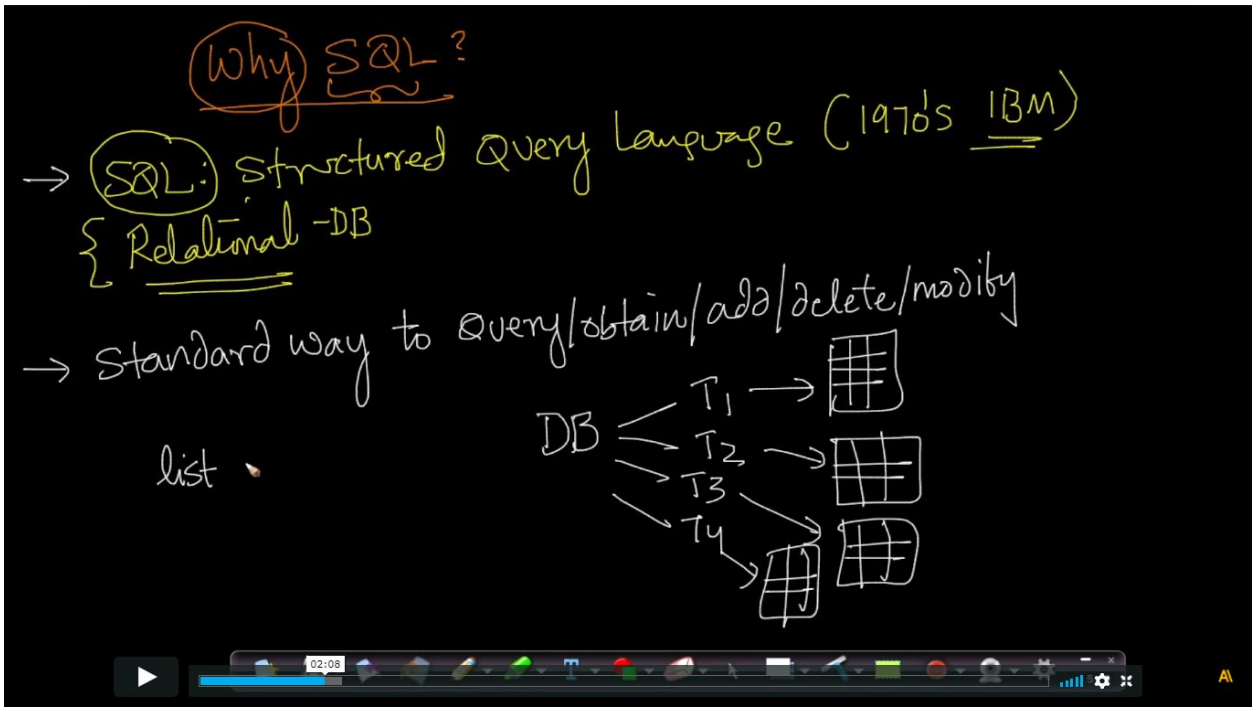
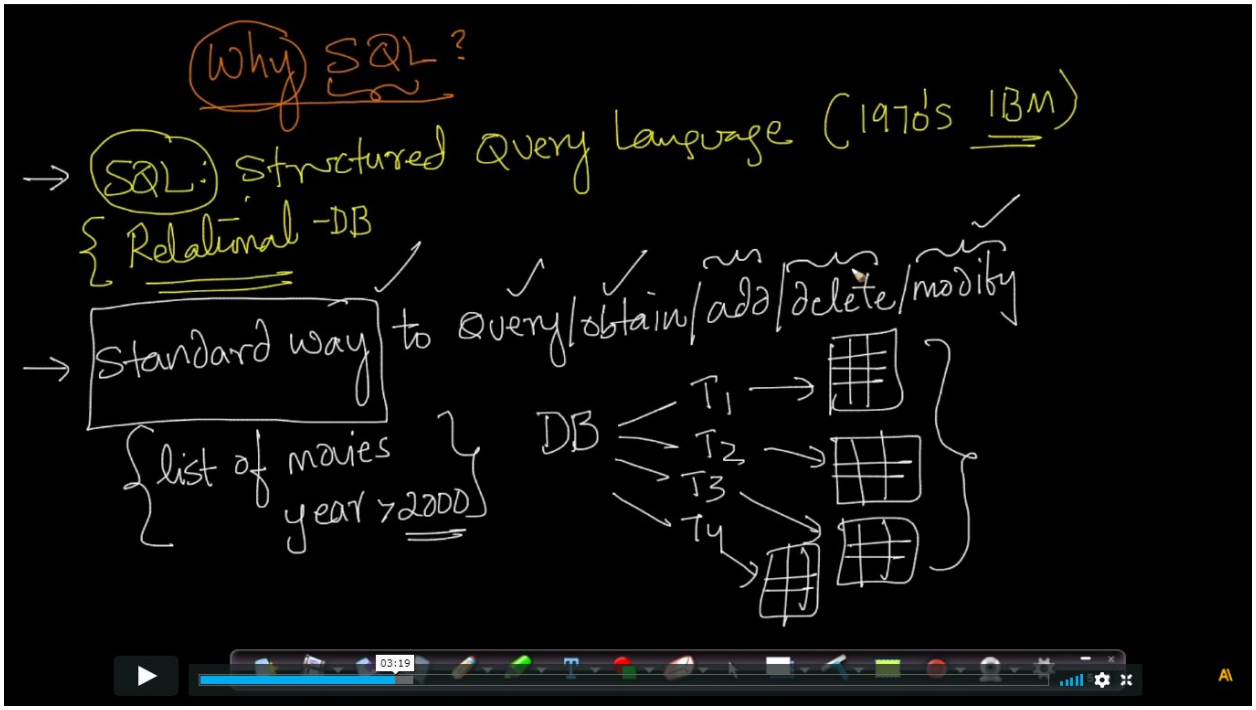
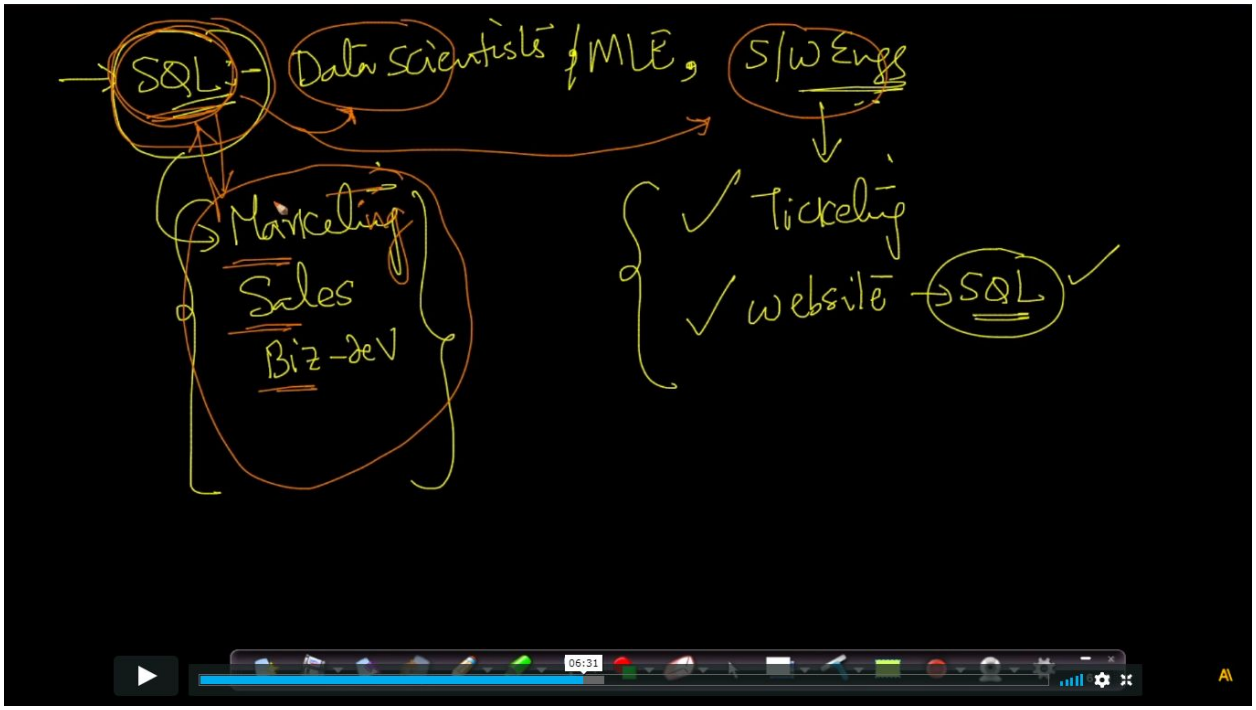
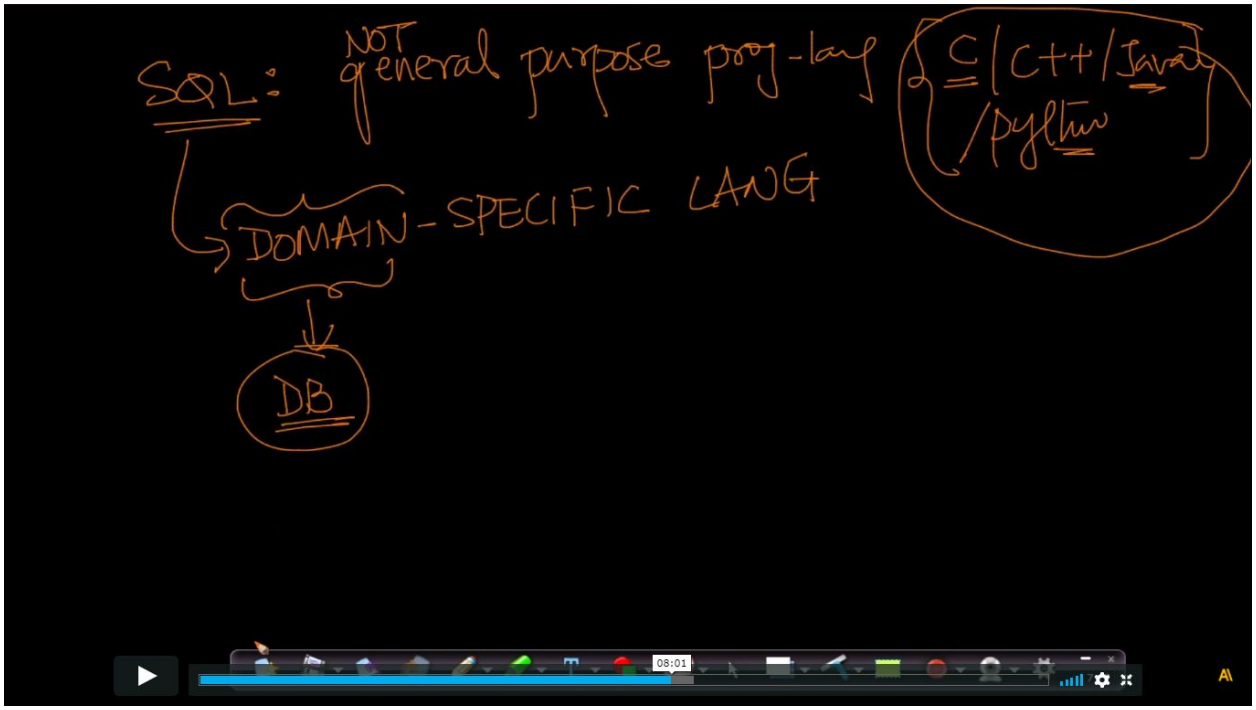
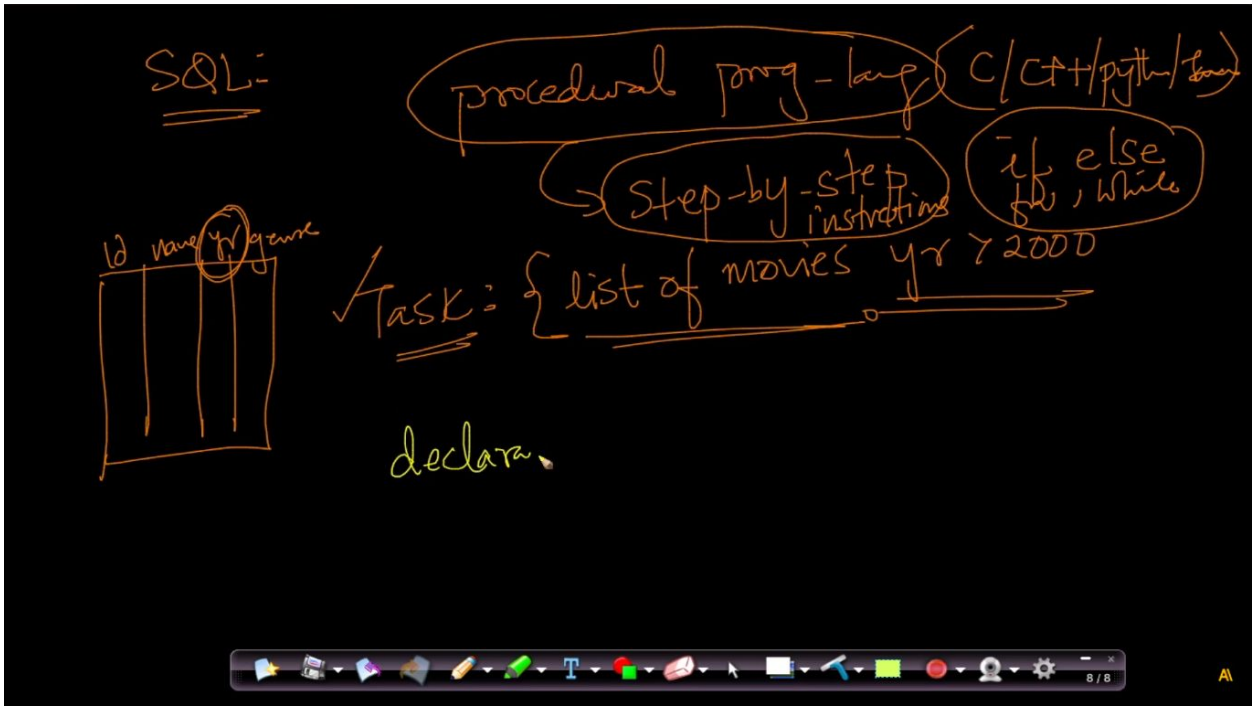
Why SQL?

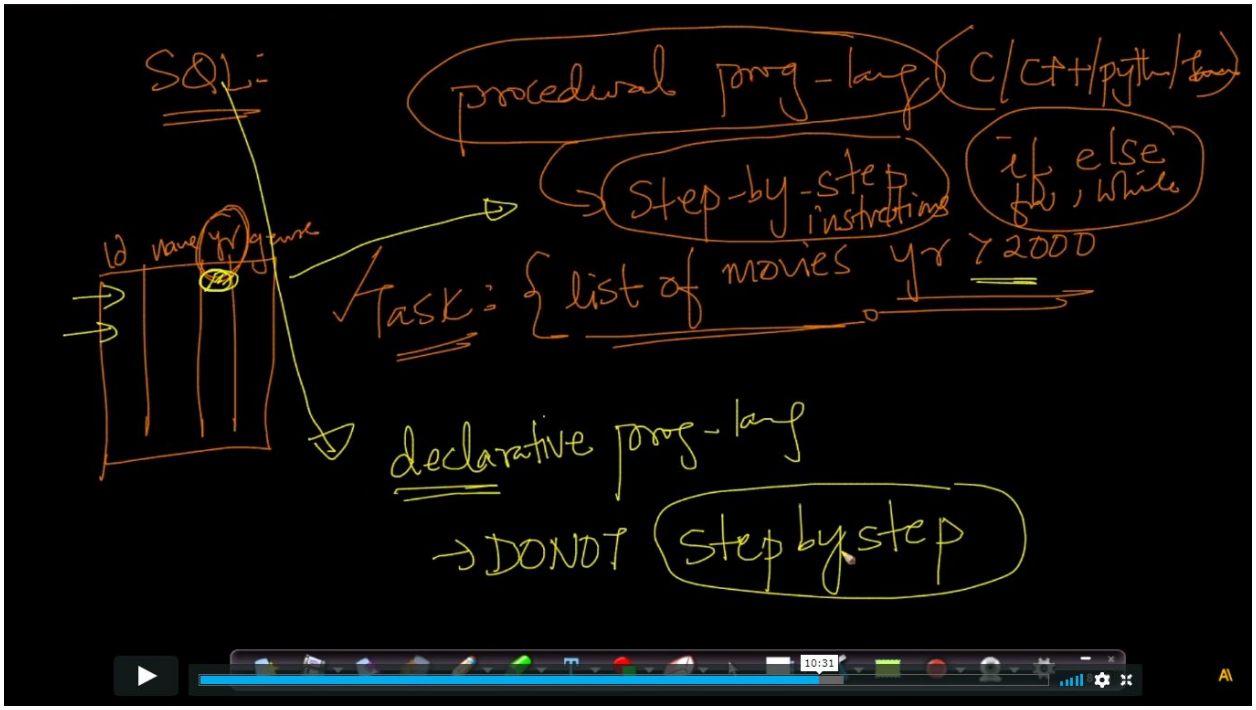

Why SQL?
SQL: Structured Query Language (1970's IBM)
{ Relational -DB
Standard way to query/obtain/add/delete/modify
list
DB
T1
T2
T3
T4
02:08

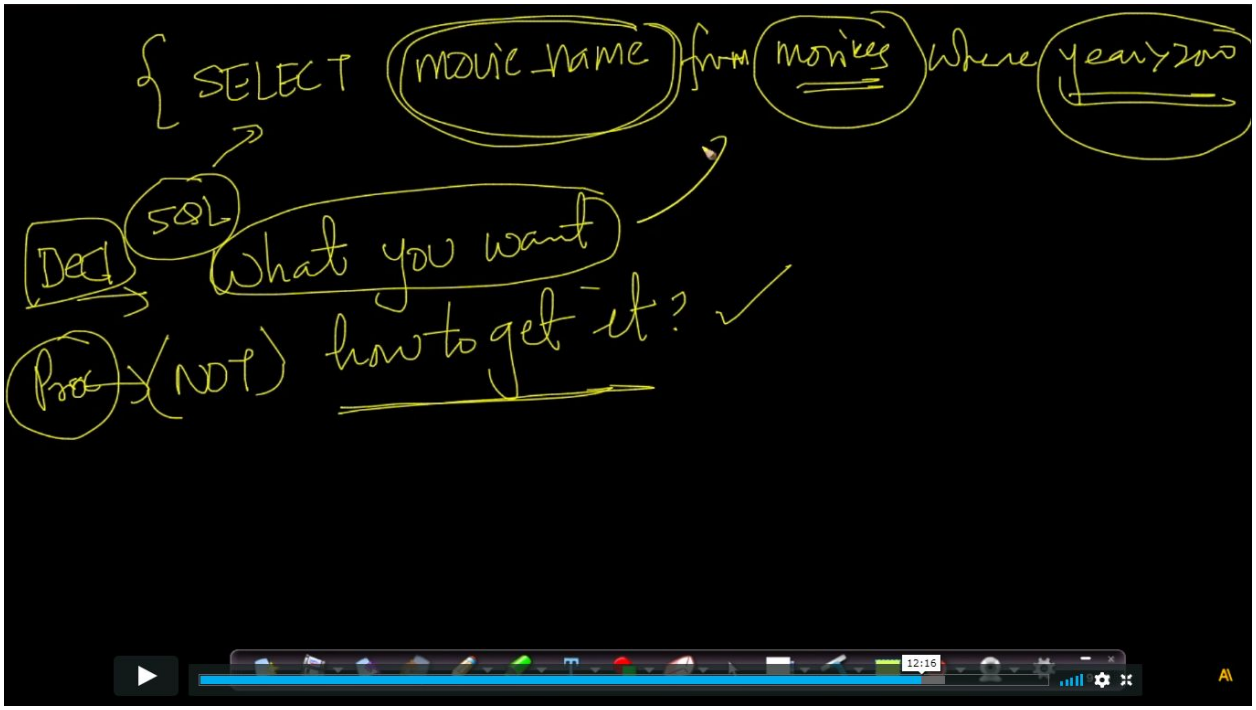

Why SQL?
SQL: Structured Query Language (1970's IBM)
{ Relational -DB
Standard way
to query/obtain/add/delete/modify
{ list of movies year >2000 }
DB
T1
T2
T3
T4
03:19

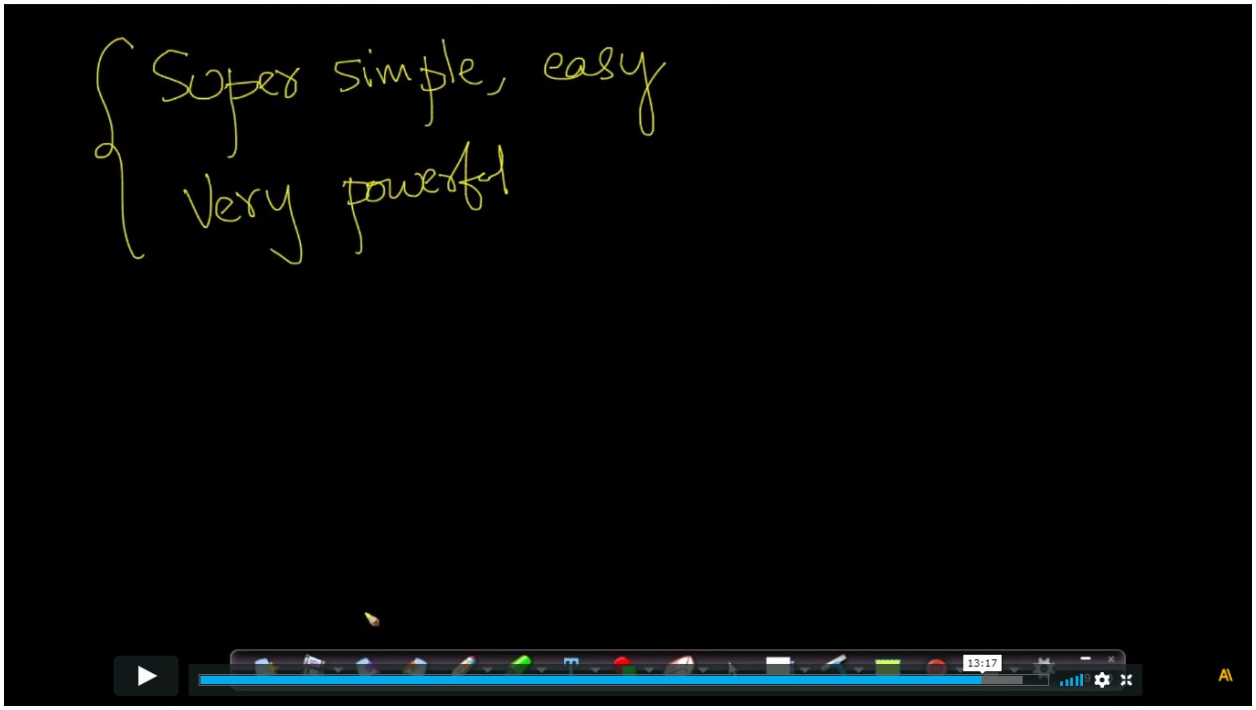

SQL
Data scientists / MLE
S/W Engg
Marketing
Sales
Biz-dev
Ticketing
website
SQL
06:31

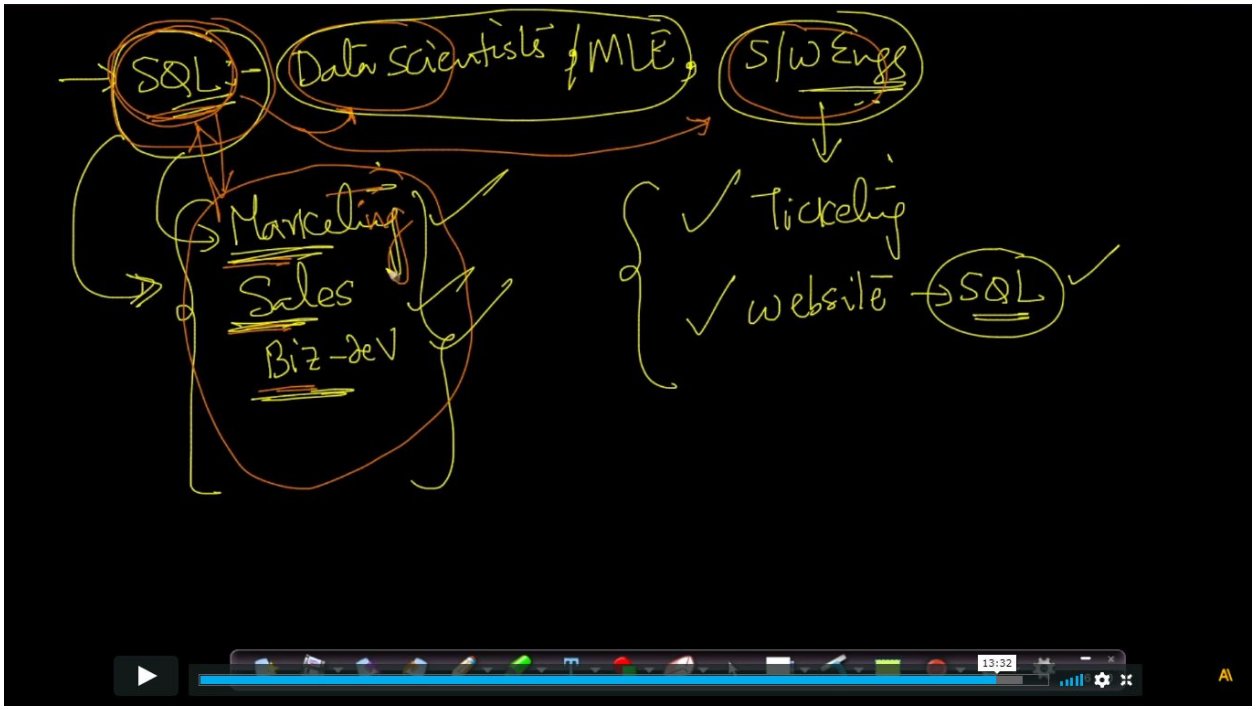

SQL: NOT general purpose prog-lang
C / C++ / Java / Python
DOMAIN-SPECIFIC LANG
DB
08:01


SQL:-
procedural prog-lang
C/C++/python/
Step-by-step instructions
if, else
for, while
Id, name, yr, genre
Task: {list of movies yr > 2000
declara.

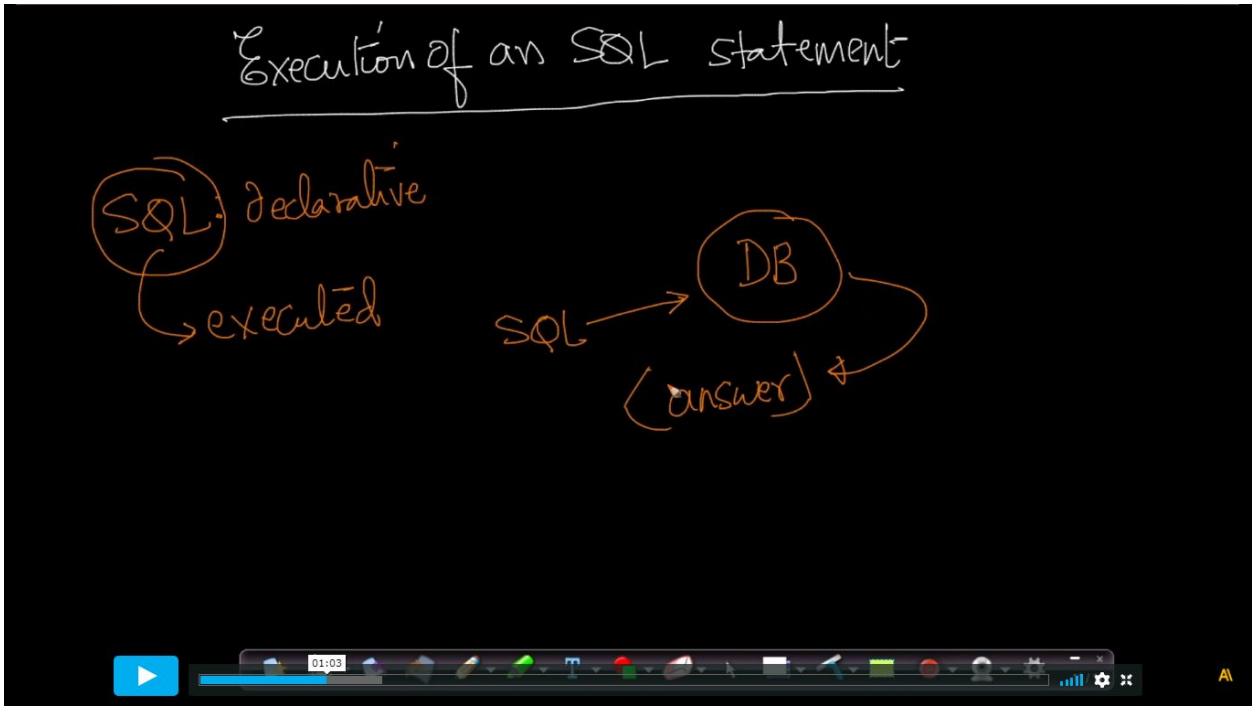

SQL:-
procedural prog-lang
C/C++/python/
Step-by-step instructions
if, else
for, while
Id, name, yr, genre
Task: {list of movies yr > 2000
declarative prog-lang
→ DONOT step by step
10:31

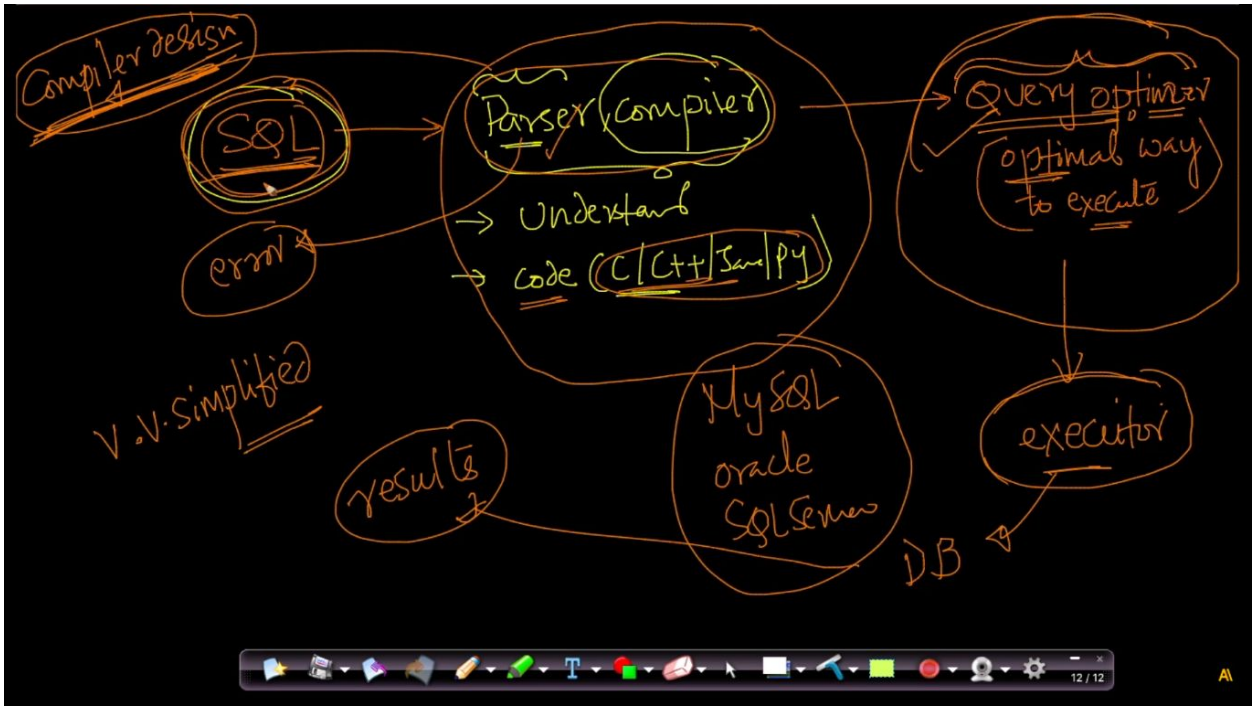

{ SELECT movie_name from movies where year>2010
SQL
Decl
What you want
Proc
(NOT) how to get it?
12:16



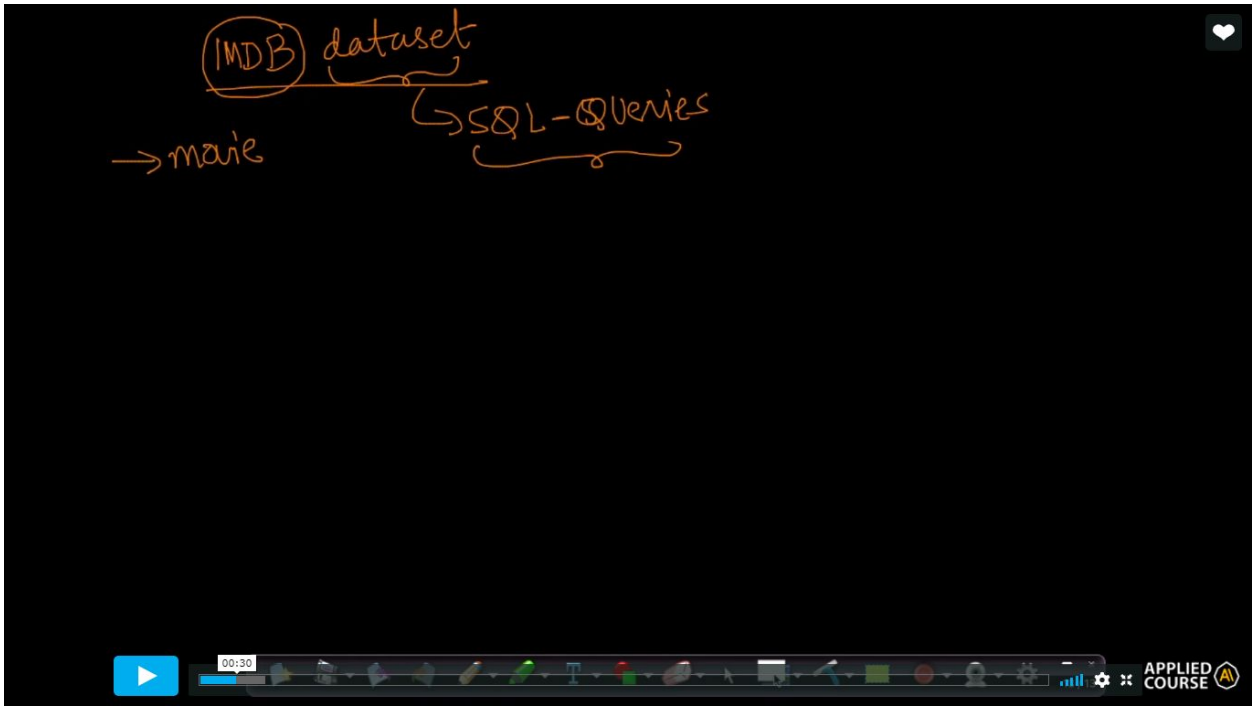


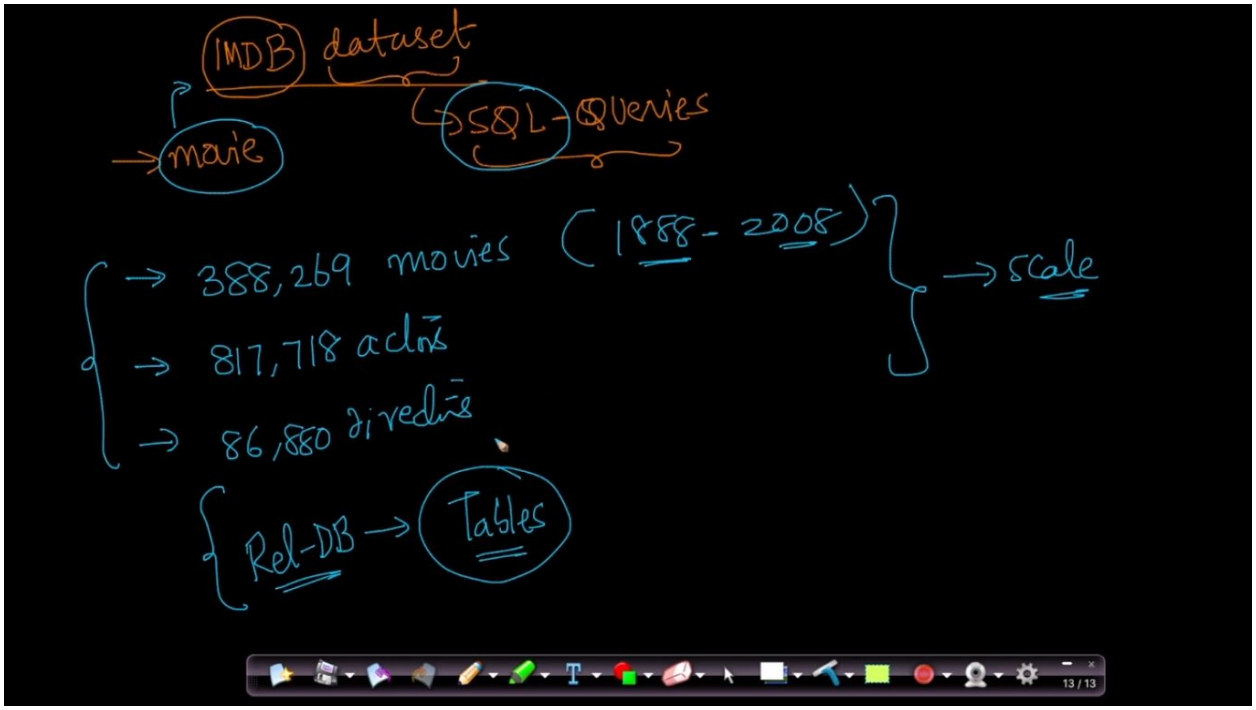
Execution of an SQL statement.

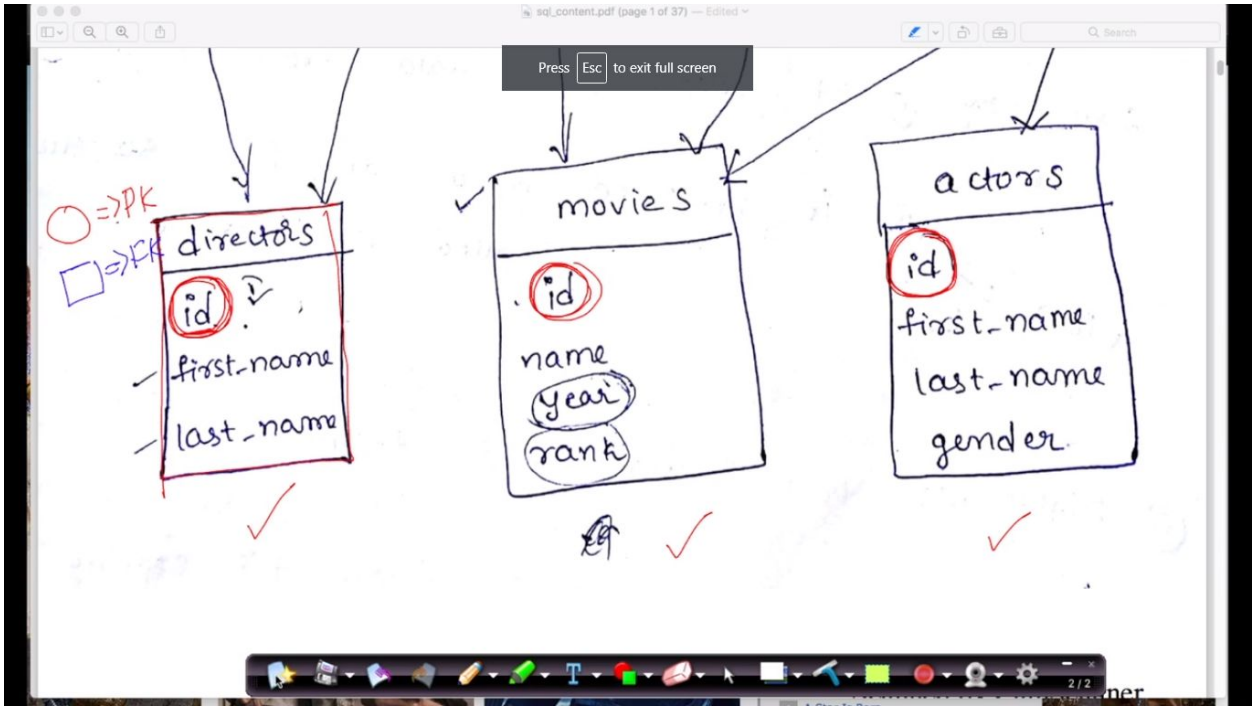


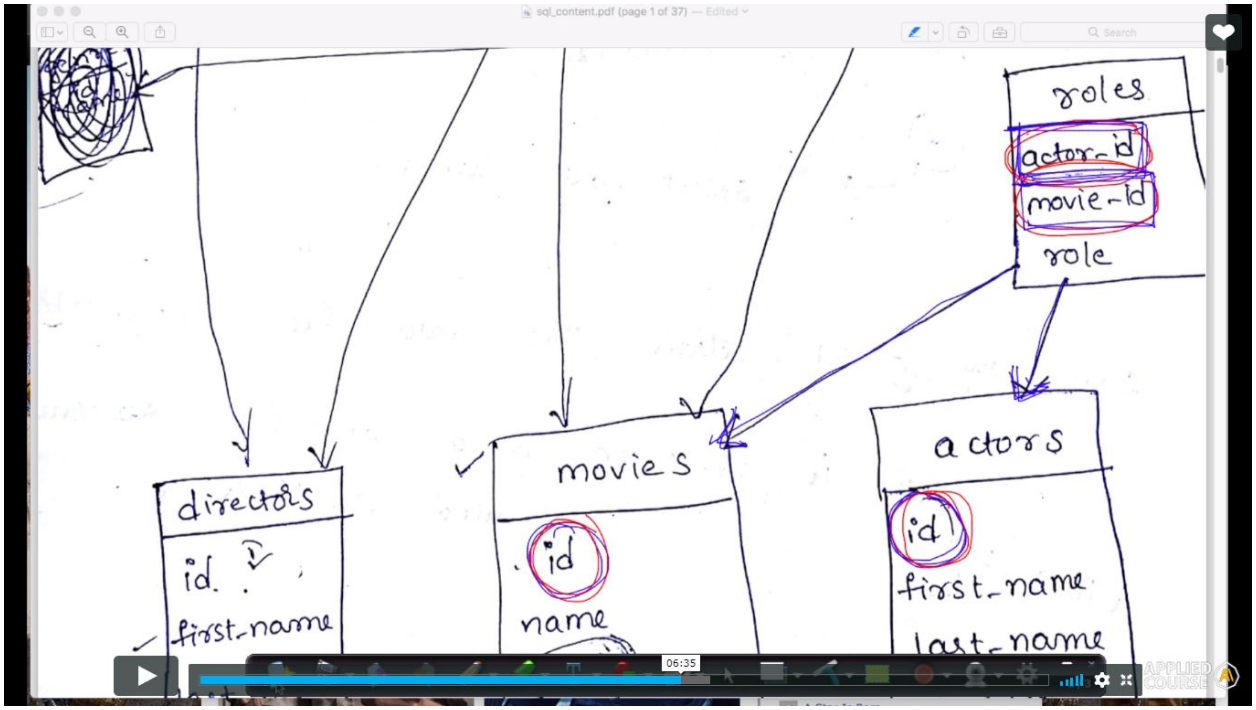


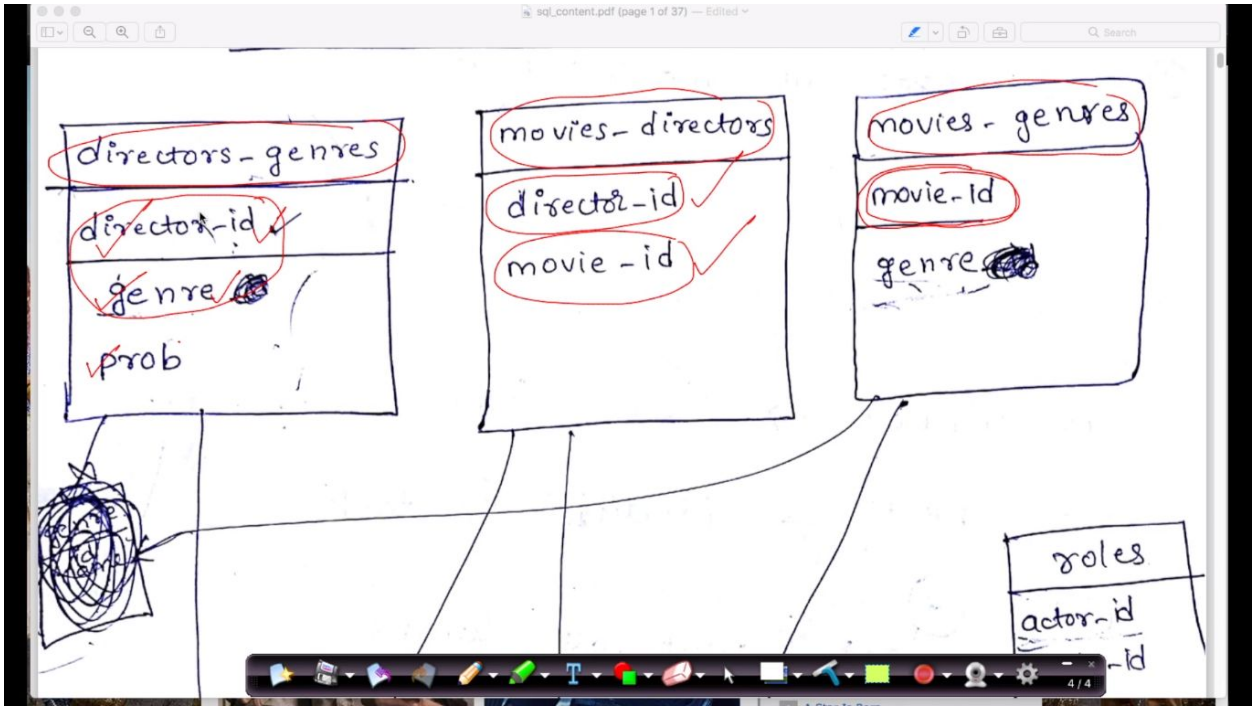
IMDB dataset

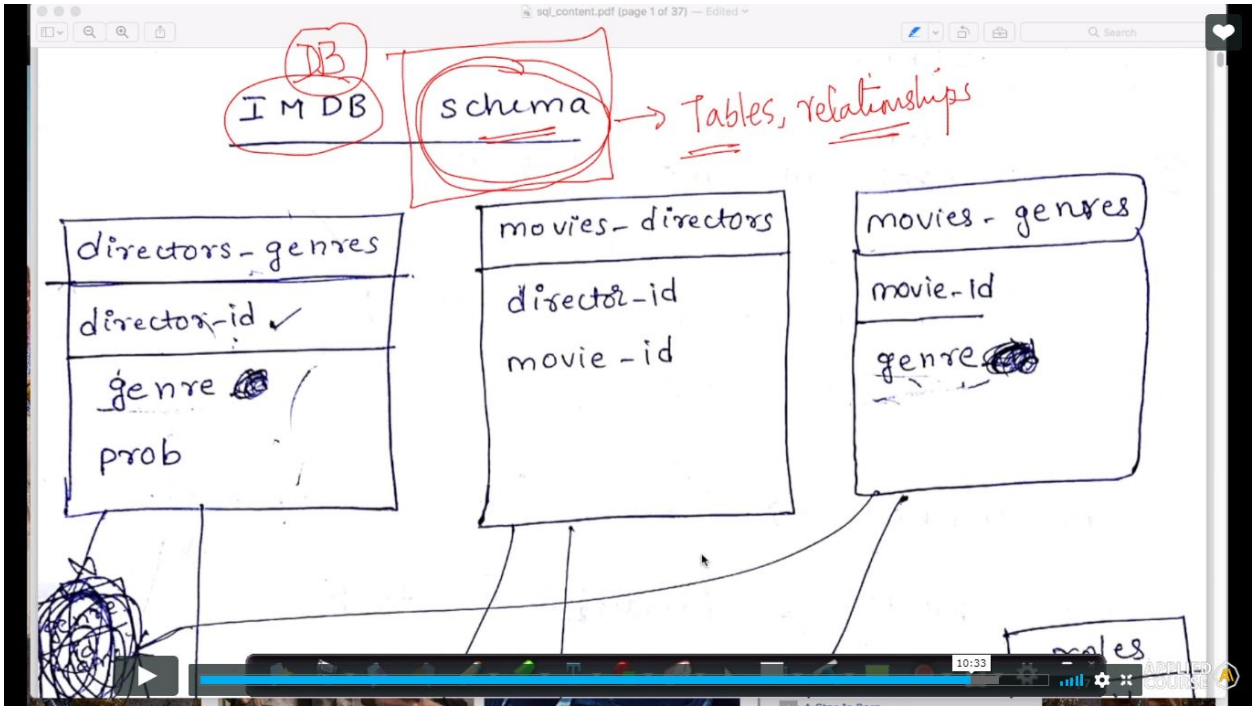

IMDB dataset
SQL-Queries
→ movie
00:30
APPLIED COURSE

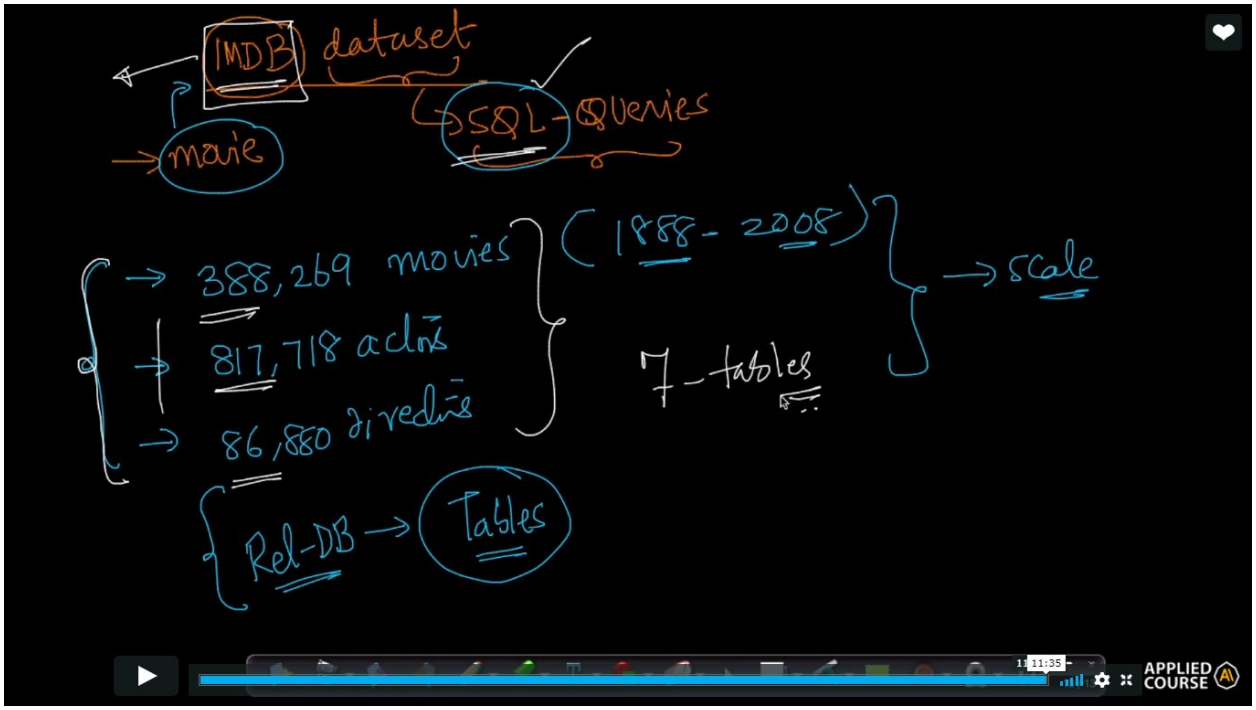

IMDB dataset
SQL-Queries
→ movie
→ 388,269 movies (1888 - 2008)
→ 817,718 actors
→ 86,880 directors
→ scale
Rel-DB → Tables
13 / 13


Press Esc to exit full screen
directors
id
first_name
last_name
movies
id
name
year
rank
actors
id
first_name
last_name
gender


roles
actor_id
movie_id
role
directors
id
first_name
movies
id
name
actors
id
first_name
last_name
06:35


directors - genres
director-id
genre
prob
movies - directors
director-id
movie-id
movies - genres
movie-id
genre
roles
actor-id
4 / 4





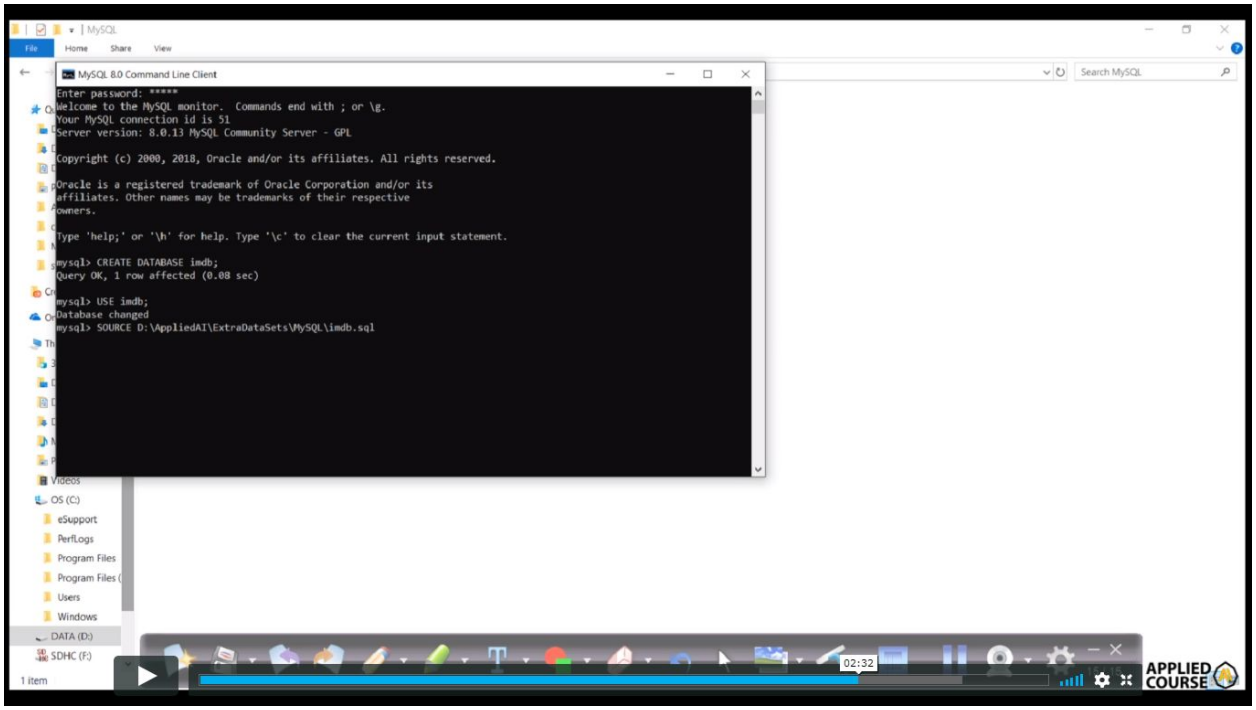
Installing MySQL

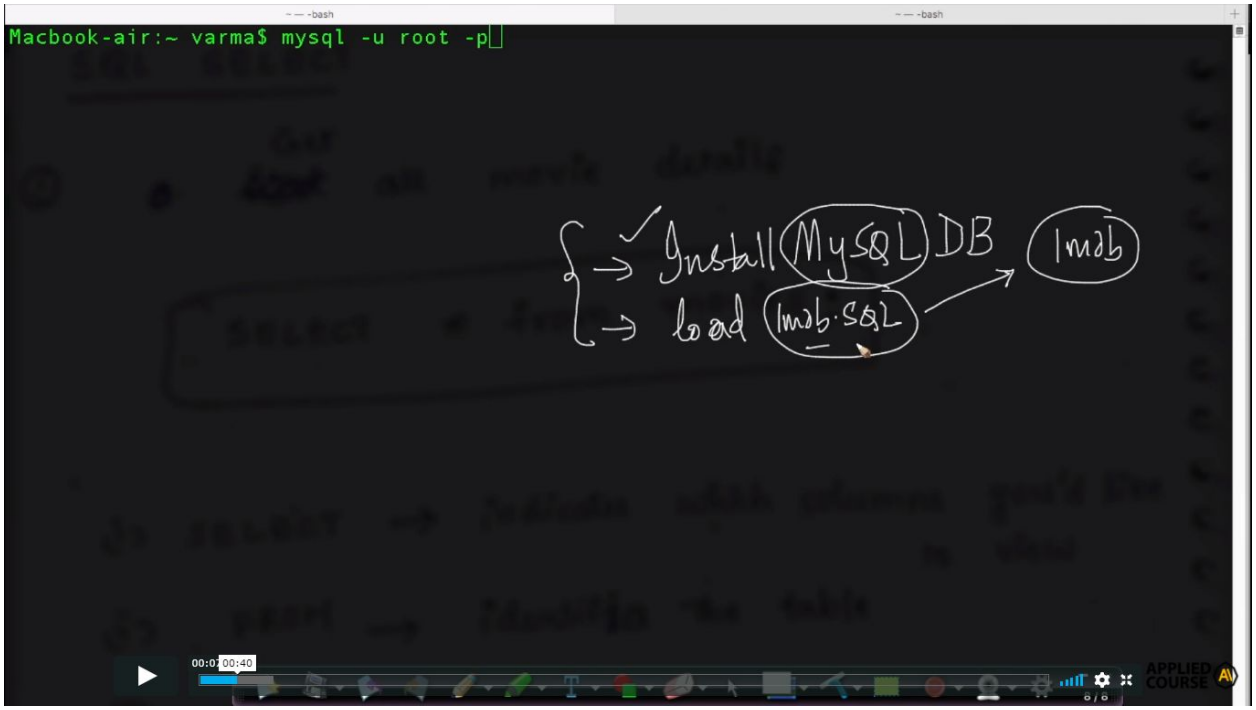
Refer:https://dev.mysql.com/downloads/windows/installer/8.0.html
https://drive.google.com/open?id=1rrqZz-ddvYGvbXoP-SH767pgY06B7YHk

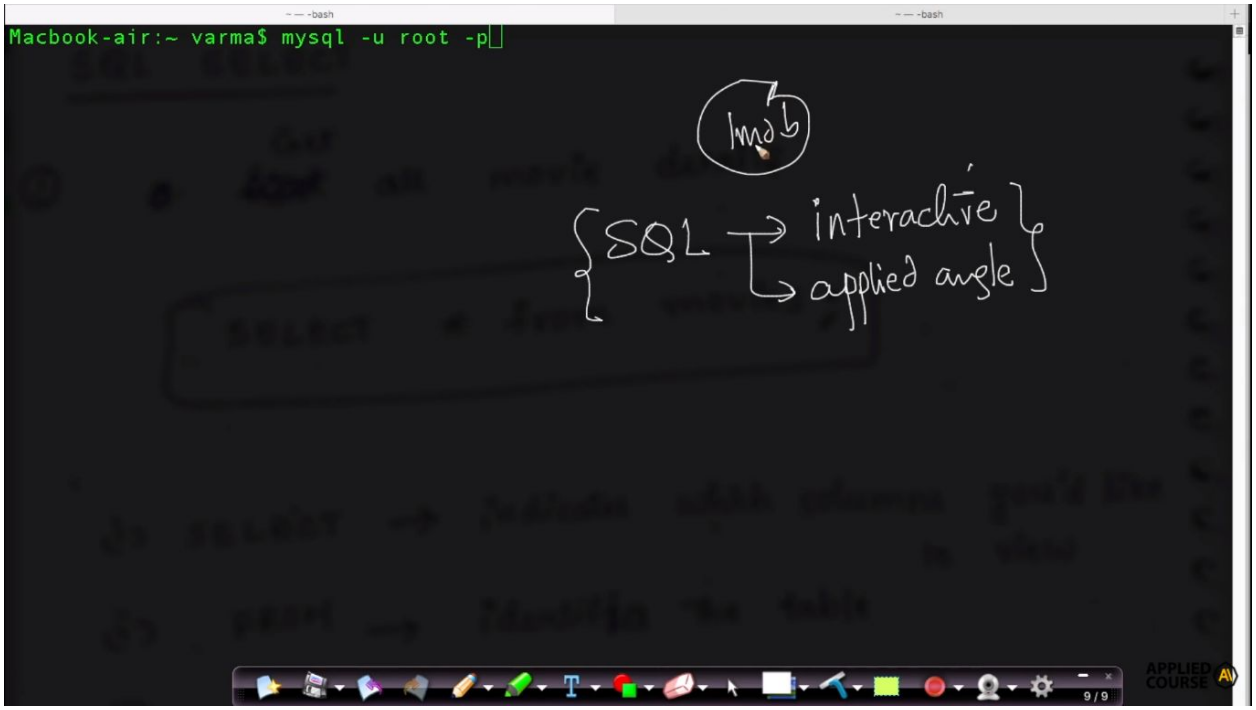
Load IMDB data.

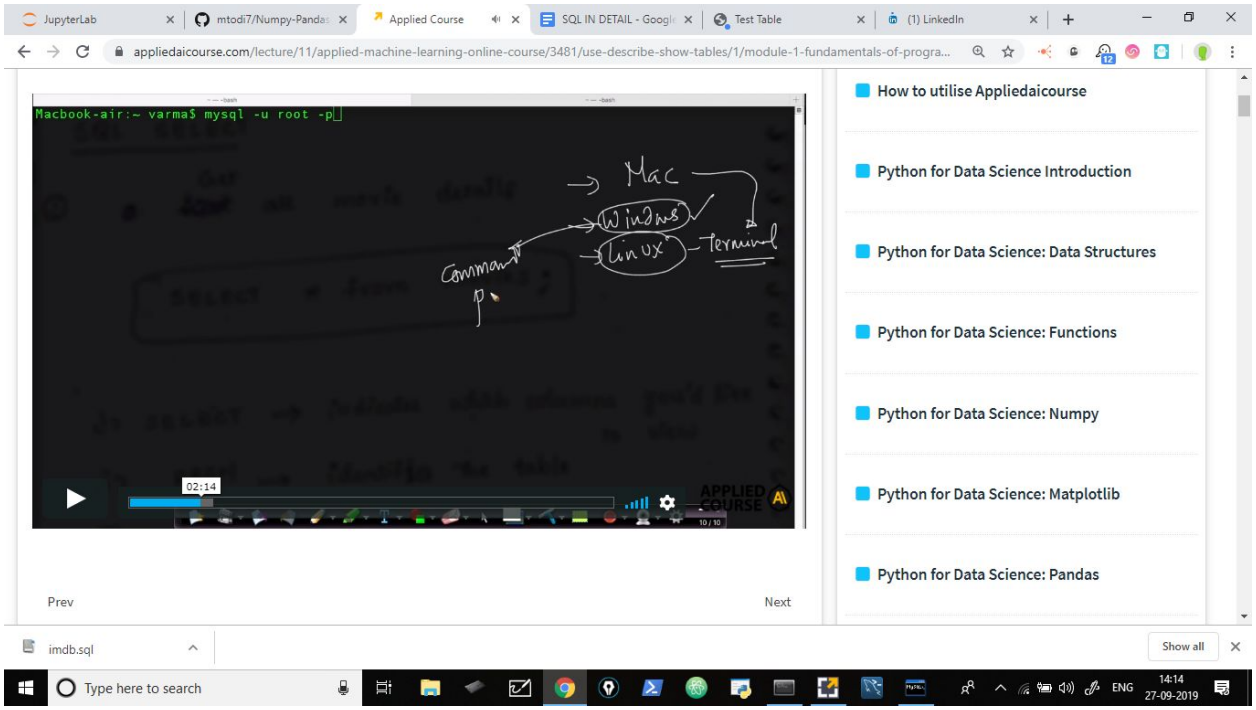
CREATE DATABASE imdb
USE imdb
SOURCE location

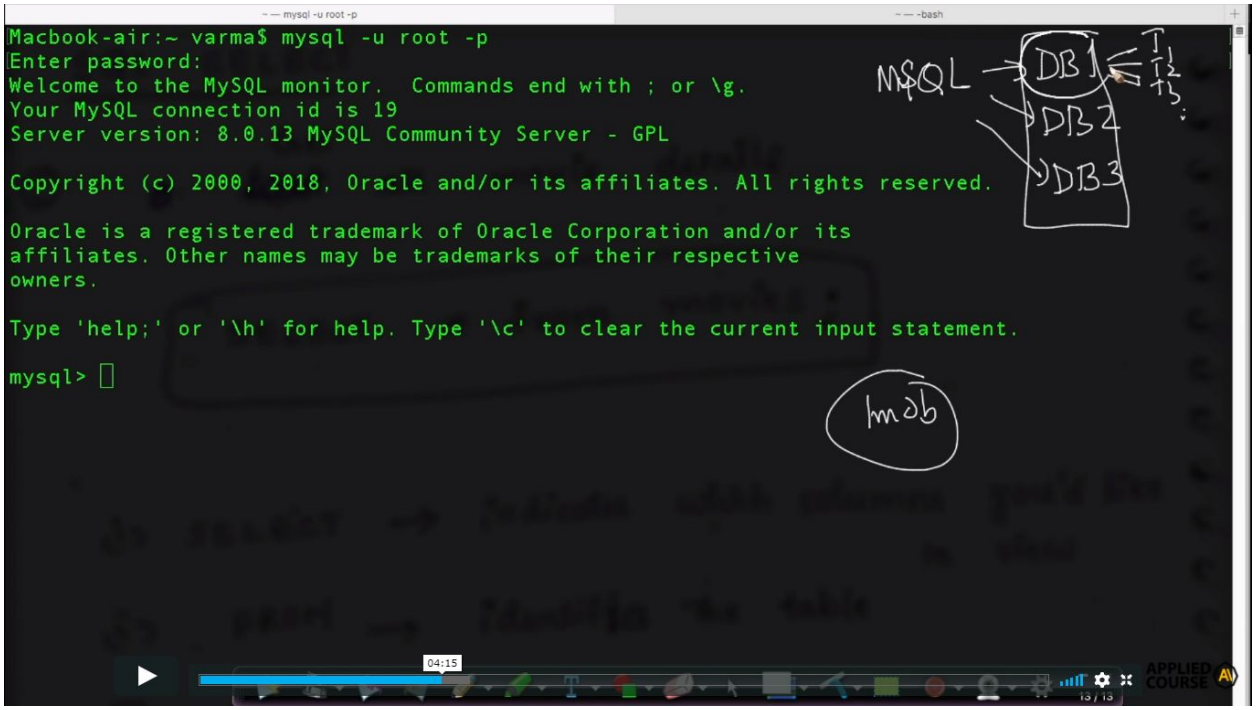
USE, DESCRIBE, SHOW TABLES

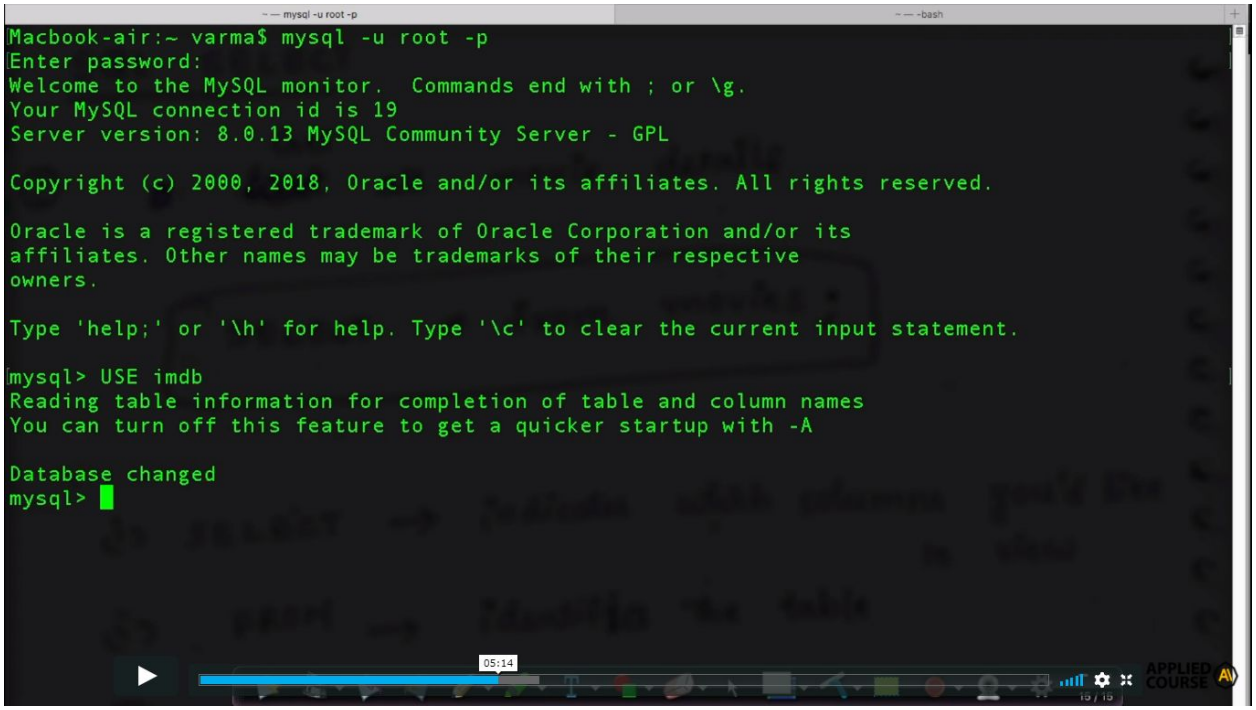

MySQL 8.0 Command Line Client
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 51
Server version: 8.0.13 MySQL Community Server - GPL
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> CREATE DATABASE imdb;
Query OK, 1 row affected (0.08 sec)
mysql> USE imdb;
Database changed
mysql> SOURCE D:\AppliedAI\ExtraDataSets\MySQL\imdb.sql
02:32
APPLIED COURSE

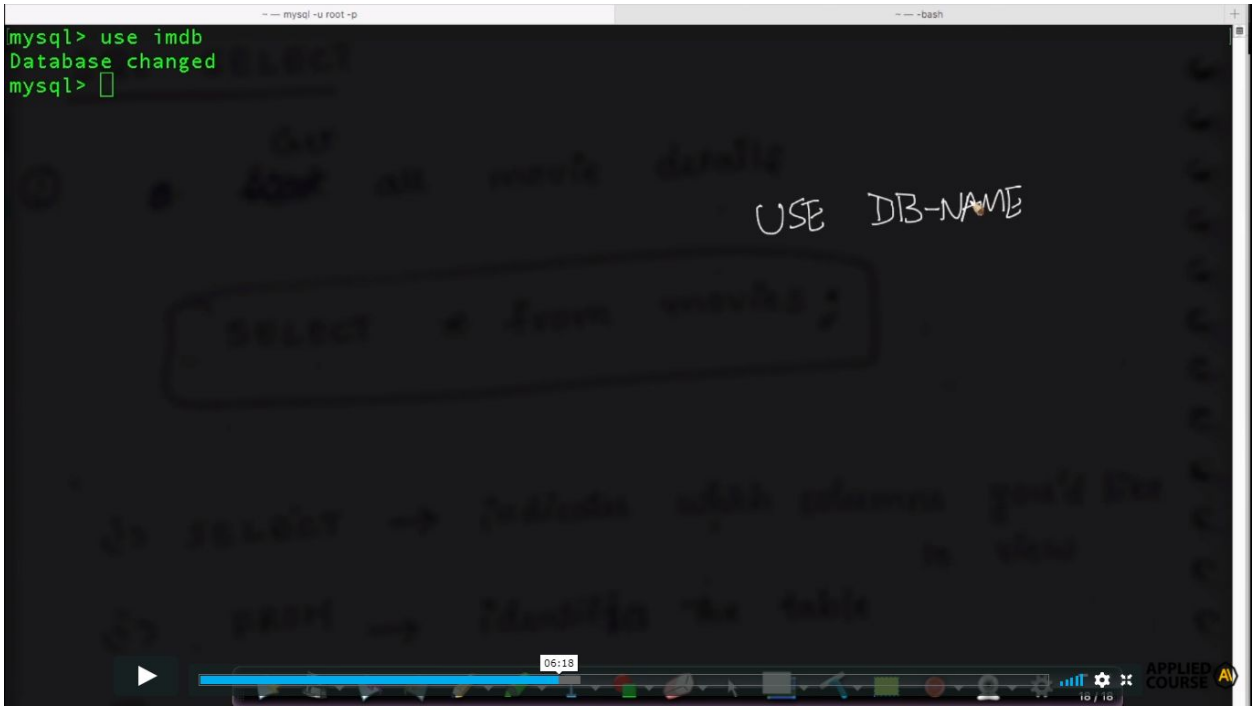

Macbook-air:~ varma$ mysql -u root -p
install MySQL DB
load imdb.sql
imdb
00:40
APPLIED COURSE

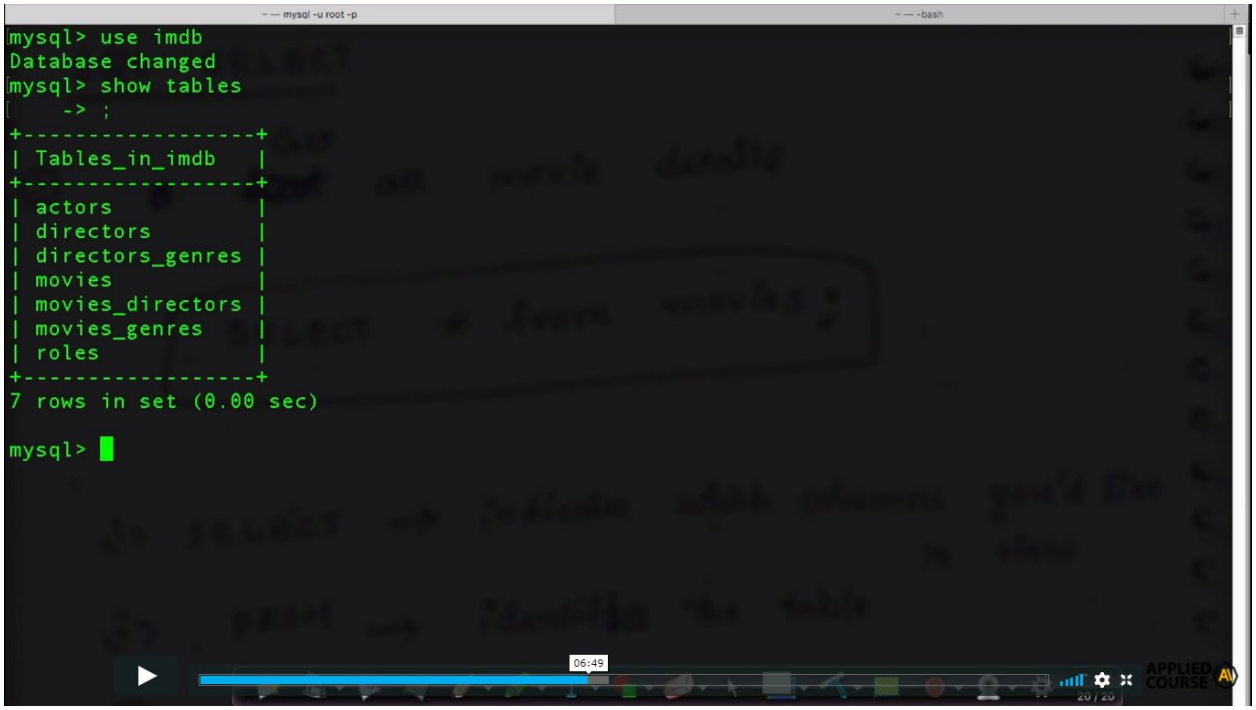

Macbook-air:~ varma$ mysql -u root -p
Imdb
SQL → interactive
applied angle
9 / 9
APPLIED COURSE

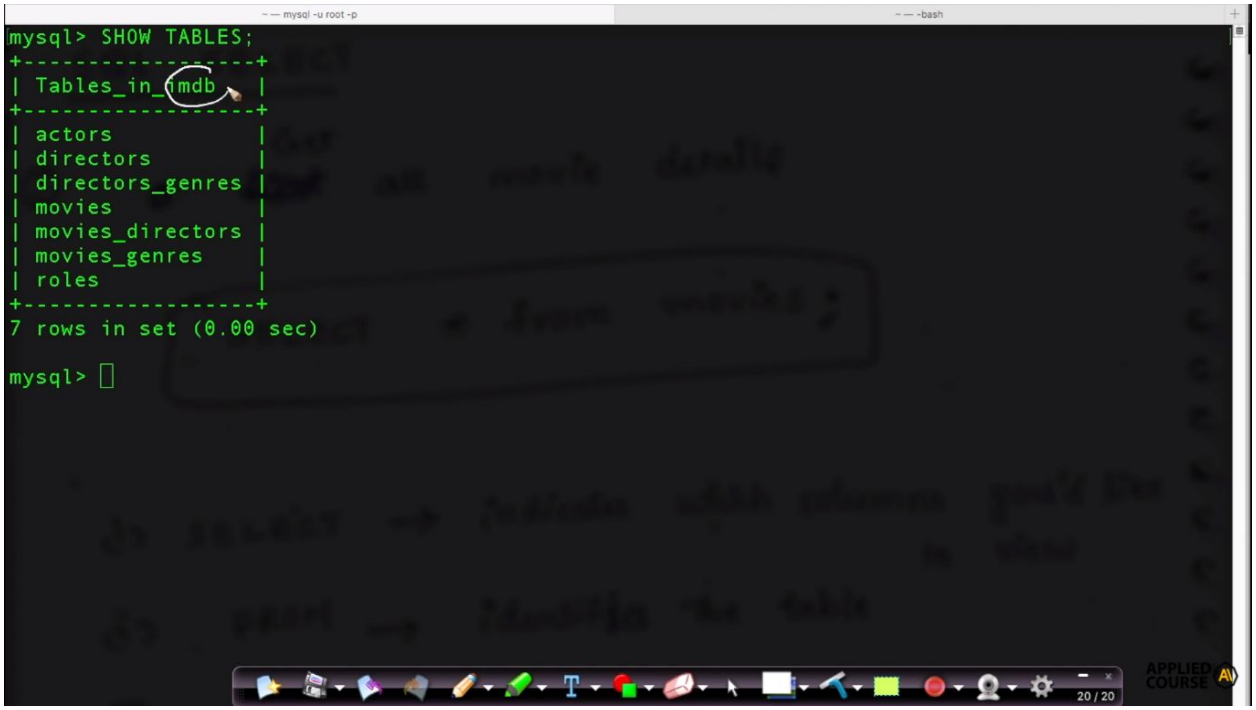

JupyterLab
mtodi7/Numpy-Panda
Applied Course
SQL IN DETAIL - Googl
Test Table
(1) LinkedIn
appliedaicourse.com/lecture/11/applied-machine-learning-online-course/3481/use-describe-show-tables/1/module-1-fundamentals-of-progra...
Macbook-air:~ varma$ mysql -u root -p
Mac
Windows
Linux
Terminal
02:14
APPLIED COURSE
How to utilise Appliedaicourse
Python for Data Science Introduction
Python for Data Science: Data Structures
Python for Data Science: Functions
Python for Data Science: Numpy
Python for Data Science: Matplotlib
Python for Data Science: Pandas
Prev
Next
imdb.sql
Show all
Type here to search
ENG
14:14
27-09-2019

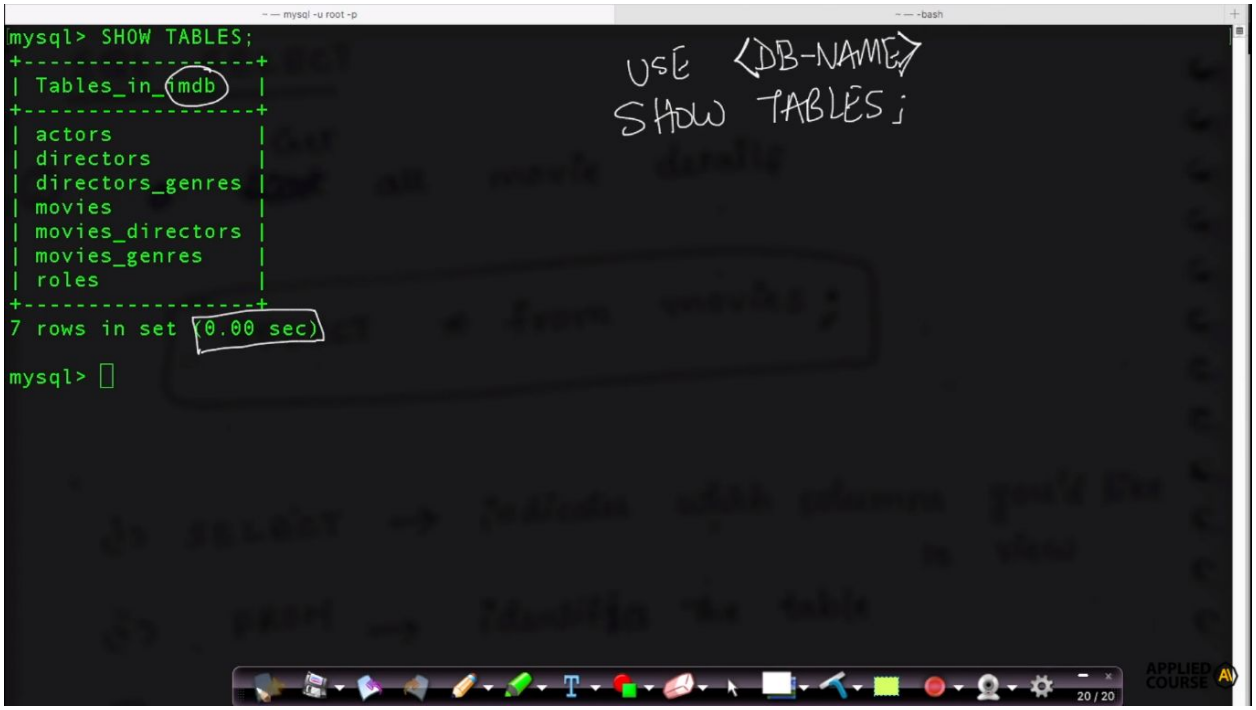

~ — mysql -u root -p
~ — -bash
Macbook-air:~ varma$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 8.0.13 MySQL Community Server - GPL
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
MySQL
DB1
DB2
DB3
T1
T2
T3
imdb
04:15
APPLIED COURSE

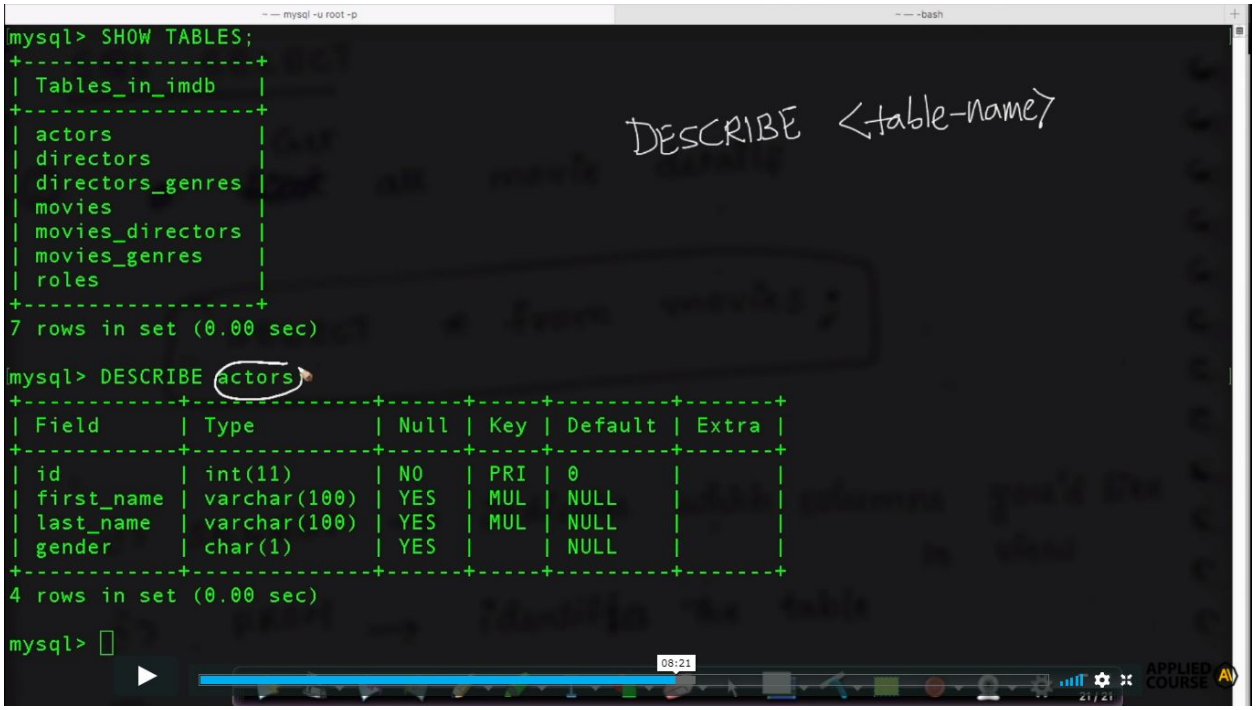

~ — mysql -u root -p
~ — -bash
Macbook-air:~ varma$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 8.0.13 MySQL Community Server - GPL
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> USE imdb
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql>
05:14
APPLIED COURSE

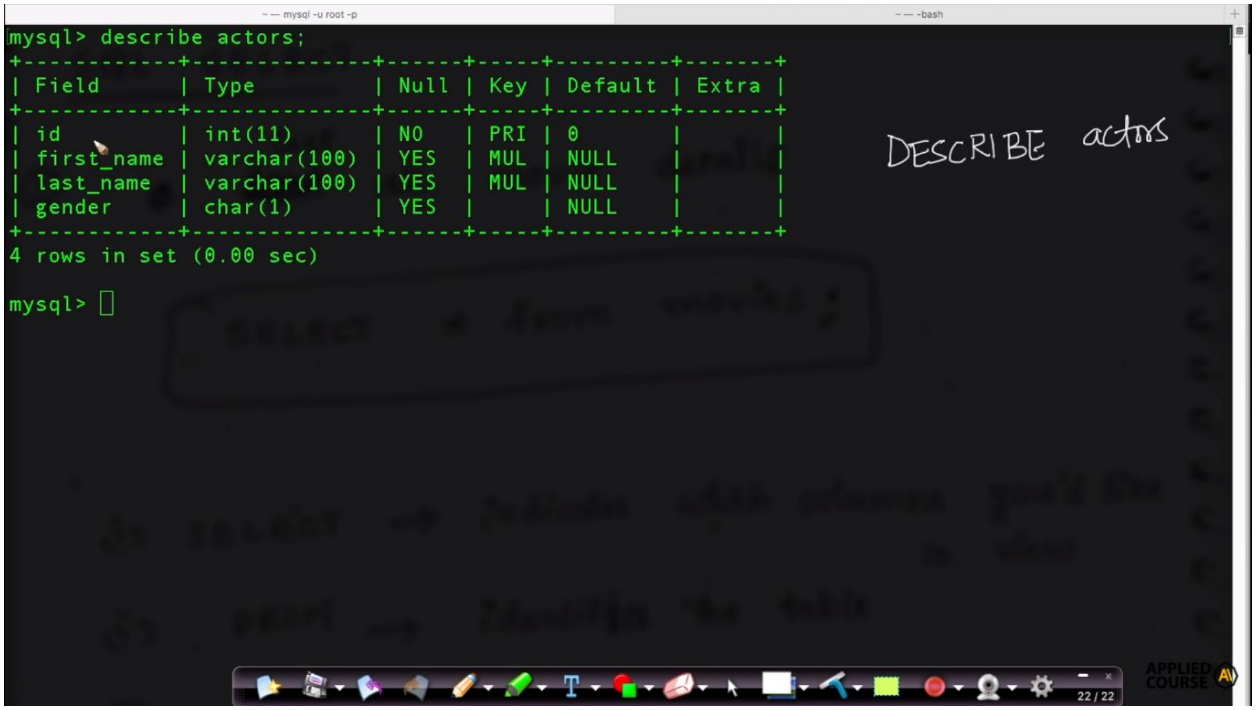

~ — mysql -u root -p
~ — -bash
mysql> use imdb
Database changed
mysql>
USE DB-NAME
06:18
APPLIED COURSE

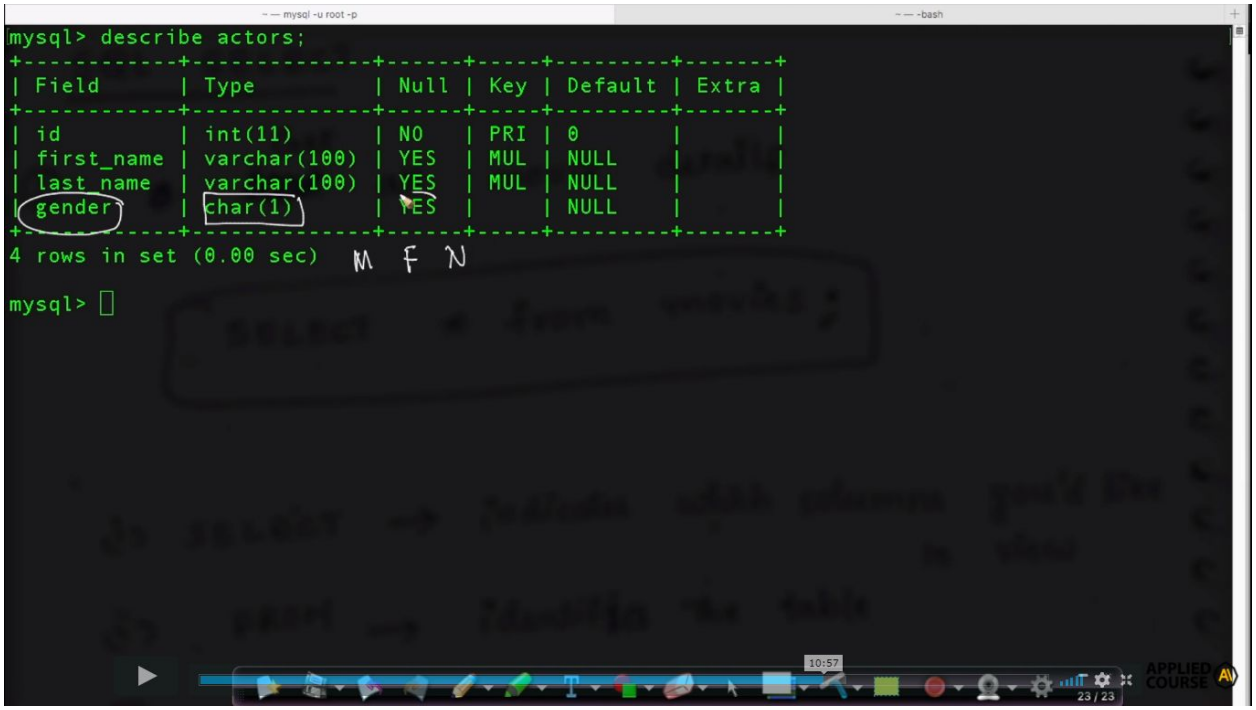

~ — mysql -u root -p
~ — -bash
mysql> use imdb
Database changed
mysql> show tables
-> ;
+------------------+
| Tables_in_imdb |
+------------------+
| actors |
| directors |
| directors_genres |
| movies |
| movies_directors |
| movies_genres |
| roles |
+------------------+
7 rows in set (0.00 sec)
mysql>
05:49
APPLIED COURSE

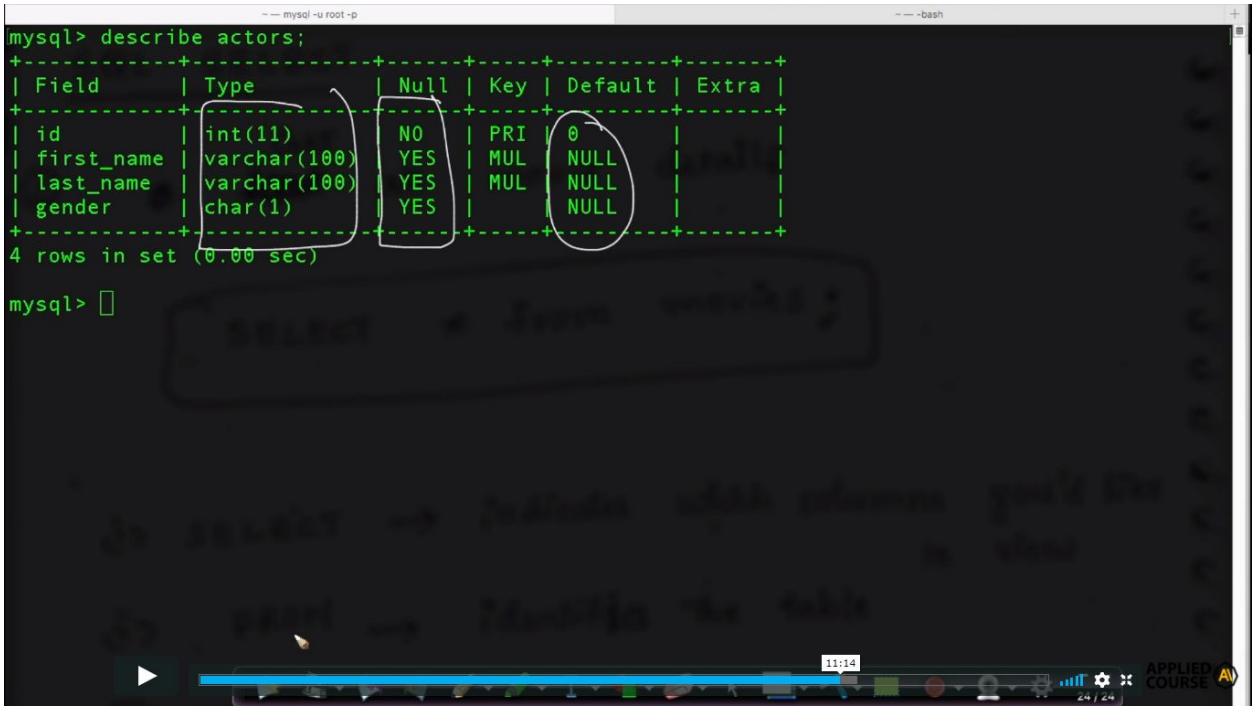

~ — mysql -u root -p
~ — -bash
mysql> SHOW TABLES;
+-----------------+
| Tables_in_imdb |
+-----------------+
| actors |
| directors |
| directors_genres |
| movies |
| movies_directors |
| movies_genres |
| roles |
+-----------------+
7 rows in set (0.00 sec)
mysql>
20 / 20
APPLIED COURSE AI

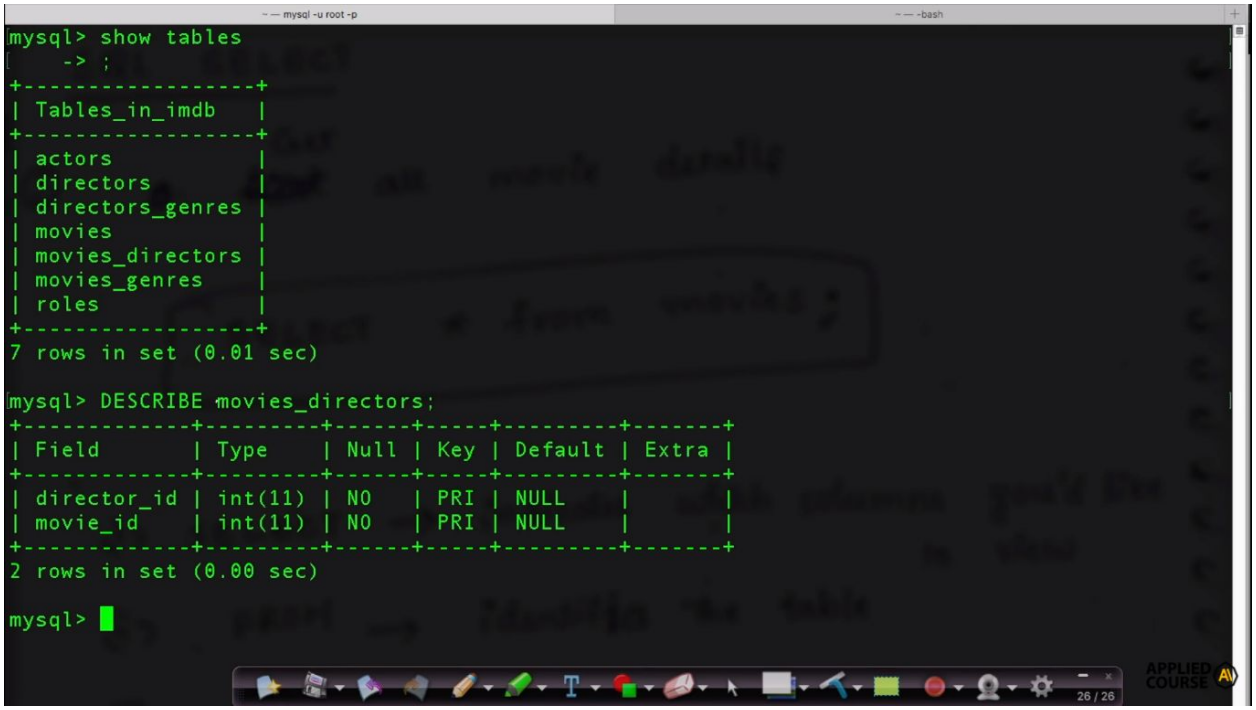

~ — mysql -u root -p
~ — -bash
mysql> SHOW TABLES;
+-----------------+
| Tables_in_imdb |
+-----------------+
| actors |
| directors |
| directors_genres |
| movies |
| movies_directors |
| movies_genres |
| roles |
+-----------------+
7 rows in set (0.00 sec)
mysql>
USE <DB-NAME>
SHOW TABLES;
20 / 20
APPLIED COURSE AI

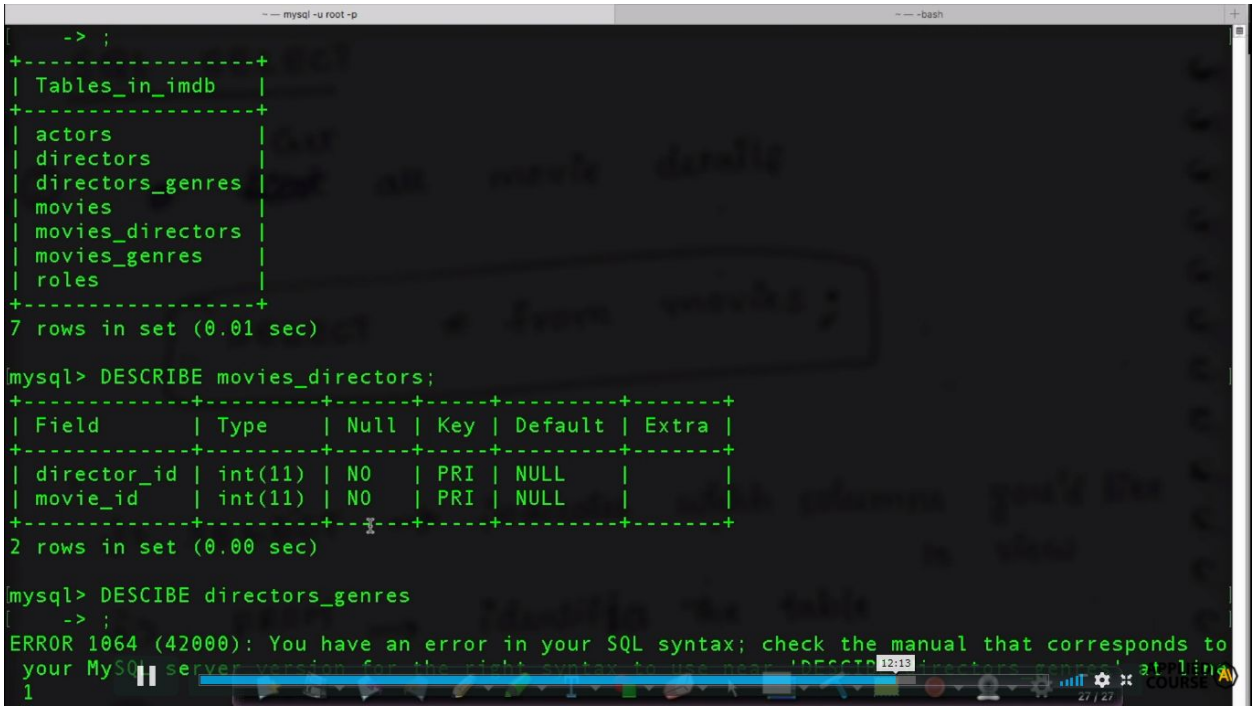

~ — mysql -u root -p
~ — -bash
mysql> SHOW TABLES;
+------------------+
| Tables_in_imdb |
+------------------+
| actors |
| directors |
| directors_genres |
| movies |
| movies_directors |
| movies_genres |
| roles |
+------------------+
7 rows in set (0.00 sec)
DESCRIBE <table-name>
mysql> DESCRIBE actors;
+------------+--------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
id	int(11)	NO	PRI	0	
first_name	varchar(100)	YES	MUL	NULL	
last_name	varchar(100)	YES	MUL	NULL	
gender	char(1)	YES		NULL	
+------------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
mysql>
08:21
APPLIED COURSE

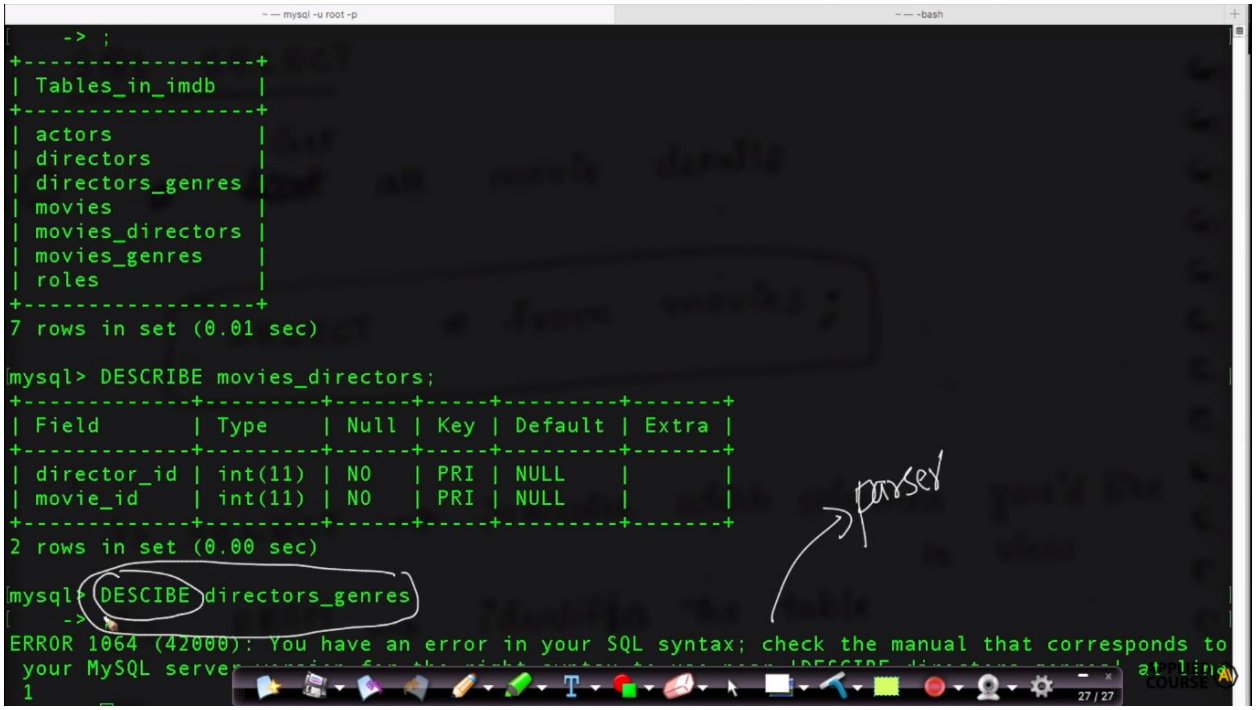

~ — mysql -u root -p
~ — -bash
mysql> describe actors;
+------------+--------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
id	int(11)	NO	PRI	0	
first_name	varchar(100)	YES	MUL	NULL	
last_name	varchar(100)	YES	MUL	NULL	
gender	char(1)	YES		NULL	
+------------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
mysql>
DESCRIBE actors
22 / 22
APPLIED COURSE

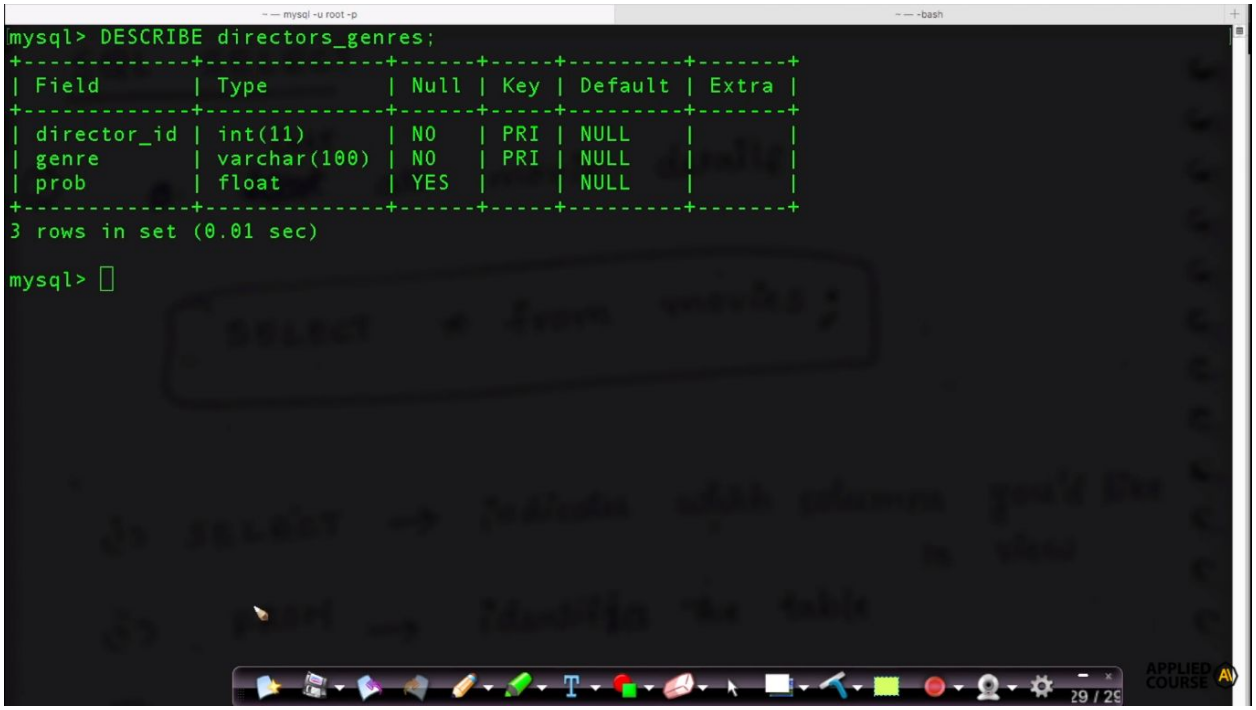

mysql> describe actors;
+------------+--------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
id	int(11)	NO	PRI	0	
first_name	varchar(100)	YES	MUL	NULL	
last_name	varchar(100)	YES	MUL	NULL	
gender	char(1)	YES		NULL	
+------------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
M F N
mysql>
10:57
23 / 23
APPLIED COURSE

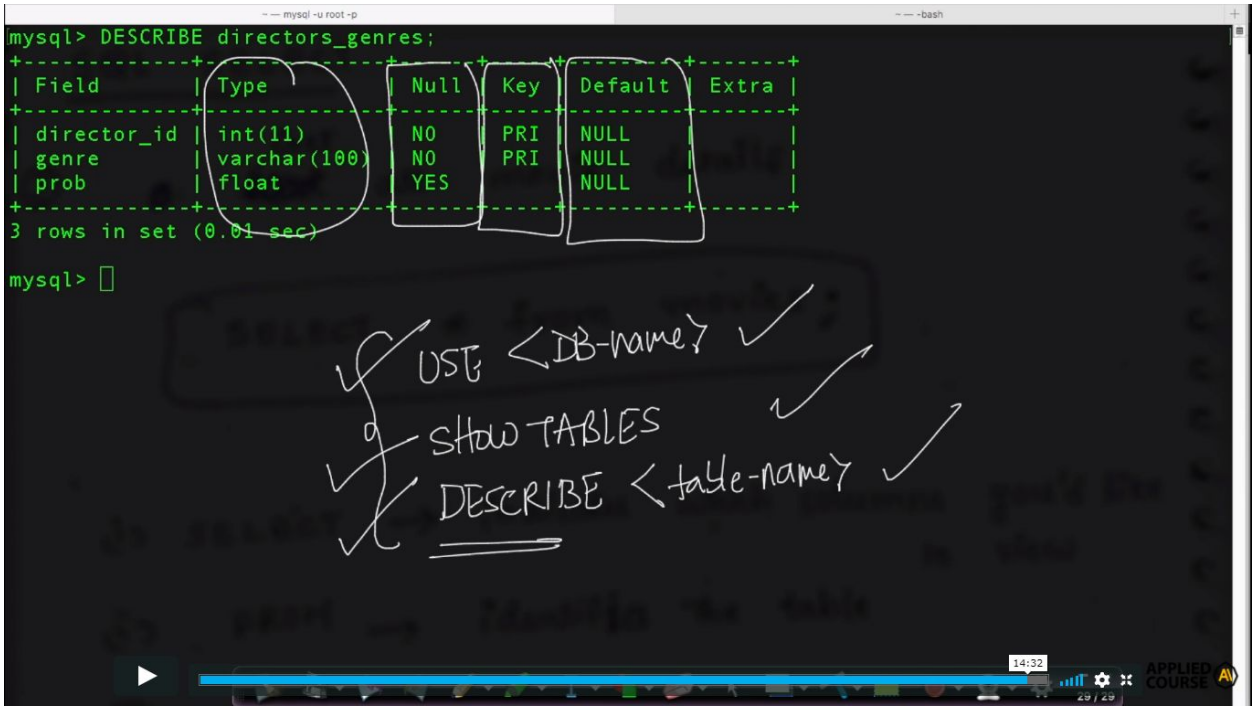

mysql> describe actors;
+------------+--------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
id	int(11)	NO	PRI	0	
first_name	varchar(100)	YES	MUL	NULL	
last_name	varchar(100)	YES	MUL	NULL	
gender	char(1)	YES		NULL	
+------------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
mysql>
11:14
APPLIED COURSE


~ — mysql -u root -p
~ — -bash
mysql> show tables
-> ;
+------------------+
| Tables_in_imdb |
+------------------+
| actors |
| directors |
| directors_genres |
| movies |
| movies_directors |
| movies_genres |
| roles |
+------------------+
7 rows in set (0.01 sec)
mysql> DESCRIBE movies_directors;
+-------------+---------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| director_id | int(11) | NO | PRI | NULL | |
| movie_id | int(11) | NO | PRI | NULL | |
+-------------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
mysql>
26 / 26
APPLIED COURSE

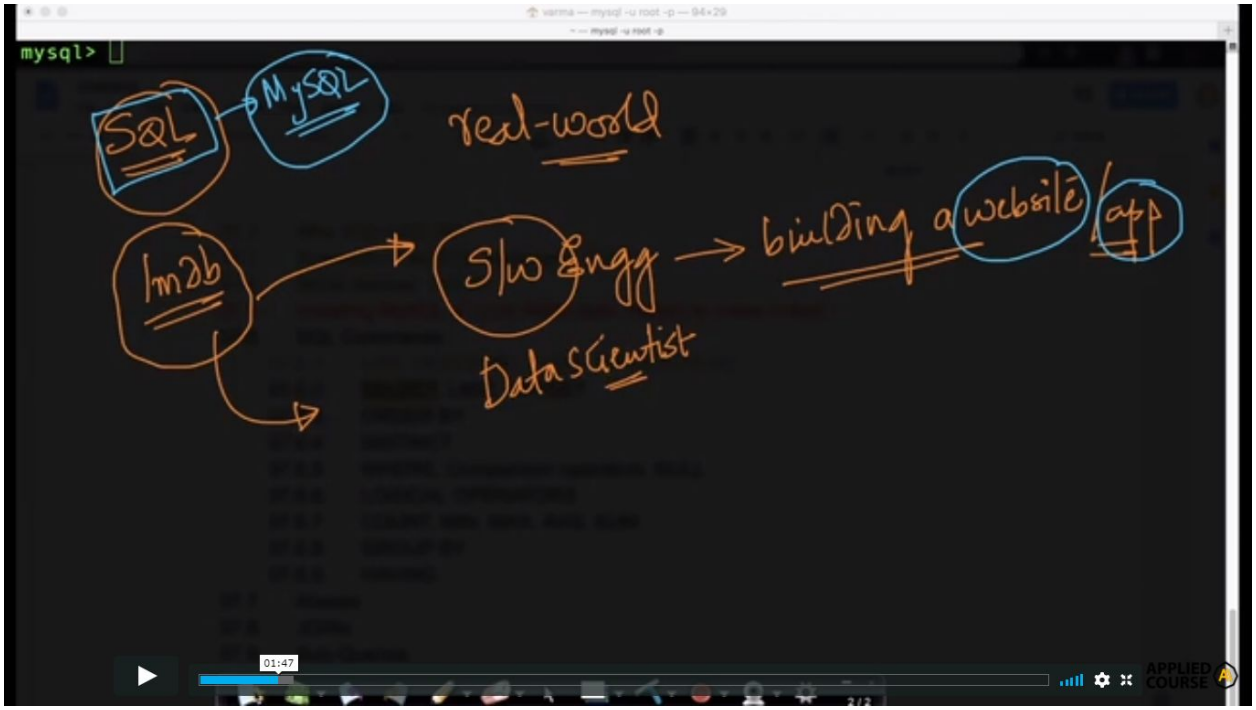

~ — mysql -u root -p
~ — -bash
-> ;
+------------------+
| Tables_in_imdb |
+------------------+
| actors |
| directors |
| directors_genres |
| movies |
| movies_directors |
| movies_genres |
| roles |
+------------------+
7 rows in set (0.01 sec)
mysql> DESCRIBE movies_directors;
+-------------+---------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| director_id | int(11) | NO | PRI | NULL | |
| movie_id | int(11) | NO | PRI | NULL | |
+-------------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
mysql> DESCIBE directors_genres
-> ;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to
your MySQL server version for the right syntax to use near 'DESCIBE directors_genres' at line
1
12:13
27 / 27
APPLIED COURSE

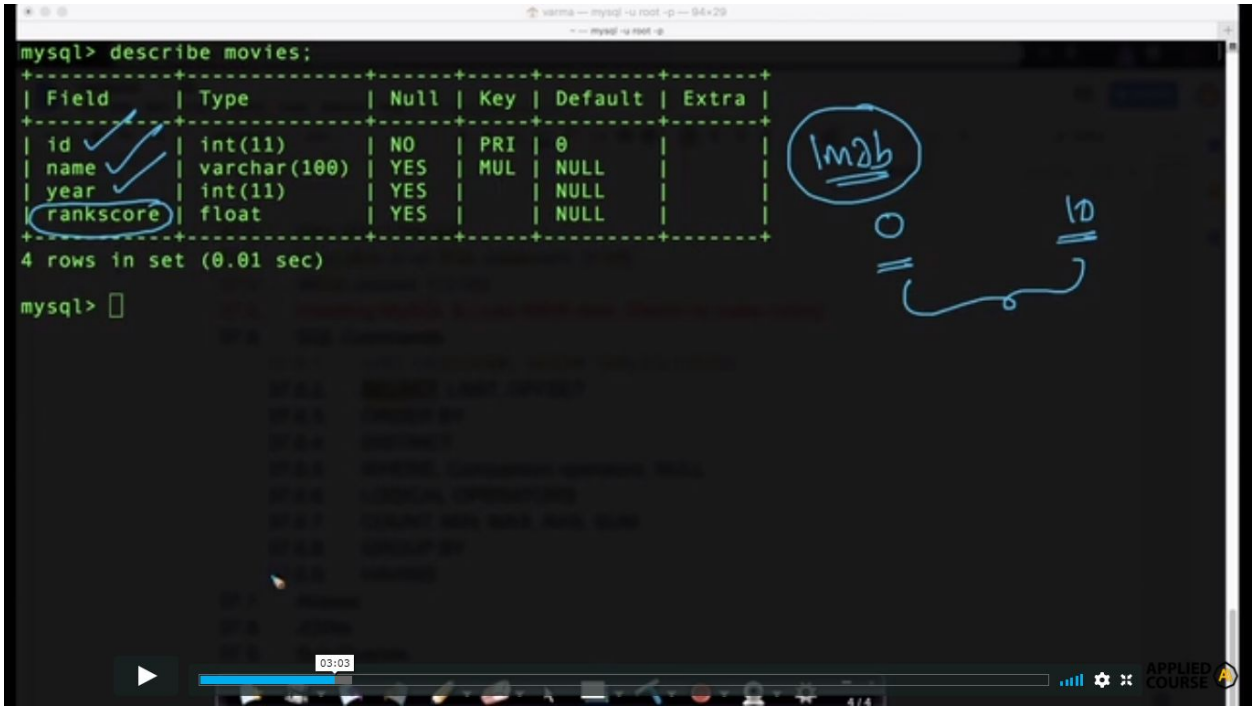

~ — mysql -u root -p
~ — -bash
-> ;
+------------------+
| Tables_in_imdb |
+------------------+
| actors |
| directors |
| directors_genres |
| movies |
| movies_directors |
| movies_genres |
| roles |
+------------------+
7 rows in set (0.01 sec)
mysql> DESCRIBE movies_directors;
+-------------+---------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| director_id | int(11) | NO | PRI | NULL | |
| movie_id | int(11) | NO | PRI | NULL | |
+-------------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
mysql> DESCIBE directors_genres
-> ;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to
your MySQL server
1
parsey
27 / 27

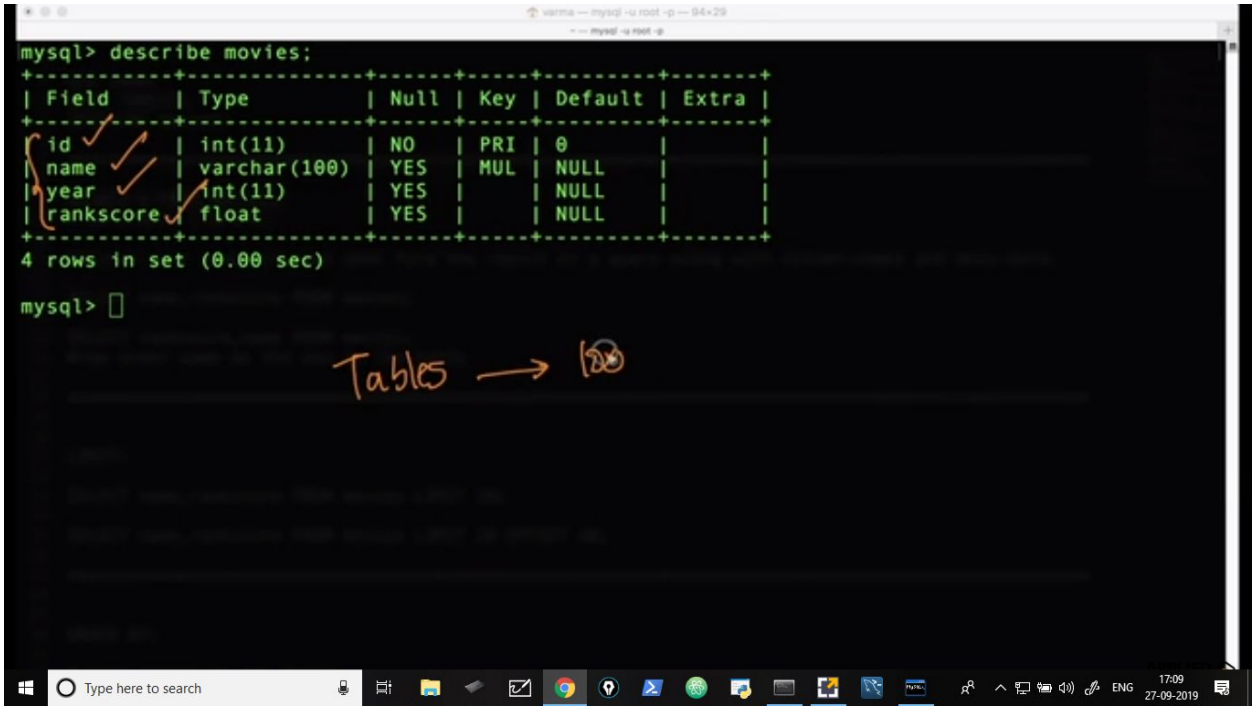

mysql> DESCRIBE directors_genres;
Field	Type	Null	Key	Default	Extra
director_id	int(11)	NO	PRI	NULL	
genre	varchar(100)	NO	PRI	NULL	
prob	float	YES		NULL	
3 rows in set (0.01 sec)
mysql>
APPLIED COURSE

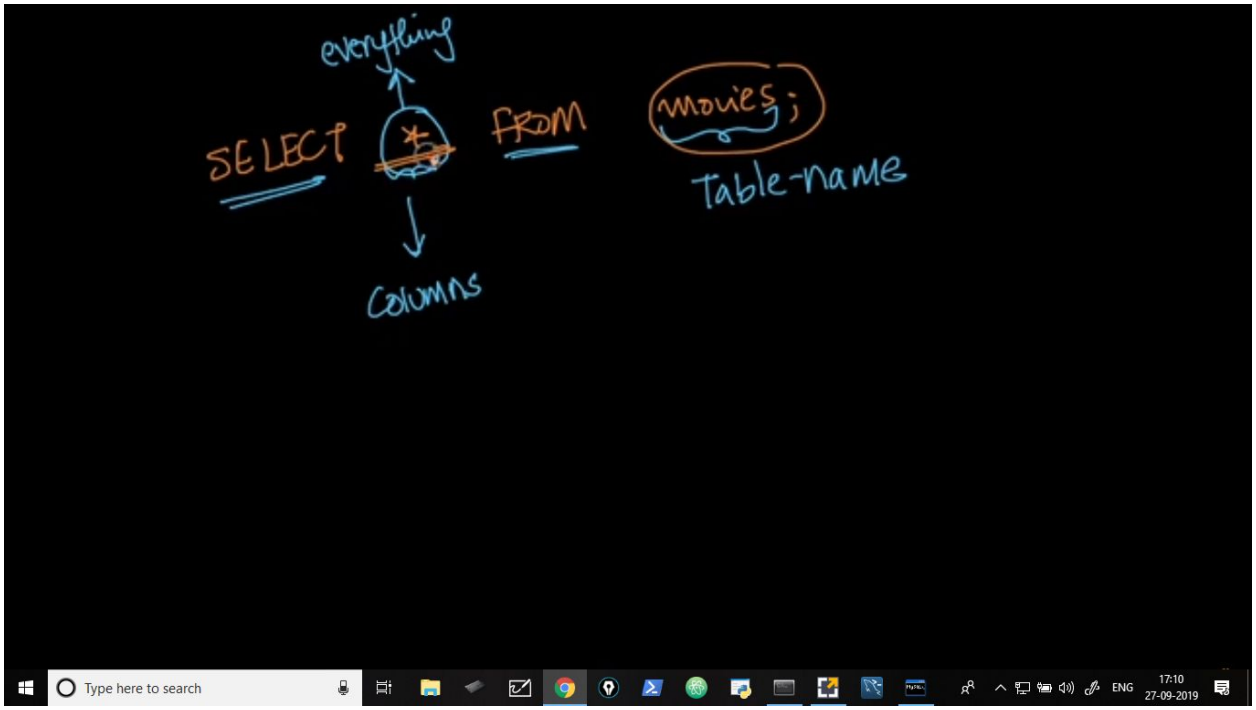

mysql> DESCRIBE directors_genres;
Field	Type	Null	Key	Default	Extra
director_id	int(11)	NO	PRI	NULL	
genre	varchar(100)	NO	PRI	NULL	
prob	float	YES		NULL	
3 rows in set (0.01 sec)
mysql>
USE <DB-name>
SHOW TABLES
DESCRIBE <table-name>
14:32
APPLIED COURSE

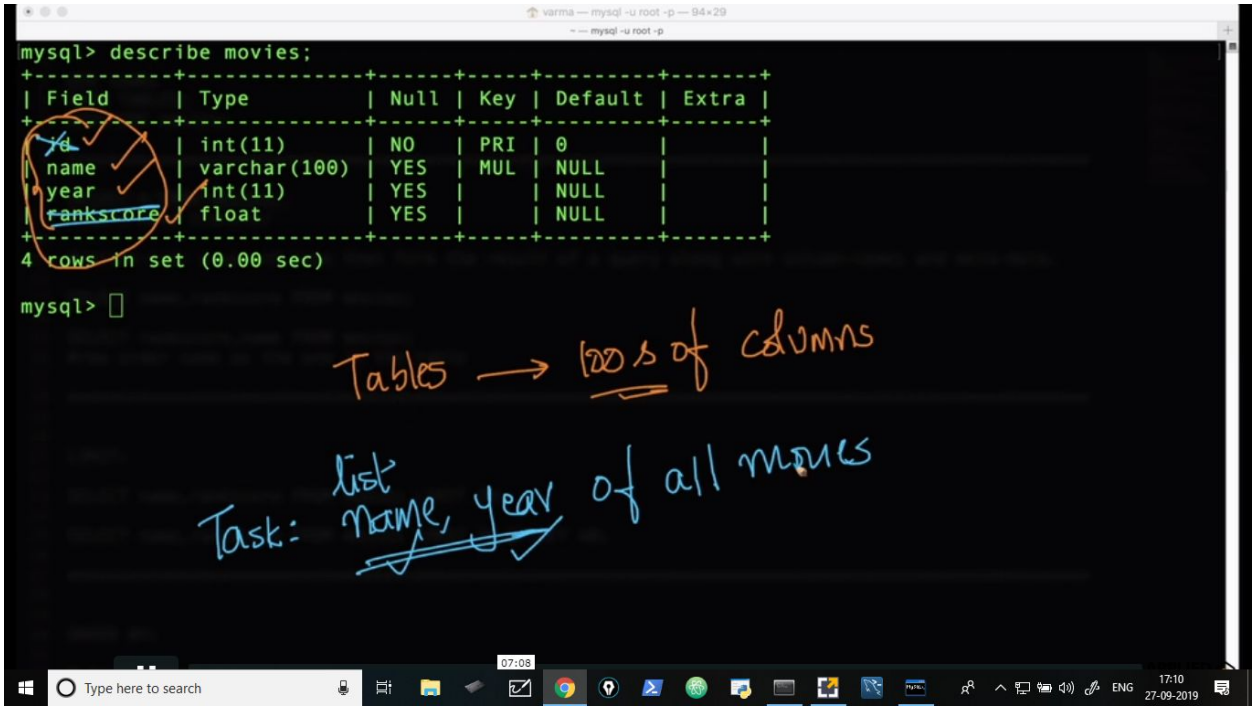
SELECT:

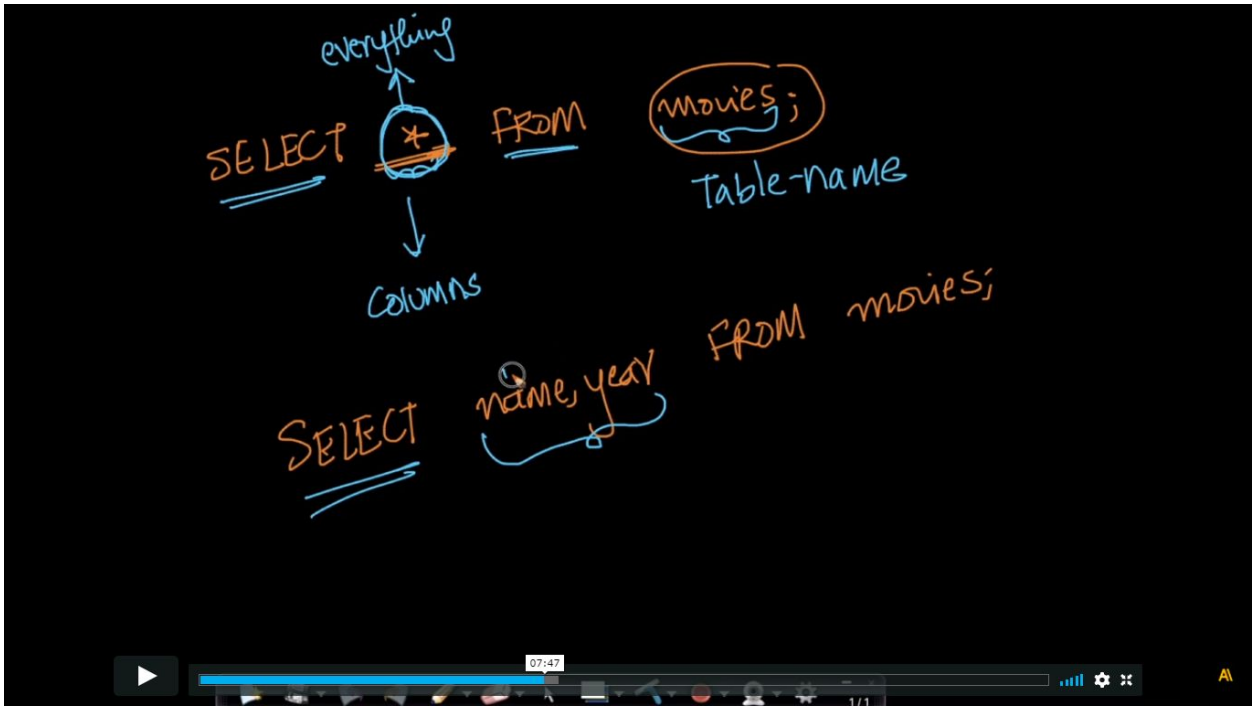

mysql>
SQL
MySQL
real-world
Imdb
S/w Engg
building a website
app
Data Scientist
01:47
APPLIED COURSE

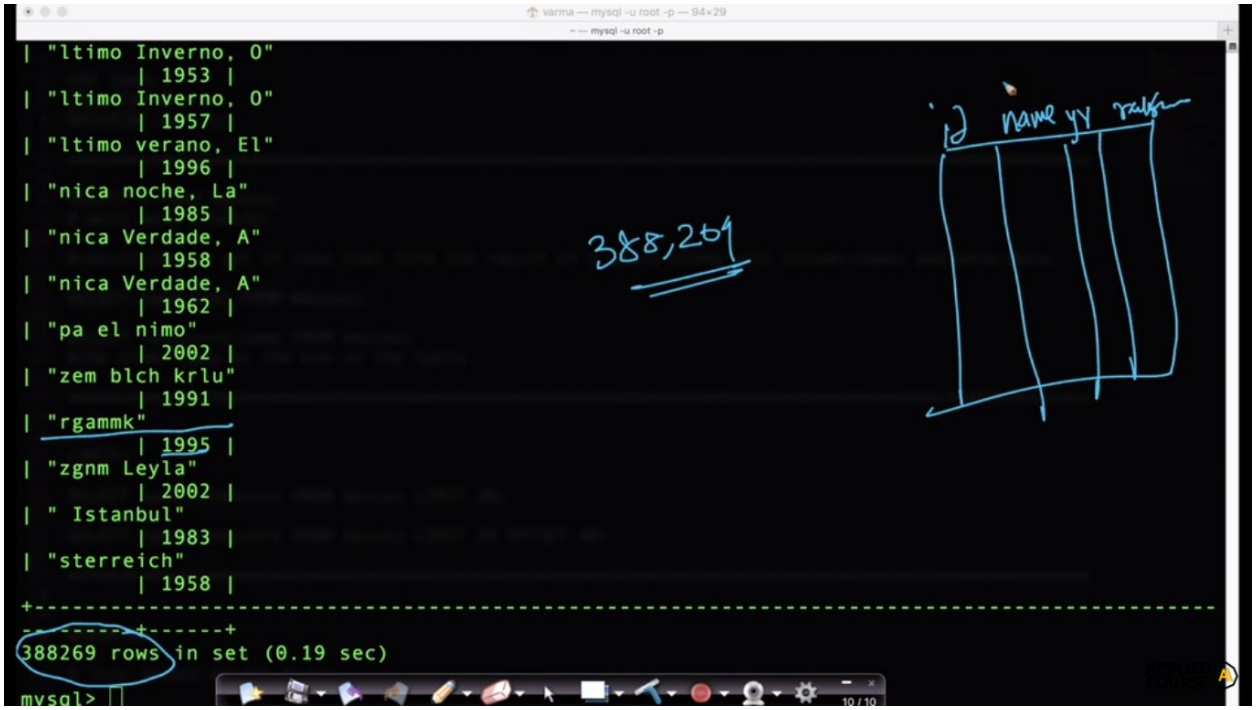

mysql> describe movies;
Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	0	
name	varchar(100)	YES	MUL	NULL	
year	int(11)	YES		NULL	
rankscore	float	YES		NULL	
4 rows in set (0.01 sec)
mysql>
Imdb
ID
03:03
APPLIED COURSE

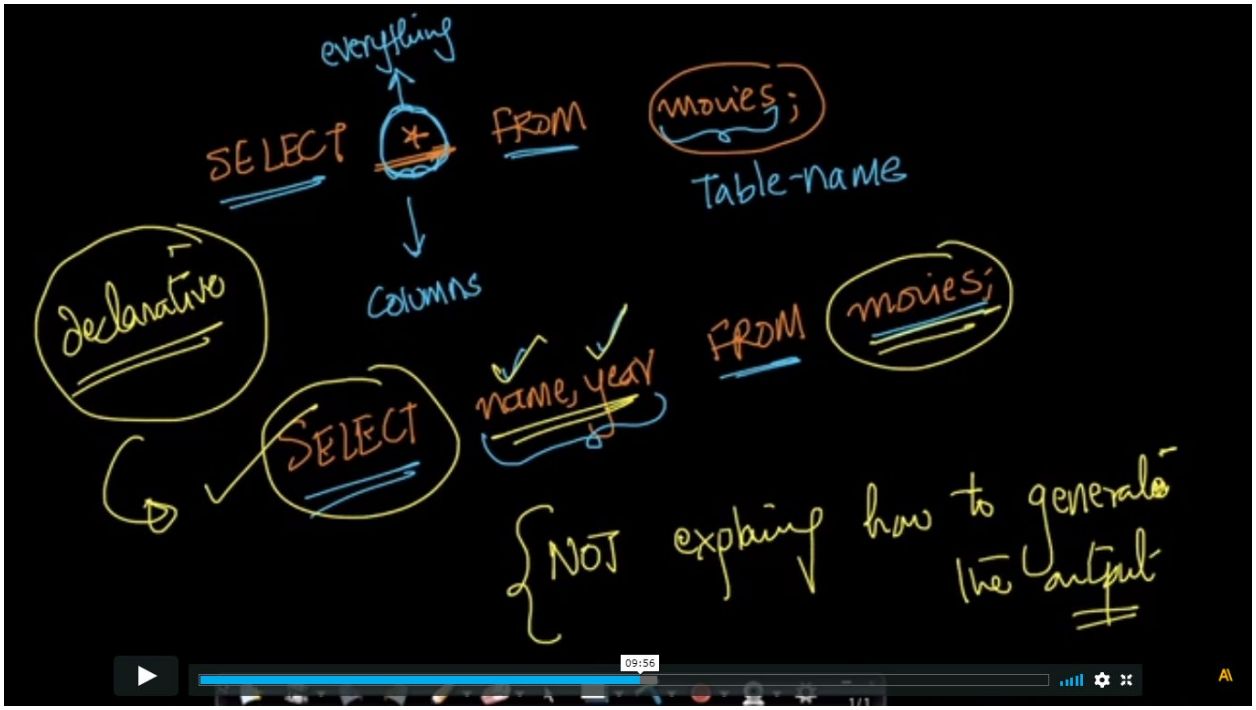

mysql> describe movies;
+-----------+--------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
id	int(11)	NO	PRI	0	
name	varchar(100)	YES	MUL	NULL	
year	int(11)	YES		NULL	
rankscore	float	YES		NULL	
+-----------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
mysql>
Tables → 100
Type here to search
ENG
17:09
27-09-2019

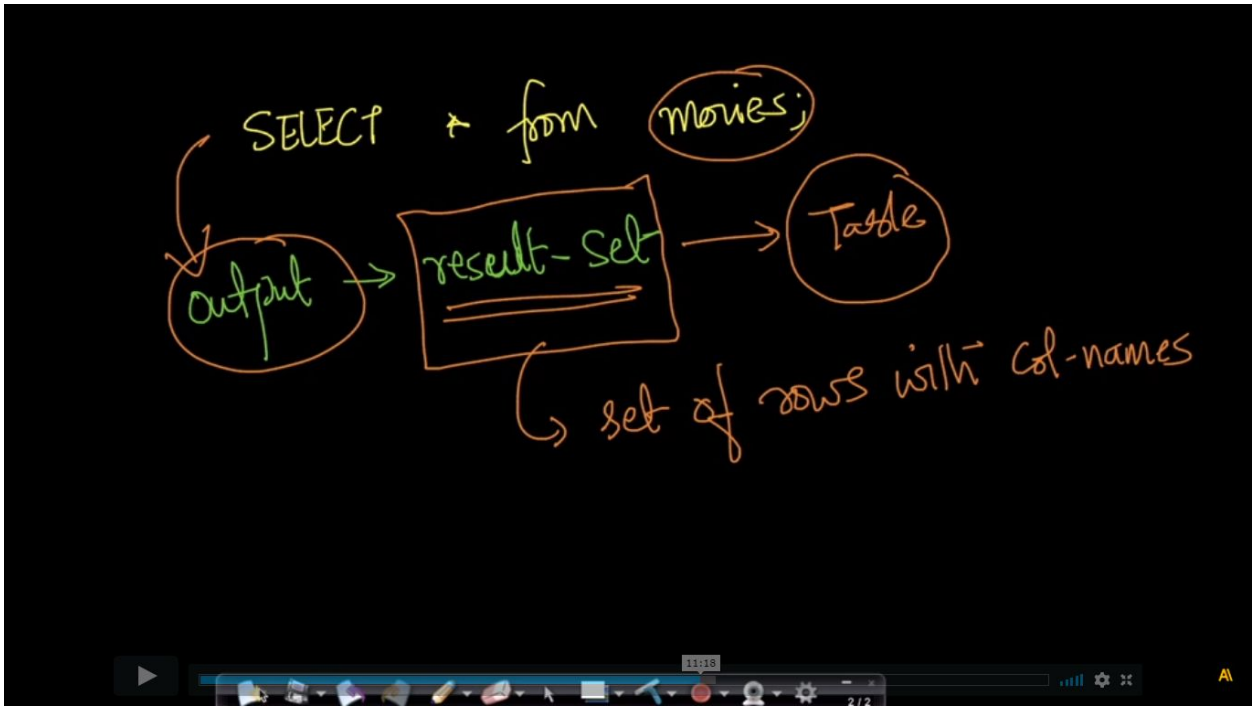

everything
SELECT
*
FROM
movies;
Table-name
Columns
Type here to search
ENG
17:10
27-09-2019

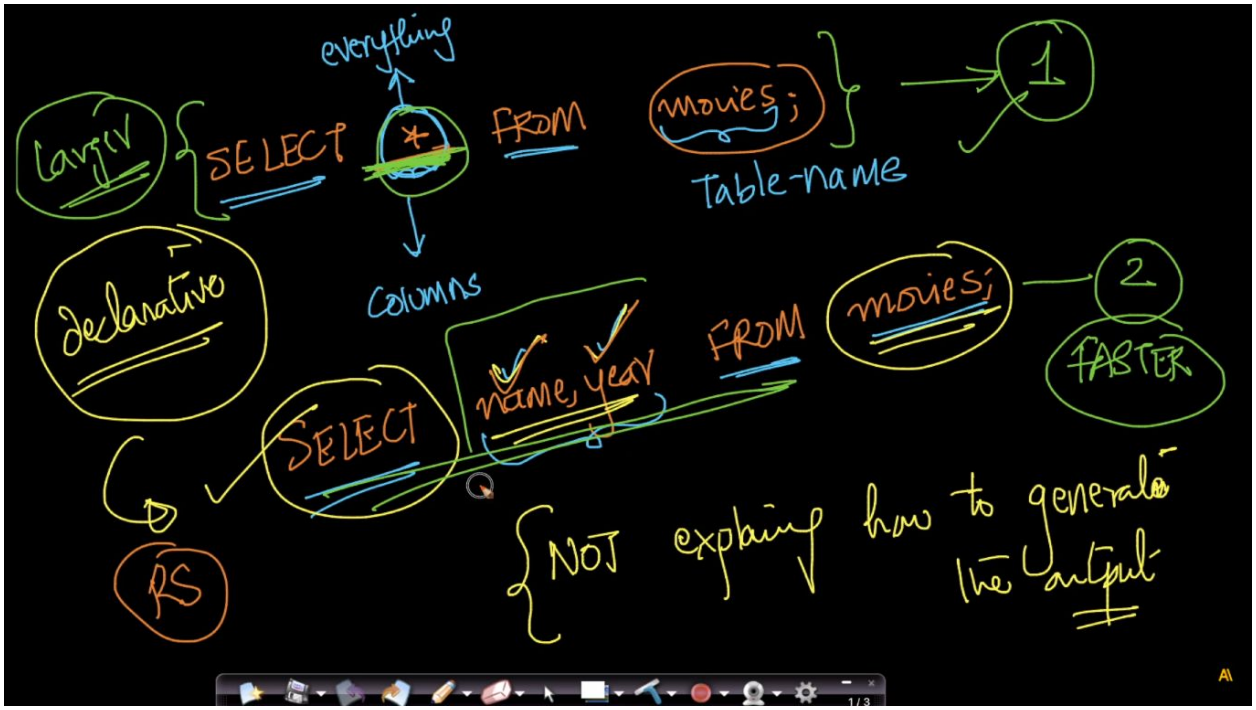

mysql> describe movies;
Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	0	
name	varchar(100)	YES	MUL	NULL	
year	int(11)	YES		NULL	
rankscore	float	YES		NULL	
4 rows in set (0.00 sec)
mysql>
Tables → 100s of columns
Task: list name, year of all movies
07:08
Type here to search
ENG
17:10
27-09-2019

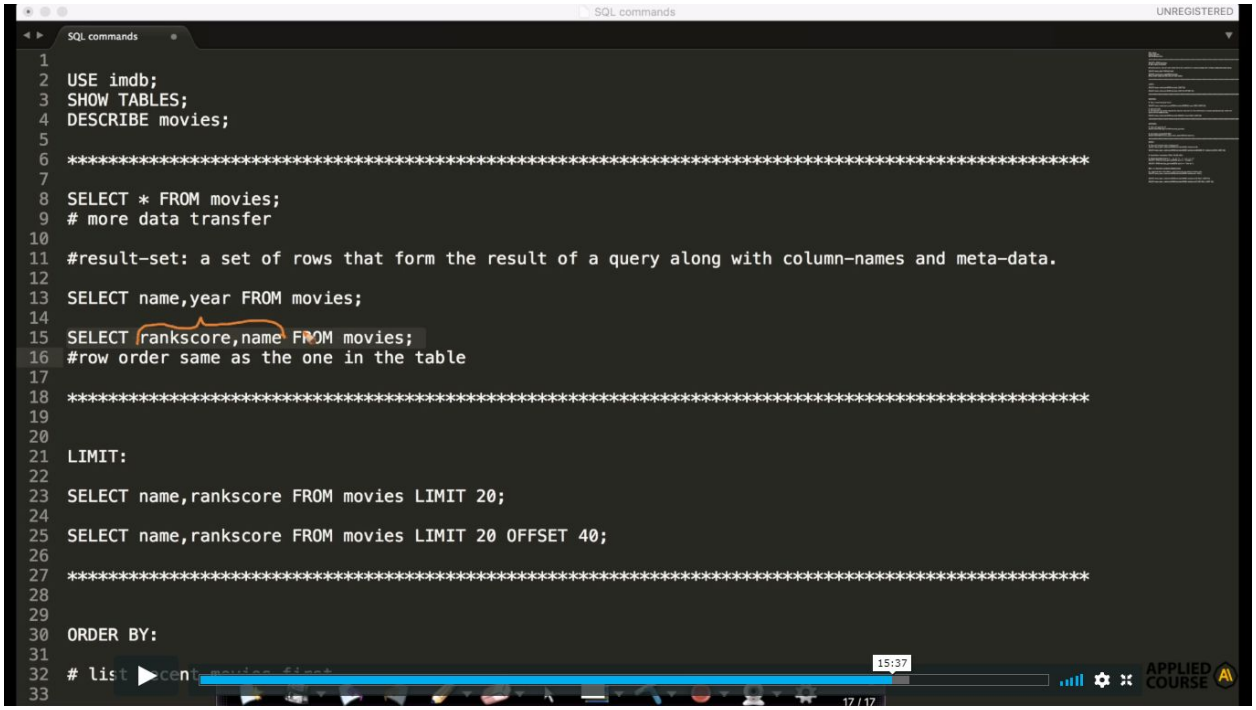

everything
SELECT * FROM movies;
Table-name
columns
SELECT name, year FROM movies;
07:47

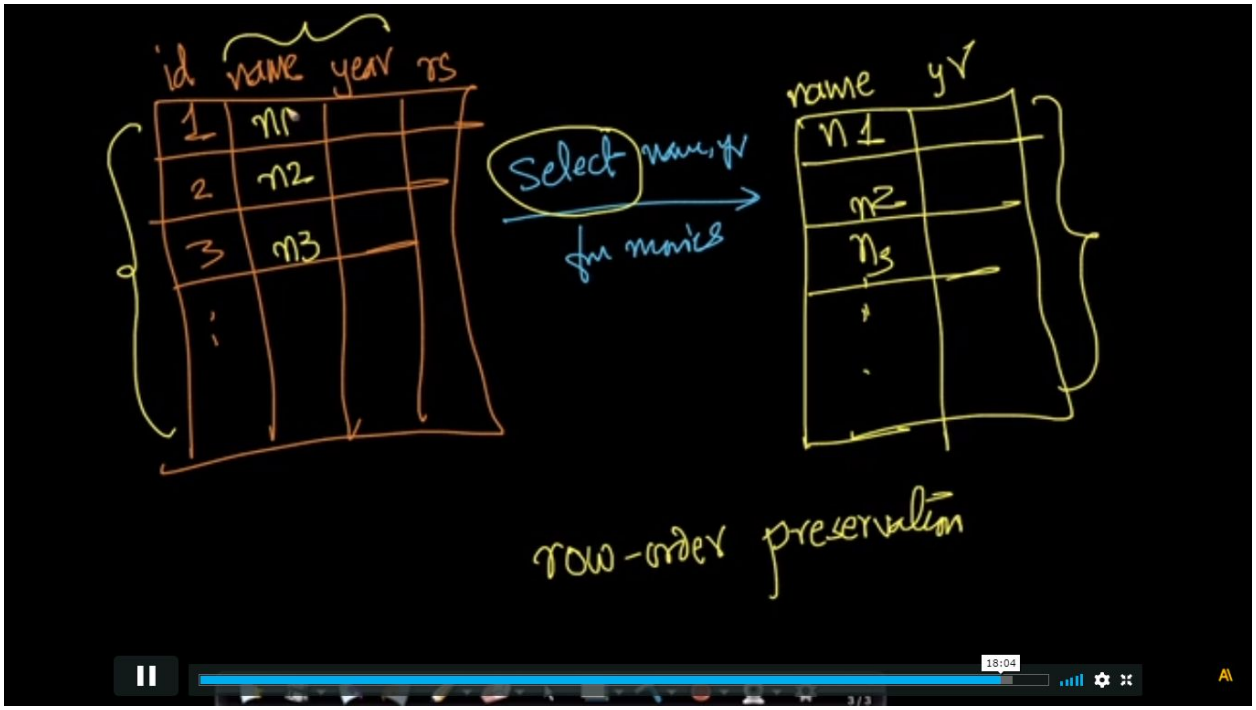

varma — mysql -u root -p — 94×29
mysql -u root -p
"ltimo Inverno, O"	1953
"ltimo Inverno, O"	1957
"ltimo verano, El"	1996
"nica noche, La"	1985
"nica Verdade, A"	1958
"nica Verdade, A"	1962
"pa el nimo"	2002
"zem blch krlu"	1991
"rgammk"	1995
"zgnm Leyla"	2002
" Istanbul"	1983
"sterreich"	1958
388269 rows in set (0.19 sec)
mysql>
388,269
name yy
10 / 10

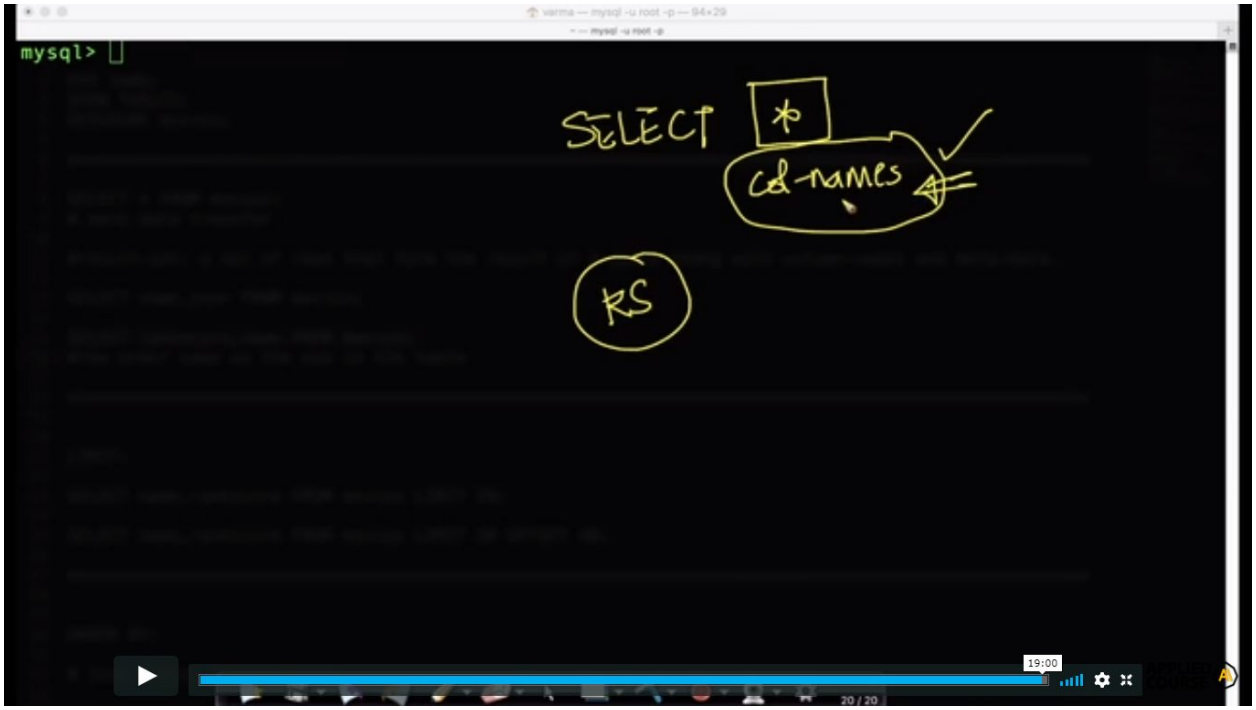

everything
SELECT
*
FROM
movies;
Table-name
columns
declarative
SELECT
name, year
FROM
movies;
{NOT explaining how to generate the output
09:56


SELECT * from movies;
output
result-set
Table
set of rows with col-names

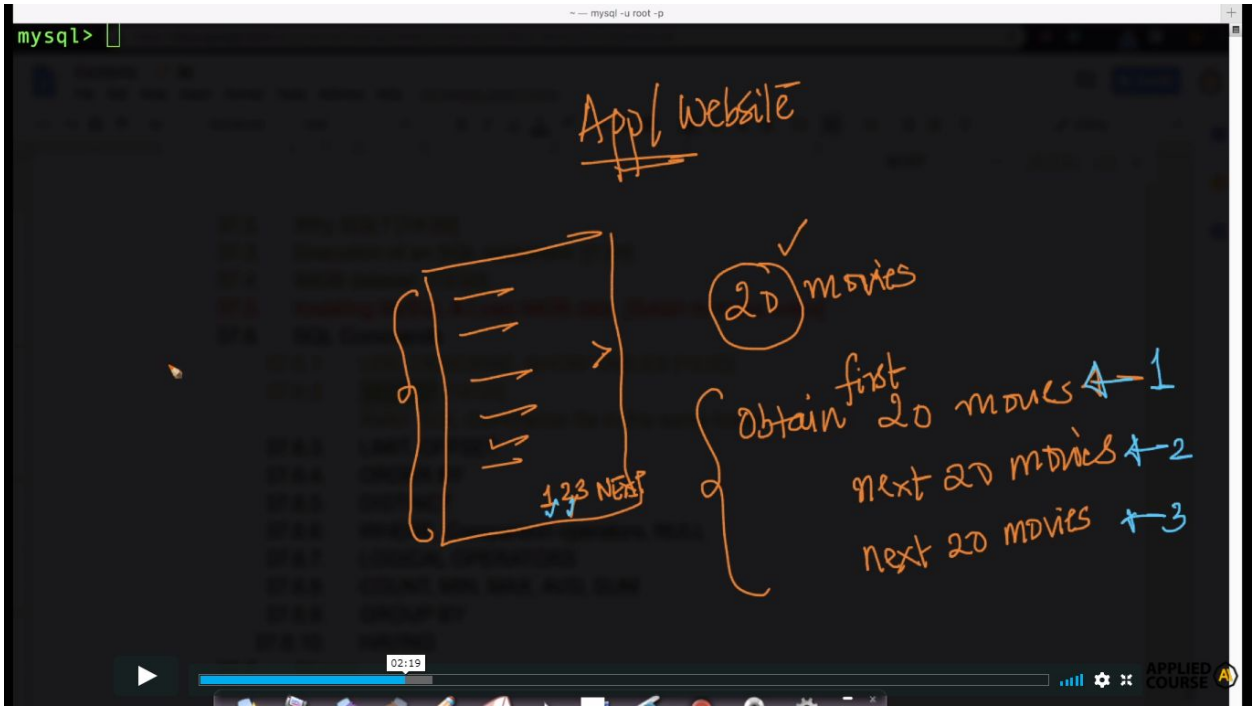

everything
Lazy
SELECT
*
FROM
movies;
Table-name
1
Declarative
Columns
SELECT
name, year
FROM
movies;
2
FASTER
RS
{NOT explaining how to generate the output

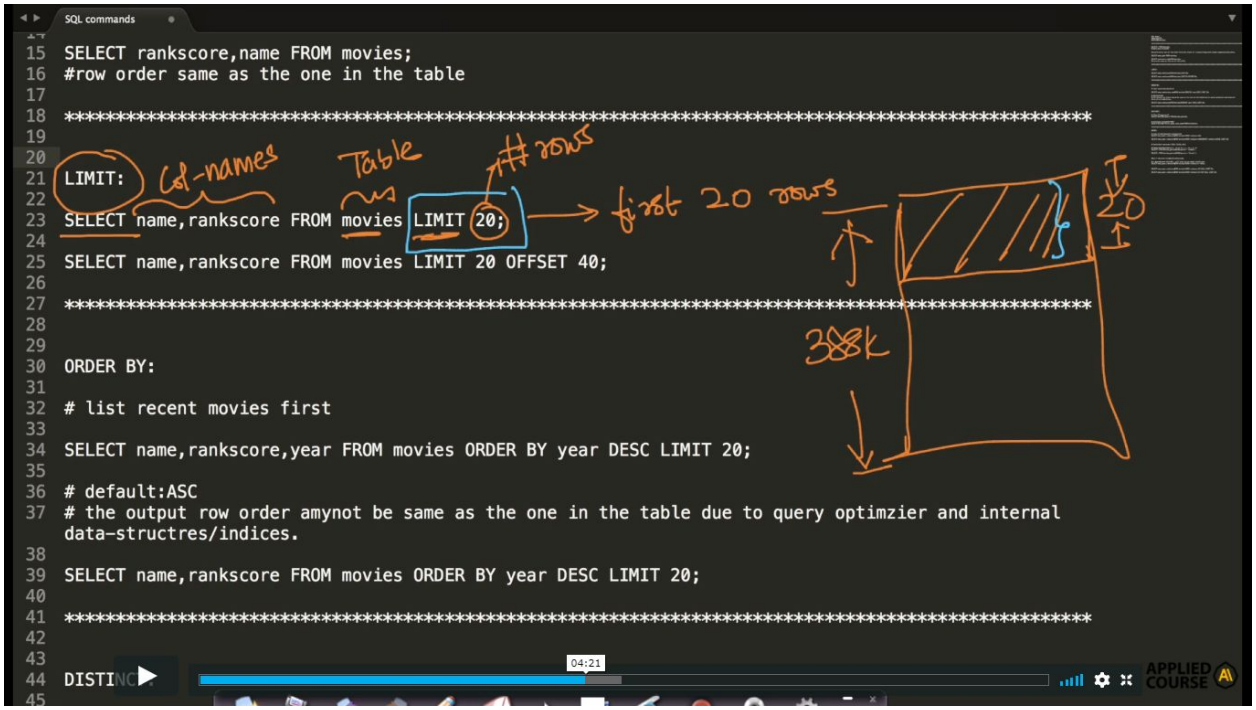

SQL commands
UNREGISTERED
SQL commands
USE imdb;
SHOW TABLES;
DESCRIBE movies;
****
SELECT * FROM movies;
more data transfer
#result-set: a set of rows that form the result of a query along with column-names and meta-data.
SELECT name,year FROM movies;
SELECT rankscore,name FROM movies;
#row order same as the one in the table
****
LIMIT:
SELECT name,rankscore FROM movies LIMIT 20;
SELECT name,rankscore FROM movies LIMIT 20 OFFSET 40;
****
ORDER BY:
15:37
APPLIED COURSE

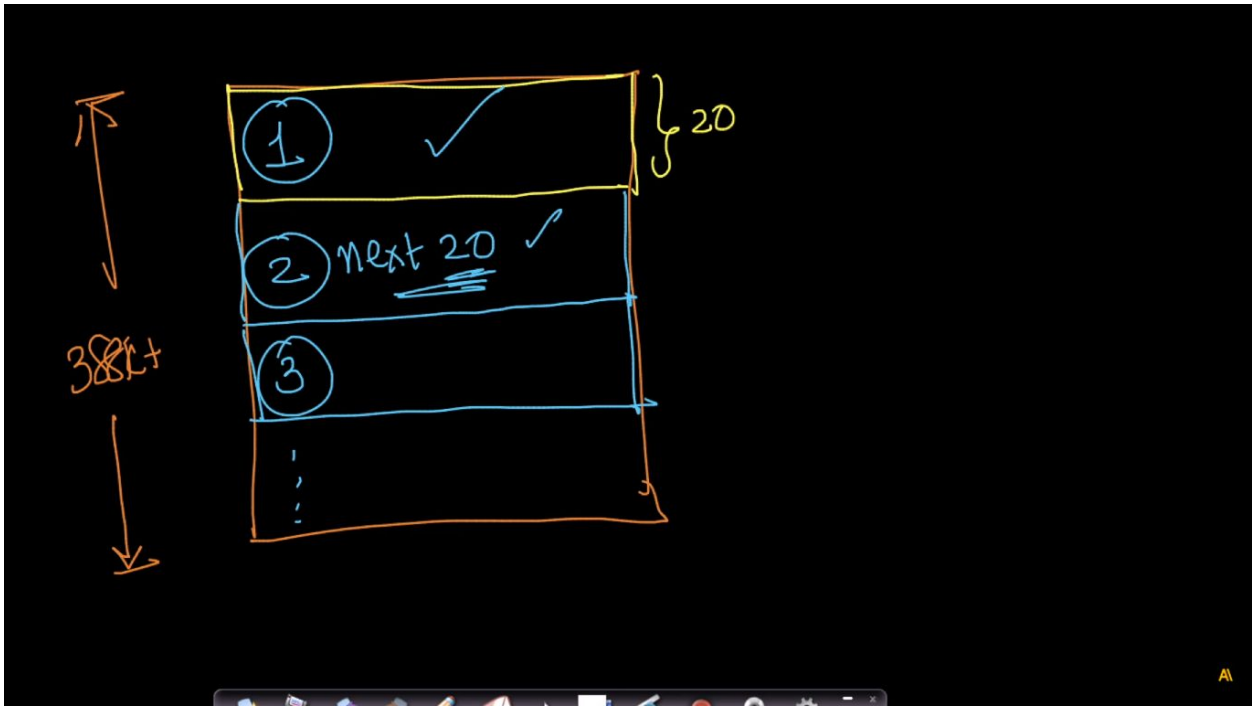


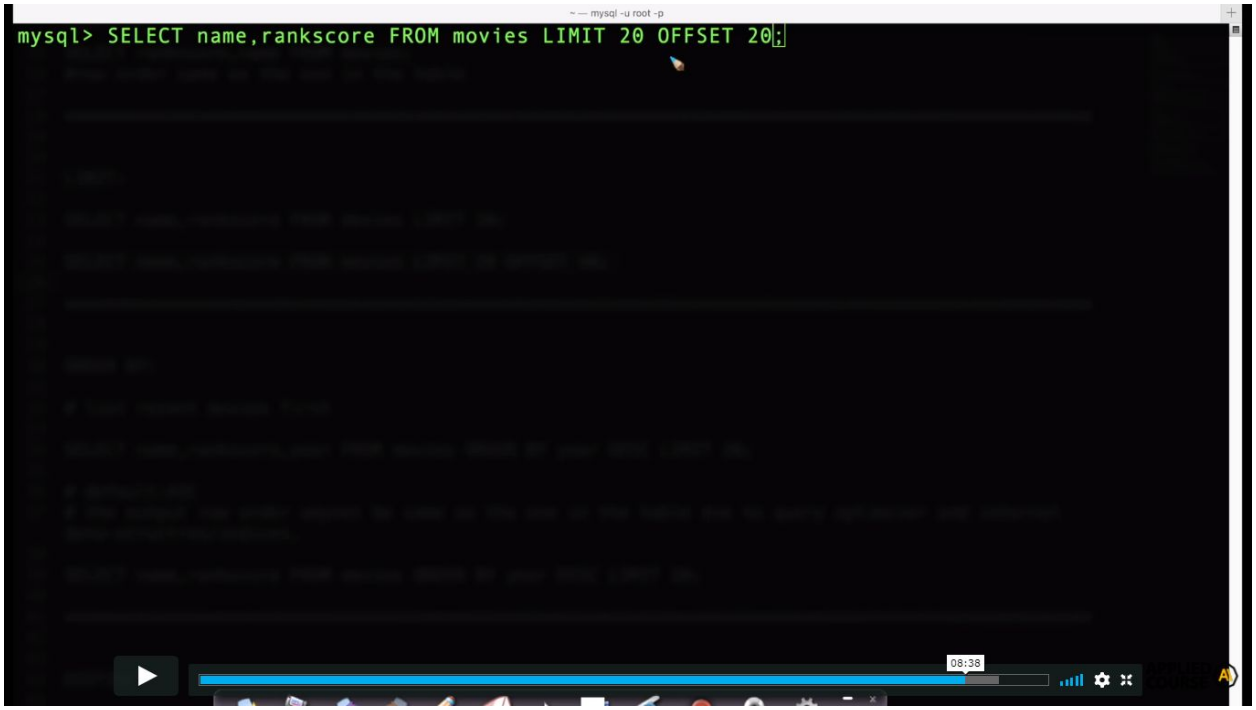


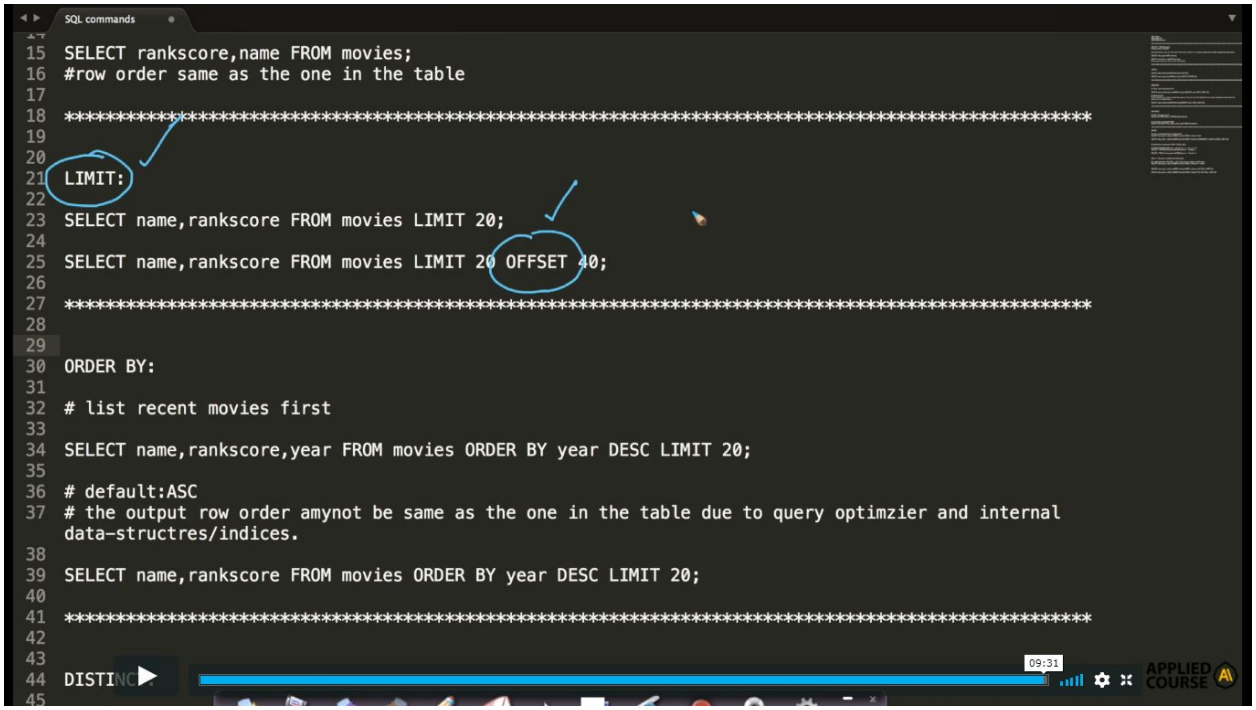
LIMIT, OFFSET


~ — mysql -u root -p
mysql>
App/ Website
20 movies
1 2 3 NEXT
Obtain first 20 movies ← 1
next 20 movies ← 2
next 20 movies ← 3
02:19
APPLIED COURSE


SQL commands
SELECT rankscore,name FROM movies;
#row order same as the one in the table
LIMIT:
Col-names
Table
rows
SELECT name,rankscore FROM movies LIMIT 20;
first 20 rows
SELECT name,rankscore FROM movies LIMIT 20 OFFSET 40;
20
388k
ORDER BY:
list recent movies first
SELECT name,rankscore,year FROM movies ORDER BY year DESC LIMIT 20;
default:ASC
the output row order amynot be same as the one in the table due to query optimzier and internal data-structres/indices.
SELECT name,rankscore FROM movies ORDER BY year DESC LIMIT 20;
DISTI
04:21
APPLIED COURSE


1
20
2
next 20
3
388



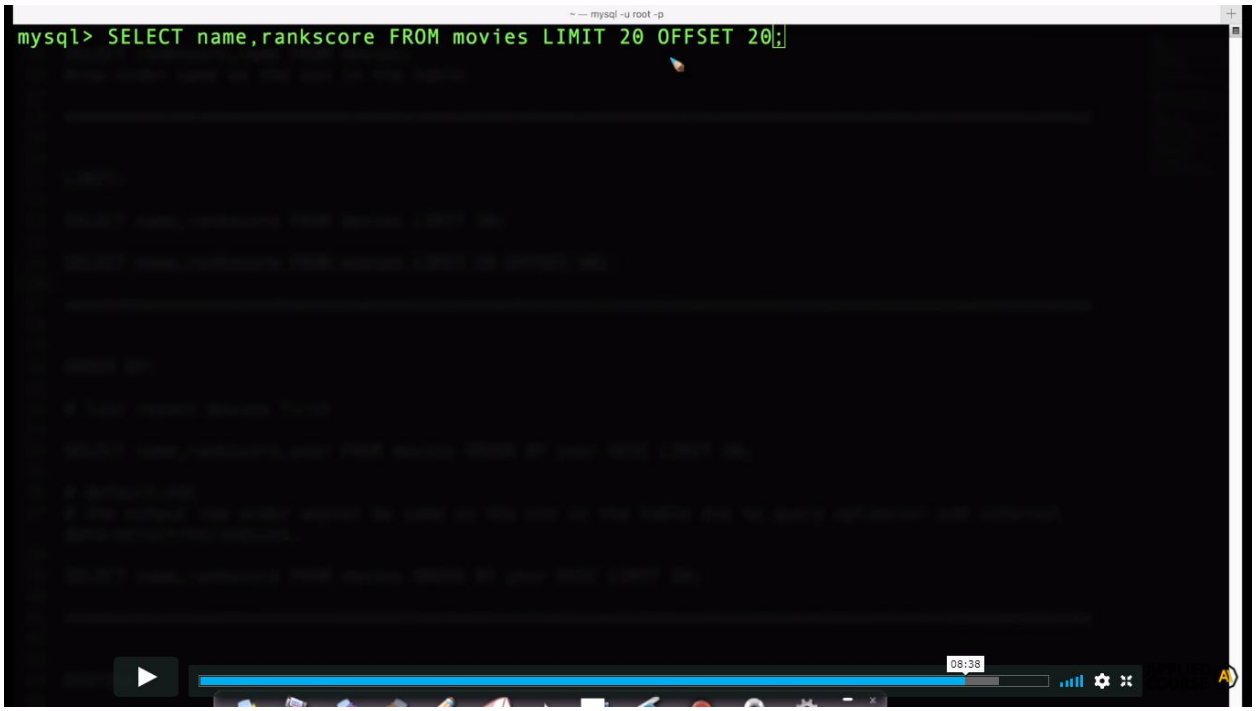


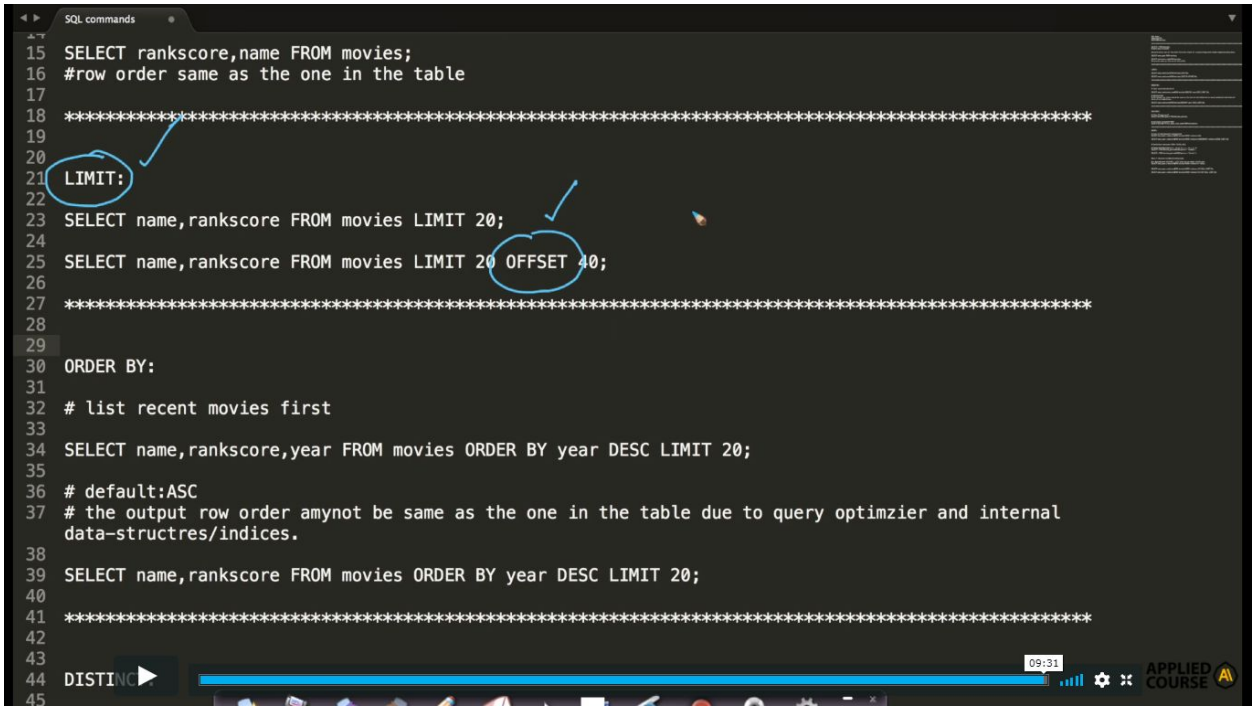
LIMIT:

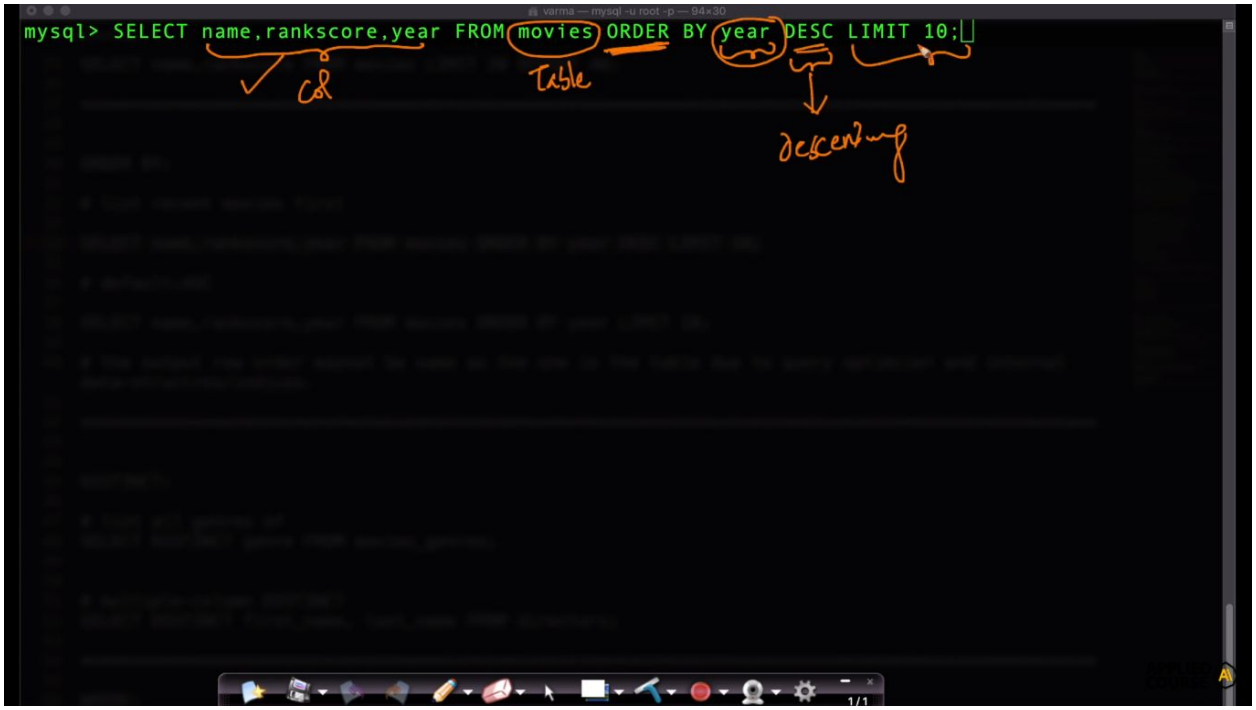
SELECT name,rankscore FROM movies LIMIT 20;

SELECT name,rankscore FROM movies LIMIT 20 OFFSET 40;
→ This command implies to ignore first 40 rows and print next 20 rows ignorning firsr 40 rows.

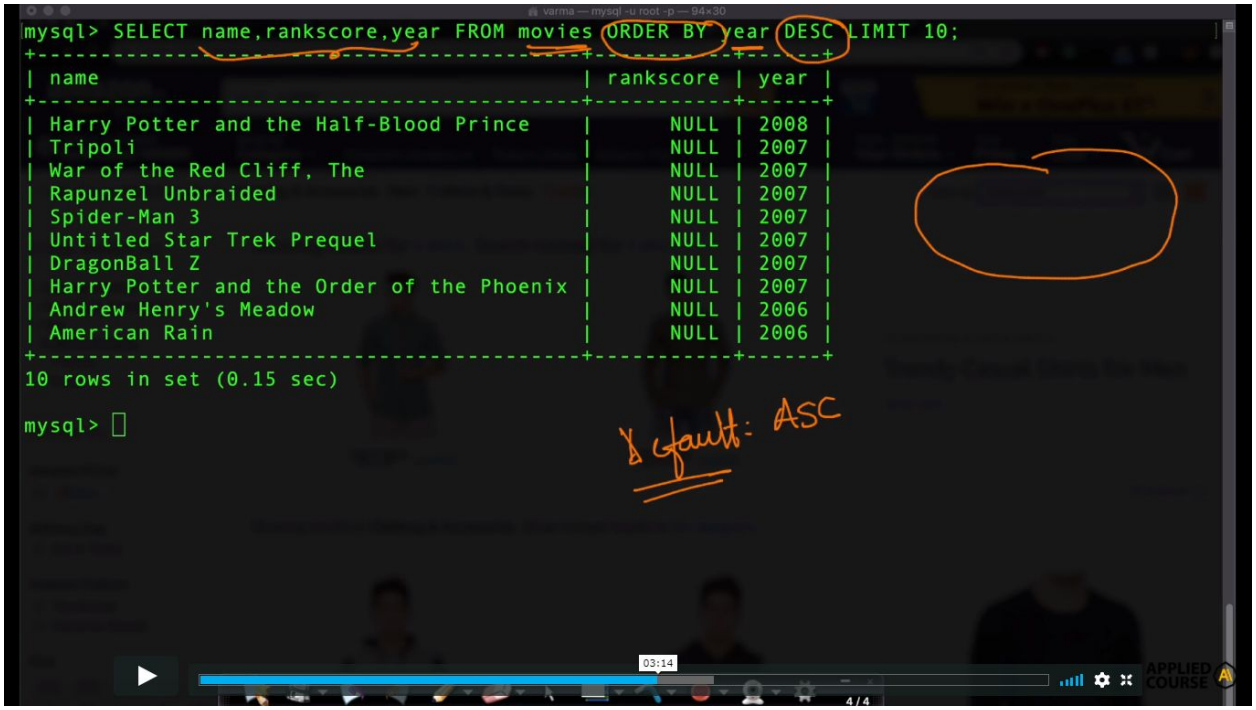
ORDER BY

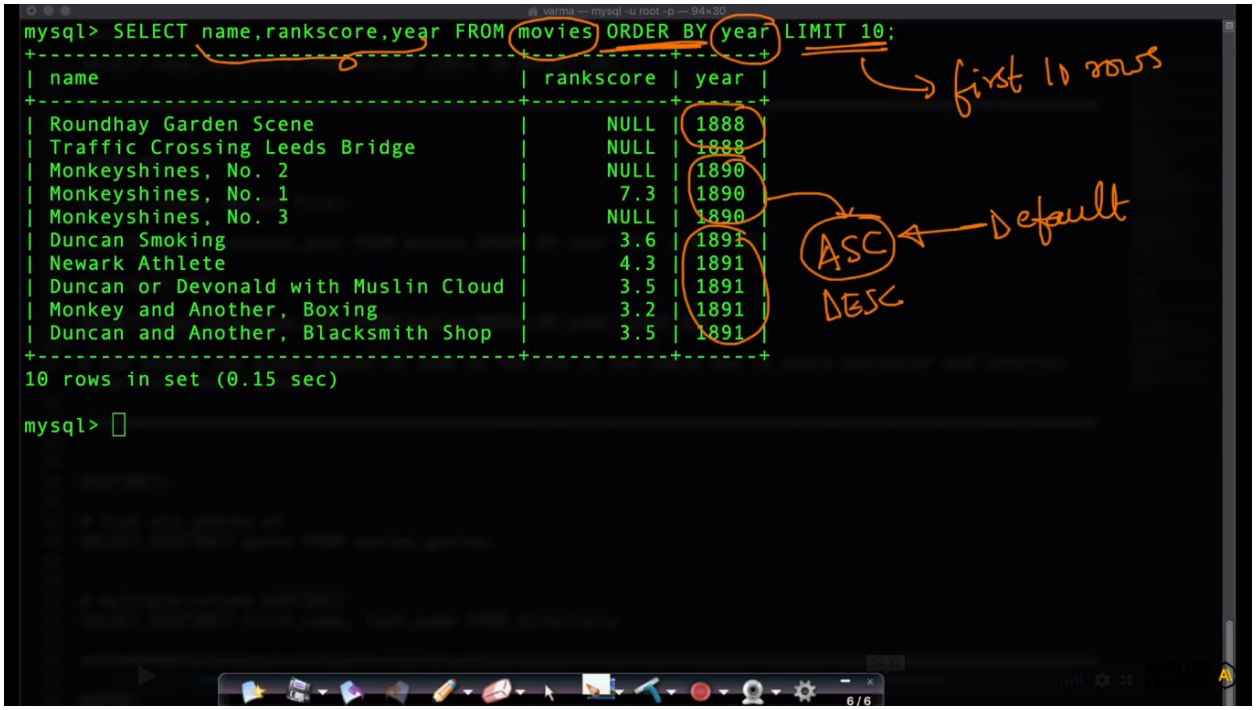

mysql> SELECT name,rankscore FROM movies LIMIT 20 OFFSET 20;
08:38

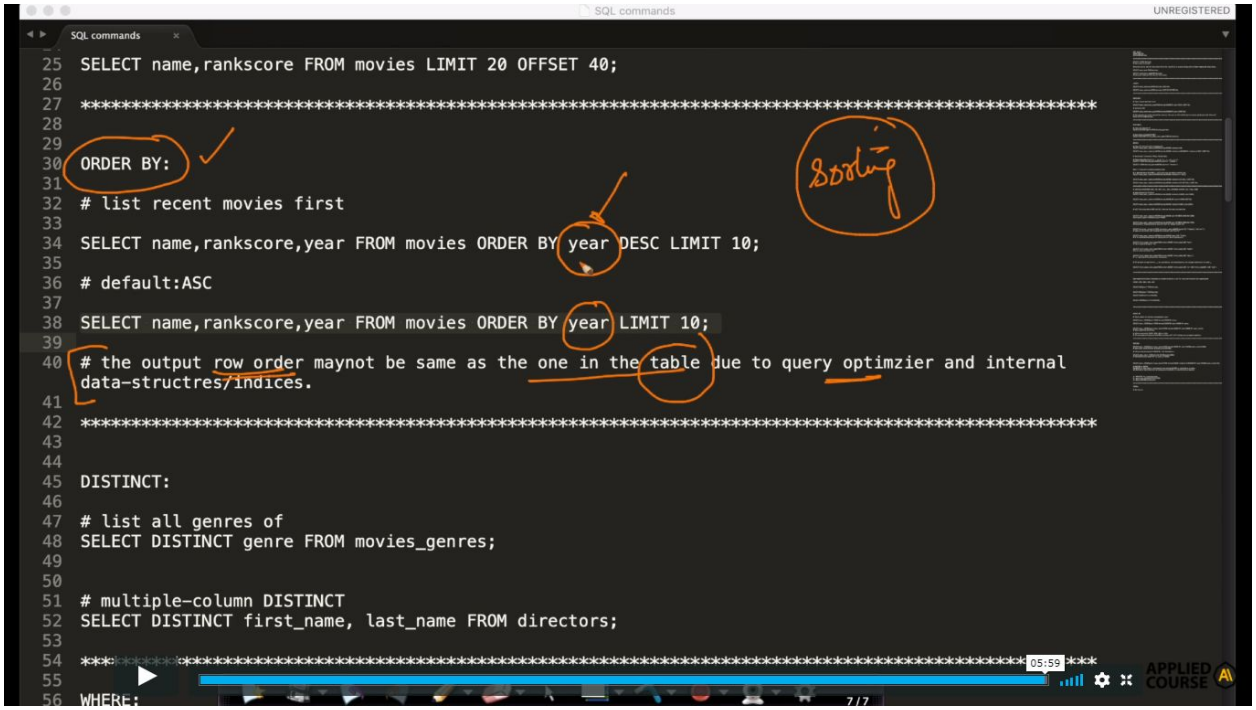

SQL commands
SELECT rankscore,name FROM movies;
#row order same as the one in the table
LIMIT:
SELECT name,rankscore FROM movies LIMIT 20;
SELECT name,rankscore FROM movies LIMIT 20 OFFSET 40;
ORDER BY:
list recent movies first
SELECT name,rankscore,year FROM movies ORDER BY year DESC LIMIT 20;
default:ASC
the output row order amynot be same as the one in the table due to query optimzier and internal data-structres/indices.
SELECT name,rankscore FROM movies ORDER BY year DESC LIMIT 20;
09:31
APPLIED COURSE




```
mysql> SELECT name,rankscore,year FROM movies ORDER BY year DESC LIMIT 10;
+----------------------------------------+-----------+------+
| name                                   | rankscore | year |
+----------------------------------------+-----------+------+
| Harry Potter and the Half-Blood Prince |      NULL | 2008 |
| Tripoli                                |      NULL | 2007 |
| War of the Red Cliff, The              |      NULL | 2007 |
| Rapunzel Unbraided                     |      NULL | 2007 |
| Spider-Man 3                           |      NULL | 2007 |
| Untitled Star Trek Prequel             |      NULL | 2007 |
| DragonBall Z                           |      NULL | 2007 |
| Harry Potter and the Order of the Phoenix |      NULL | 2007 |
| Andrew Henry's Meadow                  |      NULL | 2006 |
| American Rain                          |      NULL | 2006 |
+----------------------------------------+-----------+------+
10 rows in set (0.15 sec)

mysql>
```


varma — mysql -u root -p — 94×30
mysql> SELECT name,rankscore,year FROM movies ORDER BY year DESC LIMIT 10;
name	rankscore	year
Harry Potter and the Half-Blood Prince	NULL	2008
Tripoli	NULL	2007
War of the Red Cliff, The	NULL	2007
Rapunzel Unbraided	NULL	2007
Spider-Man 3	NULL	2007
Untitled Star Trek Prequel	NULL	2007
DragonBall Z	NULL	2007
Harry Potter and the Order of the Phoenix	NULL	2007
Andrew Henry's Meadow	NULL	2006
American Rain	NULL	2006
10 rows in set (0.15 sec)
mysql>
default: ASC
03:14
4 / 4
APPLIED COURSE





ORDER BY:

list recent movies first

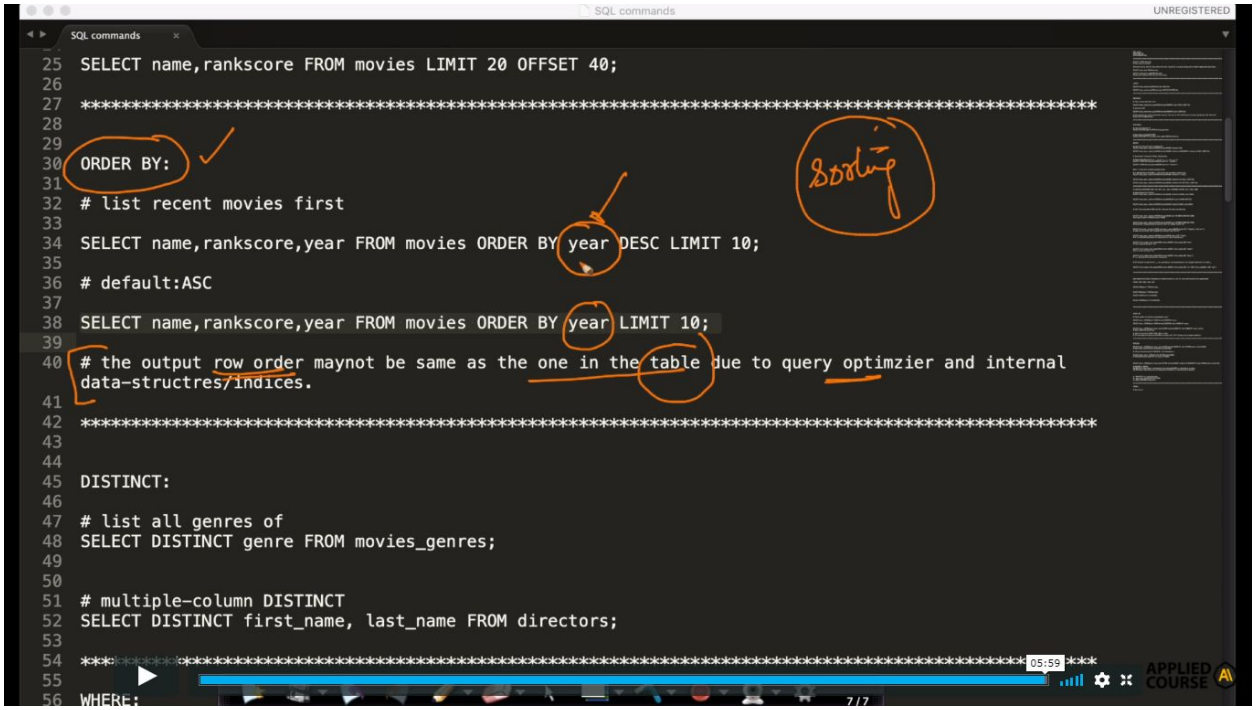
cc

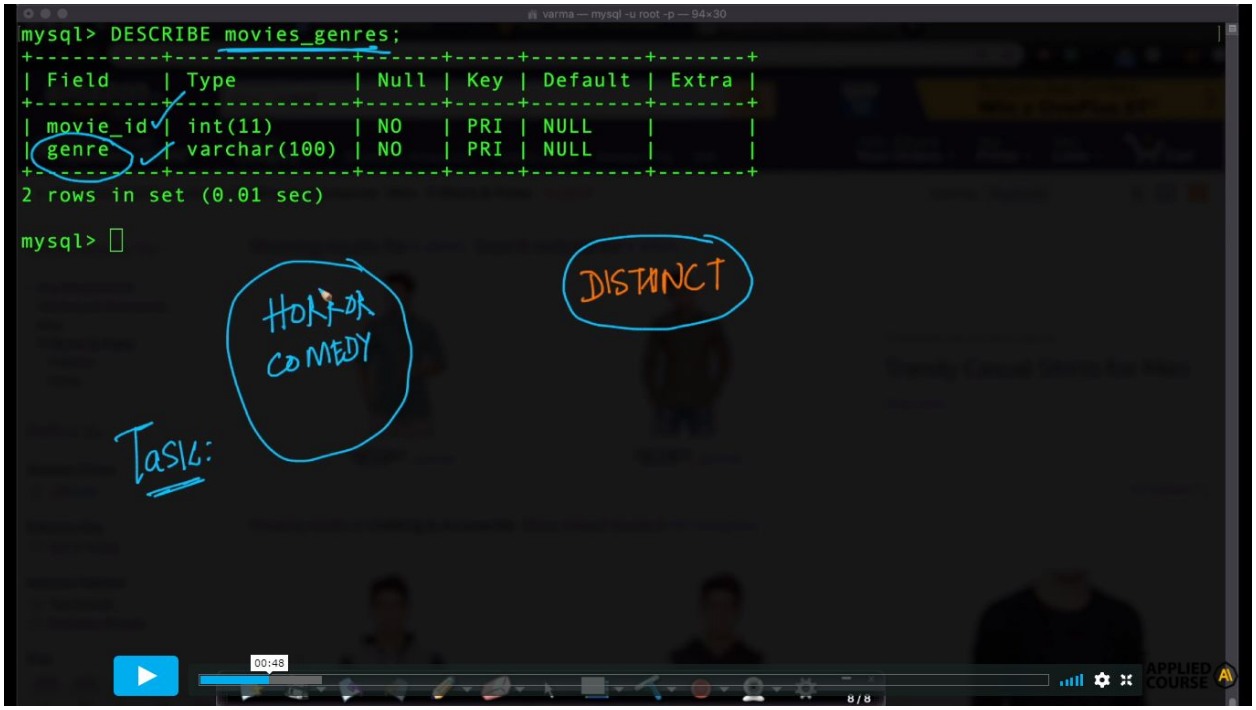
default:ASC

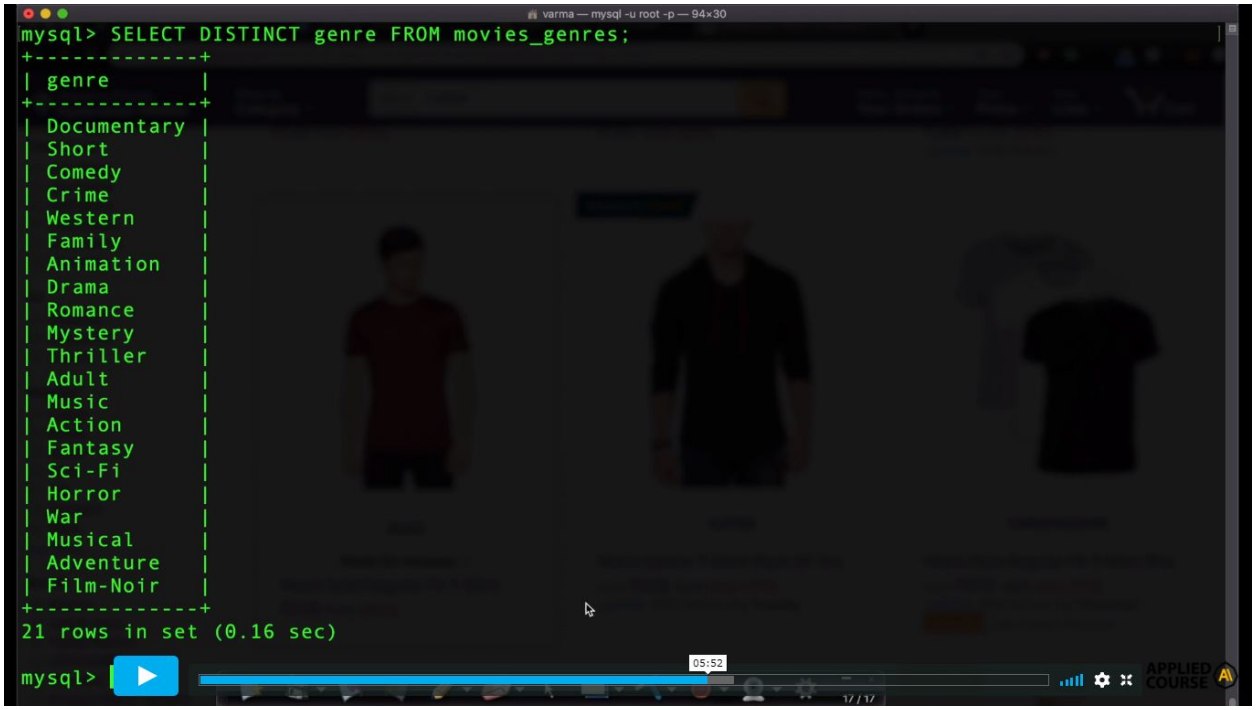
SELECT name,rankscore,year FROM movies ORDER BY year LIMIT 10;

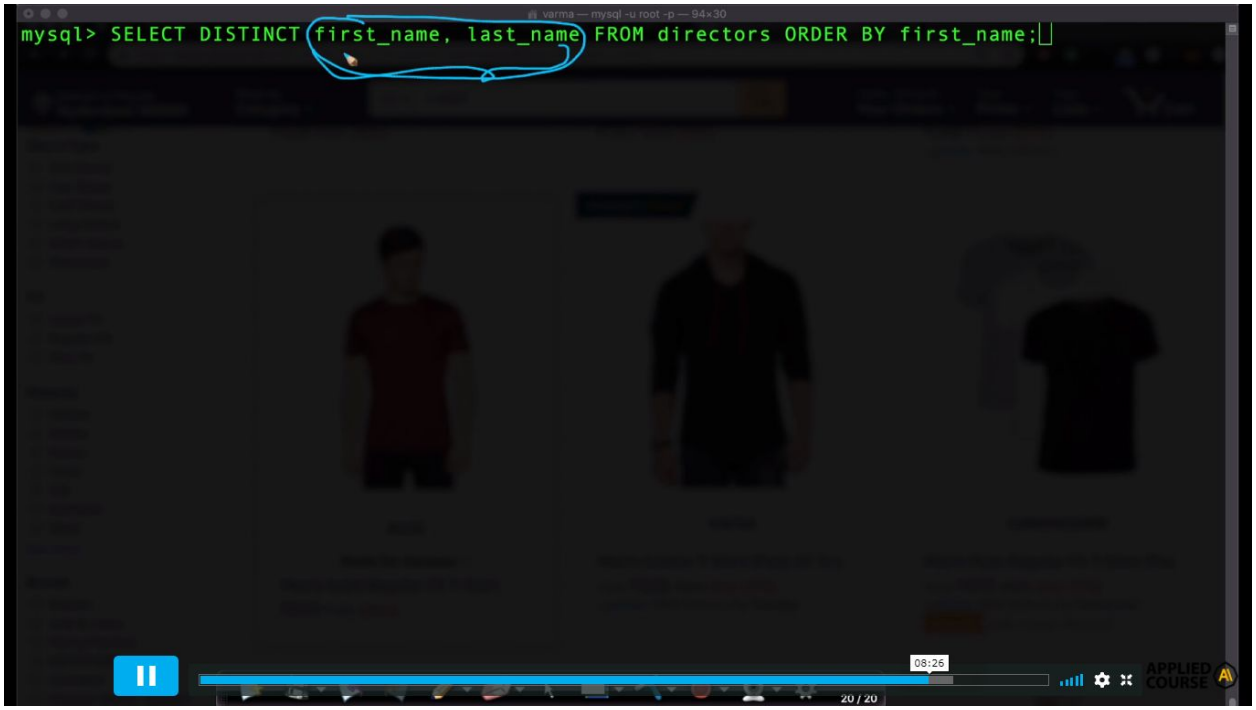
the output row order maynot be same as the one in the table due to query optimzier and internal data-structres/indices.

DISTINCT

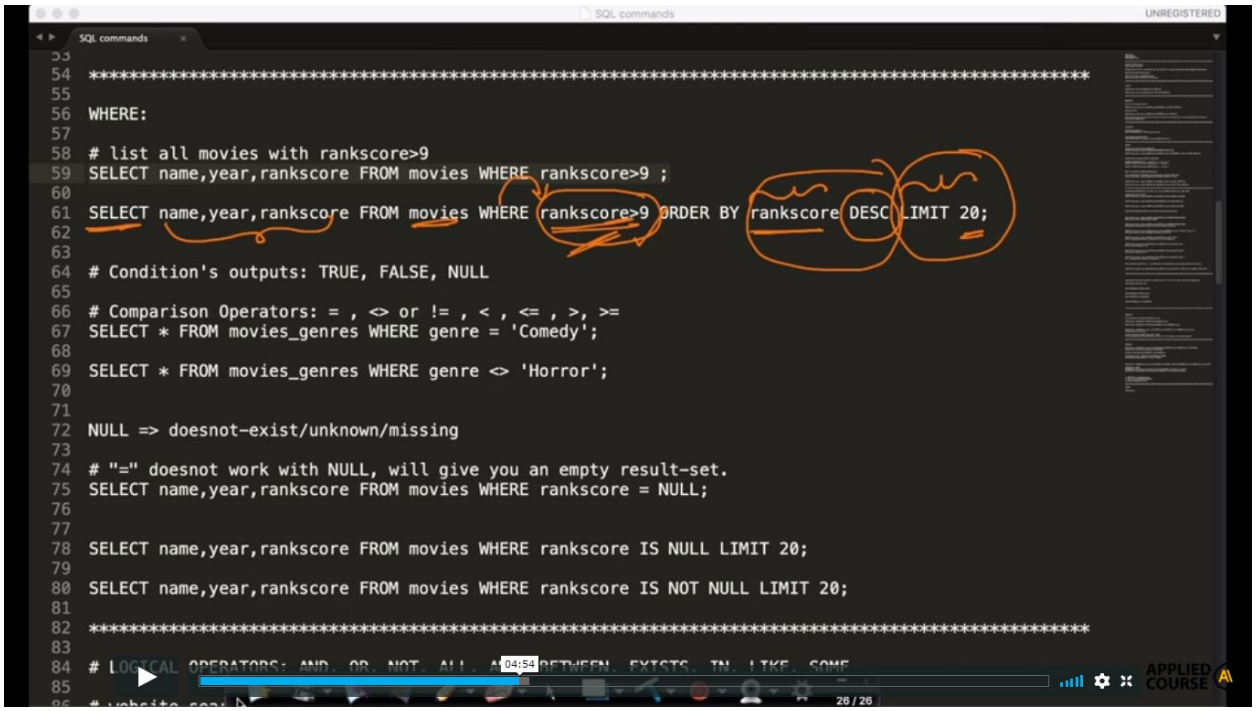

SQL commands
UNREGISTERED
SELECT name,rankscore FROM movies LIMIT 20 OFFSET 40;
ORDER BY:
Sorting
list recent movies first
SELECT name,rankscore,year FROM movies ORDER BY year DESC LIMIT 10;
default:ASC
SELECT name,rankscore,year FROM movies ORDER BY year LIMIT 10;
the output row order maynot be same as the one in the table due to query optimzier and internal data-structres/indices.
DISTINCT:
list all genres of
SELECT DISTINCT genre FROM movies_genres;
multiple-column DISTINCT
SELECT DISTINCT first_name, last_name FROM directors;
WHERE:
05:59
APPLIED COURSE

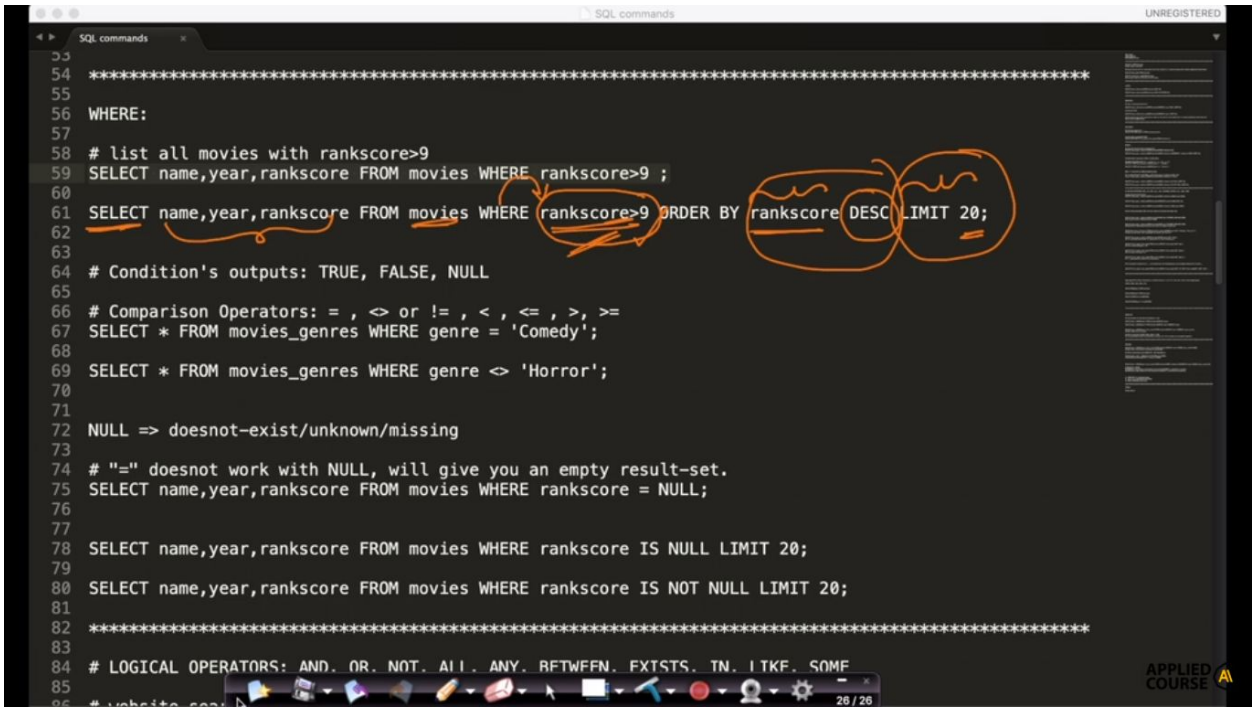

varma — mysql -u root -p — 94×30
mysql> DESCRIBE movies_genres;
+----------+--------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| movie_id | int(11) | NO | PRI | NULL | |
| genre | varchar(100) | NO | PRI | NULL | |
+----------+--------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
mysql>
HORROR COMEDY
DISTINCT
TASK:
00:48
APPLIED COURSE

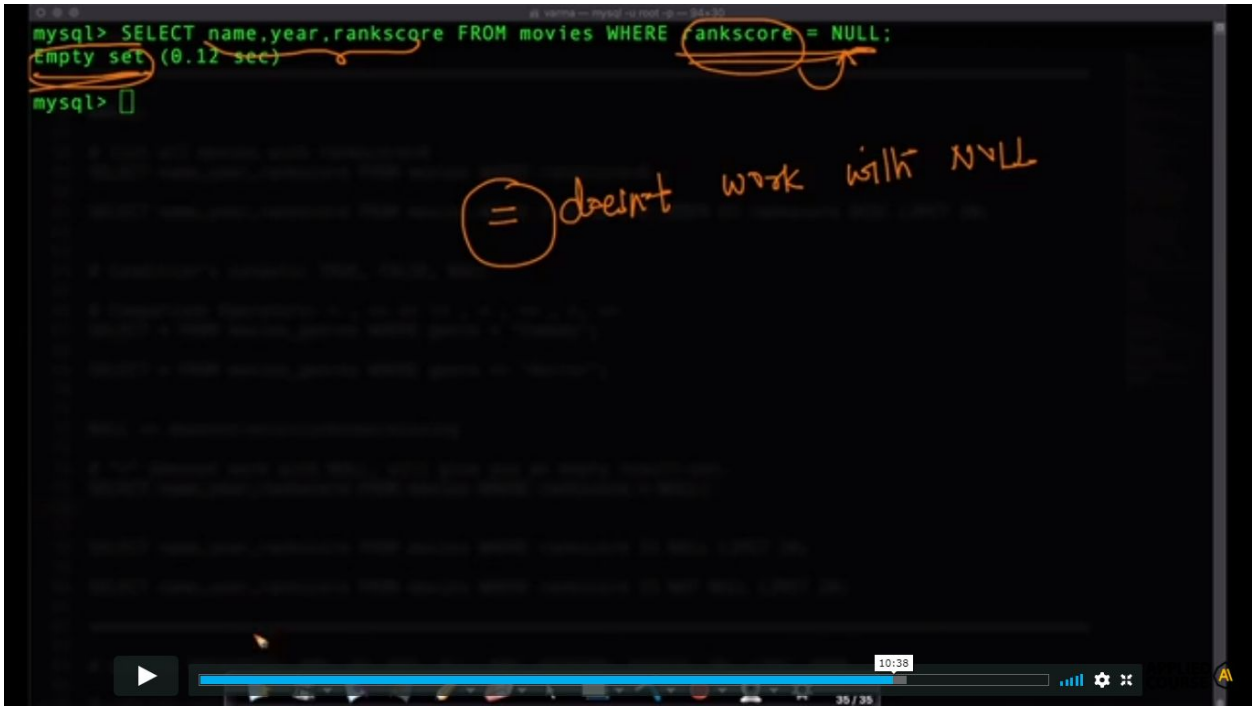

varma — mysql -u root -p — 94×30
mysql> SELECT DISTINCT genre FROM movies_genres;
+-------------+
| genre |
+-------------+
| Documentary |
| Short |
| Comedy |
| Crime |
| Western |
| Family |
| Animation |
| Drama |
| Romance |
| Mystery |
| Thriller |
| Adult |
| Music |
| Action |
| Fantasy |
| Sci-Fi |
| Horror |
| War |
| Musical |
| Adventure |
| Film-Noir |
+-------------+
21 rows in set (0.16 sec)
mysql>
05:52
17 / 17
APPLIED COURSE

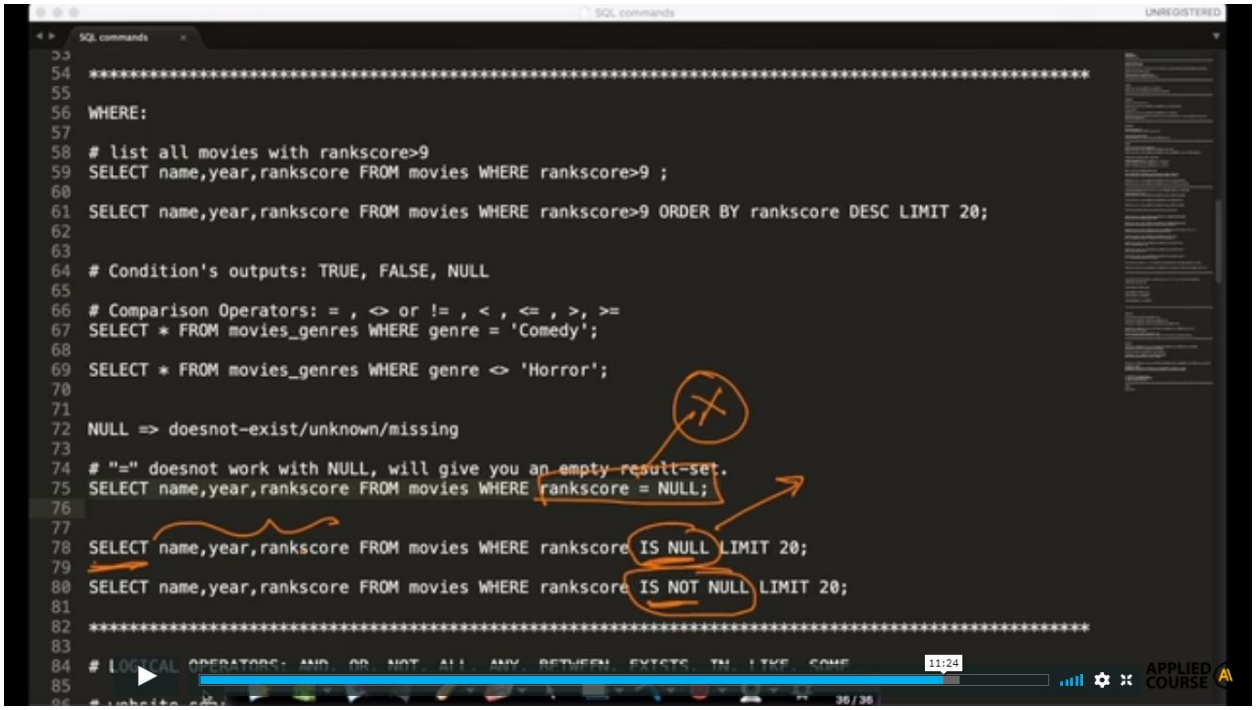

varma — mysql -u root -p — 94×30
mysql> SELECT DISTINCT first_name, last_name FROM directors ORDER BY first_name;
08:26
20 / 20
APPLIED COURSE

WHERE, Comparison operators, NULL










```
# list all movies with rankscore>9
SELECT name,year,rankscore FROM movies WHERE rankscore>9 ;

SELECT name,year,rankscore FROM movies WHERE rankscore>9 ORDER BY rankscore DESC LIMIT 20;
```

Condition's outputs: TRUE, FALSE, NULL

Comparison Operators: = , <> or != , < , <= , >, >=
SELECT * FROM movies_genres WHERE genre = 'Comedy';

SELECT * FROM movies_genres WHERE genre <> 'Horror';

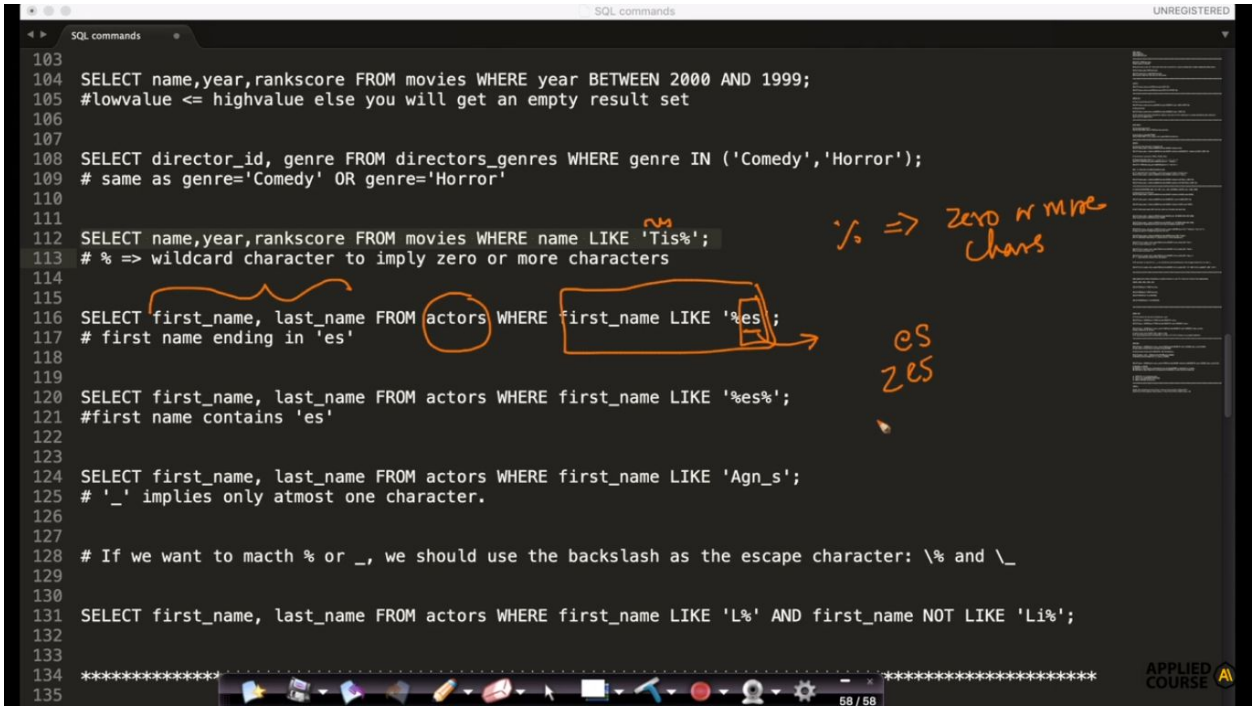
NULL => doesnot-exist/unknown/missing

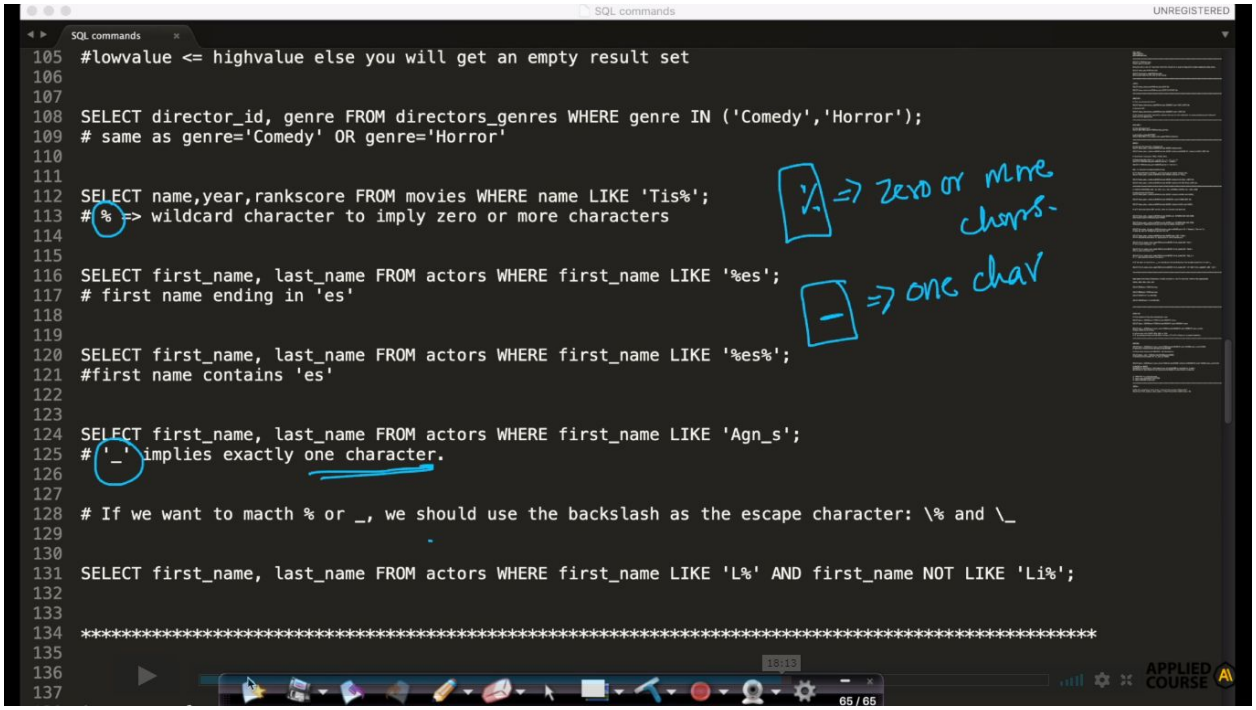
"=" doesnot work with NULL, will give you an empty result-set.
SELECT name,year,rankscore FROM movies WHERE rankscore = NULL;

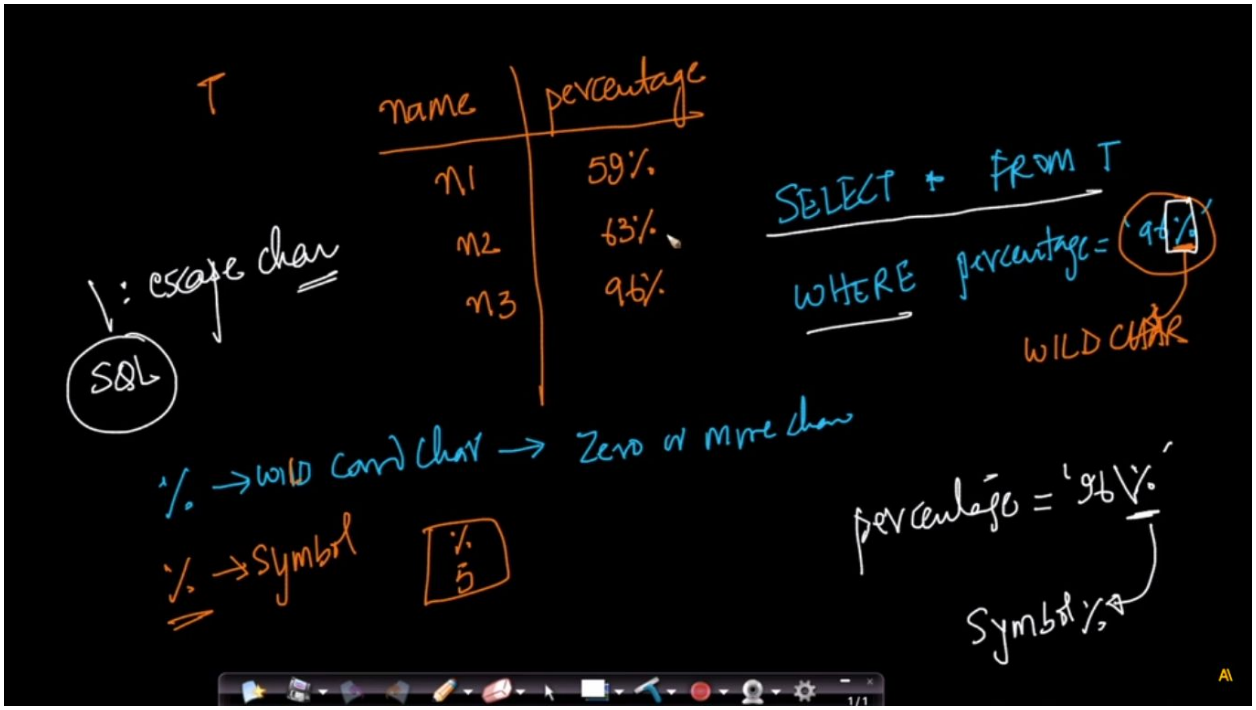
SELECT name,year,rankscore FROM movies WHERE rankscore IS NULL LIMIT 20;

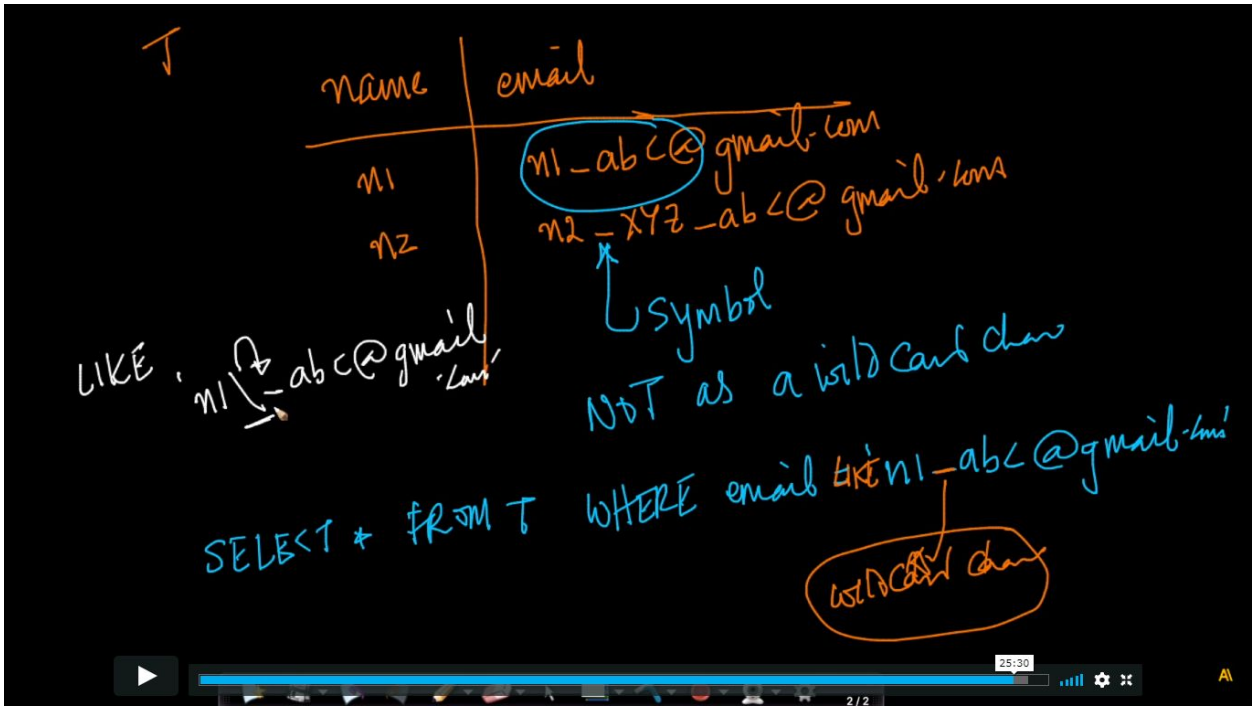
SELECT name,year,rankscore FROM movies WHERE rankscore IS NOT NULL LIMIT 20;

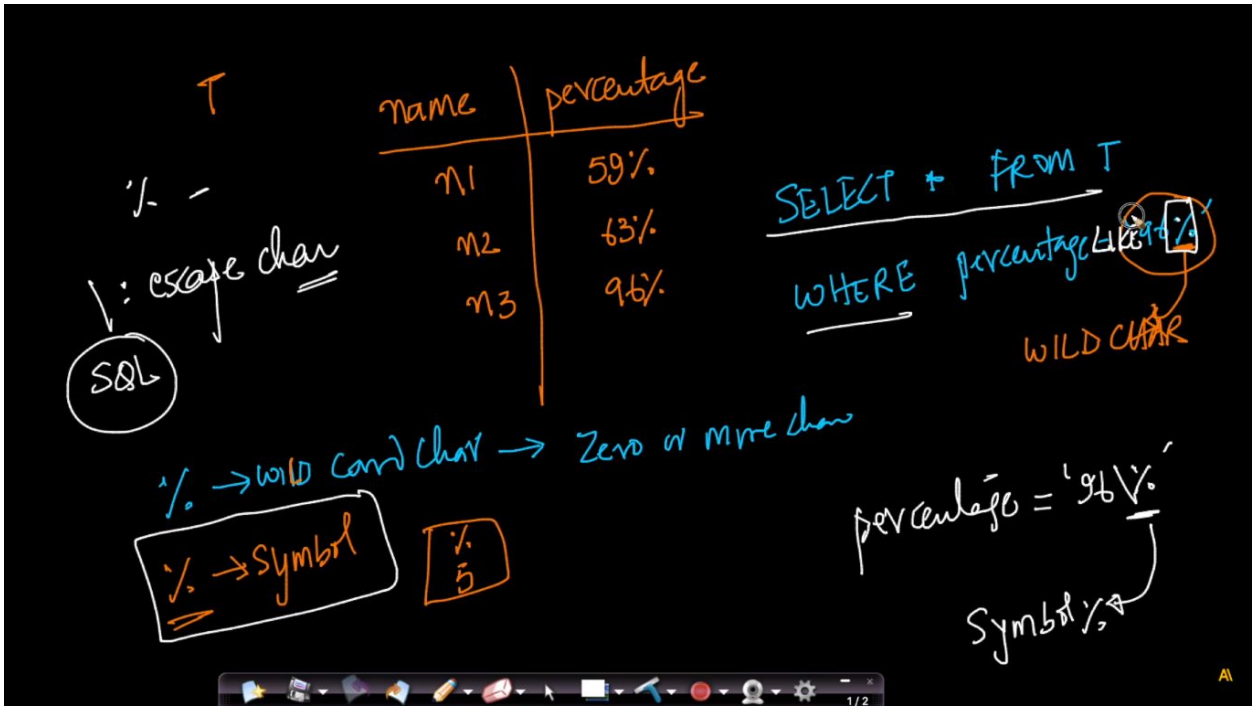
Logical Operators


SQL commands
UNREGISTERED
SQL commands
103
104 SELECT name,year,rankscore FROM movies WHERE year BETWEEN 2000 AND 1999;
105 #lowvalue <= highvalue else you will get an empty result set
106
107
108 SELECT director_id, genre FROM directors_genres WHERE genre IN ('Comedy','Horror');
109 # same as genre='Comedy' OR genre='Horror'
110
111
112 SELECT name,year,rankscore FROM movies WHERE name LIKE 'Tis%';
113 # % => wildcard character to imply zero or more characters
114
115
116 SELECT first_name, last_name FROM actors WHERE first_name LIKE '%es';
117 # first name ending in 'es'
118
119
120 SELECT first_name, last_name FROM actors WHERE first_name LIKE '%es%';
121 #first name contains 'es'
122
123
124 SELECT first_name, last_name FROM actors WHERE first_name LIKE 'Agn_s';
125 # '_' implies only atmost one character.
126
127
128 # If we want to macth % or _, we should use the backslash as the escape character: \% and _
129
130
131 SELECT first_name, last_name FROM actors WHERE first_name LIKE 'L%' AND first_name NOT LIKE 'Li%';
132
133
134 **********
135
58 / 58
APPLIED COURSE


SQL commands
UNREGISTERED
SQL commands
105 #lowvalue <= highvalue else you will get an empty result set
106
107
108 SELECT director_id, genre FROM directors_genres WHERE genre IN ('Comedy','Horror');
109 # same as genre='Comedy' OR genre='Horror'
110
111
112 SELECT name,year,rankscore FROM movies WHERE name LIKE 'Tis%';
113 # % => wildcard character to imply zero or more characters
114
115
116 SELECT first_name, last_name FROM actors WHERE first_name LIKE '%es';
117 # first name ending in 'es'
118
119
120 SELECT first_name, last_name FROM actors WHERE first_name LIKE '%es%';
121 #first name contains 'es'
122
123
124 SELECT first_name, last_name FROM actors WHERE first_name LIKE 'Agn_s';
125 # '_' implies exactly one character.
126
127
128 # If we want to macth % or _, we should use the backslash as the escape character: \% and _
129
130
131 SELECT first_name, last_name FROM actors WHERE first_name LIKE 'L%' AND first_name NOT LIKE 'Li%';
132
133
134 **********
135
136
137
18:13
65 / 65
APPLIED COURSE


T
name | percentage
n1 | 59%
n2 | 63%
n3 | 96%
SELECT * FROM T
WHERE percentage = '9%'
WILD CHAR
\ : escape char
SQL
% → WILD card char → Zero or more char
% → Symbol
%
5
percentage = '9\%'
Symbol %





****

LOGICAL OPERATORS: AND, OR, NOT, ALL, ANY, BETWEEN, EXISTS, IN, LIKE, SOME

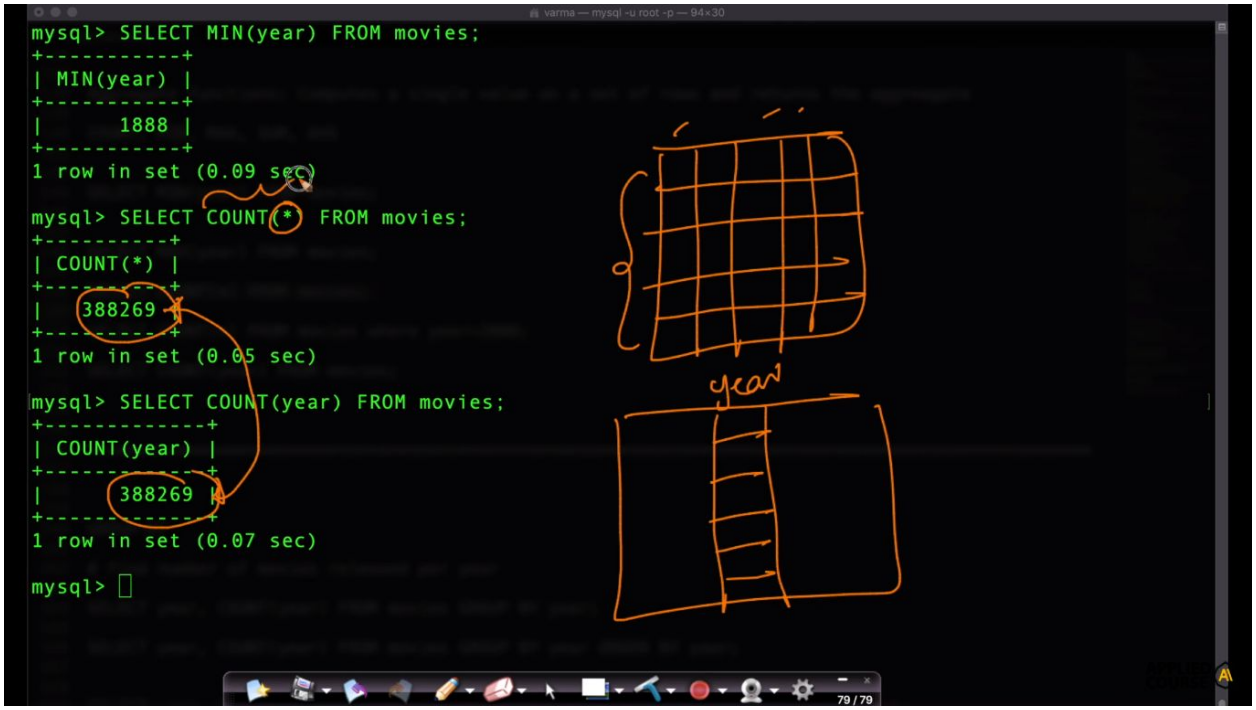
website search filters

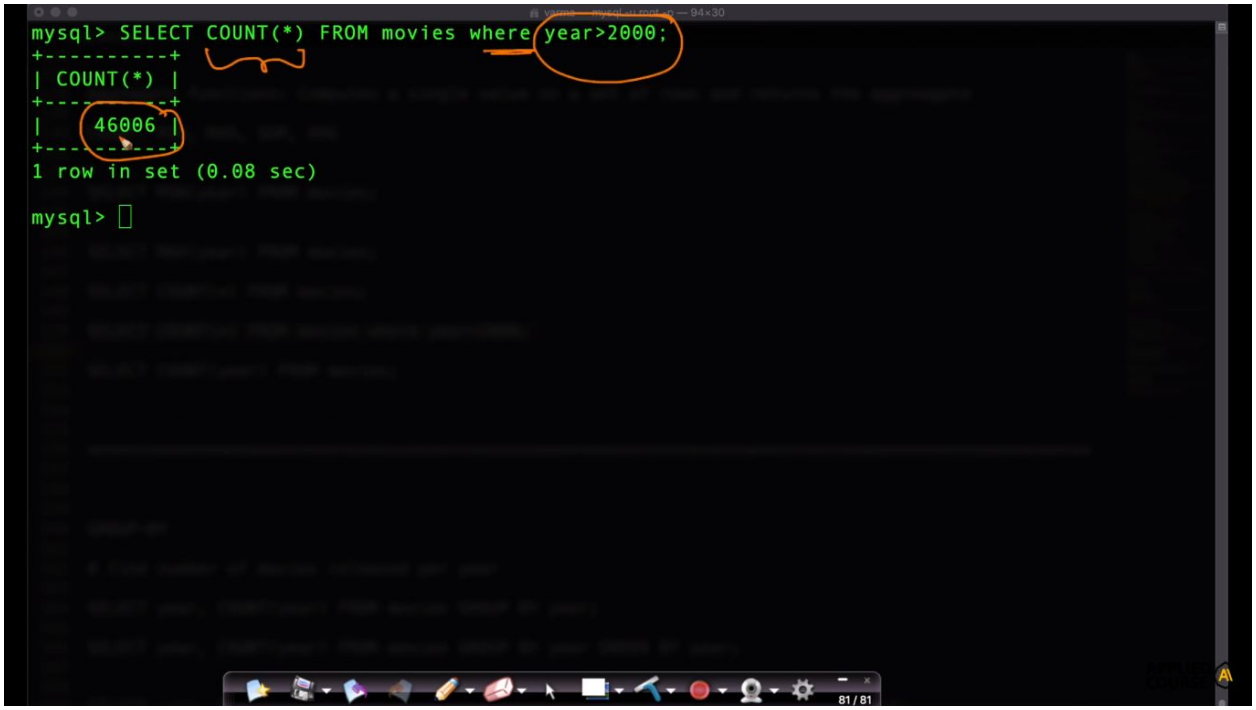
```
SELECT name,year,rankscore FROM movies WHERE rankscore>9 AND year>2000;

SELECT name,year,rankscore FROM movies WHERE NOT year<=2000 LIMIT 20;

SELECT name,year,rankscore FROM movies WHERE rankscore>9 OR year>2007;

# will discsuss about ANY and ALL when we discuss sub-queries


SELECT name,year,rankscore FROM movies WHERE year BETWEEN 1999 AND 2000;
#inclusive: year>=1999 and year<=2000

SELECT name,year,rankscore FROM movies WHERE year BETWEEN 2000 AND 1999;
#lowvalue <= highvalue else you will get an empty result set

SELECT director_id, genre FROM directors_genres WHERE genre IN ('Comedy','Horror');
# same as genre='Comedy' OR genre='Horror'

SELECT name,year,rankscore FROM movies WHERE name LIKE 'Tis%';
# % => wildcard character to imply zero or more characters

SELECT first_name, last_name FROM actors WHERE first_name LIKE '%es';
# first name ending in 'es'

SELECT first_name, last_name FROM actors WHERE first_name LIKE '%es%';
#first name contains 'es'

SELECT first_name, last_name FROM actors WHERE first_name LIKE 'Agn_s';
# '_' implies exactly one character.

# If we want to macth % or _, we should use the backslash as the escape character: \% and \_
```

SELECT first_name, last_name FROM actors WHERE first_name LIKE 'L%' AND first_name NOT LIKE 'Li%';

****

Aggregate Functions: COUNT, MIN, MAX, AVG, SUM





Aggregate functions: Computes a single value on a set of rows and returns the aggreagate

Sub Queries/Nested Queries/Inner Queries

COUNT, MIN, MAX, SUM, AVG

SELECT MIN(year) FROM movies;

SELECT MAX(year) FROM movies;

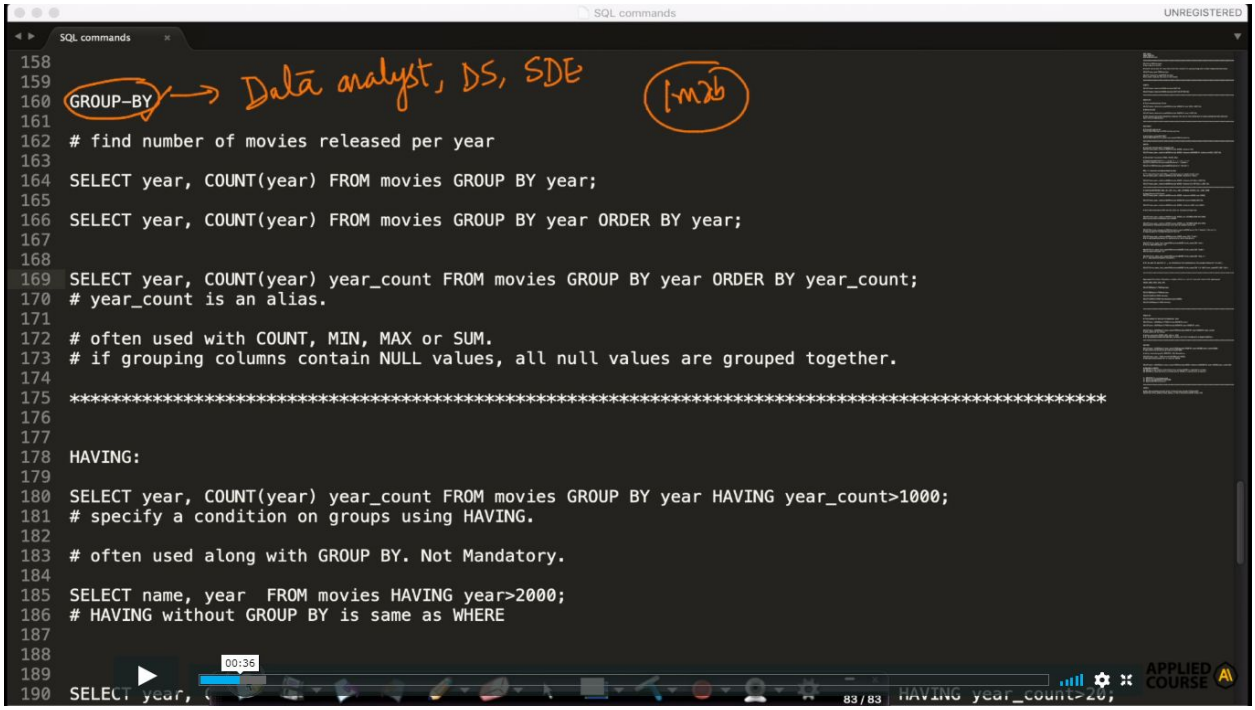
SELECT COUNT(*) FROM movies;

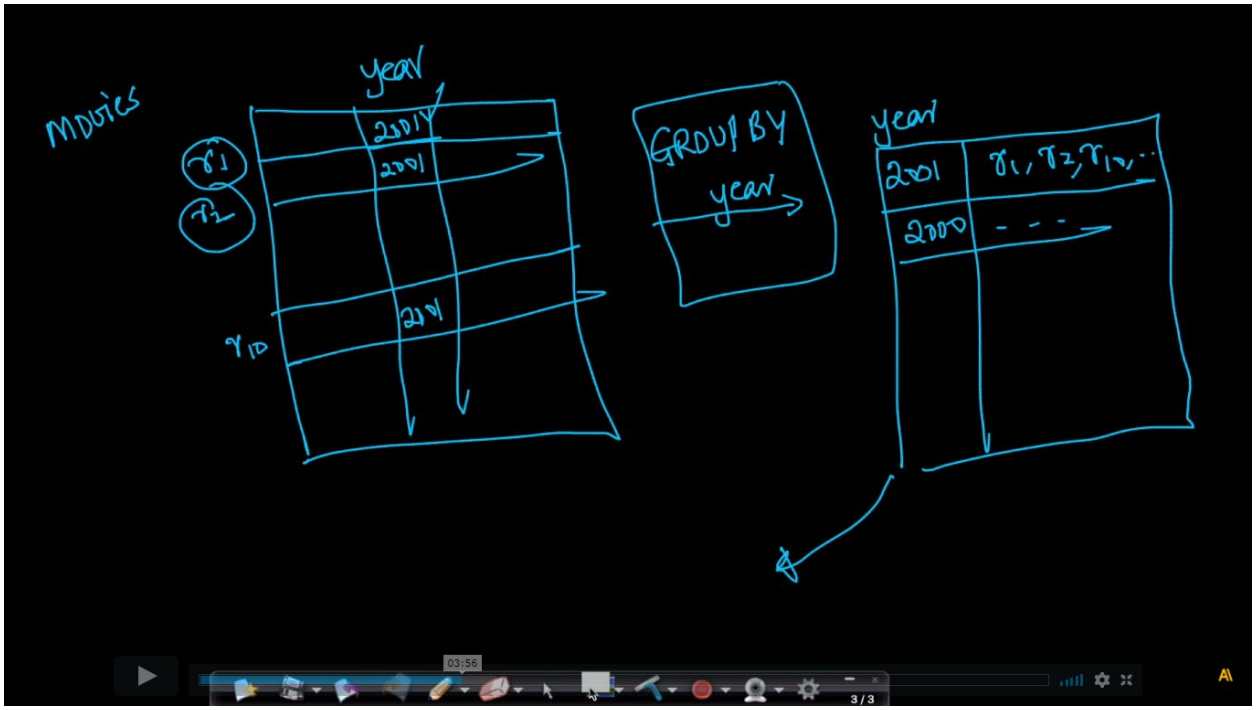
SELECT COUNT(*) FROM movies where year>2000;

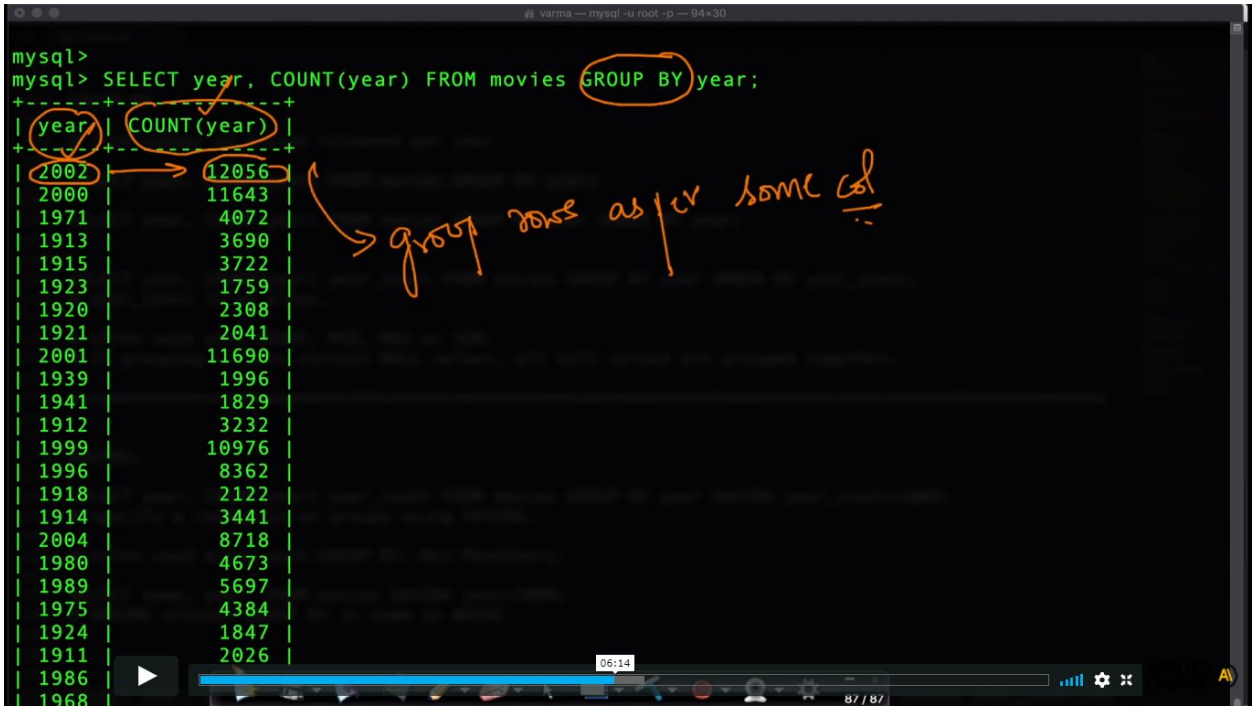
SELECT COUNT(year) FROM movies;

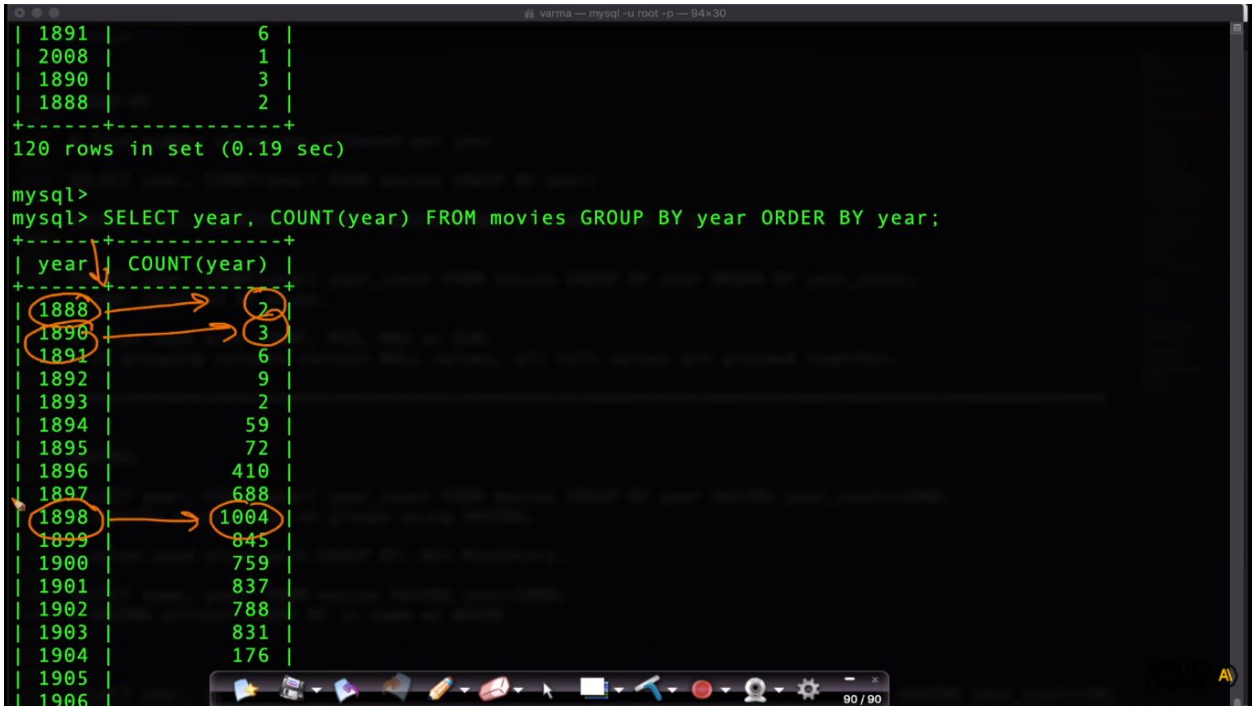
****

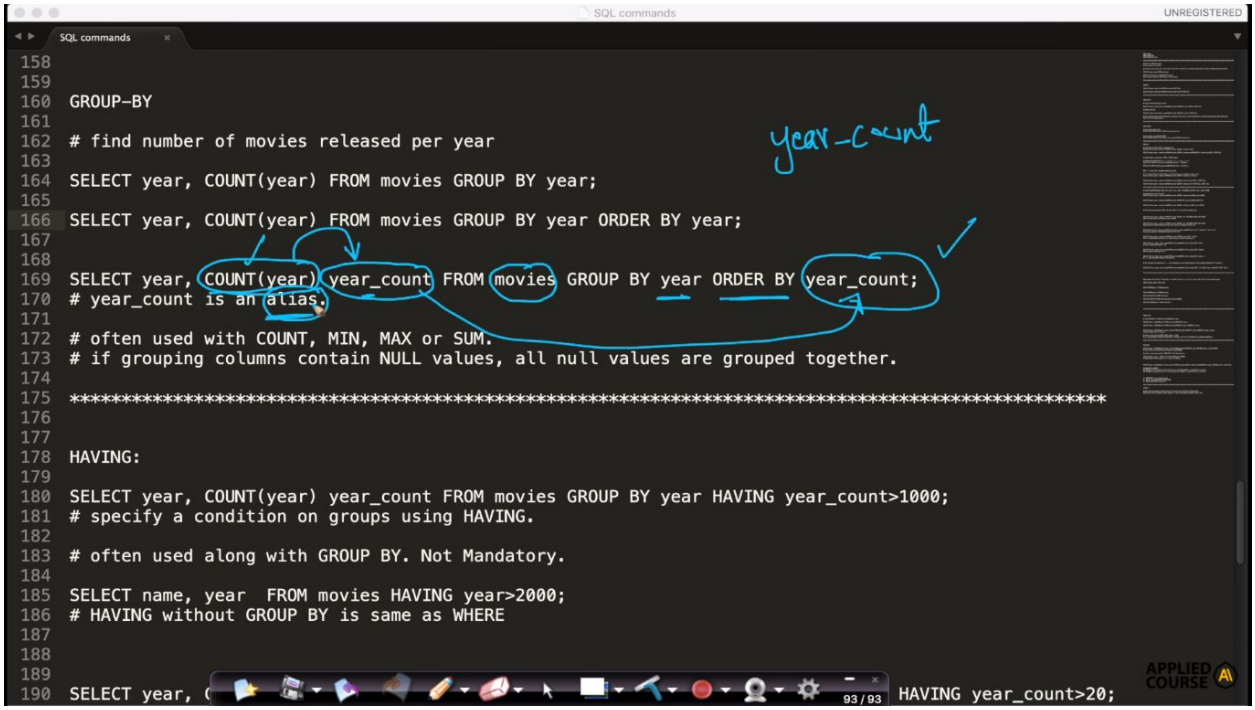
GROUP BY

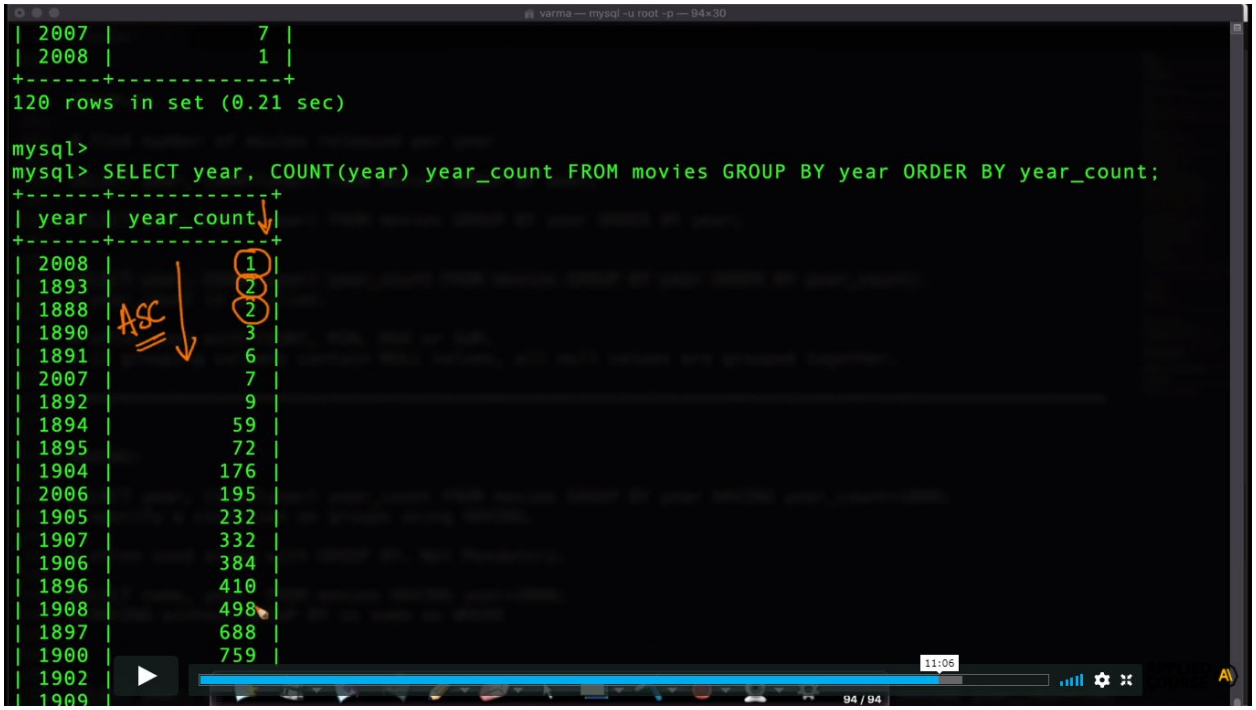

SQL commands
UNREGISTERED
GROUP-BY
Data analyst, DS, SDE
Imdb
find number of movies released per year
SELECT year, COUNT(year) FROM movies GROUP BY year;
SELECT year, COUNT(year) FROM movies GROUP BY year ORDER BY year;
SELECT year, COUNT(year) year_count FROM movies GROUP BY year ORDER BY year_count;
year_count is an alias.
often used with COUNT, MIN, MAX or SUM.
if grouping columns contain NULL values, all null values are grouped together.
HAVING:
SELECT year, COUNT(year) year_count FROM movies GROUP BY year HAVING year_count>1000;
specify a condition on groups using HAVING.
often used along with GROUP BY. Not Mandatory.
SELECT name, year FROM movies HAVING year>2000;
HAVING without GROUP BY is same as WHERE
00:36
83 / 83
APPLIED COURSE

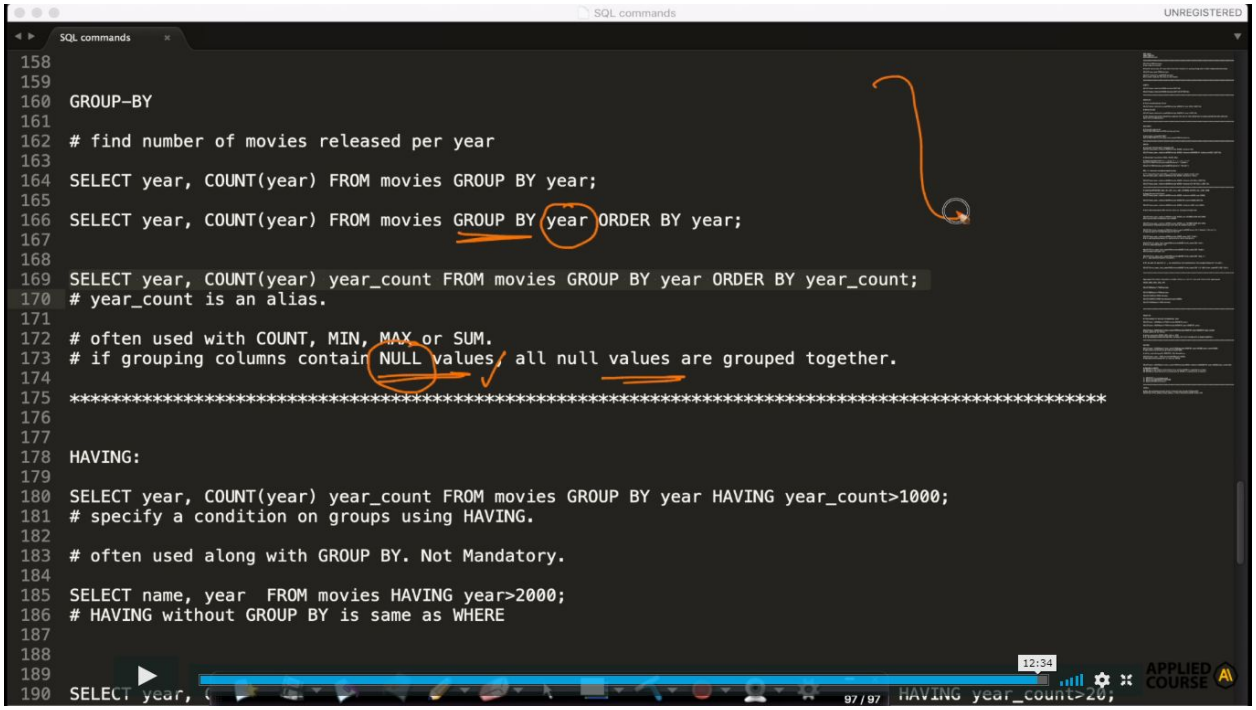

Movies
year
GROUP BY
year
03:56
3 / 3


varma — mysql -u root -p — 94×30
mysql>
mysql> SELECT year, COUNT(year) FROM movies GROUP BY year;
+------+-------------+
| year | COUNT(year) |
+------+-------------+
2002	12056
2000	11643
1971	4072
1913	3690
1915	3722
1923	1759
1920	2308
1921	2041
2001	11690
1939	1996
1941	1829
1912	3232
1999	10976
1996	8362
1918	2122
1914	3441
2004	8718
1980	4673
1989	5697
1975	4384
1924	1847
1911	2026
1986
1968
group rows as per some col
06:14
87 / 87


varma — mysql -u root -p — 94×30
1891	6
2008	1
1890	3
1888	2
+------+-------------+
120 rows in set (0.19 sec)
mysql>
mysql> SELECT year, COUNT(year) FROM movies GROUP BY year ORDER BY year;
+------+-------------+
| year | COUNT(year) |
+------+-------------+
1888	2
1890	3
1891	6
1892	9
1893	2
1894	59
1895	72
1896	410
1897	688
1898	1004
1899	845
1900	759
1901	837
1902	788
1903	831
1904	176
1905
1906
90 / 90


SQL commands
UNREGISTERED
SQL commands
158
159
160 GROUP-BY
161
162 # find number of movies released per year
163
164 SELECT year, COUNT(year) FROM movies GROUP BY year;
165
166 SELECT year, COUNT(year) FROM movies GROUP BY year ORDER BY year;
167
168
169 SELECT year, COUNT(year) year_count FROM movies GROUP BY year ORDER BY year_count;
170 # year_count is an alias.
171
172 # often used with COUNT, MIN, MAX or SUM.
173 # if grouping columns contain NULL values, all null values are grouped together.
174
175 **
176
177
178 HAVING:
179
180 SELECT year, COUNT(year) year_count FROM movies GROUP BY year HAVING year_count>1000;
181 # specify a condition on groups using HAVING.
182
183 # often used along with GROUP BY. Not Mandatory.
184
185 SELECT name, year FROM movies HAVING year>2000;
186 # HAVING without GROUP BY is same as WHERE
187
188
189
190 SELECT year, C
93 / 93
HAVING year_count>20;
year_count
APPLIED COURSE


varma — mysql -u root -p — 94×30
| 2007 | 7 |
| 2008 | 1 |
120 rows in set (0.21 sec)
mysql>
mysql> SELECT year, COUNT(year) year_count FROM movies GROUP BY year ORDER BY year_count;
year	year_count
2008	1
1893	2
1888	2
1890	3
1891	6
2007	7
1892	9
1894	59
1895	72
1904	176
2006	195
1905	232
1907	332
1906	384
1896	410
1908	498
1897	688
1900	759
1902
1909
Asc
11:06
94 / 94


SQL commands
UNREGISTERED
SQL commands
158
159
160 GROUP-BY
161
162 # find number of movies released per year
163
164 SELECT year, COUNT(year) FROM movies GROUP BY year;
165
166 SELECT year, COUNT(year) FROM movies GROUP BY year ORDER BY year;
167
168
169 SELECT year, COUNT(year) year_count FROM movies GROUP BY year ORDER BY year_count;
170 # year_count is an alias.
171
172 # often used with COUNT, MIN, MAX or SUM.
173 # if grouping columns contain NULL values, all null values are grouped together.
174
175 **********
176
177
178 HAVING:
179
180 SELECT year, COUNT(year) year_count FROM movies GROUP BY year HAVING year_count>1000;
181 # specify a condition on groups using HAVING.
182
183 # often used along with GROUP BY. Not Mandatory.
184
185 SELECT name, year FROM movies HAVING year>2000;
186 # HAVING without GROUP BY is same as WHERE
187
188
189
190 SELECT year, (
12:34
97 / 97
HAVING year_count>20;
APPLIED COURSE

****

GROUP-BY

find number of movies released per year

SELECT year, COUNT(year) FROM movies GROUP BY year;

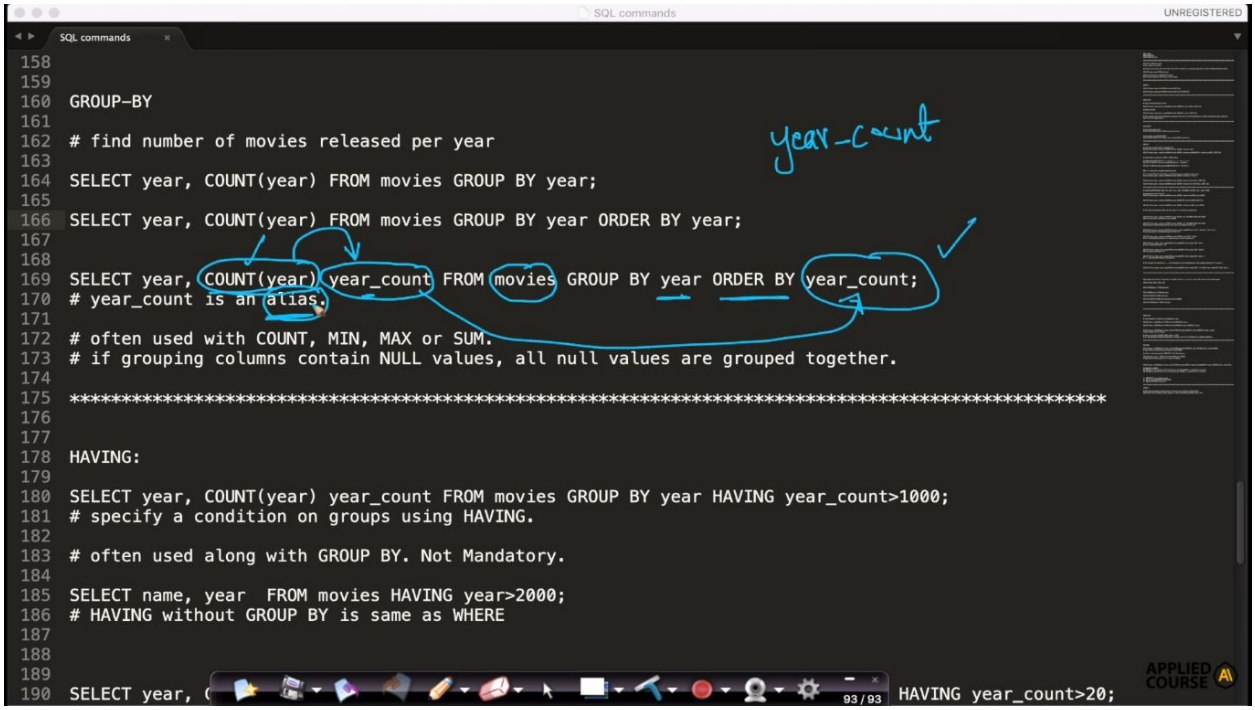
SELECT year, COUNT(year) FROM movies GROUP BY year ORDER BY year;

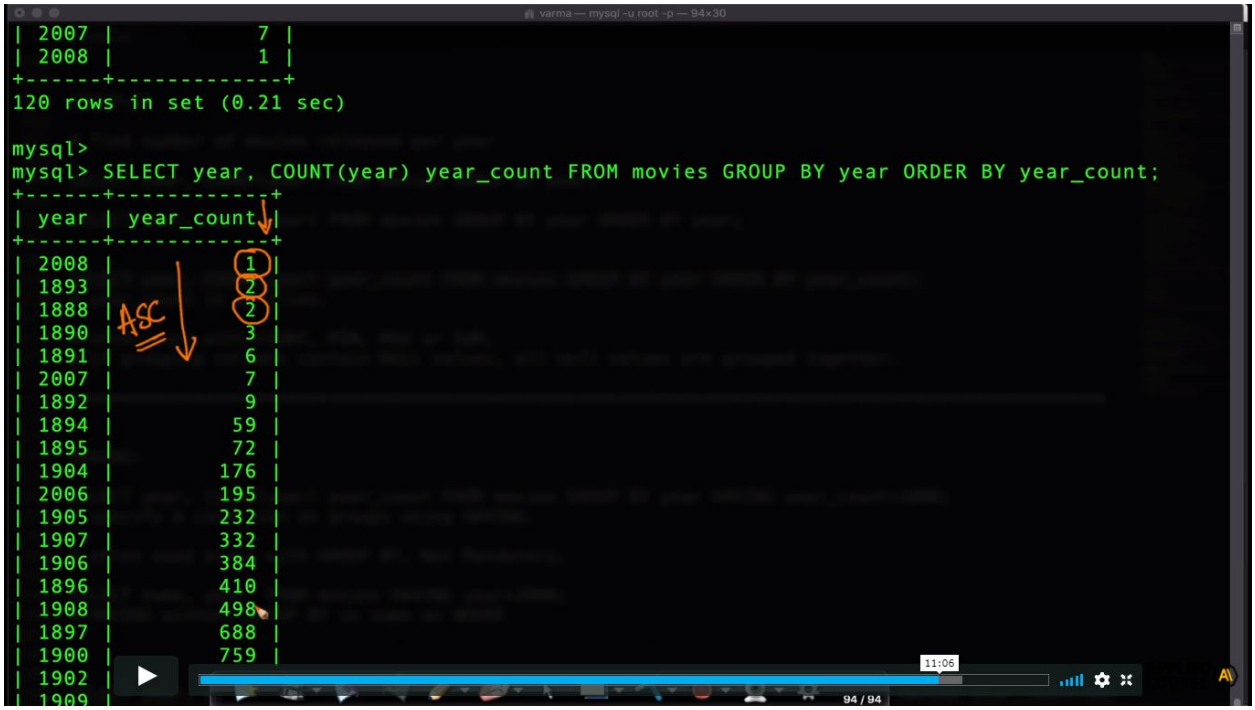
SELECT year, COUNT(year) year_count FROM movies GROUP BY year ORDER BY year_count;
year_count is an alias.

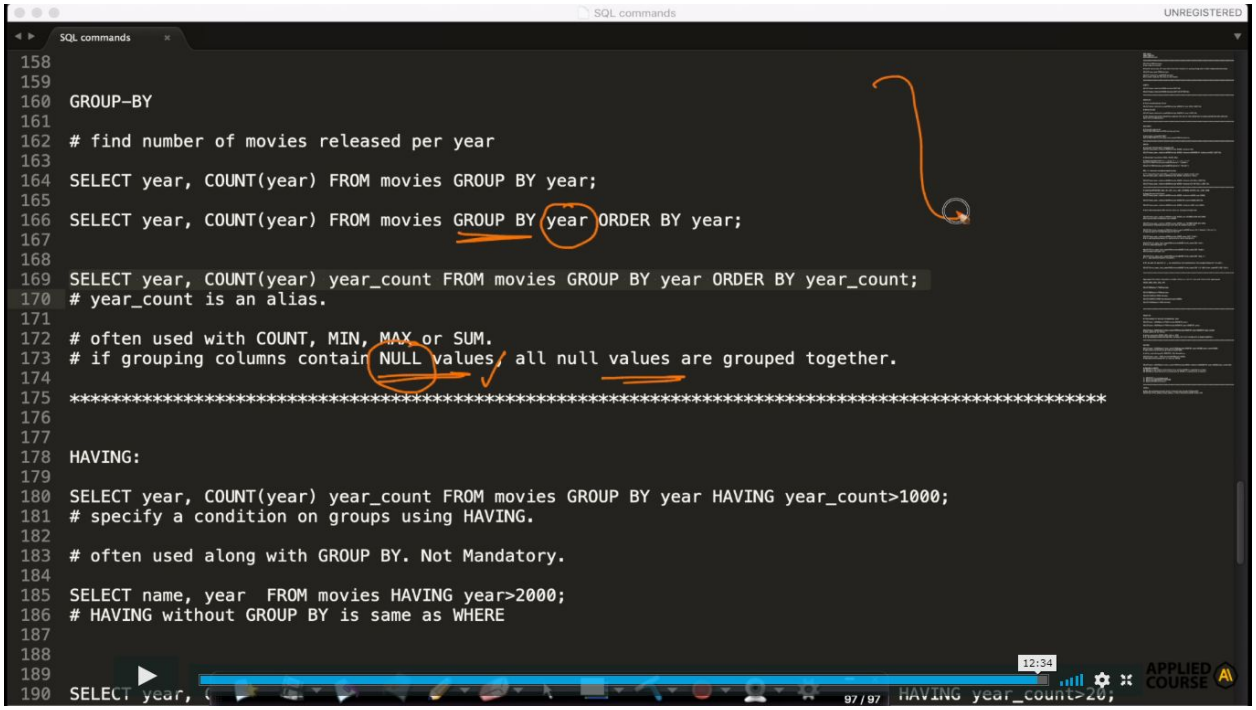
often used with COUNT, MIN, MAX or SUM.
if grouping columns contain NULL values, all null values are grouped together.

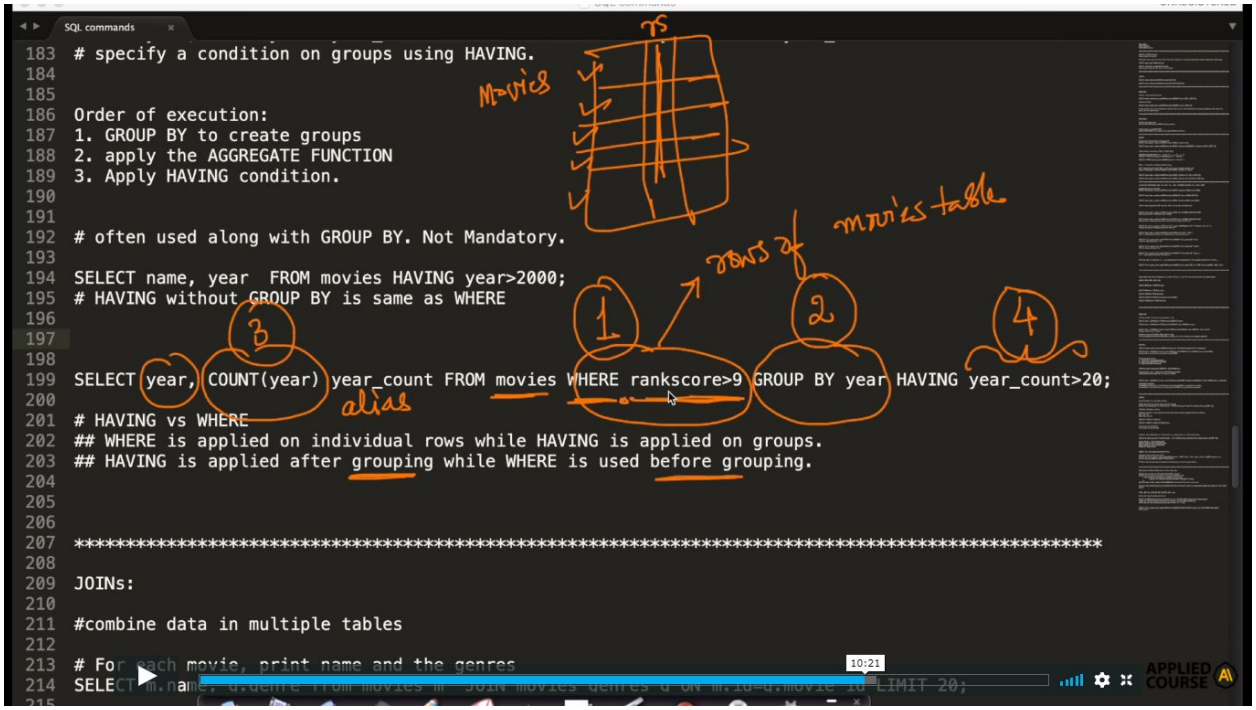
****

HAVING


SQL commands
UNREGISTERED
SQL commands
158
159
160 GROUP-BY
161
162 # find number of movies released per year
163
164 SELECT year, COUNT(year) FROM movies GROUP BY year;
165
166 SELECT year, COUNT(year) FROM movies GROUP BY year ORDER BY year;
167
168
169 SELECT year, COUNT(year) year_count FROM movies GROUP BY year ORDER BY year_count;
170 # year_count is an alias.
171
172 # often used with COUNT, MIN, MAX or SUM.
173 # if grouping columns contain NULL values, all null values are grouped together.
174
175 **
176
177
178 HAVING:
179
180 SELECT year, COUNT(year) year_count FROM movies GROUP BY year HAVING year_count>1000;
181 # specify a condition on groups using HAVING.
182
183 # often used along with GROUP BY. Not Mandatory.
184
185 SELECT name, year FROM movies HAVING year>2000;
186 # HAVING without GROUP BY is same as WHERE
187
188
189
190 SELECT year, C
93 / 93
HAVING year_count>20;
year_count
APPLIED COURSE


varma — mysql -u root -p — 94×30
| 2007 | 7 |
| 2008 | 1 |
120 rows in set (0.21 sec)
mysql>
mysql> SELECT year, COUNT(year) year_count FROM movies GROUP BY year ORDER BY year_count;
year	year_count
2008	1
1893	2
1888	2
1890	3
1891	6
2007	7
1892	9
1894	59
1895	72
1904	176
2006	195
1905	232
1907	332
1906	384
1896	410
1908	498
1897	688
1900	759
1902
1909
ASC
11:06
94 / 94





****

HAVING:

Print years which have >1000 movies in our DB [Data Scientist for Analysis]

SELECT year, COUNT(year) year_count FROM movies GROUP BY year HAVING year_count>1000;
specify a condition on groups using HAVING.

Order of execution:
1. GROUP BY to create groups
2. apply the AGGREGATE FUNCTION
3. Apply HAVING condition.

often used along with GROUP BY. Not Mandatory.

SELECT name, year FROM movies HAVING year>2000;
HAVING without GROUP BY is same as WHERE

SELECT year, COUNT(year) year_count FROM movies WHERE rankscore>9 GROUP BY year HAVING year_count>20;

HAVING vs WHERE
WHERE is applied on individual rows while HAVING is applied on groups.
HAVING is applied after grouping while WHERE is used before grouping.

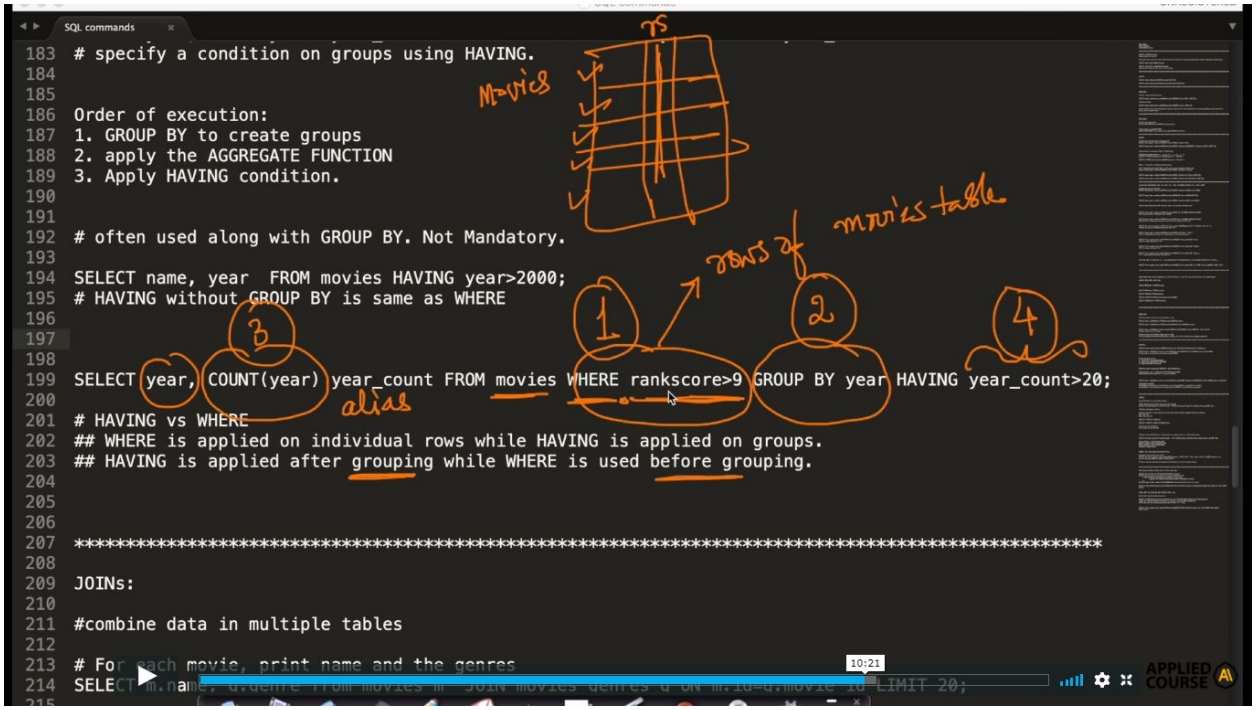
****

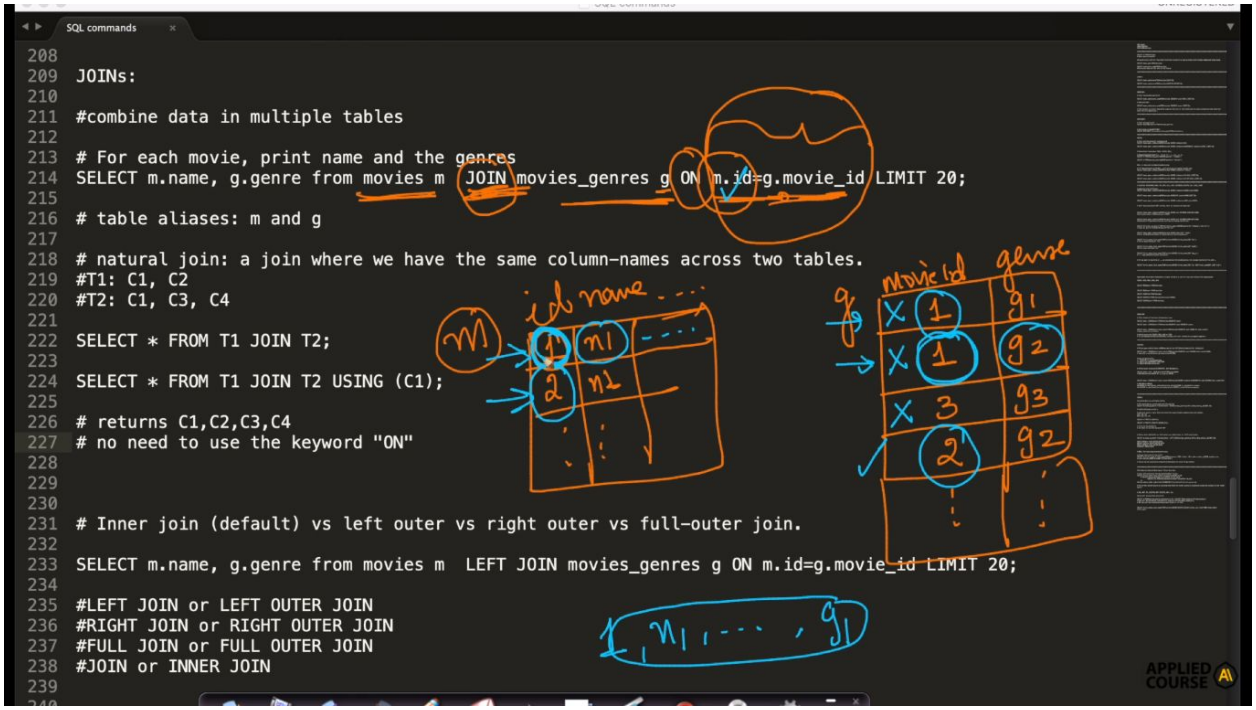
```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position}
```

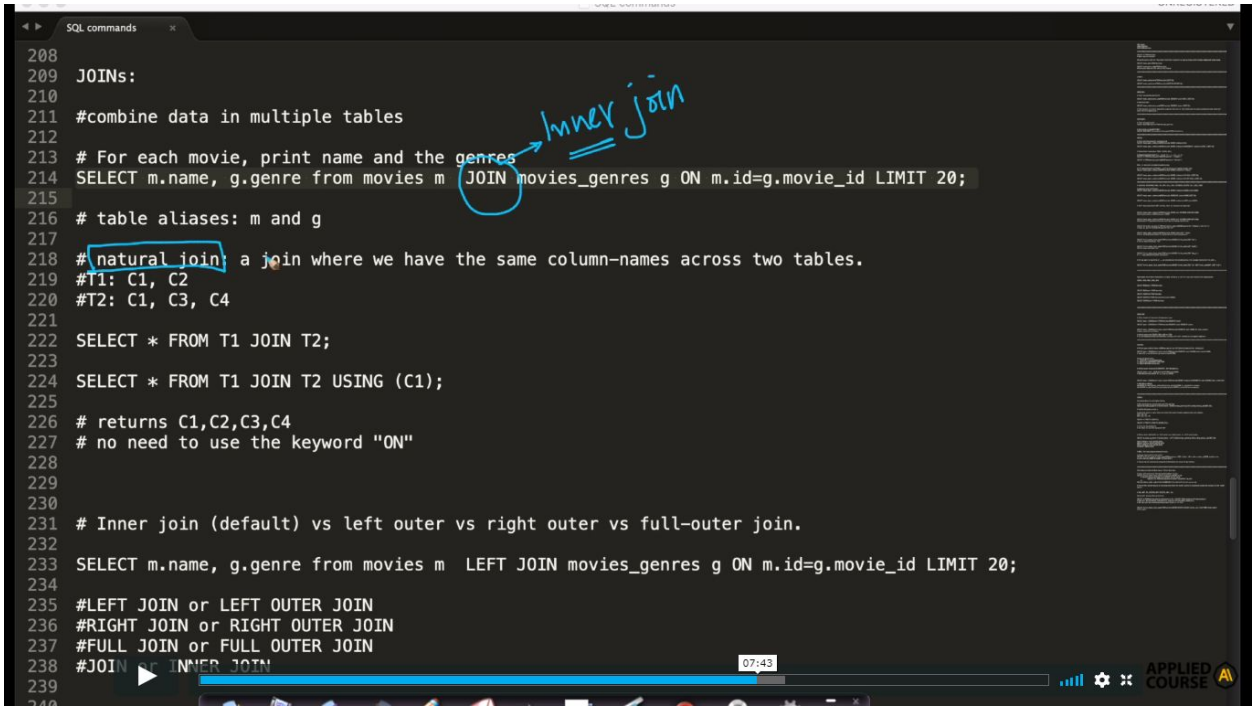
```
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR {UPDATE | SHARE} [OF tbl_name [, tbl_name] ...] [NOWAIT | SKIP LOCKED]


      | LOCK IN SHARE MODE]]
```

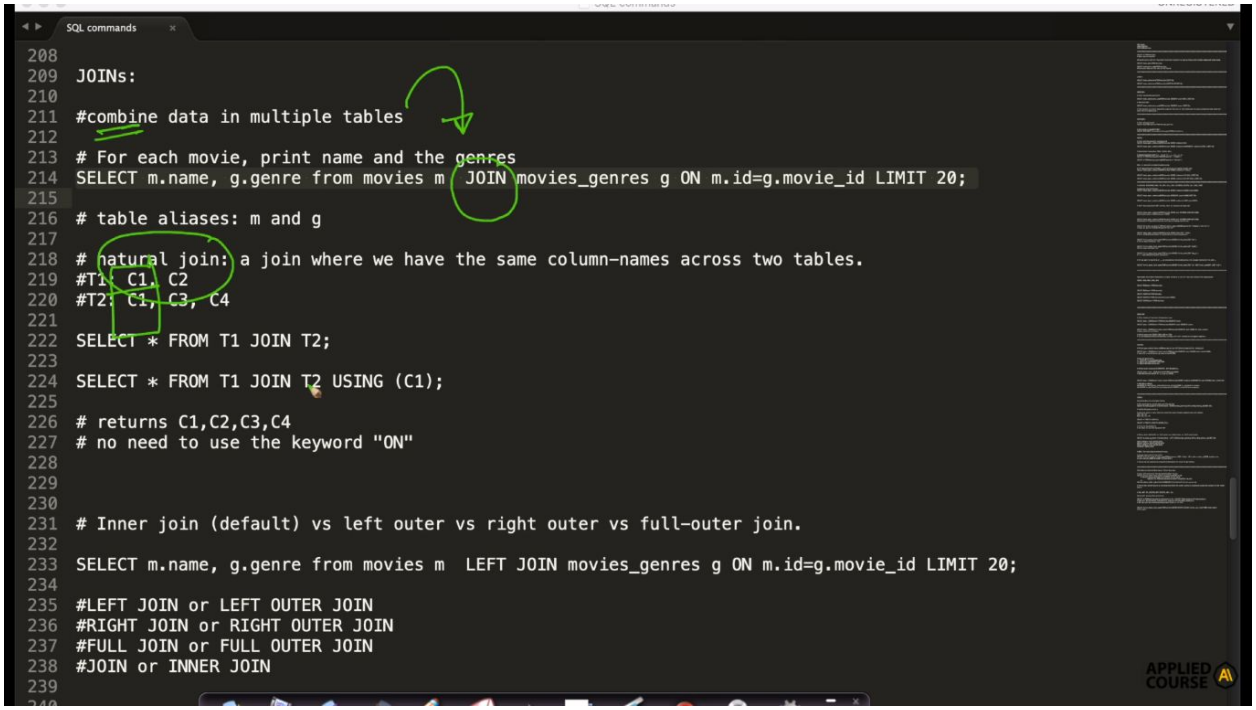
.....

Join and Natural Join


SQL commands
specify a condition on groups using HAVING.
Order of execution:
1. GROUP BY to create groups
2. apply the AGGREGATE FUNCTION
3. Apply HAVING condition.
often used along with GROUP BY. Not Mandatory.
SELECT name, year FROM movies HAVING year>2000;
HAVING without GROUP BY is same as WHERE
SELECT year, COUNT(year) year_count FROM movies WHERE rankscore>9 GROUP BY year HAVING year_count>20;
HAVING vs WHERE
WHERE is applied on individual rows while HAVING is applied on groups.
HAVING is applied after grouping while WHERE is used before grouping.
JOINs:
#combine data in multiple tables
For each movie, print name and the genres
Movies
alias
10:21
APPLIED COURSE


SQL commands
JOINs:
#combine data in multiple tables
For each movie, print name and the genres
SELECT m.name, g.genre from movies m JOIN movies_genres g ON m.id=g.movie_id LIMIT 20;
table aliases: m and g
natural join: a join where we have the same column-names across two tables.
#T1: C1, C2
#T2: C1, C3, C4
SELECT * FROM T1 JOIN T2;
SELECT * FROM T1 JOIN T2 USING (C1);
returns C1,C2,C3,C4
no need to use the keyword "ON"
Inner join (default) vs left outer vs right outer vs full-outer join.
SELECT m.name, g.genre from movies m LEFT JOIN movies_genres g ON m.id=g.movie_id LIMIT 20;
#LEFT JOIN or LEFT OUTER JOIN
#RIGHT JOIN or RIGHT OUTER JOIN
#FULL JOIN or FULL OUTER JOIN
#JOIN or INNER JOIN
movie id
genre
APPLIED COURSE





****

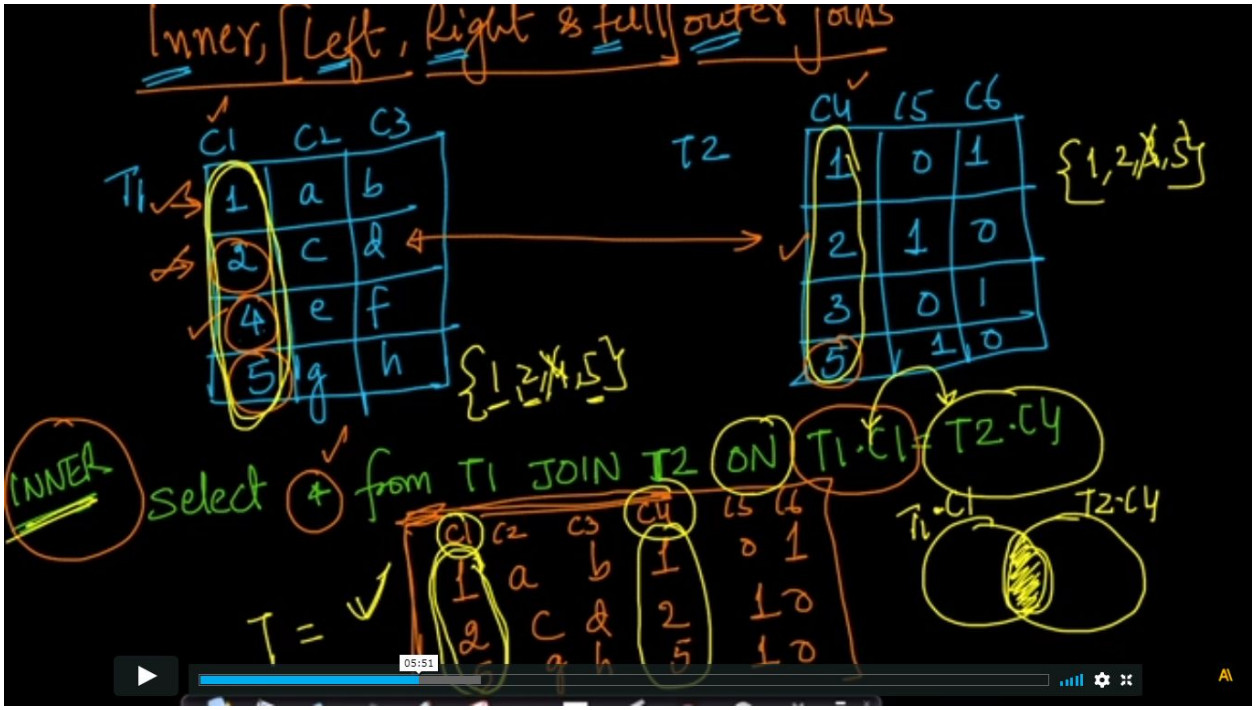
JOINs:

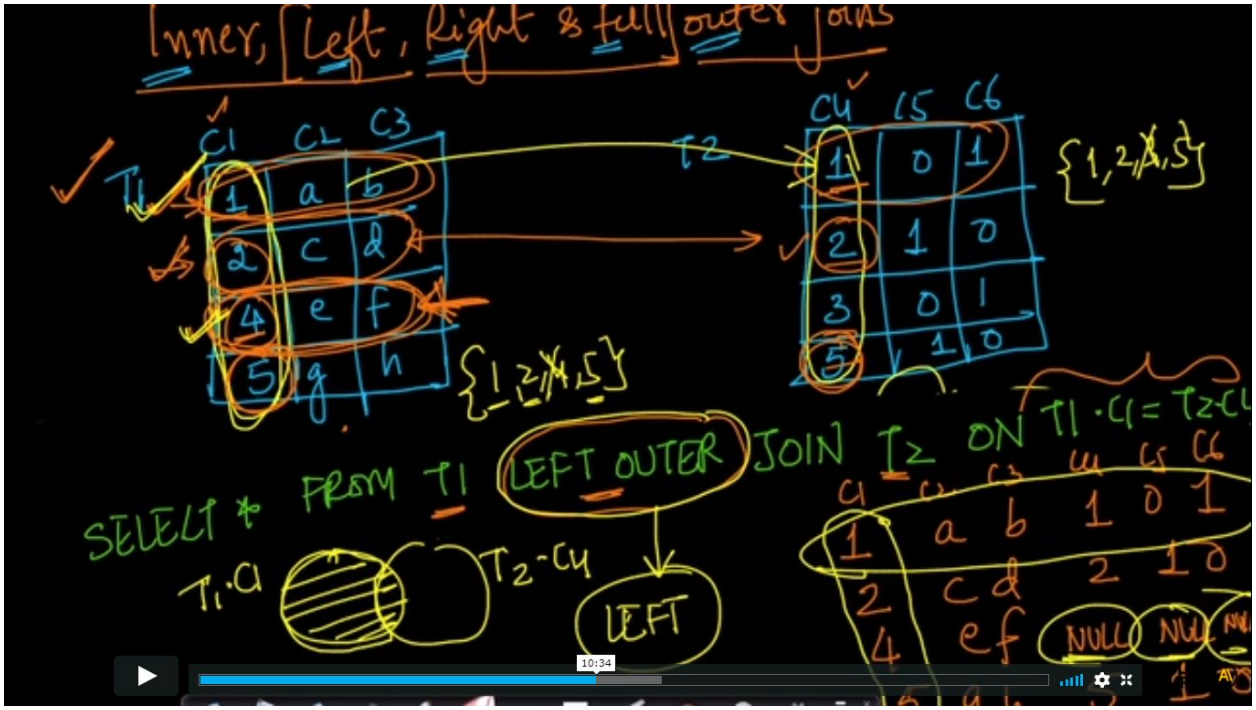
#combine data in multiple tables

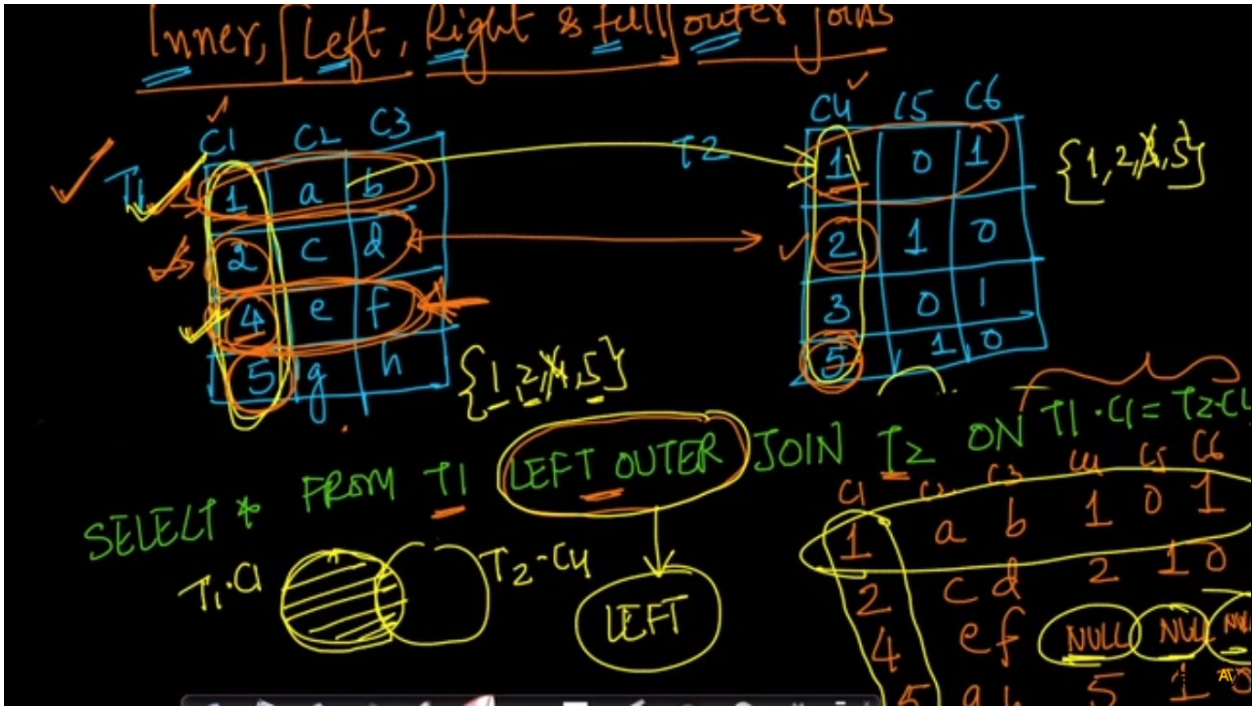
```
# For each movie, print name and the genres
SELECT m.name, g.genre from movies m  JOIN movies_genres g ON m.id=g.movie_id LIMIT
20;

# table aliases: m and g

# natural join: a join where we have the same column-names across two tables.
#T1: C1, C2
#T2: C1, C3, C4

SELECT * FROM T1 JOIN T2;

SELECT * FROM T1 JOIN T2 USING (C1);

# returns C1,C2,C3,C4
# no need to use the keyword "ON"



# Inner join (default) vs left outer vs right outer vs full-outer join.
```

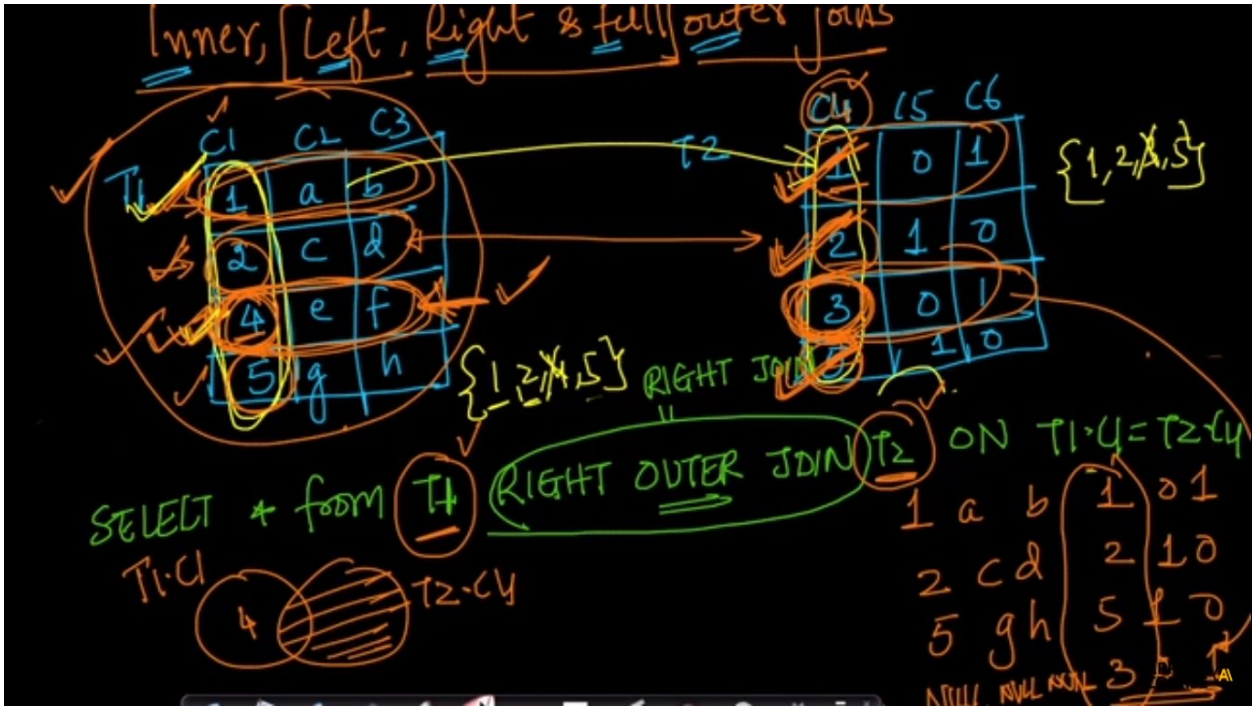
****

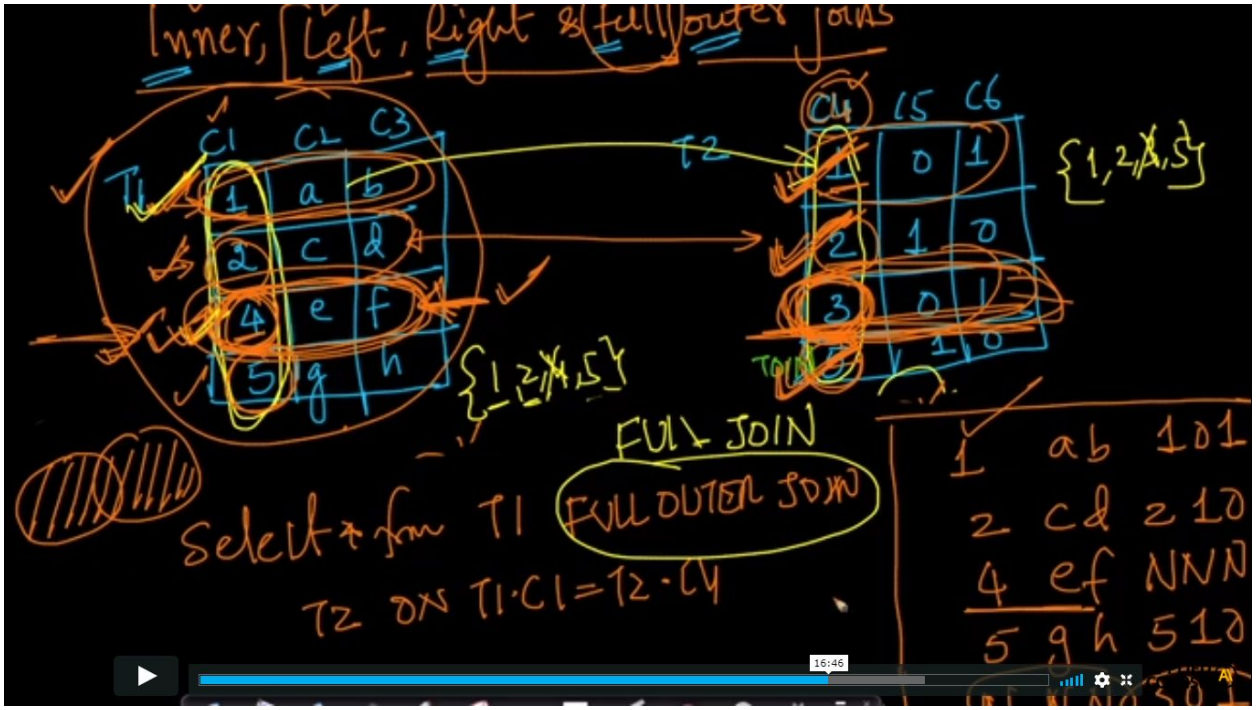
Inner, Left, Right and Outer joins.

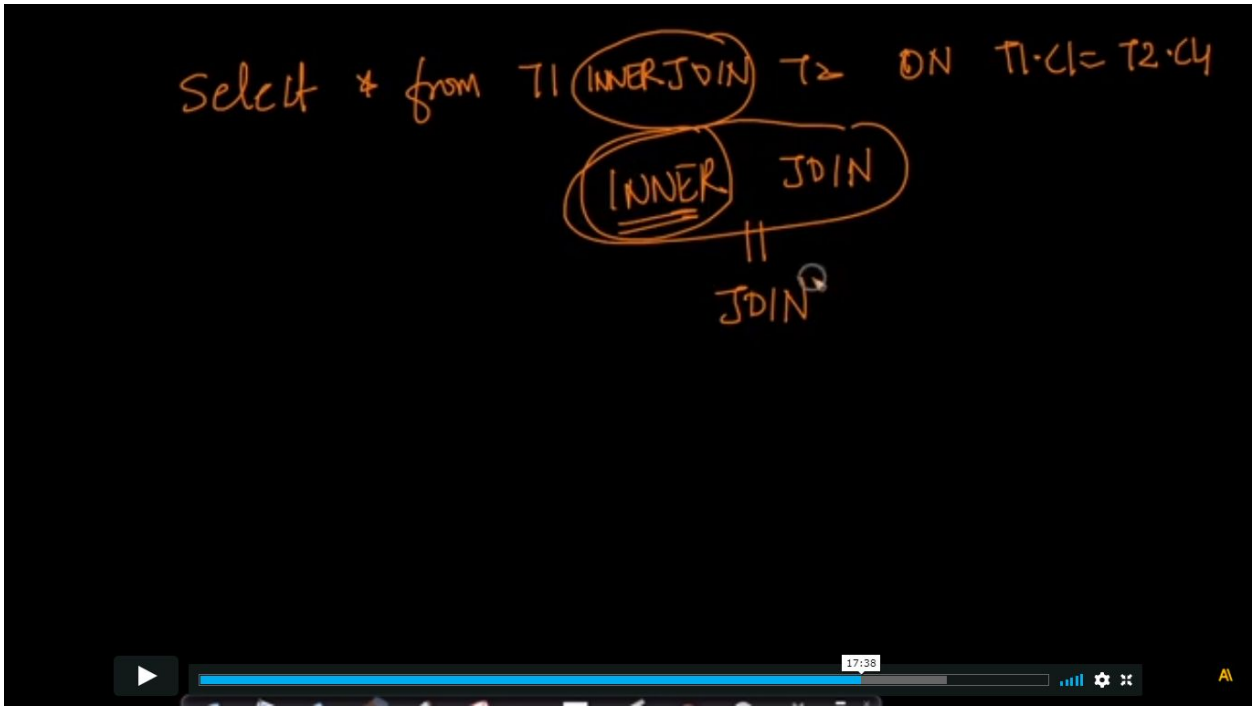

Inner, Left, Right & full outer joins
T1
T2
INNER
select * from T1 JOIN T2 ON T1.C1 = T2.C4
05:51

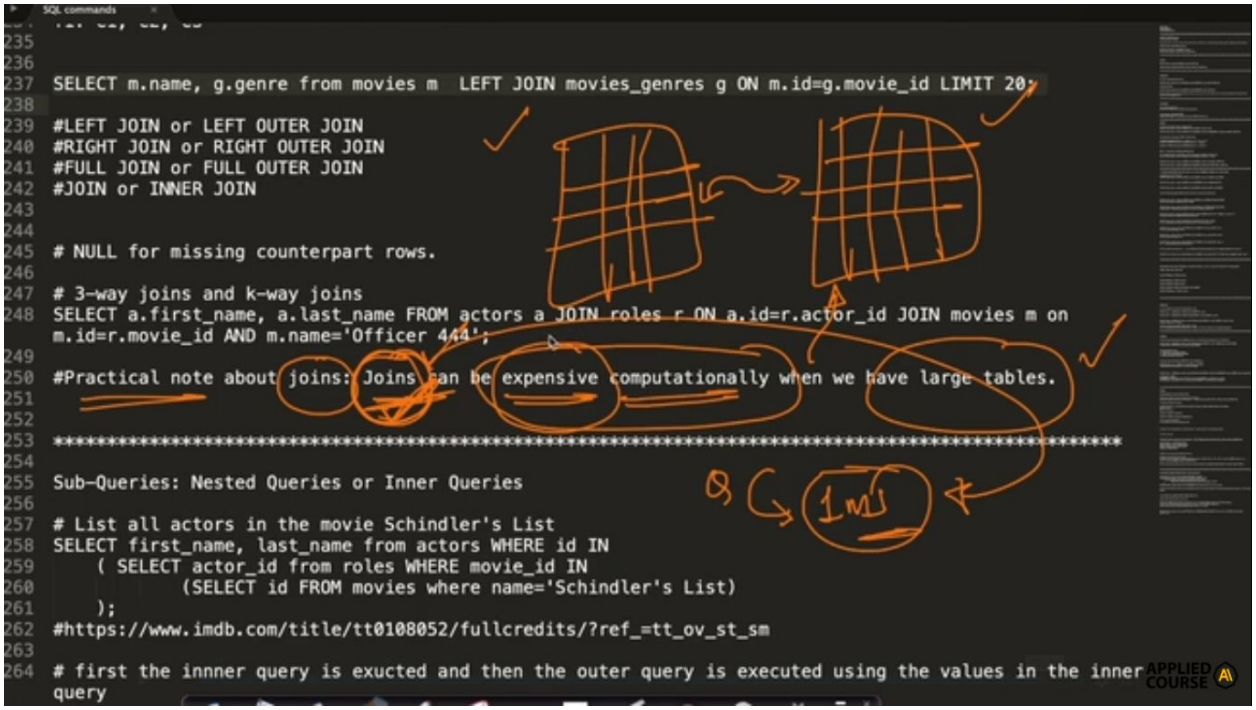

Inner, Left, Right & full outer joins
T1
T2
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1 = T2.C4
LEFT
NULL
10:34


Inner, Left, Right & full outer joins
C1 C2 C3
T1
1 a b
2 c d
4 e f
5 g h
{1,2,4,5}
T2
C4 C5 C6
1 0 1
2 1 0
3 0 1
5 1 0
{1,2,4,5}
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1 = T2.C4
T1.C1
T2.C4
LEFT
NULL


Inner, Left, Right & full outer joins
C1 C2 C3
T1
1 a b
2 c d
4 e f
5 g h
{1,2,4,5}
T2
C4 C5 C6
1 0 1
2 1 0
3 0 1
5 1 0
{1,2,4,5}
RIGHT JOIN
SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.C1 = T2.C4
T1.C1
T2.C4
4
1 a b
2 c d
5 g h
NULL


Inner, Left, Right & full outer joins
C1 C2 C3
T1
1 a b
2 c d
4 e f
5 g h
T2
C4 C5 C6
1 0 1
2 1 0
3 0 1
1 1 0
{1,2,4,5}
{1,2,4,5}
FULL JOIN
FULL OUTER JOIN
Select * from T1
T2 ON T1.C1=T2.C4
1 ab 1 0 1
2 cd 2 1 0
4 ef NNN
5 gh 5 1 0
16:46





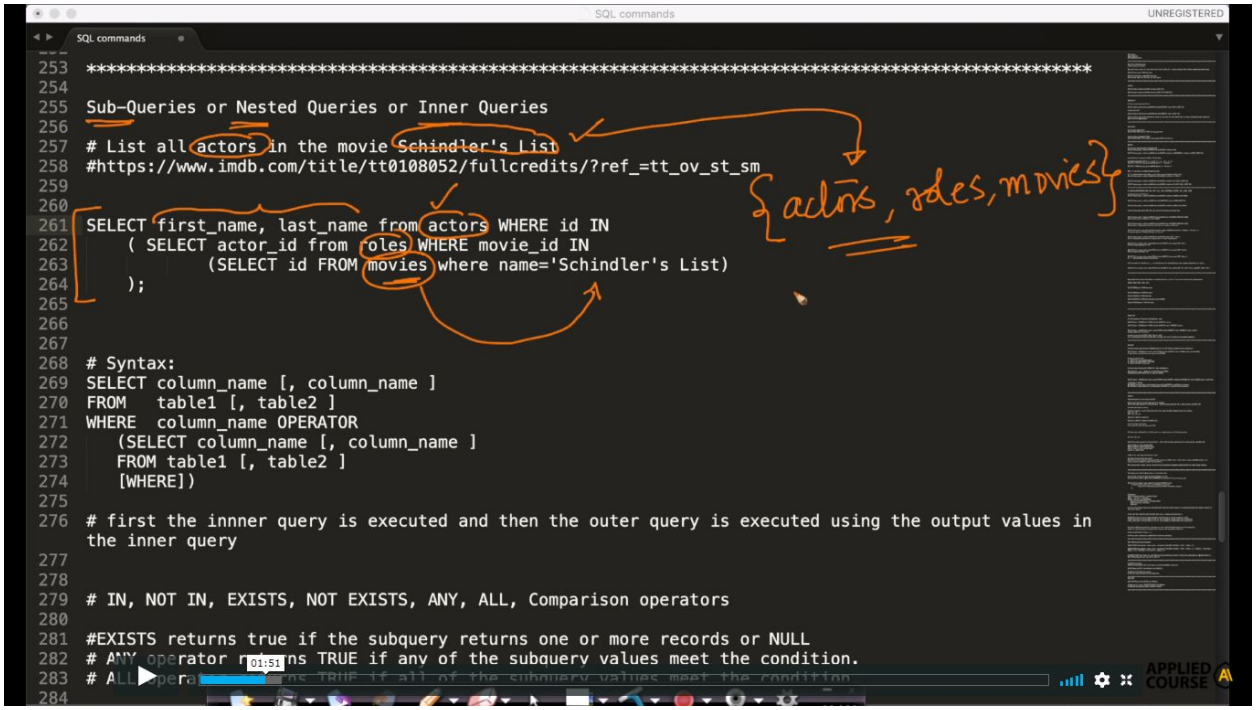
Inner join (default) vs left outer vs right outer vs full-outer join.

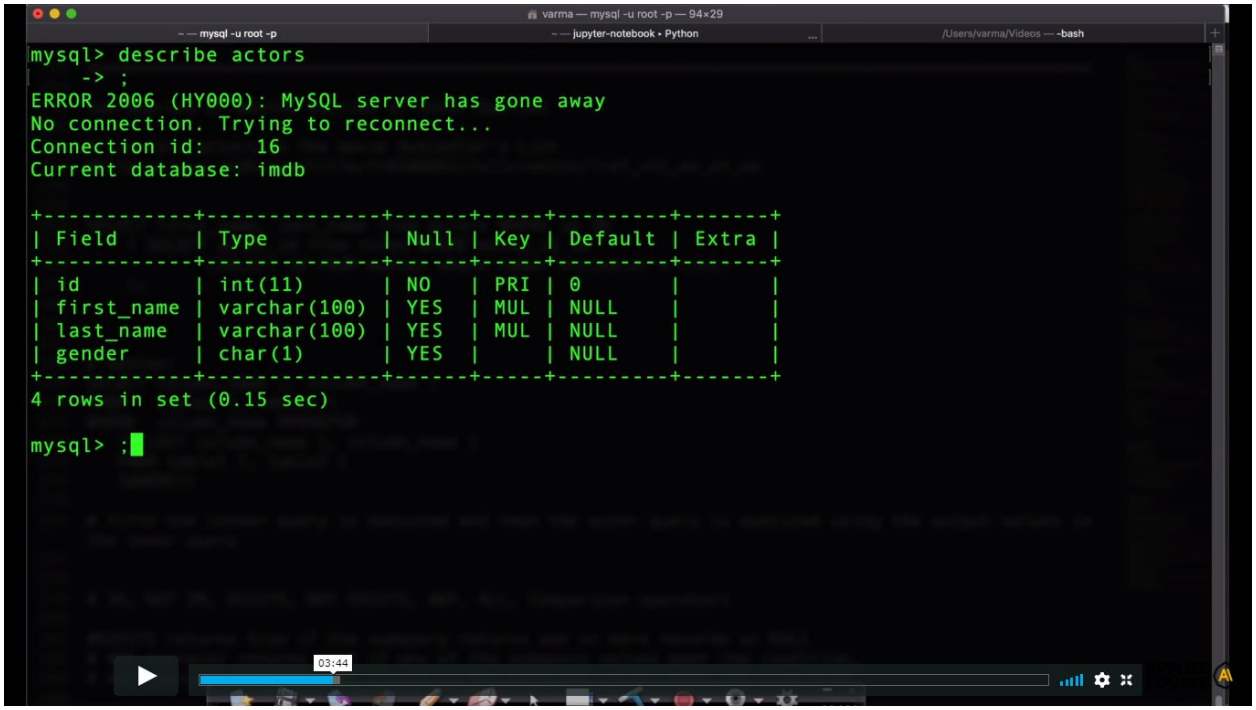
T1: C1, C2, C3

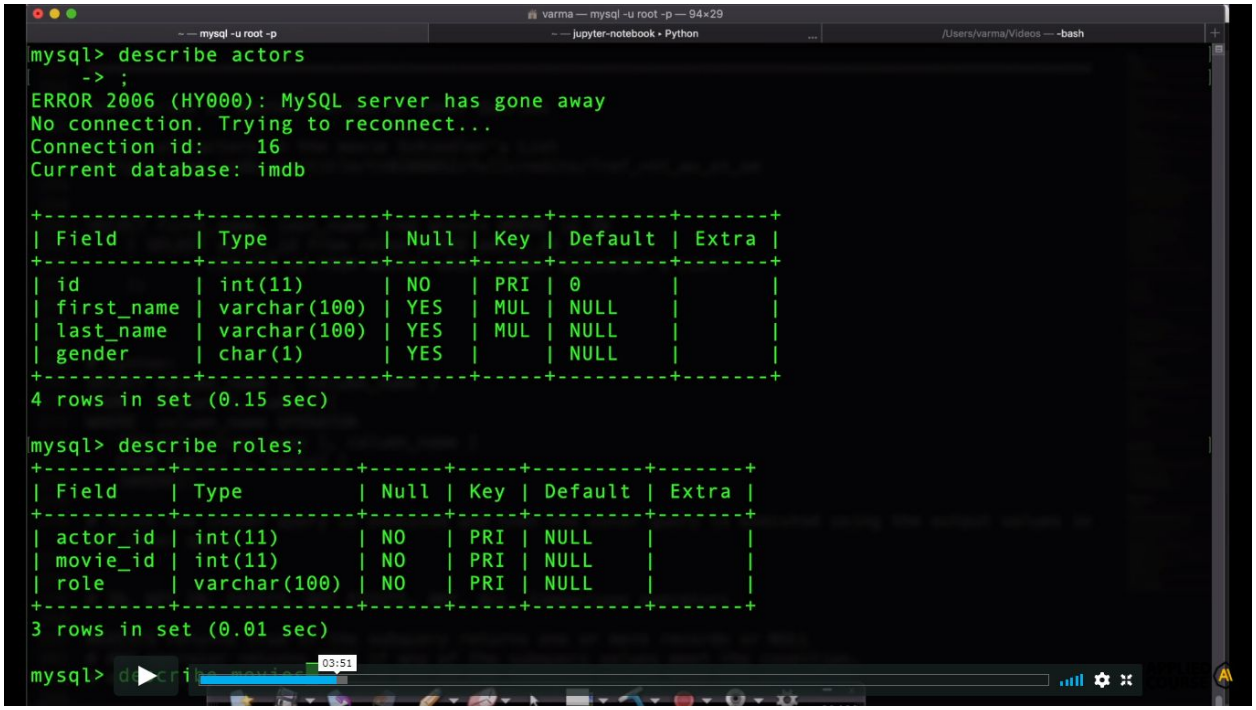
```
SELECT m.name, g.genre from movies m  LEFT JOIN movies_genres g ON m.id=g.movie_id
LIMIT 20;

#LEFT JOIN or LEFT OUTER JOIN
#RIGHT JOIN or RIGHT OUTER JOIN
#FULL JOIN or FULL OUTER JOIN
#JOIN or INNER JOIN


# NULL for missing counterpart rows.

# 3-way joins and k-way joins
SELECT a.first_name, a.last_name FROM actors a JOIN roles r ON a.id=r.actor_id JOIN
movies m on m.id=r.movie_id AND m.name='Officer 444';

#Practical note about joins: Joins can be expensive computationally when we have large tables.
```

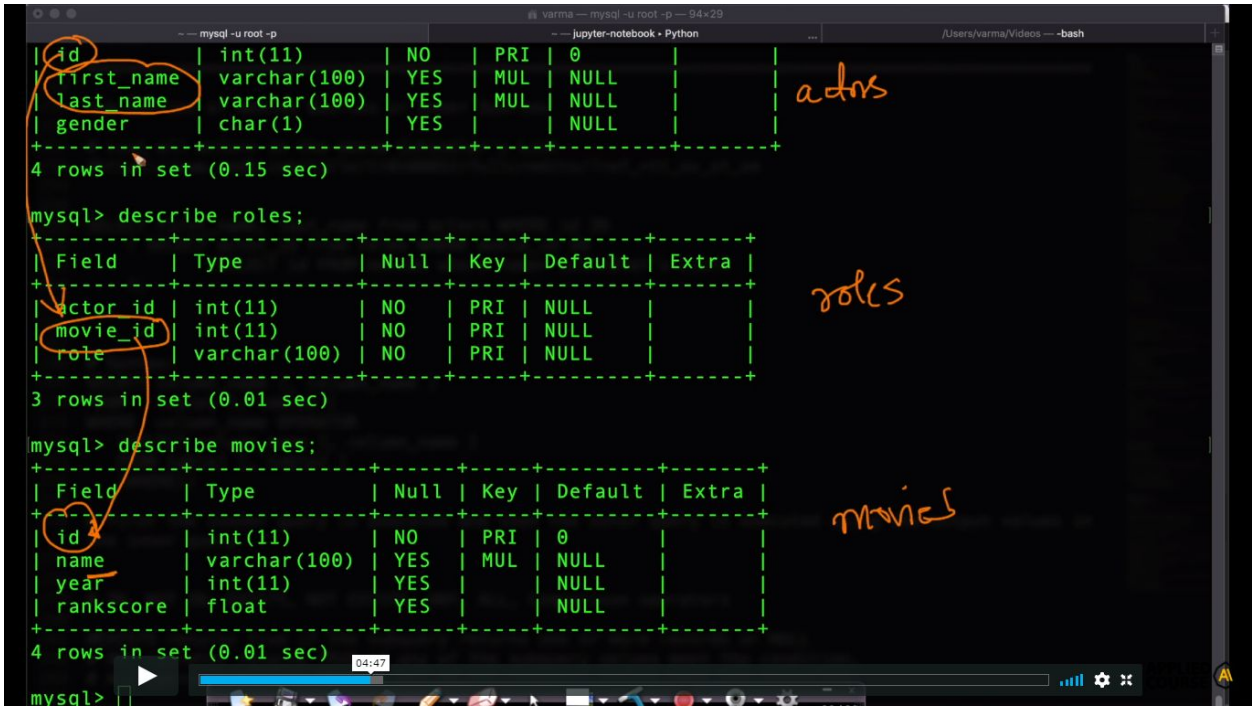
****

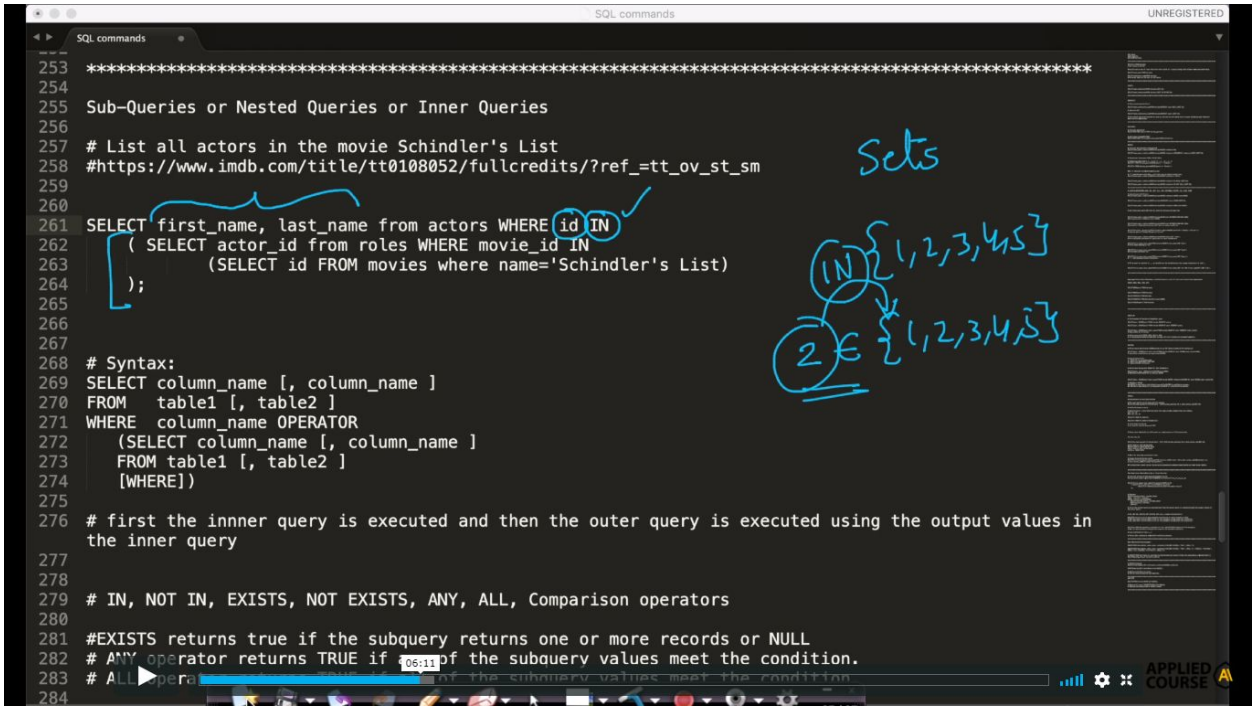
Sub Queries/Nested Queries/Inner Queries


SQL commands
UNREGISTERED
Sub-Queries or Nested Queries or Inner Queries
List all actors in the movie Schindler's List
#https://www.imdb.com/title/tt0108052/fullcredits/?ref_=tt_ov_st_sm
SELECT first_name, last_name from actors WHERE id IN
(SELECT actor_id from roles WHERE movie_id IN
(SELECT id FROM movies where name='Schindler's List')
);
{actors, roles, movies}
Syntax:
SELECT column_name [, column_name]
FROM table1 [, table2]
WHERE column_name OPERATOR
(SELECT column_name [, column_name]
FROM table1 [, table2]
[WHERE])
first the innner query is executed and then the outer query is executed using the output values in the inner query
IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL, Comparison operators
#EXISTS returns true if the subquery returns one or more records or NULL
01:51
APPLIED COURSE

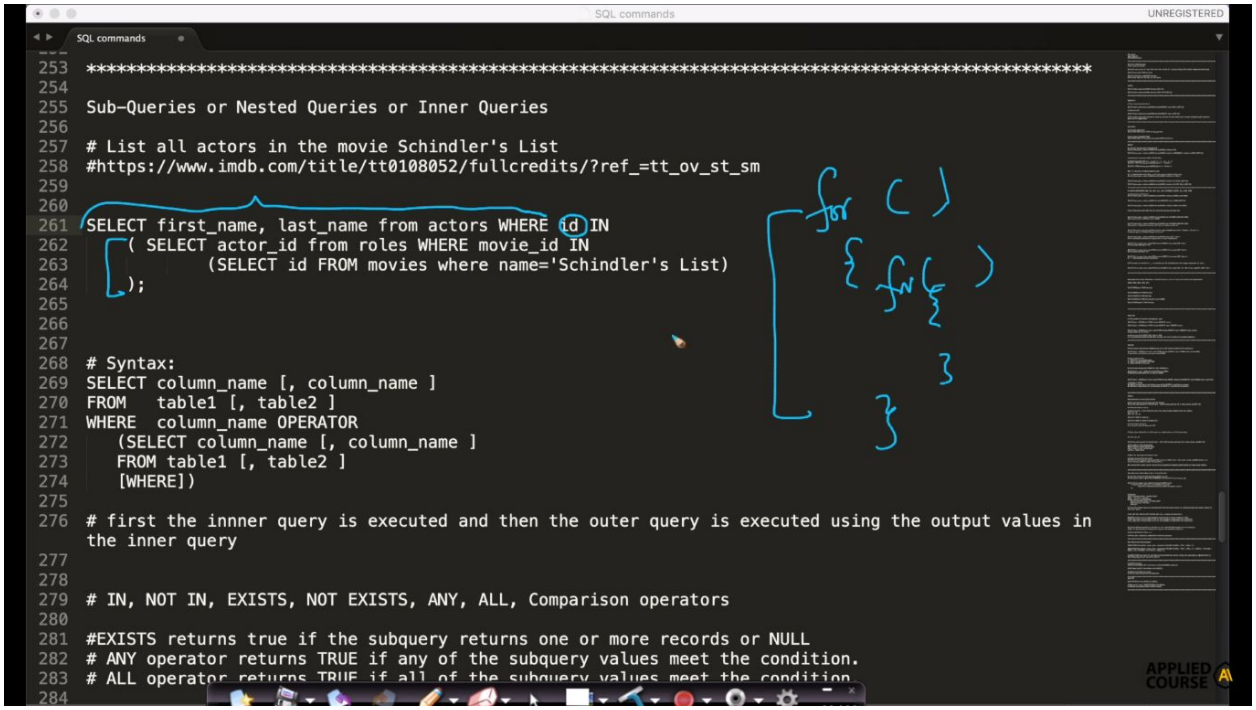

varma — mysql -u root -p — 94×29
~ — mysql -u root -p
~ — jupyter-notebook • Python
/Users/varma/Videos — -bash
mysql> describe actors
-> ;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id: 16
Current database: imdb
Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	0	
first_name	varchar(100)	YES	MUL	NULL	
last_name	varchar(100)	YES	MUL	NULL	
gender	char(1)	YES		NULL	
4 rows in set (0.15 sec)
mysql> ;
03:44


varma — mysql -u root -p — 94×29
~ — mysql -u root -p
~ — jupyter-notebook ▸ Python
/Users/varma/Videos — -bash
mysql> describe actors
-> ;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id: 16
Current database: imdb
+------------+--------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
id	int(11)	NO	PRI	0	
first_name	varchar(100)	YES	MUL	NULL	
last_name	varchar(100)	YES	MUL	NULL	
gender	char(1)	YES		NULL	
+------------+--------------+------+-----+---------+-------+
4 rows in set (0.15 sec)
mysql> describe roles;
+----------+--------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
actor_id	int(11)	NO	PRI	NULL	
movie_id	int(11)	NO	PRI	NULL	
role	varchar(100)	NO	PRI	NULL	
+----------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
mysql>
03:51

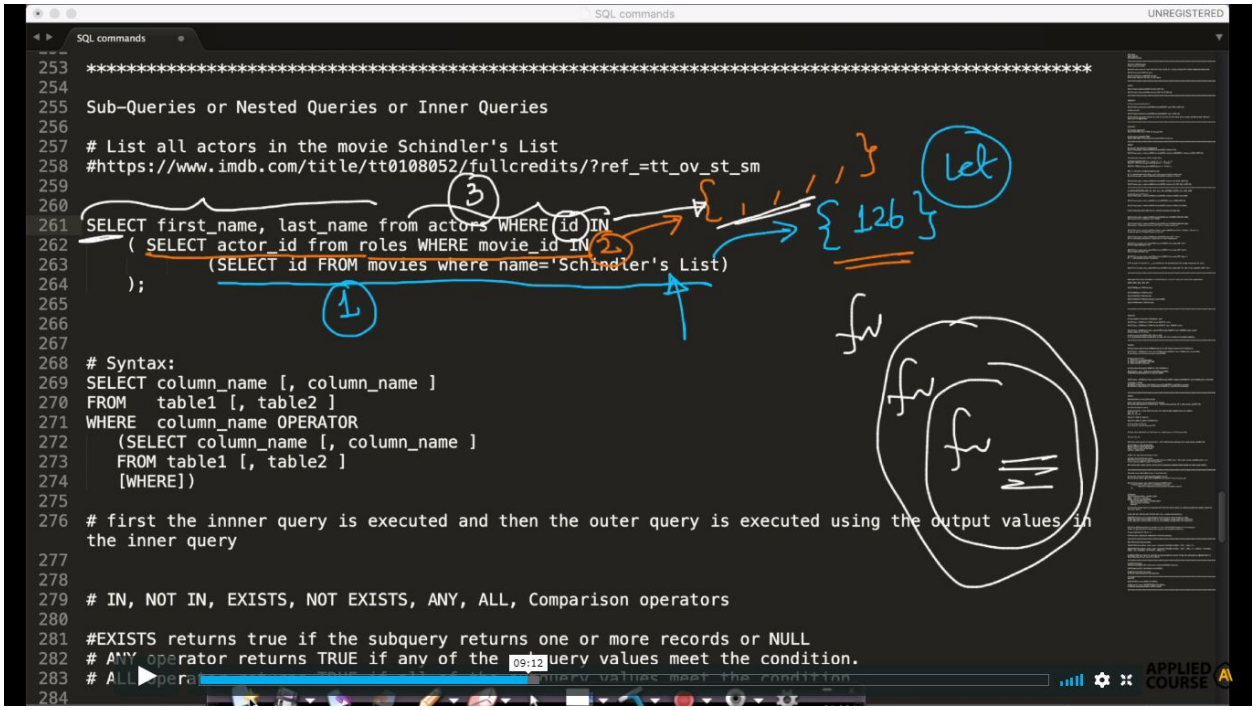

varma — mysql -u root -p — 94×29
~ — mysql -u root -p
~ — jupyter-notebook ▸ Python
/Users/varma/Videos — -bash
id	int(11)	NO	PRI	0	
first_name	varchar(100)	YES	MUL	NULL	
last_name	varchar(100)	YES	MUL	NULL	
gender	char(1)	YES		NULL	
actors
4 rows in set (0.15 sec)
mysql> describe roles;
actor_id	int(11)	NO	PRI	NULL	
movie_id	int(11)	NO	PRI	NULL	
role	varchar(100)	NO	PRI	NULL	
roles
3 rows in set (0.01 sec)
mysql> describe movies;
id	int(11)	NO	PRI	0	
name	varchar(100)	YES	MUL	NULL	
year	int(11)	YES		NULL	
rankscore	float	YES		NULL	
movies
4 rows in set (0.01 sec)
mysql>
04:47

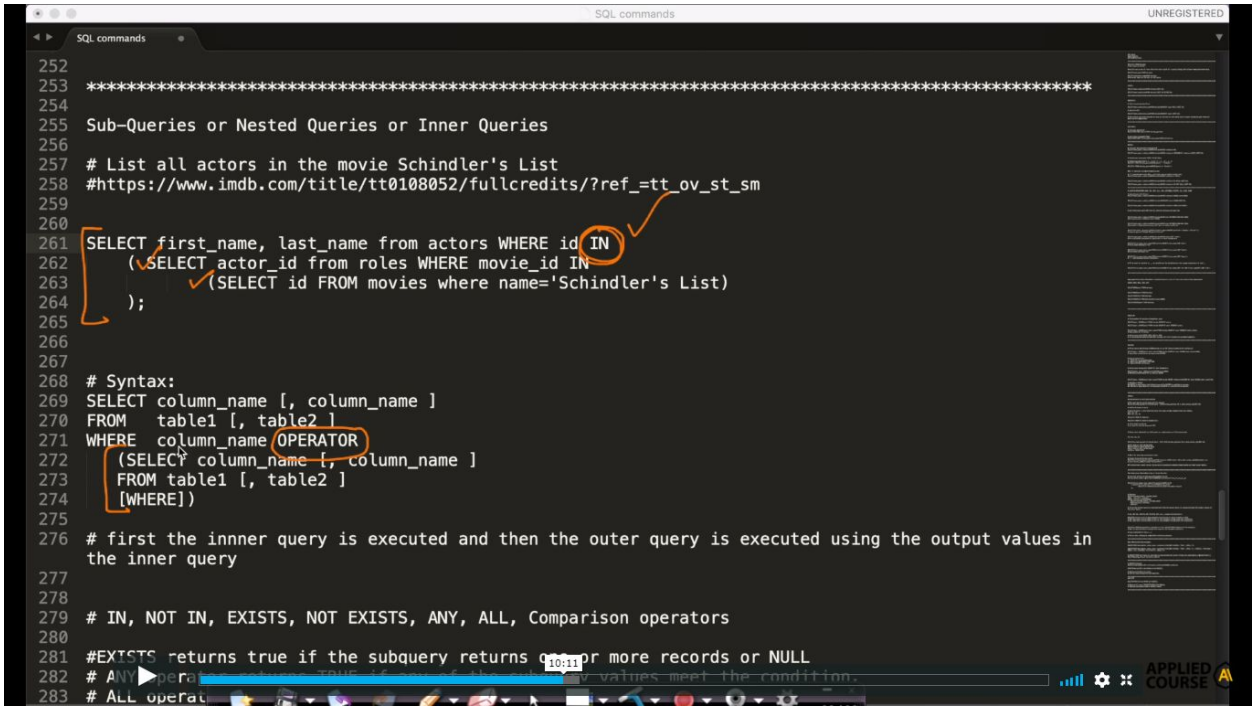


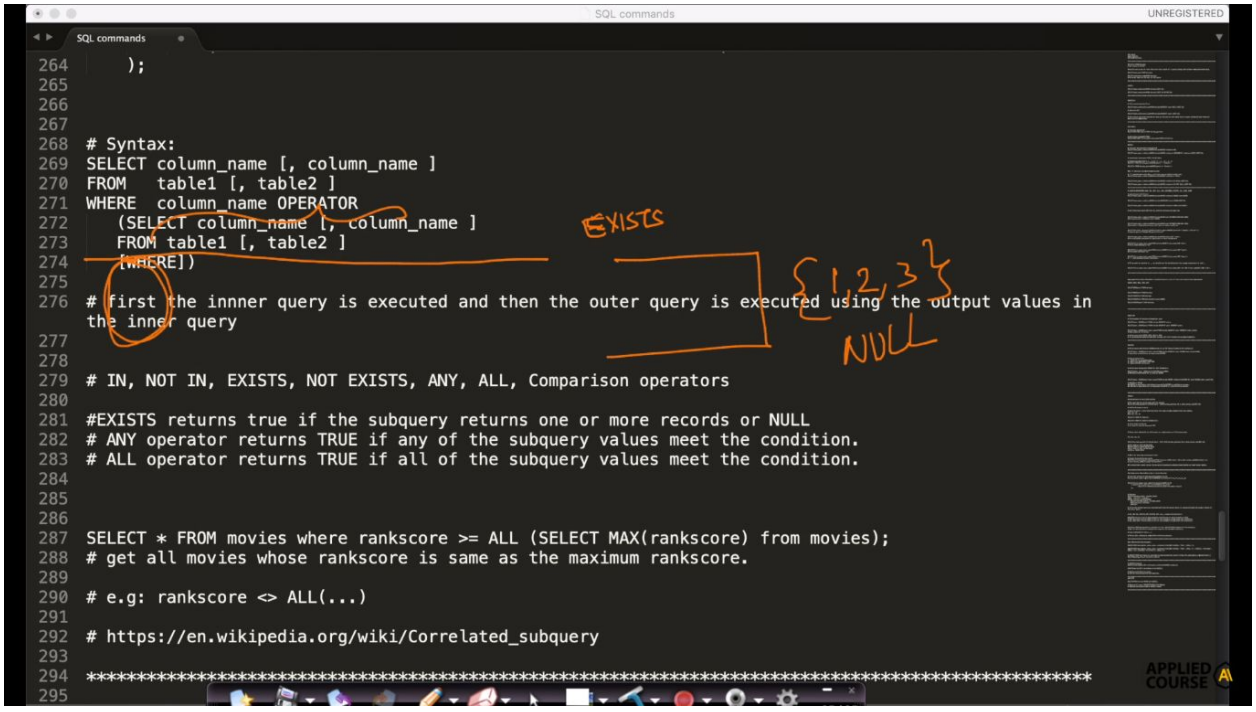

```
253 ********************************************************************************************************
254
255 Sub-Queries or Nested Queries or Inner Queries
256
257 # List all actors in the movie Schindler's List
258 #https://www.imdb.com/title/tt0108052/fullcredits/?ref_=tt_ov_st_sm
259
260
261 SELECT first_name, last_name from actors WHERE id IN
262     ( SELECT actor_id from roles WHERE movie_id IN
263             (SELECT id FROM movies where name='Schindler's List)
264     );
265
266
267
268 # Syntax:
269 SELECT column_name [, column_name ]
270 FROM   table1 [, table2 ]
271 WHERE  column_name OPERATOR
272    (SELECT column_name [, column_name ]
273    FROM table1 [, table2 ]
274    [WHERE])
275
276 # first the innner query is executed and then the outer query is executed using the output values in
    the inner query
277
278
279 # IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL, Comparison operators
280
281 #EXISTS returns true if the subquery returns one or more records or NULL
282 # ANY operator returns TRUE if any of the subquery values meet the condition.
283 # ALL operator returns TRUE if all of the subquery values meet the condition
284
```

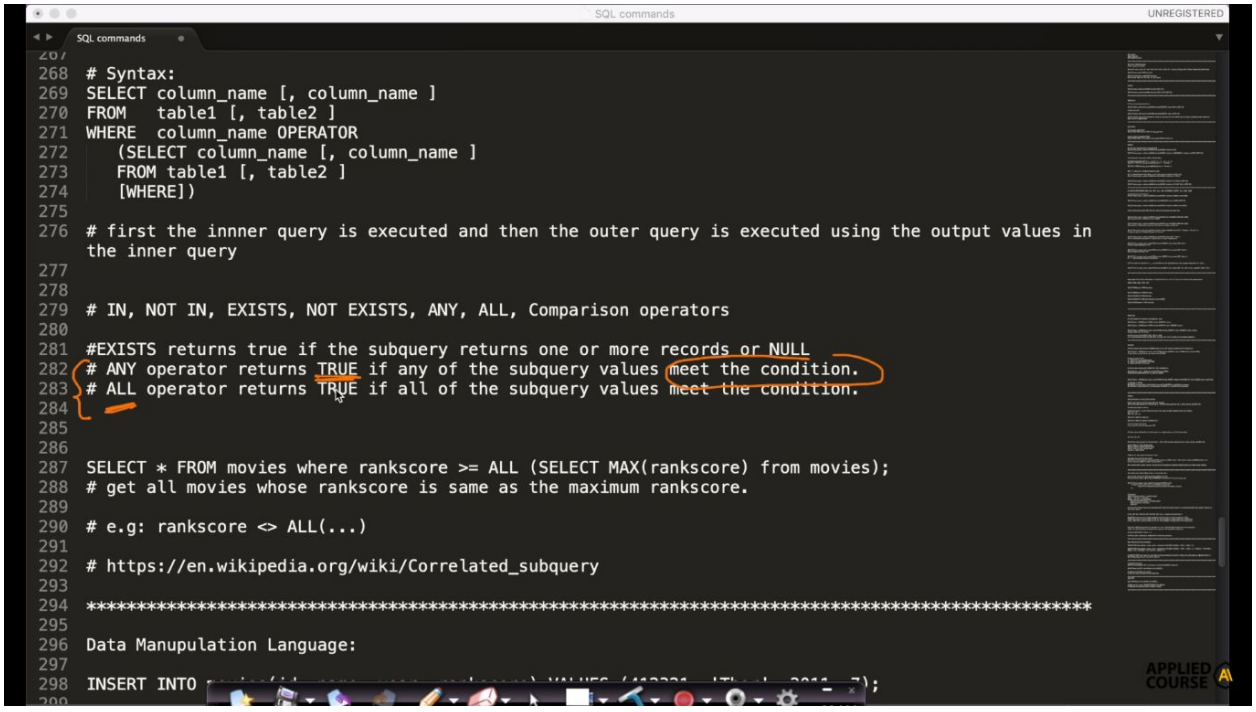


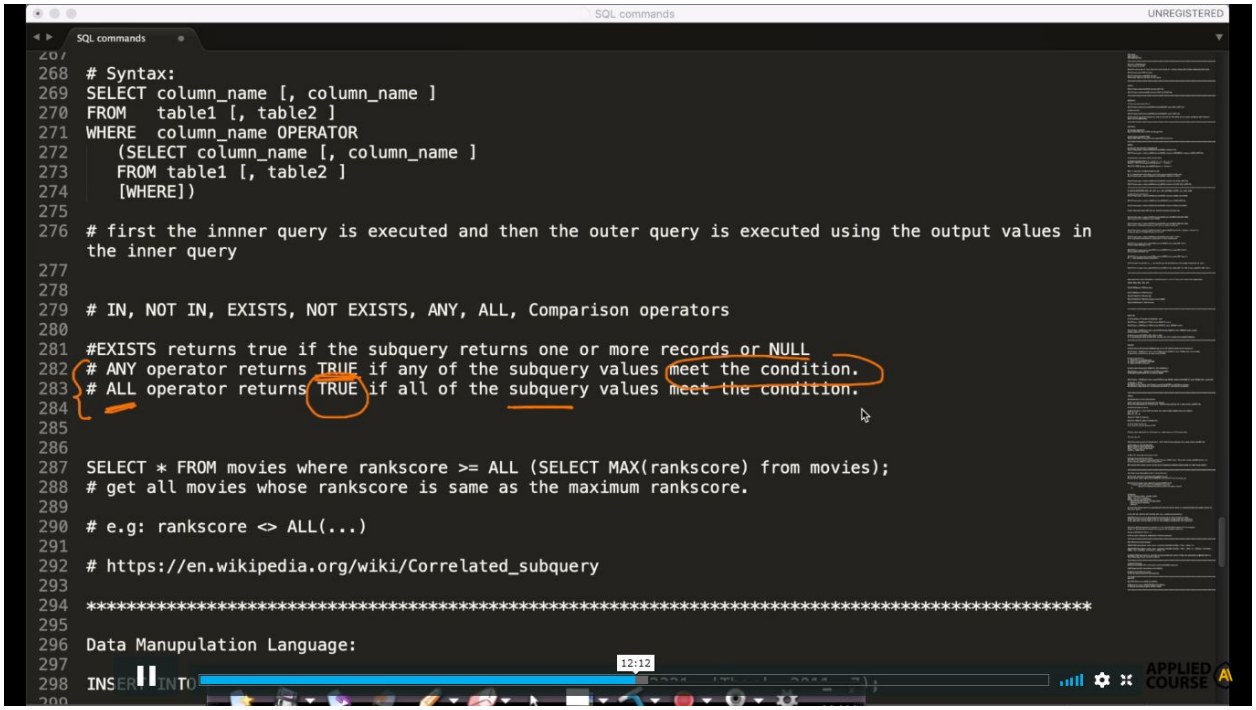

```
253 ********************************************************************************************************
254
255 Sub-Queries or Nested Queries or Inner Queries
256
257 # List all actors in the movie Schindler's List
258 #https://www.imdb.com/title/tt0108052/fullcredits/?ref_=tt_ov_st_sm
259
260
261 SELECT first_name, last_name from actors WHERE id IN
262     ( SELECT actor_id from roles WHERE movie_id IN
263             (SELECT id FROM movies where name='Schindler's List)
264     );
265
266
267
268 # Syntax:
269 SELECT column_name [, column_name ]
270 FROM   table1 [, table2 ]
271 WHERE  column_name OPERATOR
272    (SELECT column_name [, column_name ]
273    FROM table1 [, table2 ]
274    [WHERE])
275
276 # first the innner query is executed and then the outer query is executed using the output values in
    the inner query
277
278
279 # IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL, Comparison operators
280
281 #EXISTS returns true if the subquery returns one or more records or NULL
282 # ANY operator returns TRUE if any of the subquery values meet the condition.
283 # ALL operator returns TRUE if all of the subquery values meet the condition
284
```

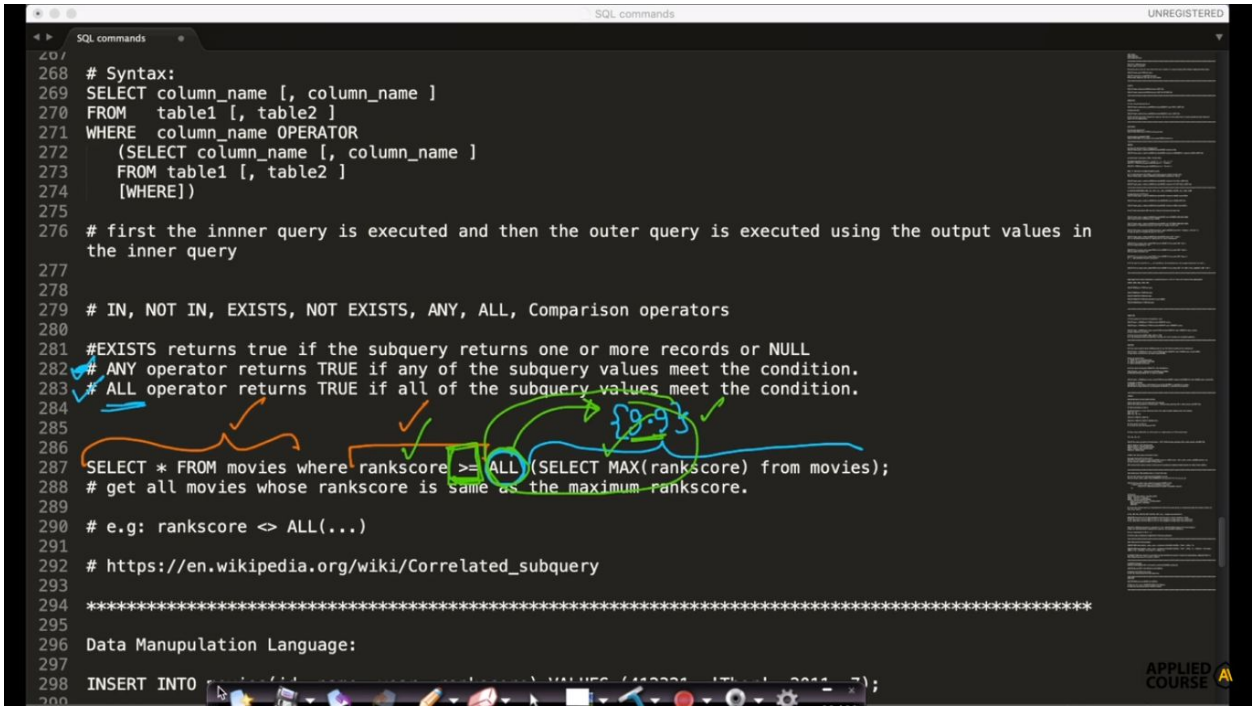

SQL commands
UNREGISTERED
SQL commands
253 **********
254
255 Sub-Queries or Nested Queries or Inner Queries
256
257 # List all actors in the movie Schindler's List
258 #https://www.imdb.com/title/tt0108052/fullcredits/?ref_=tt_ov_st_sm
259
260
261 SELECT first_name, last_name from actors WHERE id IN
262 (SELECT actor_id from roles WHERE movie_id IN
263 (SELECT id FROM movies where name='Schindler's List')
264);
265
266
267
268 # Syntax:
269 SELECT column_name [, column_name]
270 FROM table1 [, table2]
271 WHERE column_name OPERATOR
272 (SELECT column_name [, column_name]
273 FROM table1 [, table2]
274 [WHERE])
275
276 # first the innner query is executed and then the outer query is executed using the output values in the inner query
277
278
279 # IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL, Comparison operators
280
281 #EXISTS returns true if the subquery returns one or more records or NULL
282 # ANY operator returns TRUE if any of the
283 # ALL opera
284
3
2
1
126
let
fw
fw
fw
09:12
APPLIED COURSE

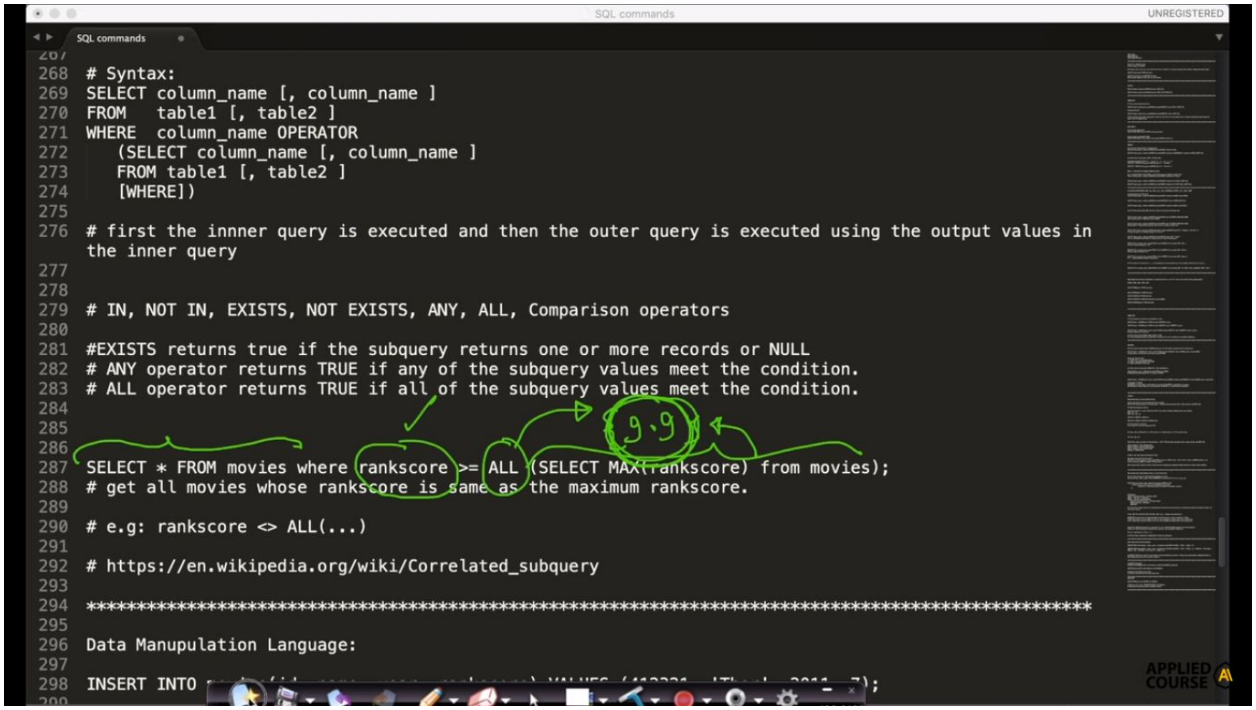

SQL commands
UNREGISTERED
SQL commands
252
253 **********
254
255 Sub-Queries or Nested Queries or Inner Queries
256
257 # List all actors in the movie Schindler's List
258 #https://www.imdb.com/title/tt0108052/fullcredits/?ref_=tt_ov_st_sm
259
260
261 SELECT first_name, last_name from actors WHERE id IN
262 (SELECT actor_id from roles WHERE movie_id IN
263 (SELECT id FROM movies where name='Schindler's List')
264);
265
266
267
268 # Syntax:
269 SELECT column_name [, column_name]
270 FROM table1 [, table2]
271 WHERE column_name OPERATOR
272 (SELECT column_name [, column_name]
273 FROM table1 [, table2]
274 [WHERE])
275
276 # first the innner query is executed and then the outer query is executed using the output values in the inner query
277
278
279 # IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL, Comparison operators
280
281 #EXISTS returns true if the subquery returns one or more records or NULL
282 # ANY operator returns TRUE if any of the subquery values meet the condition.
283 # ALL operat
10:11
APPLIED COURSE

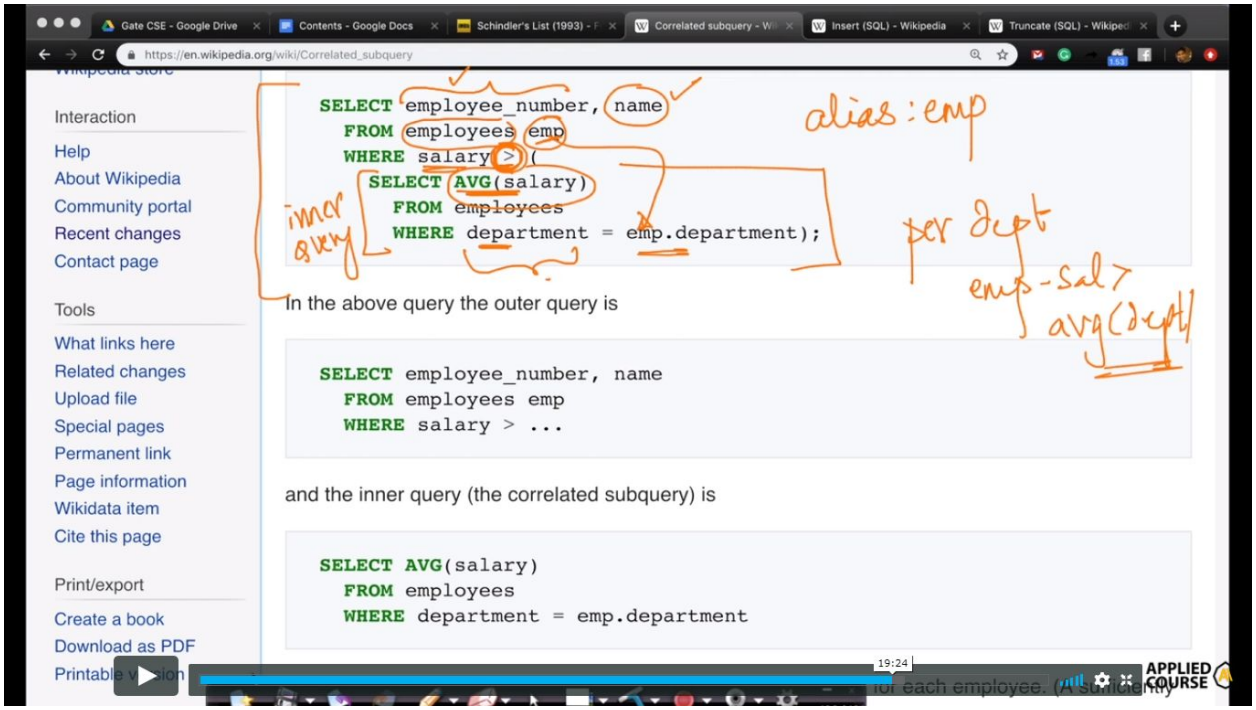

SQL commands
UNREGISTERED
SQL commands
);
Syntax:
SELECT column_name [, column_name]
FROM table1 [, table2]
WHERE column_name OPERATOR
(SELECT column_name [, column_name]
FROM table1 [, table2]
[WHERE])
EXISTS
{1,2,3}
NULL
first the innner query is executed and then the outer query is executed using the output values in the inner query
IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL, Comparison operators
#EXISTS returns true if the subquery returns one or more records or NULL
ANY operator returns TRUE if any of the subquery values meet the condition.
ALL operator returns TRUE if all of the subquery values meet the condition.
SELECT * FROM movies where rankscore >= ALL (SELECT MAX(rankscore) from movies);
get all movies whose rankscore is same as the maximum rankscore.
e.g: rankscore <> ALL(...)
https://en.wikipedia.org/wiki/Correlated_subquery
APPLIED COURSE

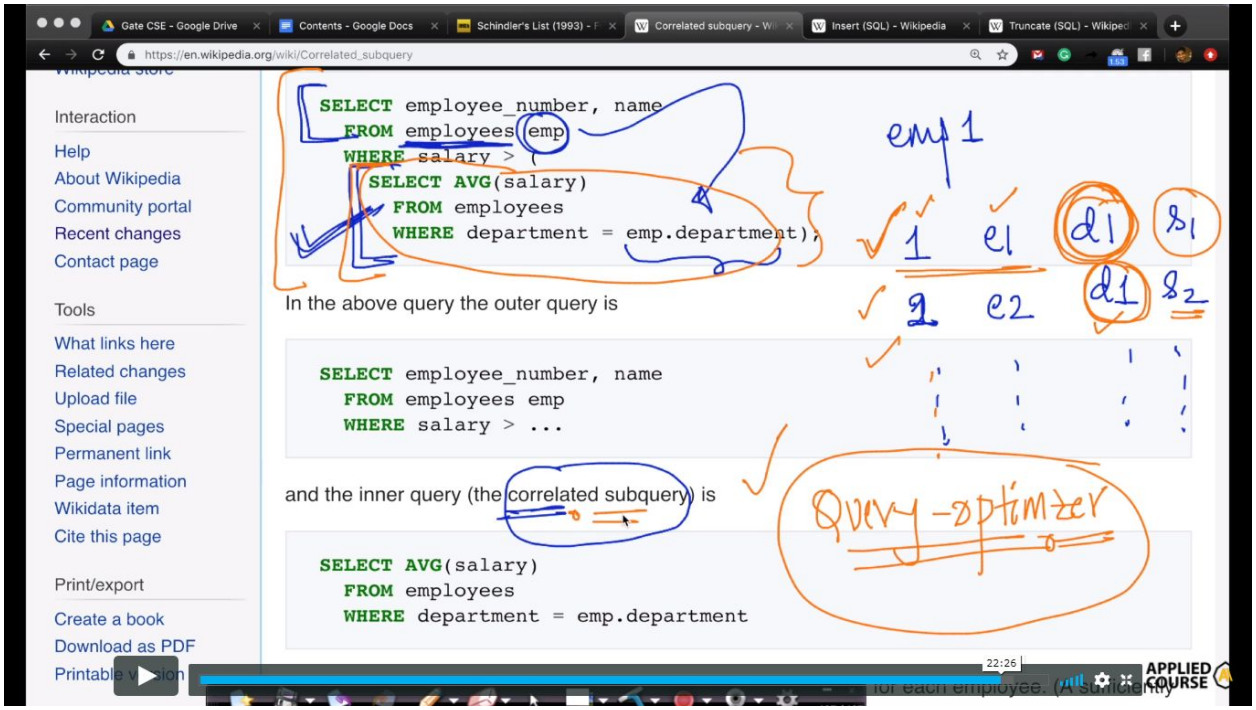

SQL commands
UNREGISTERED
SQL commands
Syntax:
SELECT column_name [, column_name]
FROM table1 [, table2]
WHERE column_name OPERATOR
(SELECT column_name [, column_name]
FROM table1 [, table2]
[WHERE])
first the innner query is executed and then the outer query is executed using the output values in the inner query
IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL, Comparison operators
#EXISTS returns true if the subquery returns one or more records or NULL
ANY operator returns TRUE if any of the subquery values meet the condition.
ALL operator returns TRUE if all of the subquery values meet the condition.
SELECT * FROM movies where rankscore >= ALL (SELECT MAX(rankscore) from movies);
get all movies whose rankscore is same as the maximum rankscore.
e.g: rankscore <> ALL(...)
https://en.wikipedia.org/wiki/Correlated_subquery
Data Manupulation Language:
INSERT INTO
APPLIED COURSE


SQL commands
UNREGISTERED
SQL commands
Syntax:
SELECT column_name [, column_name]
FROM table1 [, table2]
WHERE column_name OPERATOR
(SELECT column_name [, column_name]
FROM table1 [, table2]
[WHERE])
first the innner query is executed and then the outer query is executed using the output values in the inner query
IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL, Comparison operators
#EXISTS returns true if the subquery returns one or more records or NULL
ANY operator returns TRUE if any of the subquery values meet the condition.
ALL operator returns TRUE if all of the subquery values meet the condition.
SELECT * FROM movies where rankscore >= ALL (SELECT MAX(rankscore) from movies);
get all movies whose rankscore is same as the maximum rankscore.
e.g: rankscore <> ALL(...)
https://en.wikipedia.org/wiki/Correlated_subquery
Data Manupulation Language:
12:12
APPLIED COURSE


SQL commands
UNREGISTERED
SQL commands
Syntax:
SELECT column_name [, column_name]
FROM table1 [, table2]
WHERE column_name OPERATOR
(SELECT column_name [, column_name]
FROM table1 [, table2]
[WHERE])
first the innner query is executed and then the outer query is executed using the output values in the inner query
IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL, Comparison operators
#EXISTS returns true if the subquery returns one or more records or NULL
ANY operator returns TRUE if any of the subquery values meet the condition.
ALL operator returns TRUE if all of the subquery values meet the condition.
SELECT * FROM movies where rankscore >= ALL (SELECT MAX(rankscore) from movies);
get all movies whose rankscore is same as the maximum rankscore.
e.g: rankscore <> ALL(...)
https://en.wikipedia.org/wiki/Correlated_subquery
Data Manupulation Language:
APPLIED COURSE


SQL commands
UNREGISTERED
SQL commands
Syntax:
SELECT column_name [, column_name]
FROM table1 [, table2]
WHERE column_name OPERATOR
(SELECT column_name [, column_name]
FROM table1 [, table2]
[WHERE])
first the innner query is executed and then the outer query is executed using the output values in the inner query
IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL, Comparison operators
#EXISTS returns true if the subquery returns one or more records or NULL
ANY operator returns TRUE if any of the subquery values meet the condition.
ALL operator returns TRUE if all of the subquery values meet the condition.
9.9
SELECT * FROM movies where rankscore >= ALL (SELECT MAX(rankscore) from movies);
get all movies whose rankscore is same as the maximum rankscore.
e.g: rankscore <> ALL(...)
https://en.wikipedia.org/wiki/Correlated_subquery
Data Manupulation Language:
INSERT INTO
APPLIED COURSE





****

Sub-Queries or Nested Queries or Inner Queries

List all actors in the movie Schindler's List

#https://www.imdb.com/title/tt0108052/fullcredits/?ref_=tt_ov_st_sm

```
SELECT first_name, last_name from actors WHERE id IN
        ( SELECT actor_id from roles WHERE movie_id IN
                        (SELECT id FROM movies where name='Schindler's List)
        );
```

```
# Syntax:
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
  (SELECT column_name [, column_name ]
  FROM table1 [, table2 ]
  [WHERE])
```

first the innner query is executed and then the outer query is executed using the output values in the inner query

IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL, Comparison operators

#EXISTS returns true if the subquery returns one or more records or NULL
ANY operator returns TRUE if any of the subquery values meet the condition.
ALL operator returns TRUE if all of the subquery values meet the condition.

```
SELECT * FROM movies where rankscore >= ALL (SELECT MAX(rankscore) from movies);
```

get all movies whose rankscore is same as the maximum rankscore.

e.g: rankscore <> ALL(...)

https://en.wikipedia.org/wiki/Correlated_subquery

****

DML:INSERT

****

Data Manupulation Language: SELECT, INSERT, UPDATE, DELETE

INSERT INTO movies(id, name, year, rankscore) VALUES (412321, 'Thor', 2011, 7);

INSERT INTO movies(id, name, year, rankscore) VALUES (412321, 'Thor', 2011, 7), (412322, 'Iron Man', 2008, 7.9), (412323, 'Iron Man 2', 2010, 7);

INSERT FROM one table to annother using nnested sub query:
https://en.wikipedia.org/wiki/Insert_(SQL)#Copying_rows_from_other_tables

****

DML:UPDATE , DELETE

****

UPDATE Command
UPDATE <TableName> SET col1=val1, col2=val2 WHERE condition

UPDATE movies SET rankscore=9 where id=412321;

Update multiple rows also.
Can be used along with Sub-queries.

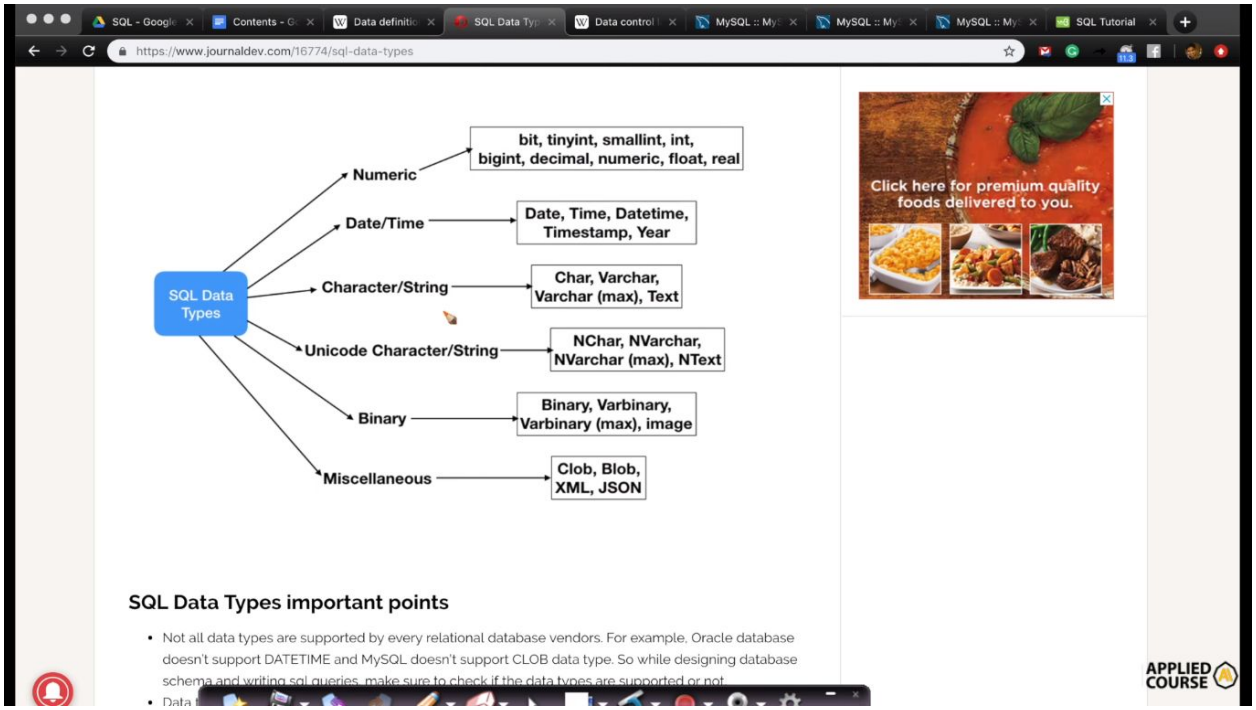
****
#DELETE

DELETE FROM movies WHERE id=412321;

Remove all rows: TRUNCATE TABLE TableName;
Same as selete without a WHERE Clause.

****

DDL:CREATE TABLE



****

Data Definition Language

CREATE TABLE language (id INT PRIMARY, lang VARCHAR(50) NOT NULL);

Datatypes: https://www.journaldev.com/16774/sql-data-types

Constraints: https://www.w3schools.com/sql/sql_constraints.asp

NOT NULL - Ensures that a column cannot have a NULL value
UNIQUE - Ensures that all values in a column are different
PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
FOREIGN KEY - Uniquely identifies a row/record in another table
CHECK - Ensures that all values in a column satisfies a specific condition
DEFAULT - Sets a default value for a column when no value is specified
INDEX - Used to create and retrieve data from the database very quickly

****

DDL:ALTER: ADD, MODIFY, DROP

****

ALTER: ADD, MODIFY, DROP

ALTER TABLE language ADD country VARCHAR(50);

ALTER TABLE language MODIFY country VARCHAR(60);

ALTER TABLE langauge DROP country;

**********D

DDL:DROP TABLE, TRUNCATE, DELETE

****

Removes both the table and all of the data permanently.
DROP TABLE Tablename;

DROP TABLE TableName IF EXISTS;

#https://dev.mysql.com/doc/refman/8.0/en/drop-table.html

TRUNCATE TABLE TableName;

as discussed earlier same as DELETE FROM TableName;

****

Data Control Language: GRANT, REVOKE

****

Data Control Language for DB Admins.

https://en.wikipedia.org/wiki/Data_control_language
https://dev.mysql.com/doc/refman/8.0/en/grant.html
https://dev.mysql.com/doc/refman/8.0/en/revoke.html
****

Learning resources

https://www.w3schools.com/sql/
https://dev.mysql.com/doc/refman/8.0/en/

****