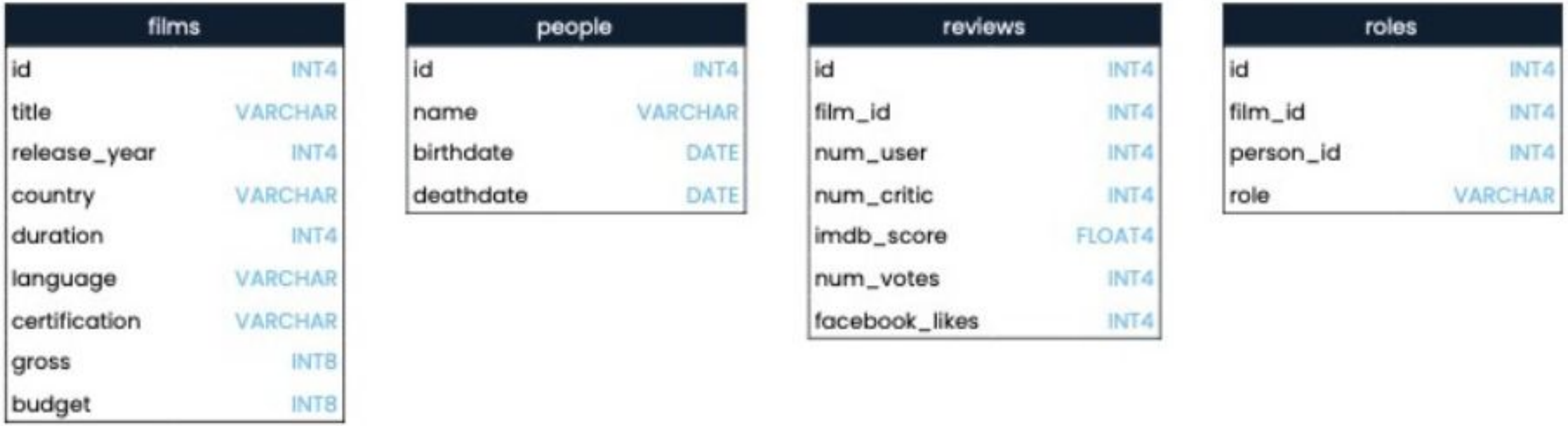
Querying a database

Selecting Data

Our films database



films	
id	INT4
title	VARCHAR
release_year	INT4
country	VARCHAR
duration	INT4
language	VARCHAR
certification	VARCHAR
gross	INT8
budget	INT8

people	
id	INT4
name	VARCHAR
birthdate	DATE
deathdate	DATE

reviews	
id	INT4
film_id	INT4
num_user	INT4
num_critic	INT4
imdb_score	FLOAT4
num_votes	INT4
facebook_likes	INT4

roles	
id	INT4
film_id	INT4
person_id	INT4
role	VARCHAR

COUNT()

- `COUNT()`
- Counts the number of records with a value in a field
- Use an alias for clarity

```
SELECT COUNT(birthdate) AS count_birthdates
FROM people;
```

```
|count_birthdates|
|----------------|
|6152            |
```

COUNT() multiple fields

```sql
SELECT COUNT(name) AS count_names, COUNT(birthdate) AS count_birthdates
FROM people;
```

```
|count_names|count_birthdates|
|-----------|----------------|
|6397       |6152            |
```

Using * with COUNT()

- `COUNT(field_name)` counts values in a field
- `COUNT(*)` counts records in a table
- `*` represents all fields

```
SELECT COUNT(*) AS total_records
FROM people;
```

```
|total_records|
|-------------|
|8397         |
```

DISTINCT

- `DISTINCT` removes duplicates to return only unique values

```
SELECT language
FROM films;
```

```
|language |
|---------|
|Danish   |
|Danish   |
|Greek    |
|Greek    |
|Greek    |
```

- Which languages are in our `films` table?

```
SELECT DISTINCT language
FROM films;
```

```
|language |
|---------|
|Danish   |
|Greek    |
```

COUNT() with DISTINCT

- Combine `COUNT()` with `DISTINCT` to count unique values

```
SELECT COUNT(DISTINCT birthdate) AS count_distinct_birthdates
FROM people;
```

```
|count_distinct_birthdates|
|-------------------------|
|5398                     |
```

- `COUNT()` includes duplicates
- `DISTINCT` excludes duplicates

Query execution

INTERMEDIATE SQL

Order of execution

- SQL is not processed in its written order

```
-- Order of execution
SELECT name
FROM people
LIMIT 10;
```

- `LIMIT` limits how many results we return
- Good to know processing order for debugging and aliasing
- Aliases are declared in the `SELECT` statement

1. From people
2. SELECT name
3. LIMIT 10;

Debugging SQL

```sql
SELECT nme
FROM people;
```

```
field "nme" does not exist
LINE 1: SELECT nme
               ^
HINT:  Perhaps you meant to reference the field "people.name".
```

- Misspelling
- Incorrect capitalization
- Incorrect or missing punctuation

Comma errors

- Look out for comma errors!

```
SELECT title, country duration
FROM films;
```

```
syntax error at or near "duration"
LINE 1: SELECT title, country duration
                      ^
```

Keyword errors

```sql
SELCT title, country, duration
FROM films;
```

```
syntax error at or near "SELCT"
LINE 1: SELCT title, country, duration
        ^
```

Final note on errors

Most common errors:

- Misspelling
- Incorrect capitalization
- Incorrect or missing punctuation, especially commas

Learn by making mistakes

SQL style

INTERMEDIATE SQL

SQL formatting

- Formatting is not required
- But lack of formatting can cause issues

```
select title, release_year, country from films limit 3
```

```
|title                                        |release_year|country|
|---------------------------------------------|------------|-------|
|Intolerance: Love's Struggle Throughout the Ages|1916        |USA    |
|Over the Hill to the Poorhouse               |1920        |USA    |
|The Big Parade                               |1925        |USA    |
```

Best practices

```
SELECT title, release_year, country
FROM films
LIMIT 3;
```

```
|title                                        |release_year|country|
|---------------------------------------------|------------|-------|
|Intolerance: Love's Struggle Throughout the Ages|1916        |USA    |
|Over the Hill to the Poorhouse               |1920        |USA    |
|The Big Parade                               |1925        |USA    |
```

- Capitalize keywords
- Add new lines

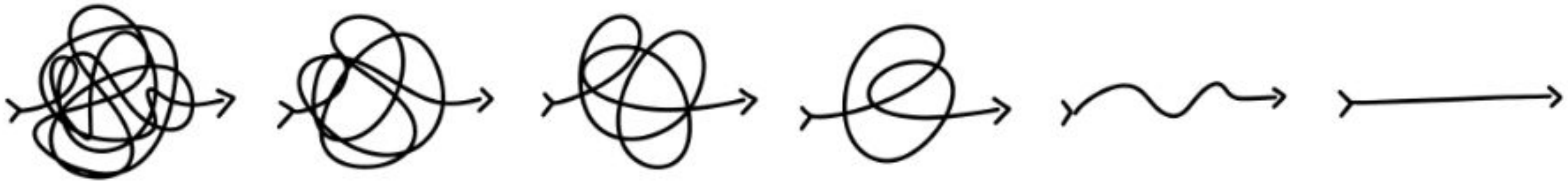
Style guides

```sql
SELECT
    title,
    release_year,
    country
FROM films
LIMIT 3;
```

```
|title                                          |release_year|country|
|-----------------------------------------------|------------|-------|
|Intolerance: Love's Struggle Throughout the Ages|1916        |USA    |
|Over the Hill to the Poorhouse                 |1920        |USA    |
|The Big Parade                                 |1925        |USA    |
```

Style guides

Holywell's style guide: **https://www.sqlstyle.guide/**



Write clear and readable code

Semicolon

```
SELECT title, release_year, country
FROM films
LIMIT 3;
```

- Best practice
- Easier to translate between SQL flavors
- Indicates the end of a query

Dealing with non-standard field names

- `release year` instead of `release_year`
- Put non-standard field names in double-quotes

```
SELECT title, "release year", country
FROM films
LIMIT 3;
```

Why do we format?

- Easier collaboration
- Clean and readable
- Looks professional
- Easier to understand
- Easier to debug

Filtering numbers

INTERMEDIATE SQL

WHERE

- `WHERE` filtering clause

WHERE

```
WHERE color = 'green'
```

WHERE with comparison operators

```sql
SELECT title
FROM films
WHERE release_year > 1960;
```

```
|title                 |
|----------------------|
|Judgment at Nuremberg|
|Pocketful of Miracles|
|The Hustler           |
|The Misfits           |
...
```

Comparison operators

```sql
SELECT title
FROM films
WHERE release_year < 1960;
```

```
|title                                        |
|---------------------------------------------|
|Intolerance:Love's Struggle Throughout the Ages|
|Over the Hill to the Poorhouse               |
|The Big Parade                               |
|Metropolis                                   |
...
```

Comparison operators

```sql
SELECT title
FROM films
WHERE release_year <= 1960;
```

```
|title                                       |
|--------------------------------------------|
|Intolerance:Love's Struggle Throughout the Ages|
|Over the Hill to the Poorhouse              |
|The Big Parade                              |
|Metropolis                                  |
...
```

Comparison operators

```sql
SELECT title
FROM films
WHERE release_year = 1960;
```

```
|title        |
|-------------|
|Elmer Gantry |
|Psycho       |
|The Apartment|
```

Comparison operators

```
SELECT title
FROM films
WHERE release_year <> 1960;
```

```
|title                                     |
|------------------------------------------|
|Intolerance:Love's Struggle Throughout the Ages|
|Over the Hill to the Poorhouse            |
|The Big Parade                            |
|Metropolis                                |
...
```

Comparison operators

- `>` Greater than or after
- `<` Less than or before
- `=` Equal to
- `>=` Greater than or equal to
- `<=` Less than or equal to
- `<>` Not equal to

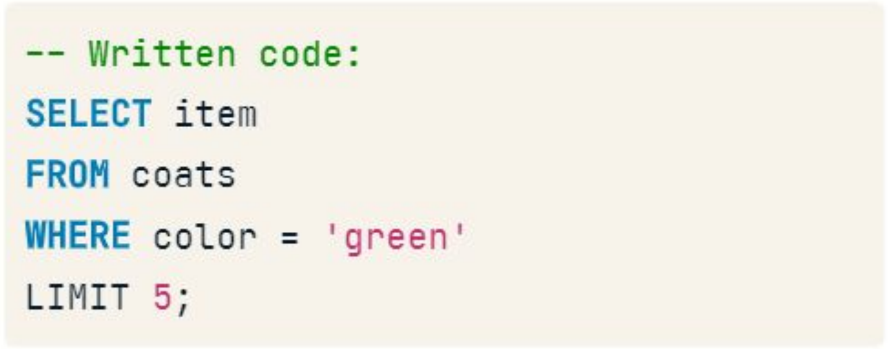
WHERE with strings

- Use single-quotes around strings we want to filter

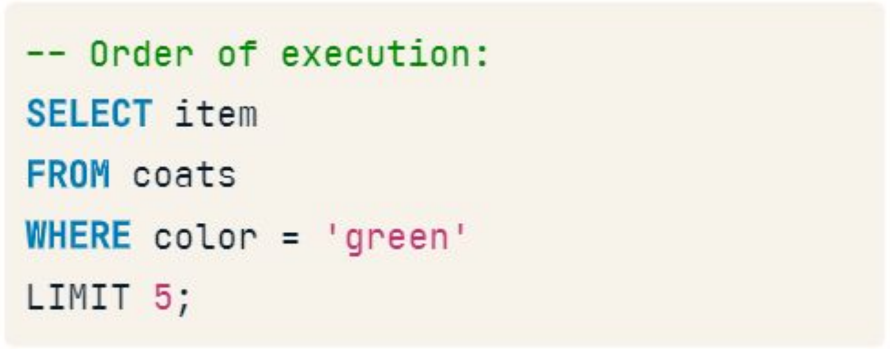
```
SELECT title
FROM films
WHERE country = 'Japan';
```

```
|title            |
|-----------------|
|Seven Samurai    |
|Tora! Tora! Tora!|
|Akira            |
|Madadayo         |
|Street Fighter   |
```

Order of execution



```sql
-- Written code:
SELECT item
FROM coats
WHERE color = 'green'
LIMIT 5;
```



```sql
-- Order of execution:
SELECT item
FROM coats
WHERE color = 'green'
LIMIT 5;
```

1. From coats
2. Where color = ‘green’
3. SELECT item
4. LIMIT 5;

Multiple criteria

INTERMEDIATE SQL

Multiple criteria

Multiple criteria

Multiple criteria

Multiple criteria

- `OR`, `AND`, `BETWEEN`

```
SELECT *
FROM coats
WHERE color = 'yellow' OR length = 'short';
```

```
SELECT *
FROM coats
WHERE color = 'yellow' AND length = 'short';
```

```
SELECT *
FROM coats
WHERE buttons BETWEEN 1 AND 5;
```

OR operator

- Use `OR` when you need to satisfy at least one condition

OR operator

- Correct:

```
SELECT title
FROM films
WHERE release_year = 1994
    OR release_year = 2000;
```

```
|title                    |
|-------------------------|
|3 Ninjas Kick Back       |
|A Low Down Dirty Shame   |
|Ace Ventura:Pet Detective|
...
```

- Invalid:

```
SELECT title
FROM films
WHERE release_year = 1994 OR 2000;
```

```
argument of OR must be type boolean,
not type integer
LINE 3: WHERE release_year = 1994
OR 2000;
   ^
```

AND operator

- Use `AND` if we need to satisfy all criteria
- Correct:

```
SELECT title
FROM films
WHERE release_year > 1994
    AND release_year < 2000;
```

```
|title                       |
|----------------------------|
|Ace Ventura:When Nature Calls|
|Apollo 13                   |
|Assassins                   |
|Babe                        |
...
```

- Invalid:

```
SELECT title
FROM films
WHERE release_year > 1994 AND < 2000;
```

```
syntax error at or near "[removed]
1994 AND < 2000;
     ^
```

AND, OR

- Filter films released in 1994 or 1995, and certified PG or R
- Enclose individual clauses in parentheses

```sql
SELECT title
FROM films
WHERE (release_year = 1994 OR release_year = 1995)
    AND (certification = 'PG' OR certification = 'R');
```

```
|title                 |
|----------------------|
|3 Ninjas Kick Back    |
|A Low Down Dirty Shame|
|Baby's Day Out        |
|Beverly Hills Cop III |
...
```

BETWEEN, AND

```sql
SELECT title
FROM films
WHERE release_year >= 1994
    AND release_year <= 2000;
```

```sql
SELECT title
FROM films
WHERE release_year
    BETWEEN 1994 AND 2000;
```

```
|title                    |
|-------------------------|
|3 Ninjas Kick Back       |
|A Low Down Dirty Shame   |
|Ace Ventura:Pet Detective|
|Baby's Day Out           |
...
```

```
|title                    |
|-------------------------|
|3 Ninjas Kick Back       |
|A Low Down Dirty Shame   |
|Ace Ventura:Pet Detective|
|Baby's Day Out           |
...
```

BETWEEN, AND, OR

```
SELECT title
FROM films
WHERE release_year
BETWEEN 1994 AND 2000 AND country='UK';
```

```
|title                      |
|---------------------------|
|Four Weddings and a Funeral|
|The Hudsucker Proxy        |
|Dead Man Walking           |
|GoldenEye                  |
...
```

Filtering text

INTERMEDIATE SQL

Filtering text

- `WHERE` can also filter text

```sql
SELECT title
FROM films
WHERE country = 'Japan';
```

```
|title            |
|-----------------|
|Seven Samurai    |
|Tora! Tora! Tora!|
|Akira            |
|Madadayo         |
|Street Fighter   |
```

Filtering text

- `WHERE` can also filter text

```sql
SELECT title
FROM films
WHERE country = 'Japan';
```

```
|title            |
|-----------------|
|Seven Samurai    |
|Tora! Tora! Tora!|
|Akira            |
|Madadayo         |
|Street Fighter   |
```

Filtering text

- Filter a pattern rather than specific text
- `LIKE`
- `NOT LIKE`
- `IN`

LIKE

- Used to search for a pattern in a field

% match zero, one, or many characters

```
SELECT name
FROM people
WHERE name LIKE 'Ade%';
```

```
|name         |
|-------------|
|Adel Karam   |
|Adelaide Kane|
|Aden Young   |
```

_ match a single character

```
SELECT name
FROM people
WHERE name LIKE 'Ev_';
```

```
|name         |
|-------------|
|Eve          |
```

- `Ev_ Mendes`

NOT LIKE

```sql
SELECT name
FROM people;
```

```
|name              |
|------------------|
|50 Cent           |
|A. Michael Baldwin|
|A. Raven Cruz     |
|A.J. Buckley      |
|A.J. DeLucia      |
...
```

```sql
SELECT name
FROM people
WHERE name NOT LIKE 'A.%';
```

```
|name         |
|-------------|
|50 Cent      |
|Aaliyah      |
|Aaron Ashmore|
|Aaron Hann   |
...
```

Wildcard position

```sql
SELECT name
FROM people
WHERE name LIKE '%r';
```

```
|name          |
|--------------|
|A.J. Langer   |
|Aaron Schneider|
|Aaron Seltzer |
|Abigail Spencer|
...
```

```sql
SELECT name
FROM people
WHERE name LIKE '__t%';
```

```
|name                   |
|-----------------------|
|Aitana Sánchez-Gijón   |
|Anthony 'Critic' Campos|
|Anthony Bell           |
|Anthony Burrell        |
...
```

WHERE, OR

```sql
SELECT title
FROM films
WHERE release_year = 1920
OR release_year = 1930
OR release_year = 1940;
```

```
|title                        |
|-----------------------------|
|Over the Hill to the Poorhouse|
|Hell's Angels                |
|Boom Town                    |
...
```

WHERE, IN

```
SELECT title
FROM films
WHERE release_year IN (1920, 1930, 1940);
```

```
|title                        |
|-----------------------------|
|Over the Hill to the Poorhouse|
|Hell's Angels                |
|Boom Town                    |
...
```

WHERE, IN

```sql
SELECT title
FROM films
WHERE country IN ('Germany', 'France');
```

```
|title        |
|-------------|
|Metropolis   |
|Pandora's Box|
|The Train    |
...
```

NULL values

INTERMEDIATE SQL

Missing values

- `COUNT(field_name)` includes only non-missing values
- `COUNT(*)` includes missing values

`null`

- Missing values:
  - Human error
  - Information not available
  - Unknown

null

```sql
SELECT COUNT(*) AS count_records
FROM people;
```

```
|count_records|
|-------------|
|8397         |
```

```sql
SELECT *
FROM people;
```

```
|id|name              |birthdate |deathdate|
|--|------------------|----------|---------|
|1 |50 Cent           |1975-07-06|null     |
|2 |A. Michael Baldwin|1963-04-04|null     |
|3 |A. Raven Cruz     |null      |null     |
...
```

IS NULL

```sql
SELECT name
FROM people
WHERE birthdate IS NULL;
```

```
|name         |
|-------------|
|A. Raven Cruz|
|A.J. DeLucia |
|Aaron Hann   |
...
```

IS NOT NULL

```
SELECT COUNT(*) AS no_birthdates
FROM people
WHERE birthdate IS NULL;
```

```
|no_birthdates|
|-------------|
|2245         |
```

```
SELECT COUNT(name) AS count_birthdates
FROM people
WHERE birthdate IS NOT NULL;
```

```
|count_birthdates|
|----------------|
|6152            |
```

COUNT() vs IS NOT NULL

```
SELECT
   COUNT(certification)
   AS count_certification
FROM films;
```

```
|count_certification|
|-------------------|
|4666               |
```

```
SELECT
   COUNT(certification)
   AS count_certification
FROM films
WHERE certification IS NOT NULL;
```

```
|count_certification|
|-------------------|
|4666               |
```

NULL put simply

- `NULL` values are missing values
- Very common
- Use `IS NULL` or `IS NOT NULL` to:
  - Identify missing values
  - Select missing values
  - Exclude missing values

Aggregate Functions

Summarizing data

INTERMEDIATE SQL

Summarizing data

- Aggregate functions return a single value

Aggregate functions

`AVG()` , `SUM()` , `MIN()` , `MAX()` , `COUNT()`

```sql
SELECT AVG(budget)
FROM films;
```

```
|avg             |
|----------------|
|39902826.2684...|
```

```sql
SELECT SUM(budget)
FROM films;
```

```
|sum         |
|------------|
|181079025606|
```

Aggregate functions

```sql
SELECT MIN(budget)
FROM films;
```

```sql
SELECT MAX(budget)
FROM films;
```

```
|min|
|---|
|218|
```

```
|max        |
|-----------|
|12215500000|
```

Non-numerical data

Numerical fields only

- `AVG()`
- `SUM()`

Various data types

- `COUNT()`
- `MIN()`
- `MAX()`

Non-numerical data

`MIN()` <-> `MAX()`

Minimum <-> Maximum

Lowest <-> Highest

A <-> Z

1715 <-> 2022

0 <-> 100

Non-numerical data

```sql
SELECT MIN(country)
FROM films;
```

```
|min        |
|-----------|
|Afghanistan|
```

```sql
SELECT MAX(country)
FROM films;
```

```
|max         |
|------------|
|West Germany|
```

Aliasing when summarizing

```
SELECT MIN(country)
FROM films;
```

```
|min        |
|-----------|
|Afghanistan|
```

```
SELECT MIN(country) AS min_country
FROM films;
```

```
|min_country|
|-----------|
|Afghanistan|
```

Summarizing subsets

INTERMEDIATE SQL

Using WHERE with aggregate functions

```sql
SELECT AVG(budget) AS avg_budget
FROM films
WHERE release_year >= 2010;
```

```
|avg_budget          |
|--------------------|
|41072235.18324607...|
```

Using WHERE with aggregate functions

```
SELECT SUM(budget) AS sum_budget
FROM films
WHERE release_year = 2010;
```

```
|sum_budget|
|----------|
|8942365000|
```

```
SELECT MIN(budget) AS min_budget
FROM films
WHERE release_year = 2010;
```

```
|min_budget|
|----------|
|65000     |
```

Using WHERE with aggregate functions

```
SELECT MAX(budget) AS max_budget
FROM films
WHERE release_year = 2010;
```

```
|max_budget|
|----------|
|600000000 |
```

```
SELECT COUNT(budget) AS count_budget
FROM films
WHERE release_year = 2010;
```

```
|count_budget|
|------------|
|194         |
```

ROUND()

- Round a number to a specified decimal

```
ROUND(number_to_round, decimal_places)
```

```
SELECT AVG(budget) AS avg_budget
FROM films
WHERE release_year >= 2010;
```

```
|avg_budget          |
|-------------------|
|41072235.18324607...|
```

```
SELECT ROUND(AVG(budget), 2) AS avg_budget
FROM films
WHERE release_year >= 2010;
```

```
|avg_budget |
|-----------|
|41072235.18|
```

ROUND() to a whole number

```
SELECT ROUND(AVG(budget)) AS avg_budget
FROM films
WHERE release_year >= 2010;
```

```
|avg_budget|
|----------|
|41072235  |
```

```
SELECT ROUND(AVG(budget), 0) AS avg_budget
FROM films
WHERE release_year >= 2010;
```

```
|avg_budget|
|----------|
|41072235  |
```

ROUND() using a negative parameter

```
SELECT ROUND(AVG(budget), -5) AS avg_budget
FROM films
WHERE release_year >= 2010;
```

```
|avg_budget|
|----------|
|41100000  |
```

- Numerical fields only

Aliasing and arithmetic

INTERMEDIATE SQL

Arithmetic

`+` , `-` , `*` , and `/`

```
SELECT (4 + 3);
```

```
|7|
```

```
SELECT (4 * 3);
```

```
|12|
```

```
SELECT (4 - 3);
```

```
|1|
```

```
SELECT (4 / 3);
```

```
|1|
```

Arithmetic

```
SELECT (4 / 3);
```

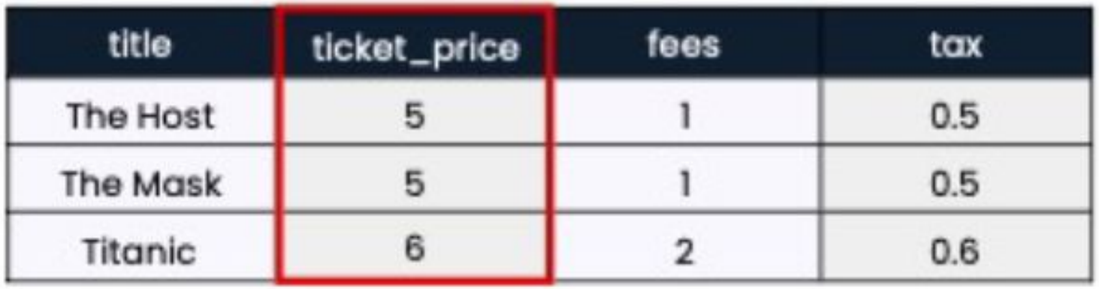
```
|1|
```

```
SELECT (4.0 / 3.0);
```

```
|1.333...|
```

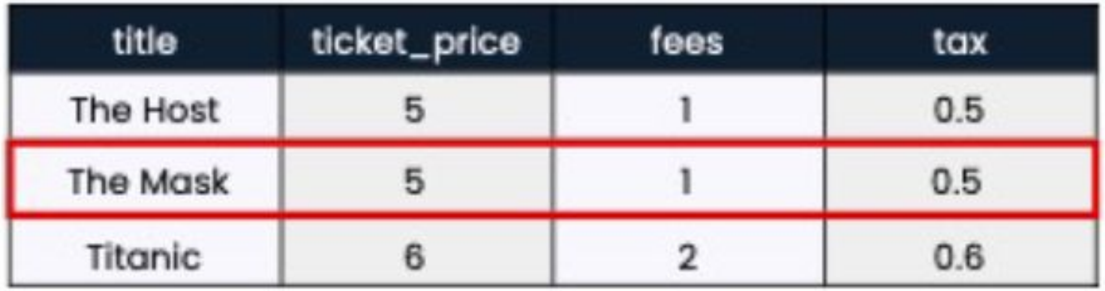
Aggregate functions vs. arithmetic

Aggregate functions



title	ticket_price	fees	tax
The Host	5	1	0.5
The Mask	5	1	0.5
Titanic	6	2	0.6

Arithmetic



title	ticket_price	fees	tax
The Host	5	1	0.5
The Mask	5	1	0.5
Titanic	6	2	0.6

Aliasing with arithmetic

```sql
SELECT (gross - budget)
FROM films;
```

```
|?column?|
|--------|
|null    |
|2900000 |
|null    |
...
```

```sql
SELECT (gross - budget) AS profit
FROM films;
```

```
|profit  |
|--------|
|null    |
|2900000 |
|null    |
...
```

Aliasing with functions

```
SELECT MAX(budget), MAX(duration)
FROM films;
```

```
|max        |max|
|-----------|---|
|12215500000|334|
```

```
SELECT MAX(budget) AS max_budget,
       MAX(duration) AS max_duration
FROM films;
```

```
|max_budget |max_duration|
|-----------|------------|
|12215500000|334         |
```

Order of execution

- Step 1: `FROM`
- Step 2: `WHERE`
- Step 3: `SELECT` (aliases are defined here)
- Step 4: `LIMIT`

- Aliases defined in the `SELECT` clause cannot be used in the `WHERE` clause due to order of execution

```sql
SELECT budget AS max_budget
FROM films
WHERE max_budget IS NOT NULL;
```

```
column "max_budget" does not exist
LINE 5: WHERE max_budget IS NOT NULL;
              ^
```

Sorting and Grouping

Sorting results

INTERMEDIATE SQL

Sorting results

ORDER BY

```sql
SELECT title, budget
FROM films
ORDER BY budget;
```

```
|title                |budget|
|-------------------|------|
|Tarnation            |218   |
|My Date with Drew    |1100  |
|A Plague So Pleasant|1400  |
|The Mongol King      |3250  |
...
```

```sql
SELECT title, budget
FROM films
ORDER BY title;
```

```
|title                      |budget  |
|---------------------------|--------|
|#Horror                    |1500000 |
|10 Cloverfield Lane        |15000000|
|10 Days in a Madhouse      |12000000|
|10 Things I Hate About You|16000000|
...
```

ASCending

```
SELECT title, budget
FROM films
ORDER BY budget ASC;
```

```
|title                |budget|
|---------------------|------|
|Tarnation            |218   |
|My Date with Drew    |1100  |
|A Plague So Pleasant|1400  |
|The Mongol King      |3250  |
...
```

DESCending

```
SELECT title, budget
FROM films
ORDER BY budget DESC;
```

```
|title                         |budget|
|------------------------------|------|
|Love and Death on Long Island |null  |
|The Chambermaid on the Titanic|null  |
|51 Birch Street               |null  |
...
```

```
SELECT title, budget
FROM films
WHERE budget IS NOT NULL
ORDER BY budget DESC;
```

```
|title                   |budget     |
|------------------------|-----------|
|The Host                |12215500000|
|Lady Vengeance          |4200000000 |
...
```

Sorting fields

```sql
SELECT title
FROM films
ORDER BY release_year;
```

```
|title                                 |
|--------------------------------------|
|Intolerance: Love's Struggle Throu...|
|Over the Hill to the Poorhouse        |
|The Big Parade                        |
|Metropolis                            |
...
```

```sql
SELECT title, release_year
FROM films
ORDER BY release_year;
```

```
|title                    |release_year|
|-------------------------|------------|
|Intolerance: Love's S...|1916        |
|Over the Hill to the ...|1920        |
|The Big Parade           |1925        |
|Metropolis               |1927        |
...
```

ORDER BY multiple fields

- `ORDER BY field_one, field_two`

```
SELECT title, wins
FROM best_movies
ORDER BY wins DESC;
```

```
|title                              |wins|
|-----------------------------------|----|
|Lord of the Rings:Return of t...|11  |
|Titanic                         |11  |
|Ben-Hur                         |11  |
```

- Think of `field_two` as a tie-breaker

```
SELECT title, wins, imdb_score
FROM best_movies
ORDER BY wins DESC, imdb_score DESC;
```

```
|title                 |wins|imdb_score|
|----------------------|----|----------|
|Lord of the Rings:...|11  |9         |
|Ben-Hur              |11  |8.1       |
|Titanic              |11  |7.9       |
```

Different orders

```sql
SELECT birthdate, name
FROM people
ORDER BY birthdate, name DESC;
```

```
|birthdate |name            |
|----------|----------------|
|1990-01-01|Robert Brown    |
|1990-02-02|Anne Smith      |
|1991-05-14|Amy Miller      |
|1991-11-22|Adam Waters     |
...
```

Order of execution

```
-- Written code:
SELECT item
FROM coats
WHERE color = `yellow`
ORDER BY length
LIMIT 3;
```

```
-- Order of execution:
SELECT item
FROM coats
WHERE color = `yellow`
ORDER BY length
LIMIT 3;
```

Grouping data

INTERMEDIATE SQL

Grouping data

GROUP BY single fields

```
SELECT certification, COUNT(title) AS title_count
FROM films
GROUP BY certification;
```

```
|certification|title_count|
|-------------|-----------|
|Unrated      |62         |
|M            |5          |
|G            |112        |
|NC-17        |7          |
...
```

Error handling

```sql
SELECT certification, title
FROM films
GROUP BY certification;
```

```
column "films.title" must appear in the
GROUP BY clause or be used in an
aggregate function
LINE 1: SELECT certification, title
                              ^
```

```sql
SELECT
   certification,
   COUNT(title) AS count_title
FROM films
GROUP BY certification;
```

```
|certification|count_title|
|-------------|-----------|
|Unrated      |62         |
|M            |5          |
|G            |112        |
...
```

GROUP BY multiple fields

```sql
SELECT certification, language, COUNT(title) AS title_count
FROM films
GROUP BY certification, language;
```

```
|certification|language |title_count|
|-------------|---------|-----------|
|null         |null     |5          |
|Unrated      |Japanese |2          |
|R            |Norwegian|2          |
...
```

GROUP BY with ORDER BY

```sql
SELECT
	certification,
	COUNT(title) AS title_count
FROM films
GROUP BY certification;
```

```
|certification|title_count|
|-------------|-----------|
|Unrated      |62         |
|M            |5          |
|G            |112        |
...
```

```sql
SELECT
	certification,
	COUNT(title) AS title_count
FROM films
GROUP BY certification
ORDER BY title_count DESC;
```

```
|certification|title_count|
|-------------|-----------|
|R            |2118       |
|PG-13        |1462       |
...
```

Order of execution

```
-- Written code:
SELECT
       certification,
       COUNT(title) AS title_count
FROM films
GROUP BY certification
ORDER BY title_count DESC
LIMIT 3;
```

```
-- Order of execution:
SELECT
       certification,
       COUNT(title) AS title_count
FROM films
GROUP BY certification
ORDER BY title_count DESC
LIMIT 3;
```

Filtering grouped data

INTERMEDIATE SQL

HAVING

```
SELECT
       release_year,
       COUNT(title) AS title_count
FROM films
GROUP BY release_year
WHERE COUNT(title) > 10;
```

```
syntax error at or near "WHERE"
LINE 4: WHERE COUNT(title) > 10;
        ^
```

```
SELECT
       release_year,
       COUNT(title) AS title_count
FROM films
GROUP BY release_year
HAVING COUNT(title) > 10;
```

```
|release_year|title_count|
|------------|-----------|
|1988        |31         |
|null        |42         |
|2008        |225        |
...
```

Order of execution

```
-- Written code:
SELECT
       certification,
       COUNT(title) AS title_count
FROM films
WHERE certification
       IN ('G', 'PG', 'PG-13')
GROUP BY certification
HAVING COUNT(title) > 500
ORDER BY title_count DESC
LIMIT 3;
```

```
-- Order of execution:
SELECT
       certification,
       COUNT(title) AS title_count
FROM films
WHERE certification
       IN ('G', 'PG', 'PG-13')
GROUP BY certification
HAVING COUNT(title) > 500
ORDER BY title_count DESC
LIMIT 3;
```

HAVING vs WHERE

- WHERE filters individual records, HAVING filters grouped records
- What films were released in the year 2000?

```sql
SELECT title
FROM films
WHERE release_year = 2000;
```

```
|title        |
|-------------|
|102 Dalmatians|
|28 Days      |
...
```

- In what years was the average film duration over two hours?

HAVING vs WHERE

- In what years was the average film duration over two hours?

```sql
SELECT release_year
FROM films
GROUP BY release_year
HAVING AVG(duration) > 120;
```

```
|release_year|
|------------|
|1954        |
|1959        |
...
```