SQL

Intro

- "S.Q.L." or "sequel"
- Supported by all major commercial DBMS
- Standardized
- Interactive via GUI or command line, or embedded in programs (e.g., in Python programs)
- Declarative

Terminology

- Data Definition Language (DDL)
  - Create Table
  - Drop Table
  - Indexes
- **Data Manipulation language (DML)**
  - **Select**
  - Insert
  - Delete
  - Update
- Other Commands
  - constraints, views, triggers etc.

Choosing a database in MySQL

- SHOW DATABASES;
  - Lists the available databases in MySQL
- USE <database>;
  - Chooses which database to work with
  - *Example: USE imdb;*
- SHOW TABLES;
  - Lists the tables in the database

The SELECT statement

SELECT $\boxed{A_1, A_2, A_3, \ldots, A_n}$ ⟵ What to return

FROM $\boxed{T_1, T_2, \ldots\ T_m}$ ⟵ Tables (or “relations”)

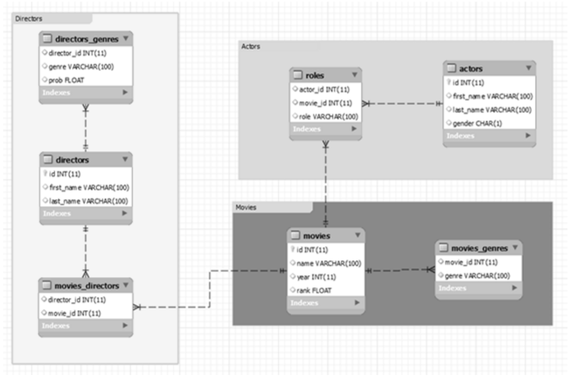
WHERE $\boxed{\text{condition}}$ ⟵ Combines/Filters

ORDER BY A_1 [ASC|DESC], A_2 [ASC|DESC] ⟵ Sort

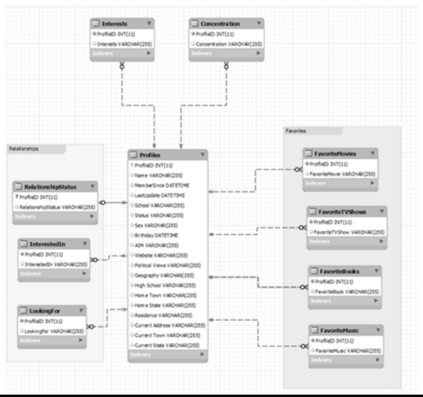
The result of a query is a relation.

Note that a table is always a relation, but not vice versa.

Example 1: IMDB



Example 2: Facebook



Practice queries: IMDB

1. Find all movie titles with id less than 100.
2. Find all information about movies that were released before 1895 (excl)
3. Find all information about movies that were released before 1895 and after 2006 (inclusive)
4. Find all information about movies released between 1895 and 1898 (excl)
5. Find all information about ***actresses*** who are have first name Skyler
6. Find the director ID of Steven Spielberg
7. Find the director IDs and the first and last names of directors with the last name Spielberg and Hitchcock
8. Find all genres of films and the corresponding probabilities for the director ID that corresponds to Steven Spielberg. Sort the results by probability.
9. Find the id of the movie Schindler's List.
10. List all the roles for the movie with id 290070. Sort them alphabetically

The DISTINCT statement

SELECT **DISTINCT** $\boxed{A_1, A_2, A_3, \ldots, A_n}$ ← What to return
FROM $\boxed{T_1, T_2, \ldots T_m}$ ← Tables (or "relations")
WHERE $\boxed{\text{condition}}$ ← Combines/Filters
ORDER BY A_1 [ASC|DESC], A_2 [ASC|DESC] ← Sort

Used to eliminate duplicates in the results

Practice queries: Facebook

1. Find all names of students from the Profiles table
2. Get the names and sex of all liberal students
3. Get the High Schools of the students in the database
    - Need to use “backticks” (`) for attribute names with space in them
4. Find all the possible political views, eliminating duplicate entries
5. Find all possible relationship statuses
6. Find all possible values for the “status” attribute in Profiles
7. Find all possible values for the “Residence” attribute in Profiles, eliminating duplicates
8. *Find all students living in Palladium*
9. *Find all students who attended Stuyvesant*

LIKE

- LIKE allows a (limited) regular expression query
    - “_” to match any single character
    - “%” to match an arbitrary number of characters

Example: Find all names that start with B

```
SELECT *
FROM Profiles
WHERE name LIKE ‘B%';
```

Example: Find all names with exactly 10 characters

```
SELECT *
FROM Profiles
WHERE name LIKE ‘__________';
```

REGEXP

- REGEXP allows a standard regular expression query

Example: Find all names that contain a digit

```
SELECT *
FROM Profiles
WHERE name REGEXP '[0-9]+'
```

Renaming columns: The “AS” clause

- Instead of using the existing attribute name, we can change it using the “AS” clause

```
SELECT      A1 AS name1, A2 AS name2 ...
FROM        T1, T2, … Tm                    ⟵ Tables (or “relations”)
WHERE       condition
ORDER BY A1 [ASC|DESC], A2 [ASC|DESC]
```

Example: Find the names of all students who attended Stuyvesant and rename the “High School” column to HS and the rename the “name” column to “StudentName”

The NULL value

- When columns do not have a value, they are assigned a “NULL” value, which is a special way that SQL handles the “empty”
  - *Notice: NULL is **not** identical to “” (empty string). In practice, you may see both, although NULL is always superior choice*
- To check if something is NULL you use the expression: “**attr IS NULL**”
  - Example: Find all students that have not listed their birthday

```
SELECT ProfileID
FROM Profiles
WHERE Birthday IS NULL
```

- Similarly, you use “attr IS NOT NULL” is you want only results that have non-NULL values

Practice queries

- Write down three queries that you would like to answer
  (Ensure that the information exists in a ***single*** table, for now)
- Let’s answer them in class…

Group By

Basic aggregation functions

Operator	*Description*
max	Row with maximum value
min	Row with minimum value
sum	Sums values of selected rows
count	Counts the number of rows
avg	Estimates the average of selected rows

Group by

count(*), sum(*), avg(*), min, max:
Applied to groups!!!!

SELECT A_1, $\boxed{\text{Aggregation Function}}$
FROM T_1, T_2, … T_m
WHERE condition
Group By A_1

Note: Whatever attribute you select (in this case A_1) **<u>must</u>** appear in the group by clause.

Group by Toy Example

Table1

Student_id	Class	Grade
1	Algebra	19
2	Algebra	16
3	Algebra	20
2	Analysis	18
2	Physics	13
1	Analysis	17
1	Physics	19
1	History	14

```
SELECT      Student_id, count(*)
 FROM       Table1
GROUP BY    Student_id
```

$\downarrow$

Student_id	Count
1	4
2	3
3	1

Group by Toy Example

Table1

Student_id	Class	Grade
1	Algebra	19
2	Algebra	16
3	Algebra	20
2	Analysis	18
2	Physics	13
1	Analysis	17
1	Physics	19
1	History	14

```
SELECT      Student_id, avg(Grade)
 FROM       Table1
GROUP BY    Student_id
```

$\downarrow$

Student_id	Avg(grade)
1	(19+17+19+14)/4
2	(16+18+13)/3
3	20

Group by Toy Example

Table1

Student_id	Class	Grade
1	Algebra	19
2	Algebra	16
3	Algebra	20
2	Analysis	18
2	Physics	13
1	Analysis	17
1	Physics	19
1	History	14

```
SELECT      Class, avg(Grade)
 FROM       Table1
GROUP BY    Class
```

$\downarrow$

Class	Avg(grade)
Algrebra	(19+16+20)/3
Analysis	(18+17)/2
History	14
Physics	(19+13)/2

Having

```
SELECT     A1, Aggregation Function
FROM       T1, T2, … Tm
WHERE      condition
GROUP BY   A1
```

HAVING Aggregation Function **Condition**

Group by Toy Example

Table1

Student_id	*Class*	*Grade*
1	Algebra	19
2	Algebra	16
3	Algebra	20
2	Analysis	18
2	Physics	13
1	Analysis	17
1	Physics	19
1	History	14

```
SELECT     Class, avg(Grade)
FROM       Table1
GROUP BY   Class
HAVING     avg(Grade) > 15
```

$\downarrow$

Class	*Avg(grade)*
Algrebra	(19+16+20)/3
Analysis	(18+17)/2
Physics	(19+13)/2

History class is *not* included in the result because its average is 14 (less than 15)

Group by Toy Example

Table1

Student_id	*Class*	*Grade*
1	Algebra	19
2	Algebra	16
3	Algebra	20
2	Analysis	18
2	Physics	13
1	Analysis	17
1	Physics	19
1	History	14

```
SELECT     Student_id, count(*)
FROM       Table1
GROUP BY   Student_id
HAVING     count(*) > 2
```

$\downarrow$

Student_id	*count*
1	4
2	3

Student with id=3 is not included in the results
because he/she is taking only one class

Differences of WHERE and HAVING

- WHERE applies to rows, **before** computing the aggregate
- HAVING applies to aggregate value only

Aggregation practice queries: IMDB

- Find the number of movies for each director
- Rank directors by the number of movies they directed
- Find the number of actors in each movie
- Find the movies with more than 100 actors
- Find the most popular genres (based on the number of movies)
- Find the average rank of the movies in the database, per year of release

Aggregation practice queries: Facebook

- List the most number of males and females
- List the number of students for each political view
- List the number of males and female students for each political view
- List the number of students per each birth year
  - Use the YEAR(date) function to get the year value from a datetime column
  - List only years that have at least 10 students
- Find the most popular TV Shows and Books
- Find the number of students in various relationship statuses
- Find the most popular majors (concentration)

Joins

A SQL join clause combines records from two or more tables in a database. (Wikipedia)

Joins

Student Has Class

Student_id	Class Id	Grade
1	1	19
2	1	16
3	1	20
2	2	18
1	3	19
1	4	14

Class

Class_id	Class
1	Algebra
2	Analysis
3	Physics
4	History

Question: Find the class name for all the classes that each student is taking.

Answer 1:
```
select student_id, class
from Student_Has_Class s, Class c
where c.class_id = s.class_id
```

Answer 2:
```
select student_id, class
from Student_Has_Class s inner join Class c
on c.class_id = s.class_id
```

Result

Student_id	Class Id	Grade
1	1	19
2	1	16
3	1	20
2	2	18
1	3	19
1	4	14

Inner Join on Class_id

Class_id	Class
1	Algebra
2	Analysis
3	Physics
4	History

↓

Student_id	Class
1	Algebra
2	Algebra
3	Algebra
2	Analysis
1	Physics
1	History

Outer Join

Student_id	Class Id	Grade
1	1	19
2	1	16
3	1	20
2	2	18
1	3	19
1	4	14
3	6	17

Class_id	Class
1	Algebra
2	Analysis
3	Physics
4	History

Question: Find the class name for all the classes that each student is taking.

Note: No class with id=6 exists in the Class table

select student_id, class
from Student_Has_Class s $\boxed{\text{left}}$ outer join Class c
on c.class_id = s.class_id

Refers to the "left" table: Student_Has_Class

Outer Join

Student_id	Class Id	Grade
1	1	19
2	1	16
3	1	20
2	2	18
1	3	19
1	4	14
3	6	17

Left outer Join on Class_id

Class_id	Class
1	Algebra
2	Analysis
3	Physics
4	History

Student_id	Class
1	Algebra
2	Algebra
3	Algebra
2	Analysis
1	Physics
1	History
3	NULL

A left outer join returns all the values from an inner join plus all values in the left table that do not match to the right table.

Joins Practice Queries

- List all the actors that worked with Steven Spielberg
- Compute the average rank for the movies directed by Steven Spielberg
- List the movies of Brad Pitt
  - Exclude the movies where he plays himself
  - Compute the average rank for his movies
- List the genre of the movies where Sean Connery appears, and rank them in descending order by count.
  - Exclude the movies where Sean Connery plays himself
- Compute the average rank for the movies of each actor and rank the actors in descending order based on that rank

Outer joins
- List all the actors that have not worked with Francis Ford Coppola
  - *Important: Understand why we need an outer join here*

Subqueries

Subqueries / FROM

SELECT $\boxed{A_1, A_2, A_3, \ldots, A_n}$ ← What to return
FROM $\boxed{T_1, T_2, \ldots T_m}$ ← Tables (or queries)
WHERE $\boxed{\text{condition}}$ ← Combines/Filters

The table can be directly replaced by another query, placed within parentheses

SELECT $\boxed{A_1, A_2, A_3, \ldots, A_n}$ ← What to return
FROM $\boxed{T_1, (\text{SELECT * FROM...}}$),.. ← Tables (or queries)
WHERE $\boxed{\text{condition}}$ ← Combines/Filters

Subqueries / WHERE

The "IN" clause allows us to check if an attribute appears within a list returned by another SQL query

SELECT $\boxed{A_1, A_2, A_3, \ldots, A_n}$
FROM $\boxed{T_1, T_2, \ldots T_m}$
WHERE $\boxed{A_j \text{ attribute IN (SELECT attr FROM ....)}}$

Subqueries Practice Queries

- Find the average number of movies directed by each director
- Find the average number of movies played by each actor
- Find the maximum number of genres associated with a movie

- Compare the favorite books of liberal and conservative students
    - Subquery 1: Get the list of books (with counts) of all liberal students
    - Subquery 2: Get the list of books (with counts) of all conservative students
    - Join the two on book name and compare counts

Saving Queries: CREATE VIEW

We can save the results of a query in order to reuse the results easier, without having to always rewrite the subquery using the "CREATE VIEW" command

Example:

```
CREATE VIEW StuyHS AS
        SELECT id, name AS StudentName, `High School` AS HS
        FROM Profiles
        WHERE `High School` LIKE 'Stuy%'
        ORDER BY StudentName
```

Comparison Operators

Operator	*Description*
=	equals
<>	is not equal to
!=	>>
<	less than
>	greater than
AND	logical and
OR	logical or
NOT	logical not

Other operators

SQL	*Description*
as	used to change the name of a column in the result
distinct	no duplicate rows
order by column(s)	sorts by column(s) in ascending order
order by .. desc	sorts by column(s) in descending order
*	select all columns
like '%pattern_'	$: any sequence of characters _: any single character
attribute is null	rows that have null values for the specific attribute
is not null	rows that have not null values for the specific attribute
between this and that	between **this** value and **that** value
in	set membership
limit n	fetches only the top n rows from the database