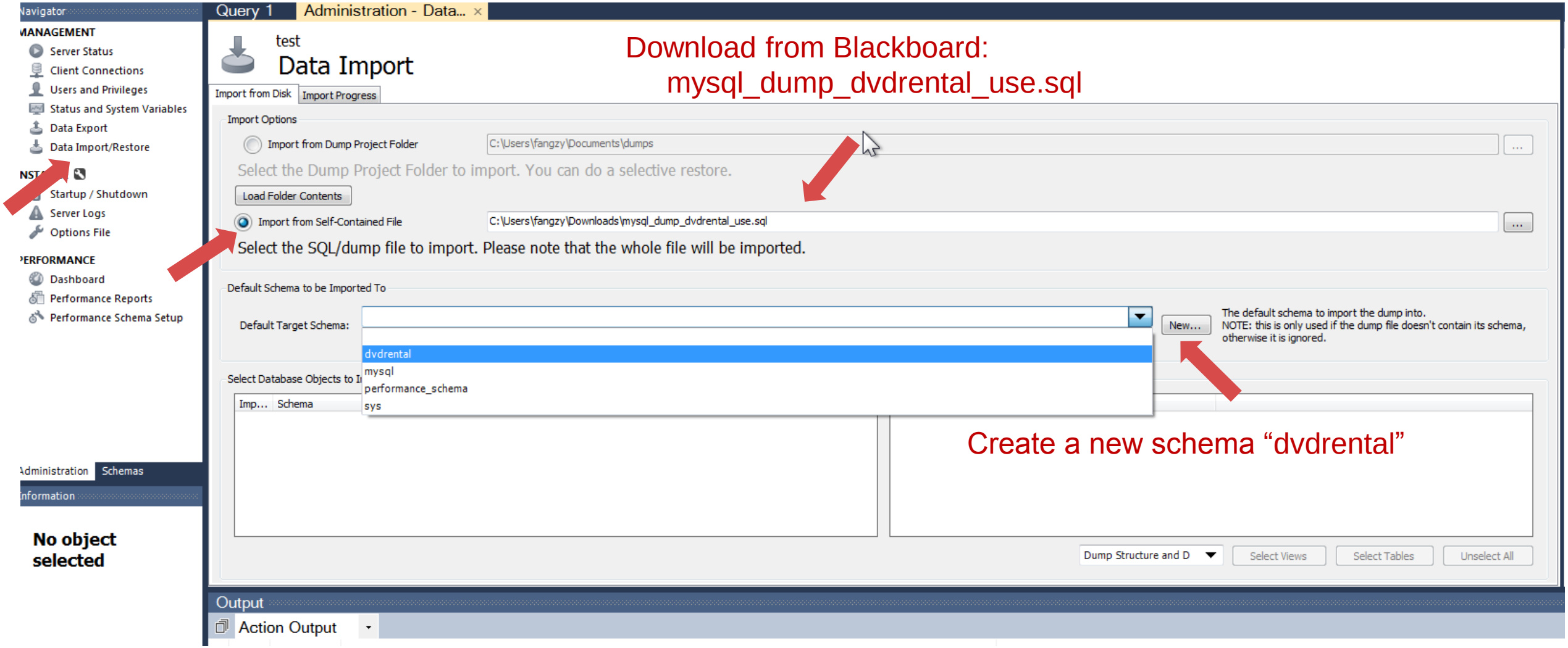
test
root
127.0.0.1:3306


MySQL Workbench
Mysql@127.0.0.1:3...
File Edit View Query Database Server Tools Scripting Help
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
Administration Schemas
Information
No object selected
Object Info Session
Query 1
Limit to 100
1 show da
SQL Additions
Jump to
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Context Help Snippets
Output
Action Output
Time Action Message Duration / Fetch


MySQL Workbench
Mysql@127.0.0.1:3...
File Edit View Query Database Server Tools Scripting Help
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
Administration Schemas
Information
No object selected
Object Info Session
Query 1
Limit to 100
1 • show databases;
Execute the selected portion of the script or everything, if there is no selection
SQL Additions
Jump to
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Context Help Snippets
Output
Action Output
Time Action Message Duration / Fetch


MySQL Workbench
Mysql@127.0.0.1:3...
File Edit View Query Database Server Tools Scripting Help
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
Administration Schemas
Information
No object selected
Object Info Session
Query 1
Limit to 100
1 • show databases;
Result Grid Filter Rows: Export: Wrap Cell Content:
Database
dvdrental
information_schema
mysql
performance_schema
sys
Result 1
Read Only
SQL Additions
Jump to
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Context Help Snippets
Output
Action Output
Time Action Message Duration / Fetch
1 14:38:22 show databases 5 row(s) returned 0.000 sec / 0.000 sec

Loading demo database into mysql



Alternatively, you can use DOS/CMD to load demo database into mysql

Open DOS/CMD: Windows DOS, Mac Terminal, Linux Shell

In DOS/CMD:

```
mysql -u root -p    (Login into MySQL, after adding path to environment variable)
mysql> create database dvdrental;    (Create a database schema for loading.
                                       must end with a semicolon ";")
mysql> quit;    (Quit MySQL to return to DOS/CMD)
mysql -u root -p   dvdrental <  mysql_dump_dvdrental_use.sql
                                (Load the demo database into MySQL,
                                or the direct path, e.g. C:\mysql….sql)
mysql -u root -p              (Login into MySQL again)
mysql> use dvdrental;     (Choose a database)
mysql> show tables;       (Show the tables in the database)
```

Week 9 Learning Objectives

- Retrieve specified columns of data from a database
- Join multiple tables in a single SQL query
- Restrict data retrievals to rows that match complex criteria
- Aggregate data across groups of rows
- Create subqueries to preprocess data for inclusion in other queries
- Explain the key principles in crafting a SELECT query

SQL queries

- At the heart of SQL is the query. A query is a spur-of-the-moment question. Actually, in the SQL environment, the word query covers both questions and actions. Most SQL queries are used to answer questions such as these:
  - "What products currently held in inventory are priced over $100, and what is the quantity on hand for each of those products?"
  - "How many employees have been hired since January 1, 2016, by each of the company's departments?"
- In most database-related jobs, retrieving data is by far the most common type of task.
- Data retrieval is done in SQL using a **SELECT** query.

Basic SELECT Queries

Each clause in a SELECT query performs a specific function.

- SELECT—selects the attributes (columns) to be returned
- FROM—specifies the table(s) from which the data will be retrieved
- WHERE—chooses the rows based on a conditional expression
- GROUP BY—groups the selected rows based on one or more attributes
- HAVING—chooses the grouped rows (by GROUP BY clause) based on a condition
- ORDER BY—orders the selected rows based on one or more attributes

SQL language format convention

SELECT *columnlist*
FROM *tablelist*;

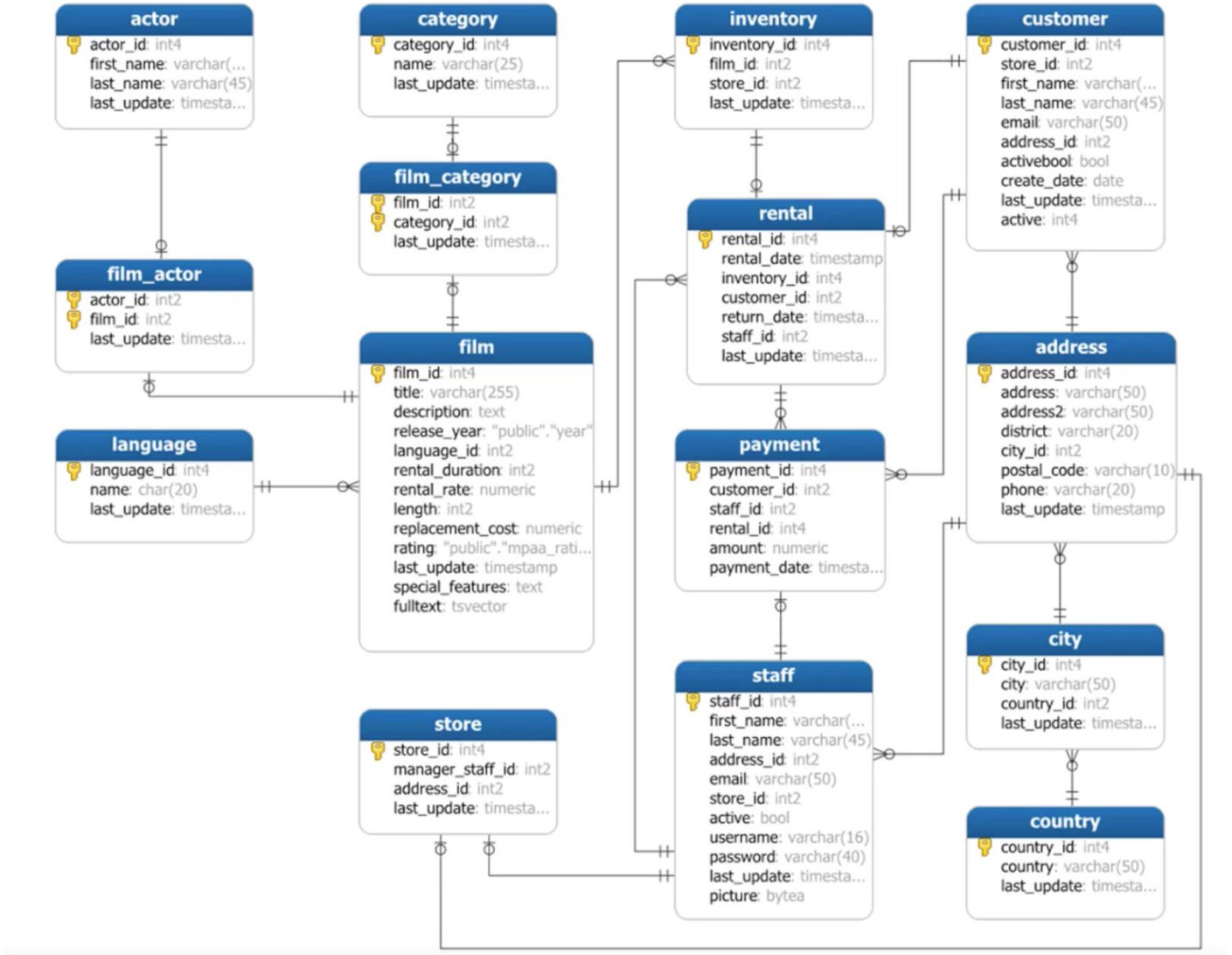
SELECT *columnlist* FROM *tablelist*;

select *columnlist* from *tablelist*;

Using that formatting convention makes it much easier to see the components of the SQL statements, which in turn makes it easy to trace the SQL logic and make corrections if necessary.

- with **space** between the SQL command and the command's components.
- The number of spaces used in the indention is up to you.
- Even though lower case for keywords like 'select' would work, but we usually use **upper case** to make the code easier to read.
- complex command sequences are best shown on **separate lines.**

The demo database



with some revisions

SELECT Statement options

The SELECT query specifies the columns to be retrieved as a column list. The syntax for a basic SELECT query that retrieves data from a table is:

```
SELECT columnlist
FROM tablelist;
```

The *columnlist* represents one or more attributes, separated by commas.

If you wants all of the columns to be returned, then the asterisk (*) wildcard can be used. A **wildcard character** is a symbol that can be used as a general substitute for other characters or commands. This wildcard means "all columns."

DEMO:

```
SELECT *
FROM actor;
```

Task 1 :

Select actor id, first name, last name from actor table

```
SELECT actor_id,first_name,last_name
FROM actor;
```

SELECT Statement options

SELECT - aliases

If you wants a different name to be used as the label in the output, a new name can be specified. The new name is referred to as an **alias**.

```
SELECT columnlist  AS newname
FROM tablelist;
```

- Not all columns in a query must use an alias
- AS is optional, but recommended
- Aliases that contain a space must be inside a delimiter (quotes)

DEMO:

Select actor id, first name, last name from actor table and rename actor id to ‘id’ and first name to ’first name’

```
SELECT actor_id AS id, first_name AS “first name”,last_name
FROM actor;
```

SELECT Statement options

SELECT limit

The SQL SELECT LIMIT statement is used to retrieve records from one or more tables in a database and limit the number of records returned based on a limit value.

```
SELECT columnlist
FROM tablelist
LIMIT number_rows;
```

Task 2 :

select the first 5 actors (id first_name, last_name)

```
SELECT actor_id, first name,last_name
FROM actor
LIMIT 5;
```

SELECT Statement options

SELECT – computed columns

A computed column (also called a calculated column) represents a derived attribute. For example, suppose that you want to know the length of a film in seconds.

DEMO:

```
SELECT film_id, title, length, length*60 AS length_in_secs
FROM film
LIMIT 5;
```

- If you forget AS, the new column would not be labeled automatically.
  e.g. length*60

SELECT Statement options

SELECT – Arithmetic Operators: The Rule of Precedence

As you saw in the previous example, you can use arithmetic operators with table attributes in a column list or in a conditional expression. In fact, SQL commands are often used in conjunction with the arithmetic operators shown in

OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide
^	Raise to the power of

rules of precedence

1. Perform operations within parentheses.
2. Perform power operations.
3. Perform multiplications and divisions.
4. Perform additions and subtractions.

DEMO:

suppose the renting cost is 10+rental_duration*2, compute it.

```
SELECT film_id, title, length, 10+rental_duration*2 AS cost
FROM film
LIMIT 5;
```

SELECT Statement options

SELECT – Timestamp Arithmetic

For timestamp, we use interval to add or subtract time from timestamp.

In mysql, intervals could be: “interval x day”, interval 10 hours, etc.

DEMO:

```
SELECT payment_date, payment_date + interval 3 day AS payment_duration
FROM payment;
```

Task 3 :

Assume the payment_date is recored in LA time, and now we want to know the payment_date timestamps in Bejing (+16 hours). Generate a new attribute called payment_date_Beijing.

```
SELECT payment_date, payment_date + interval 16 hour AS payment_date_Beijing
FROM payment;
```

SELECT Statement options

SELECT – Listing Unique Values

How many *different* vendors are currently represented in the PRODUCT table? A simple listing (SELECT) is not very useful if the table contains several thousand rows and you have to sift through the vendor codes manually. Fortunately, SQL's **DISTINCT** clause produces a list of only those values that are different from one another. For example, the command

```
SELECT DISTINCT vendor_name FROM product;
/* not for running */
```

Task 4 :

different films have different rating, we want to know what unique ratings are out there, try to use SELECT DISTINCT to find it out

```
SELECT DISTINCT rating
FROM film;
```

ORDER BY Clause options

The **ORDER BY** clause is especially useful when the listing order is important to you.
The syntax is:

```
SELECT    columnlist
FROM      tablelist
[ORDER BY columnlist [ASC|DESC]];
```

DEMO: we want to find out the film information ordered by the rental_duration

```
SELECT *
FROM film
ORDER BY rental_duration
LIMIT 5;
```

Task 5 :

find out rental_id for each rental, and then order them by the payment amount in descending orders

```
SELECT rental_id
FROM payment
ORDER BY amount DESC
LIMIT 5;
```

ORDER BY Clause options

Ordered listings are used frequently. For example, suppose that you want to create a phone directory. It would be helpful if you could produce an ordered sequence (last name, first name, initial) in three stages:

1. ORDER BY last name.
2. Within matching last names, ORDER BY first name.
3. Within matching first and last names, ORDER BY middle initial.

Such a multilevel ordered sequence is known as a **cascading order sequence**, and it can be created easily by listing several attributes, separated by commas, after the ORDER BY clause.


DEMO:

```
SELECT *
FROM actor
ORDER BY first_name, last_name, actor_id;
```


Task 6 :

Order the title of the most popular films in each language and each rating.

```
SELECT title, rental_rate, language_id, rating
FROM film
ORDER BY rental_rate DESC, language_id, rating;
```

WHERE Clause options

SELECT is an incredibly powerful tool that enables you to transform data into information.

With WHERE clause, you can create queries that can answer questions such as these:

"*What products were supplied by a particular vendor?*"

"*Which products are priced below $10?*"

"*How many products supplied by a given vendor were sold between January 5, 2018, and March 20, 2018?*"

WHERE Clause options

Selecting Rows with Conditional Restrictions

You can select partial table contents by placing restrictions on the rows to be included in the output. Use the **where** clause to add conditional restrictions to the SELECT statement that limit the rows returned by the query. The following syntax enables you to specify which rows to select:

```
SELECT columnlist
FROM    tablelist
[WHERE conditionlist]
[ORDER BY columnlist [ASC | DESC]];
```

DEMO:

Let's find out the films with rental duration of 5.

```
SELECT *
FROM film
WHERE rental_duration=5;
```

DEMO:

Let's find out the films with R rating.

```
SELECT *
FROM film
WHERE rating = 'R';
```

Task 7 :

Let's find out the films less than 1 hrs.

```
SELECT *
FROM film
WHERE length<=60;
```

symbol	meaning
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> Or !=	Not equal to

WHERE Clause options

Using Comparison Operators on Dates

Date procedures are often more software-specific than other SQL procedures.

In MySQL, we compare timestamp like:

DEMO:

```
SELECT    *
FROM      rental
WHERE     return_date <= timestamp '2005-05-27 00:00:00';
```

Task 8 : find out DVD that were returned on 2005-06-01 and then order by the return date?

```
SELECT    *
FROM      rental
WHERE     return_date >= timestamp '2005-06-01 00:00:00'
AND       return_date < timestamp '2005-06-02 00:00:00'
ORDER BY return_date;
```

WHERE Clause options

Logical Operators: AND, OR, and NOT

In the real world, a search of data normally involves multiple conditions. For example, when you are buying a new house, you look for a certain area, a certain number of bedrooms, bathrooms, stories, and so on. In the same way, SQL allows you to include multiple conditions in a query through the use of logical operators. **AND, OR and NOT.**

DEMO:

Let's find out the films with rental duration of 5 and rating or R.

```
SELECT *
FROM film
WHERE rental_duration=5
AND rating='R';
```

Task 9 :

Find out the payment amount that is larger than 5 and not processed by Mike Hillyer.

```
SELECT staff_id
FROM staff
WHERE first_name = "Mike"
AND last_name = "Hillyer";
```

```
SELECT *
FROM payment
WHERE amount>=5
AND NOT staff_id=2;
```

Special Operators

SQL allows the use of special operators in conjunction with the WHERE clause. These special operators include:

- ***BETWEEN***: Used to check whether an attribute value is within a range
- ***IN***: Used to check whether an attribute value matches any value within a value list
- ***LIKE***: Used to check whether an attribute value matches a given string pattern
- ***IS NULL***: Used to check whether an attribute value is null

- **BETWEEN**

DEMO:

Let's find out the films with rental duration between 5 and 10.

```
SELECT *
FROM film
WHERE rental_duration BETWEEN 5 AND 10;
```

Special Operators

- **IN**

Task 10 :

Let’s find out the customers with a last name of “Harris” or “White”.

```
SELECT *
FROM customer
WHERE last_name IN ("Harris", "White");
```

```
SELECT *
FROM customer
WHERE last_name = "Harris" OR last_name =
"White";
```

- **IS NULL**

Standard SQL allows the use of **IS NULL** to check for a null attribute value.

DEMO:

Let’s find out the staff who don’t have a profile picture.

```
SELECT *
FROM staff
WHERE picture IS NULL;
```

Special Operators

- **LIKE**

  The **Like** special operator is used in conjunction with wildcards to find patterns within string attributes. Standard SQL allows you to use the percent sign (%) and underscore (_) wildcard characters to make matches when the entire string is not known:

  % means any and all *following* or *preceding* characters are eligible. For example:
  - 'J%' includes Johnson, Jones, Jernigan, July, and J-231Q.
  - 'Jo%' includes Johnson and Jones.
  - '%n' includes Johnson and Jernigan.

  _ means any *one* character may be substituted for the underscore. For example:
  - '_23-456-6789' includes 123-456-6789, 223-456-6789, and 323-456-6789.
  - '_23-_56-678_' includes 123-156-6781, 123-256-6782, and 823-956-6788.
  - '_o_es' includes Jones, Cones, Cokes, totes, and roles.

Special Operators

- **LIKE**

DEMO:

Find out customers whose last name start with J.

```
SELECT *
FROM customer
WHERE last_name LIKE "J%";
```

Task 11 :

Find out the address with a postal code end with 00.

```
SELECT *
FROM address
WHERE postal_code LIKE “___00";
```

Or:

```
SELECT *
FROM address
WHERE postal_code LIKE “%00";
```

Aggregate Processing

SQL can perform various mathematical summaries for you, such as counting the number of rows that contain a specified condition, finding the minimum or maximum values for a specified attribute, summing the values in a specified column, and averaging the values in a specified column.

COUNT The function **counts** the number of non-null values of an attribute. e.g.:

SELECT COUNT (*column*|*) FROM *table*;

DEMO:
how many films have a rating of R?

```
SELECT COUNT(film_id)
FROM film
WHERE rating="R";  /* try count(*) */
```

Task 12 : What's the total payment amount for staff 2?

```
SELECT SUM(amount)
FROM payment
WHERE staff_id=2;
```

FUNCTION	OUTPUT
COUNT	The number of rows containing non-null values
MIN	The minimum attribute value encountered in a given column
MAX	The maximum attribute value encountered in a given column
SUM	The sum of all values for a given column
AVG	The arithmetic mean (average) for a specified column

Grouping Data

Rows can be grouped into smaller collections quickly and easily using the **GROUP BY** clause within the SELECT statement. The aggregate functions will then summarize the data within each smaller collection. The syntax is:

```
SELECT      columnlist
FROM        tablelist
[WHERE      conditionlist ]
[GROUP BY   columnlist ];
```

DEMO:

What is the minimal length for films with different rating?

```
SELECT rating, MIN(length)
FROM film
GROUP BY rating;
```

Task 13 :

What is the number of rentals and the total amount for **each** customer checked out with **each** staff?

```
SELECT customer_id, staff_id, COUNT(*), SUM(amount)
FROM payment
GROUP BY customer_id,staff_id;
```

HAVING Clause

However, restricting data based on an aggregate value is slightly more complicated and can require the use of a **HAVING** clause. The syntax for a HAVING clause is:

```
SELECT      columnlist
FROM        tablelist
[WHERE      conditionlist ]
[GROUP BY   columnlist ]
[HAVING     conditionlist ]
[ORDER BY   columnlist [ASC | DESC] ];
```

HAVING vs WHERE

- WHERE clause applies to columns and expression for individual rows
- HAVING clause is applied to the output of a GROUP BY operation

Task 14 :

What is the minimal length (>46) for films with different rating?

```
SELECT rating, MIN(length)
FROM film
GROUP BY rating
HAVING MIN(length) >46;
```

JOIN

The **FROM** clause of the query specifies the table or tables from which the data is to be retrieved.

In practice, most SELECT queries will need to retrieve data from multiple tables. So we need to decompose our data into separate entities to create a flexible, stable structure for storing and manipulating the data. We usually do this through **JOIN**.

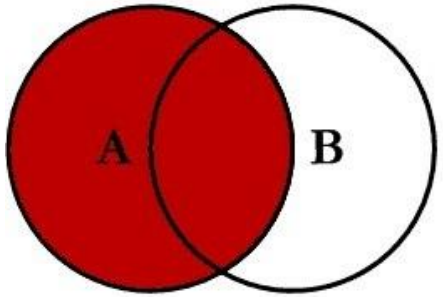
- **Inner joins vs Outer joins**

Inner joins: return only rows from the tables that match on a common value. (almost the same as **natural-join** except allowing different attribute names)

Outer joins: return the same matched rows as the inner join, plus unmatched rows from one table or the other.

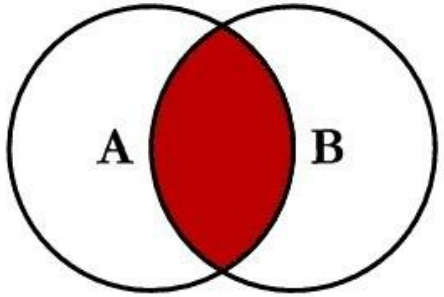
SQL JOINS

Left outer join



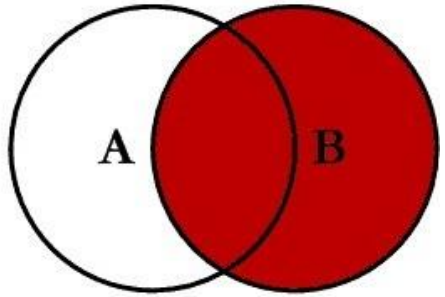

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

Inner Join



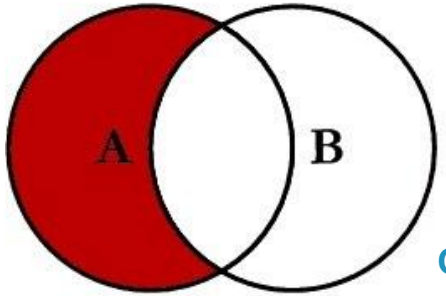

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

Right outer Join



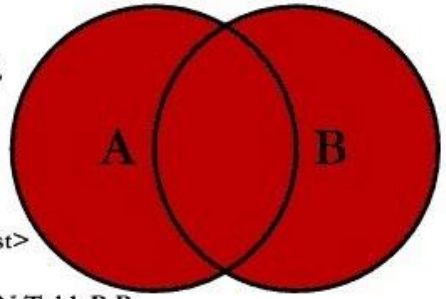

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

Left excluding join



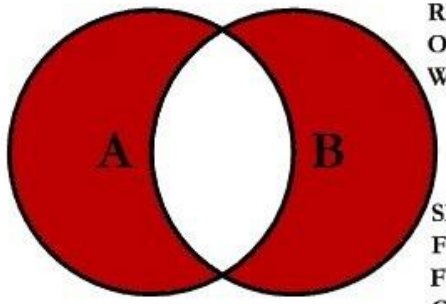

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

OUTER JOIN or FULL OUTER JOIN or Full join



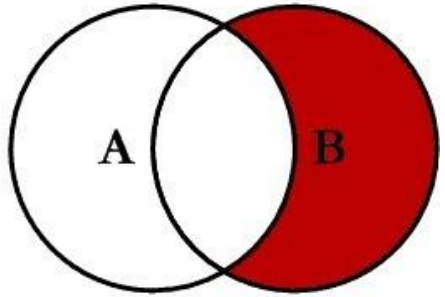

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

Outer excluding Join

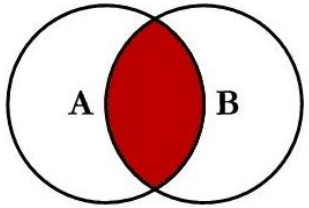
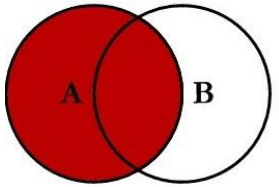
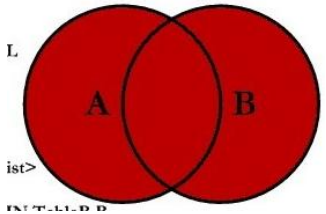
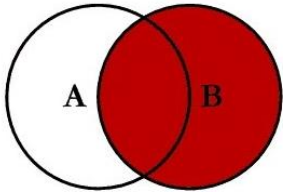



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

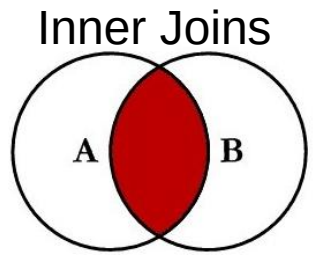
Right excluding join




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

Table	Join Type		Table	Statement	What we use	Visualization
A		Inner	B	A inner join B	A Inner Join B	
	Left	Outer		A Left outer join B	A Left Join B	
	Full			A Full outer Join B	A Full Join B	
	Right			A Right Outer Join B	A Right Join B	

Inner Joins



To get data from both tables, you use the INNER JOIN clause in the SELECT statement as follows:

SELECT *column-list* FROM *table1* INNER JOIN *table2* ON *join-condition;*

DEMO:

inner join table city and country with country_id.

```
SELECT *
FROM city
INNER JOIN country ON
city.country_id=country.country_id;
```

Another way to write this:

```
SELECT *
FROM city, country
WHERE city.country_id = country.country_id;
```

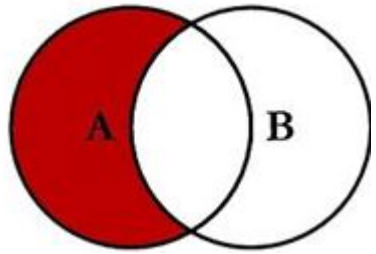
Task 15 :

Find out the customer information (id, first_name, last_name, email, amount) and his/her payment information (amount, payment_date) for those customers whose first_name starts with A.

```
SELECT
customer.customer_id,
customer.first_name,
customer.last_name,
customer.email,
payment.amount,
payment.payment_date
FROM customer
INNER JOIN payment ON
customer.customer_id=payment.customer_id
WHERE first_name LIKE 'A%';
```

Left/Right Outer Join

Left excluding join



An outer join returns not only the rows matching the join condition, but it also returns the rows with unmatched values.

```
SELECT column-list
FROM table1 LEFT/RIGHT [OUTER] JOIN table2 ON join-condition
```

DEMO:

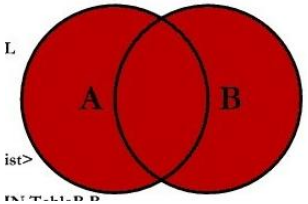
Find out film information (id, title) and inventory information (id) that present in film table but not inventory table. Then find out those with a null film_id in inventory table, and order them by the film title (descending).

```
SELECT film.film_id, film.title, inventory_id
FROM film
LEFT OUTER JOIN inventory ON film.film_id = inventory.film_id
WHERE inventory.film_id IS NULL
ORDER BY film.title DESC;
```

select film.film_id,film.title,inventory_id from film left outer join inventory on film.film_id=inventory.film_id where inventory.film_id is null order by film.title DESC;

Full Outer Join

Full outer join



An outer join returns not only the rows matching the join condition, but it also returns the rows with unmatched values. The first table named in the FROM clause will be the left side, and the second table named will be the right side.

```
SELECT column-list
FROM table1 FULL [OUTER] JOIN table2 ON join-condition
```

Task 16 : We want get the information of which staff process which rental events. Use Full outer join to complete this task. Then we want to know the customer with a first name start with M who rented in store_id =1, what their rental histories? Please also order those histories by their address id.

```
SELECT *
FROM rental
FULL OUTER JOIN staff ON
staff.staff_id=rental.staff_id
WHERE store_id=1 AND first_name LIKE 'M%'
ORDER BY address_id;
```

Subqueries

It is often necessary to process data based on *other* processed data.
SELECT **within** SELECT.

DEMO: Suppose we want to know the films with a rental rate above the average.
First, we can get the average rental rate of all films:

```
SELECT AVG(rental_rate)
FROM film;    /* 2.98 */
```

DEMO: Then, we can get films whose rental rate is higher than the average rental rate:

```
SELECT film_id, title,rental_rate
FROM film
WHERE rental_rate > 2.98;
```

DEMO: The code is not so elegant, which requires two steps. We want a way to pass the result of the first query to the second query in one query. The solution is to use a subquery.

```
SELECT film_id, title,rental_rate
FROM film
WHERE rental_rate > (
        SELECT AVG(rental_rate)
        FROM film);
```

Subqueries

Task 16 :

to get films (id and title) that have the returned date between 2005-05-29 and 2005-05-30

```
SELECT film_id, title
FROM  film
WHERE film_id IN (
  SELECT inventory.film_id
  FROM
     rental
     INNER JOIN
     inventory
     ON
     inventory.inventory_id = rental.inventory_id
  WHERE
     return_date
  BETWEEN '2005-05-29' AND '2005-05-30'
);
```

Relational Set operators

UNION, INTERSECT, and EXCEPT (MINUS) work properly only if relations are *union-compatible*, which means that the number of attributes must be the same and their corresponding data types must be alike.

In MySQL, The **UNION** statement combines rows from two or more queries without including duplicate rows. The syntax of the UNION statement is:

query UNION *query* /*combine the output of two SELECT queries*/

DEMO:

find out actor whose first name is ‘Joe’ and customer whoes first name is ‘Lisa’.

```
SELECT first_name,last_name
FROM actor
WHERE first_name=’Joe’
UNION
SELECT first_name,last_name
FROM customer
WHERE first_name=’Lisa’;
```

Hint:
UNION is set strict and no duplicates; if you want to keep duplicates, you can use **UNION ALL**

```
select first_name,last_name from actor where first_name='Joe' union select first_name,last_name from customer where first_name='Lisa';
```

Relational Set operators

In MySQL, INTERSECT statement is not supported. To combine rows from two queries and return only the rows that appear in both sets, we can use:

query IN *query* /*take intersection on the output of two SELECT queries*/

Task 17 :

Find out the film_id of the film with length < 60 and category belongs to ‘Action’ type.

```
SELECT film_id
FROM film
WHERE length < 60 AND film_id
IN
(SELECT film_id  FROM film_category
WHERE category_id = (
        SELECT category_id
        FROM category
        WHERE name=‘Action’
));
```

```
select film_id from film where length<60 and film_id in (select film_id from film_category where category_id = (select category_id from category where name='action'));
```

Relational Set operators

In MySQL, the EXCEPT statement is not supported. To combines rows from two queries and returns only the rows that appear in the first set but not in the second, we can use:

query NOT IN *query* /*take except on the output of two SELECT queries*/

Task 18 :

get the films (id and title) that are not in the ‘inventory’. NOTE ‘inventory’ has only ‘id’ but title is in ‘film’.

```
SELECT  film_id,title
FROM  film
WHERE (film_id,title) NOT IN
(   SELECT
      DISTINCT inventory.film_id,title
   FROM
      inventory
   INNER JOIN film ON film.film_id = inventory.film_id
);
```

select film_id,title from film where (film_id,title) not in (select distinct inventory.film_id,title from inventory inner join film on film.film_id = inventory.film_id) order by title;

Crafting SELECT Queries

As you have seen in this chapter, the SQL language is both simple and complex. Each clause and function on its own is simple and performs a well-defined task. However, because of the flexibility of the SQL language, combining the appropriate clauses and functions to satisfy an information request can become rather complex.

So bear in mind the following rules for query crafting:

- **Know your data**
- **Know the problem**
  - The average price for all of the sales that have occurred:
    10 + 10 + 10 + 10 + 10 + 10 + 10 + 10 + 20 + 30 = 130 / 10 = $13
    - Coded as: SELECT AVG(sale_price)
  - The average of the prices at which any sale has occurred: 10 + 20 + 30 = 60 / 3 = $20
    - Coded as: SELECT AVG(DISTINCT sale_price)
- **Build one clause at a time**
  - it may be helpful to build your clauses in the following order:
    - FROM , WHERE , GROUP BY , HAVING , SELECT , ORDER BY

Week 9 Summary

- Retrieve specified columns of data from a database
  - SELECT, WHERE, ORDER BY, special operators, LIMIT
- Restrict data retrievals to rows that match complex criteria
- Aggregate data across groups of rows
  - GROUP, HAVING
- Join multiple tables in a single SQL query
  - INNER JOIN, OUTER JOIN
- Create subqueries to preprocess data for inclusion in other queries
  - SELECT in SELECT
- Explain the key principles in crafting a SELECT query

Week 9 KEY TERMS

- Alias
- AND
- AVG
- BETWEEN
- boolean algebra
- COUNT
- DISTINCT
- EXISTS
- FROM
- GROUP BY
- LIMIT
- HAVING IN
- IS NULL
- LIKE
- MAX
- MIN
- NOT
- OR
- ORDER BY
- Subquery
- SUM
- WHERE
- wildcard character

Week 9 more SQL tasks

🎯 Additional Task :
find out the actor (id,first_name,last_name) with most films.

```
SELECT actor_id, first_name, last_name
 FROM actor
WHERE actor_id = (
            SELECT actor_id
            FROM film_actor
            GROUP BY actor_id
            ORDER BY COUNT(film_id) DESC
            LIMIT 1);
```

Week 9 more SQL tasks

Additional Task :

find out the rental events for customer first/last name, phone, film title and return_date that returned before 2005-06-01.

```
SELECT
customer.first_name, customer.last_name,
address.phone, film.title, rental.return_date
FROM rental
INNER JOIN customer ON rental.customer_id=customer.customer_id
INNER JOIN address ON customer.address_id=address.address_id
INNER JOIN inventory ON rental.inventory_id=inventory.inventory_id
INNER JOIN film ON inventory.film_id=film.film_id
WHERE rental.return_date < ‘2005-06-01’;
```