# Scientific Data and Databases

CDS-302/502, SPRING 2025

*Mariia Belaia*

Lecture 4

# Today’s class

I. SQL queries
II. Aggregate functions

# Last class

**I. DDL commands:**

-CREATE TABLE

-DROP TABLE <TableName> - delete the table

-ALTER TABLE

ALTER TABLE *r* ADD *A B* - add attribute A of type B to existing relation r

ALTER TABLE *r* DROP *A* - remove attribute from existing relation r

ALTER TABLE *r* MODIFY *A* NEWTYPE - change attribute type

ALTER TABLE t1 RENAME t2;

**II. DML commands:**

INSERT into <TableName>
values (A1..An);

UPDATE <TableName>
SET attribute=value
WHERE <predicate>

DELETE FROM <RelationName>
WHERE <Predicate>;

Safe update/delete mode:
off: SET sql_safe_updates=0;
on: SET sql_safe_updates=1;

**Integrity constraints include:**

PRIMARY KEY(AttributeName);
FOREIGN KEY(AttributeName) REFERENCING RelationName(AttributeName);
NOT NULL constraint: *attribute$_1$ datatype NOT NULL;*

**In addition:**
DEFAULT command – for specified attribute you can set all values to default.

**Example:**

```
CREATE TABLE movie(
   movieid smallint,
  moviename char(50) NOT NULL
  seeingstatus char(25) DEFAULT "haven't seen"
  PRIMARY KEY(movieid)
);
```

# I. DQL: SQL queries

# DQL: SELECT FROM WHERE

- SQL queries: three fundamental clauses

  SELECT
  FROM
  WHERE

- Result of a query is a **relation**!

Simplest form:

```
SELECT <AttributeName>
FROM <RelationName>
```

pick attribute

```
SELECT * FROM <RelationName>
```

pick all attributes

```
SELECT <AttributeName>
FROM <RelationName>
WHERE <Predicate>;
```

pick attribute for tuples where values satisfy predicate

# DQL: SELECT FROM WHERE

E1. Open and execute the following SQL scripts (in this order):

Execute schema “movieDB”
INSERT data

**Insert from the CSV data format**

1.Use MySQL workbench wizard.

2.Command LOAD DATA INFILE: https://dev.mysql.com/doc/refman/8.0/en/load-data.html).

3.Convert CSV to INSERT statements.

# Single-relation queries

**Single-relation queries**

**Exercise E2:**

1. To display the instance of each relation in a database:

```
SELECT * FROM actor
SELECT * FROM movie
SELECT * FROM mcast
```

**2.** For each actor in actor table, list first name only.

**3.** List first names of actors whose first name starts with m (hint: instead of equality, use c

**4.** List first and last names of actors born before 1950.

**5.** List last names of actors with a name Patricia and born before the year 1950.

Hint: see date operations
https://dev.mysql.com/doc/refman/8.4/en/date-and-time-functions.html

# Single-relation queries

**Sort rows:**

ORDER BY ascending (ASC) or descending (DESC).

Example: SELECT aid ORDER BY ASC;

**To force elimination of duplicates:**

use DISTINCT:

Example: SELECT DISTINCT aid;

You can use **arithmetic operations** in the SELECT clause:

Example: SELECT release_year+2 FROM movie;

**Exercises:**

**2. 7.** List last names of actors vs. list distinct last names of actors in ascending order.

**2. 8.** Example**:** How many years ago was each movie released? List title and number of years.

# Single-relation queries

**Single relation query**

Count all rows in a table:

```
SELECT COUNT(*) FROM <table>;
```

Count rows in a table that satisfy some predicate:

```
SELECT COUNT(*) FROM <table> WHERE <predicate>;
```

Count number of distinct values of  attribute:

```
SELECT COUNT(DISTINCT <attribute>) FROM <table>
```

**10**. How many last names of actors vs. how many distinct last names of actors are there in our database.

**11**. How many actors do play in movies listed in our DB?

**12.** How many movies were released before 1950?

# Single-relation queries

**SET OPERATIONS**

UNION means "or"

INTERSECT means "and" – MySQL doesn't support intersect – use nested subqueries.

EXCEPT means set A minus set B. MySQL does not implement it at all;
Oracle: uses MINUS keyword.
In MySQL: use nested subqueries.

Select… UNION Select …
(adding up rows)

**Exercise E3:**

3.1 List first names of actors born before 1930 or after 1950 (use UNION operation)

3.2 List actors (all info) with a name Margot or George.

# Multiple-relation query

**Multiple-relation query**

```
SELECT COUNT(*)
FROM actors

SELECT COUNT(*)
FROM actors, movies
```

How many tuples (i.e. rows) in actors?
How many tuples in movies?
How many tuples in movies x actors?

**Recall from last class:** Cartesian-product operation – a set of *n*-tuples with a set of *m*-tuples yields a set of "flattened" (*n* + *m*)-tuples

Output is a single tuple (as opposed to a pair of tuples) for each possible pair of tuples – one from $r_1$ and one from $r_2$:

$$r_1 \times r_2 = \{(a_1, a_2, a_n, b_1, b_2 \dots b_m),$$
$$where\ (a_1, a_2, \dots a_n) \in r_1\ and\ (b_1 \dots b_m) \in r_2\}$$

# DQL: SELECT

**Exercise E4:** List actors and movies they played in: first name, last name, and movie title.

Here, we need information from 3 tables: movie, actor, and mcast.
If we were to use cartesian product -> loss of information.

-> use matching condition to only join matching tuples:
WHERE mcast.aid=actor.aid AND movie.movie_id=mcast.movie_id;

# Working with NULL

- Result of any algebraic operation (+,-,x, /) that has null is NULL.
  Example: NULL/12 is NULL
- Comparisons: “NULL>1”
  TRUE?
  FALSE?
- SQL: any comparison involving NULL returns NULL. Note: We can ask whether value is NULL

NULL is a third logical value (in addition to TRUE and FALSE).

**Definitions of Boolean operations are extended to include the value unknown:**

1. **AND:**
   true **AND** unknown **= unknown**
   false **AND** unknown **= false**
   unknown **AND** unknown **= unknown**
2. **OR:**
   true **OR** unknown = true; false **OR** unknown = unknown
   unknown **OR** unknown=unknown
3. **NOT**

**NOT** unknown=unknown

# Working with NULL

| x | y | x AND y | x OR y |
|---|---|---|---|
| false | false | false | false |
| true | false | false | true |
| false | true | false | true |
| true | true | true | true |
| true | null | null | true |
| false | null | false | null |
| null | null | null | null |

# Working with NULL

is NULL command means “equals NULL”, you can use it in WHERE clause

**Exercise 5**:

1. List titles of movies where we ado not know the imdbrank.
2. List titles of movies where imdbrank is not NULL.

# II. Aggregate functions

**Commands:**

**Avg()**
**Min()**
**Max()**
**Sum()**
**Count()**

Example: SELECT AVG(attribute_name)

**Exercise E6:**

6.1 Find average imdbrank of movies.
6.2 Find smallest imdbrank of movies.
6.3 Find largest imdbrank of movies.

We can also use command “as” to save new result!

6.4 Example:

```
SELECT MAX(…) AS largest_rank
FROM …;
```

# II. Aggregate functions

**Aggregate functions and NULL?**

6.5 Find average imdbrank using AVG() and average imdbrank using summation of ranks/total number of movie.

# II. Aggregation with grouping

**Group by - tuples with same value on attributes in the group by statement are placed in one group.**

```
SELECT <aggregate-function>(<attribute1>)
FROM <relation>
GROUP BY <attribute2>;
```

Example: for each name, find average age

```
SELECT name, avg(age)
FROM database
GROUP BY name;
```

| name | age |
|---|---|
| Ana | 12 |
| Ana | 16 |
| Leo | 30 |
| Leo | 40 |
| Leo | 50 |

database

| name | Average age |
|---|---|
| Ana | 14 |
| Leo | 40 |

output

# II. Aggregation with grouping

**Exercise E7:**

7.1 Find average imdbrank for each movie genre.

# II. Aggregation with grouping

**Having – condition imposed on each group**

```
SELECT <attribute(s)>
FROM <relation>
WHERE <predicate>
GROUP BY <attribute>
HAVING <group predicate> ;
```

**Evaluation sequence:**

1. **FROM** - get relation
2. **WHERE** – predicate is applied to the relation
3. **GROUP BY** - tuples that satisfy WHERE clause are placed into groups
4. **HAVING** clause is applied to each group.
5. **SELECT** clause uses groups to generate tuples.

# II. Aggregation with grouping

**Example:**

7.2 Find average imdbrank for each movie genre except comedy.

7.3 For each movie, find number of actors who play in it.

7.4 For each movie, find number of actors who play in it. Only list movies with at least 2 actors.

# Please practice writing SELECT statements!

**Go to:**

https://sqlzoo.net

**Today's material fits the following quizzes:**

SELECT basics
SELECT from world
SELECT from nobel
SUM and COUNT
NULL
UNION
GROUP BY

**Last class's material fits the following quizzes:**

CREATE TABLE
DROP
ALTER
UPDATE
DELETE

This is not an assignment, but a suggestion