# Solutions and Notes for SQLBolt Tutorial Exercises

## 1. `SELECT` queries 101

### Table

`movies`

| `id` | `title` | `director` | `year` | `length_minutes` |
|---|---|---|---|---|
| 1 | Toy Story | John Lasseter | 1995 | 81 |
| 2 | A Bug's Life | John Lasseter | 1998 | 95 |
| 3 | Toy Story 2 | John Lasseter | 1999 | 93 |
| 4 | Monsters, Inc. | Pete Docter | 2001 | 92 |
| 5 | Finding Nemo | Andrew Stanton | 2003 | 107 |
| 6 | The Incredibles | Brad Bird | 2004 | 116 |
| 7 | Cars | John Lasseter | 2006 | 117 |
| 8 | Ratatouille | Brad Bird | 2007 | 115 |
| 9 | WALL-E | Andrew Stanton | 2008 | 104 |
| 10 | Up | Pete Docter | 2009 | 101 |

### Tasks

1. Find the `title` of each film.

```
SELECT
    title
FROM
    movies;
```

2. Find the `director` of each film.

```
SELECT
    director
FROM
    movies;
```

3. Find the `title` and `director` of each film.

```sql
SELECT
    title,
    director
FROM
    movies;
```

4. Find the `title` and `year` of each film.

```sql
SELECT
    title,
    year
FROM
    movies;
```

5. Find *all* the information about each film.

```sql
SELECT
    *
FROM
    movies;
```

## 2. Queries with constraints (Pt. 1)

### Table

`movies` ↑

### Tasks

1. Find the movie with a row `id` of 6.

```
SELECT
    *
FROM
    movies
WHERE
    id = 6;
```

2. Find the movies released in the `year` s between 2000 and 2010.

```
SELECT
    *
FROM
    movies
WHERE
    year BETWEEN 2000 AND 2010;
```

3. Find the movies **not** released in the `year` s between 2000 and 2010.

```
SELECT
    *
FROM
    movies
WHERE
    year NOT BETWEEN 2000 AND 2010;
```

4. Find the first 5 Pixar movies and their release `year` .

```
SELECT
    title,
    year
FROM
    movies
WHERE
    id < 6;
```

## 3. Queries with constraints (Pt. 2)

### Table

`movies` ↑

### Tasks

1. Find all the Toy Story movies.

```
SELECT
    *
FROM
    movies
WHERE
    title LIKE 'Toy Story%';
```

2. Find all the movies directed by John Lasseter.

```
SELECT
    *
FROM
    movies
WHERE
    director = 'John Lasseter';
```

3. Find all the movies (and director) not directed by John Lasseter.

```
SELECT
    title,
    director
FROM
    movies
WHERE
    director != 'John Lasseter';
```

4. Find all the WALL-* movies.

```
SELECT
    *
FROM
    movies
WHERE
    title like 'WALL-%';
```

## 4. Filtering and sorting query results

### Table

`movies` ↑

### Tasks

1. List all directors of Pixar movies (alphabetically), without duplicates.

   ```
   SELECT DISTINCT
       director
   FROM
       movies
   ORDER BY
       director;
   ```

   Note: `DISTINCT` removes duplicate column *tuples* in the resulting collection of rows. (In this context, a *tuple* is the combination of column values in a single row.)

2. List the last four Pixar movies released (ordered from most recent to least).

   ```
   SELECT
       *
   FROM
       movies
   ORDER BY
       year DESC
   LIMIT 4;
   ```

3. List the **first** five Pixar movies sorted alphabetically.

   ```
   SELECT
       *
   FROM
       movies
   ORDER BY
       title
   LIMIT 5;
   ```

4. List the **next** five Pixar movies sorted alphabetically.

```sql
SELECT
    *
FROM
    movies
ORDER BY
    title
LIMIT 5 OFFSET 5;
```

## 5. Review: Simple `SELECT` Queries

### Table

`north_american_cities`

| `city` | `country` | `population` | `latitude` | `longitude` |
|---|---|---|---|---|
| Guadalajara | Mexico | 1500800 | 20.659699 | -103.349609 |
| Toronto | Canada | 2795060 | 43.653226 | -79.383184 |
| Houston | United States | 2195914 | 29.760427 | -95.369803 |
| New York | United States | 8405837 | 40.712784 | -74.005941 |
| Philadelphia | United States | 1553165 | 39.952584 | -75.165222 |
| Havana | Cuba | 2106146 | 23.05407 | -82.345189 |
| Mexico City | Mexico | 8555500 | 19.432608 | -99.133208 |
| Phoenix | United States | 1513367 | 33.448377 | -112.074037 |
| Los Angeles | United States | 3884307 | 34.052234 | -118.243685 |
| Ecatepec de Morelos | Mexico | 1742000 | 19.601841 | -99.050674 |
| Montreal | Canada | 1717767 | 45.501689 | -73.567256 |
| Chicago | United States | 2718782 | 41.878114 | -87.629798 |

### Tasks

1. List all the Canadian cities and their populations.

```
SELECT
    *
FROM
    north_american_cities
WHERE
    country = 'Canada';
```

2. Order all the cities in the United States by their latitude from north to south.

```
SELECT
    *
FROM
    north_american_cities
WHERE
    country = 'United States'
ORDER BY
    latitude DESC;
```

3. List all the cities west of Chicago, ordered from west to east.

```
SELECT
    *
FROM
    north_american_cities
WHERE
    longitude < -87.629798
ORDER BY
    longitude;
```

4. List the two largest cities in Mexico (by population).

```
SELECT
    *
FROM
    north_american_cities
WHERE
    country = 'Mexico'
ORDER BY
    population DESC
LIMIT 2;
```

5. List the third and fourth largest cities (by population) in the United States and their population.

```
SELECT
    *
FROM
    north_american_cities
WHERE
    country = 'United States'
ORDER BY
    population DESC
LIMIT 2 OFFSET 2;
```

## 6. Multi-table queries with `JOIN`s

### Tables

`movies` ↑

`boxoffice`

| `movie_id` | `rating` | `domestic_sales` | `international_sales` |
|---|---|---|---|
| 5 | 8.2 | 380843261 | 555900000 |
| 14 | 7.4 | 268492764 | 475066843 |
| 8 | 8 | 206445654 | 417277164 |
| 12 | 6.4 | 191452396 | 368400000 |
| 3 | 7.9 | 245852179 | 239163000 |
| 6 | 8 | 261441092 | 370001000 |
| 9 | 8.5 | 223808164 | 297503696 |
| 11 | 8.4 | 415004880 | 648167031 |
| 1 | 8.3 | 191796233 | 170162503 |
| 7 | 7.2 | 244082982 | 217900167 |
| 10 | 8.3 | 293004164 | 438338580 |
| 4 | 8.1 | 289916256 | 272900000 |
| 2 | 7.2 | 162798565 | 200600000 |
| 13 | 7.2 | 237283207 | 301700000 |

### Tasks

1. Find the domestic and international sales for each movie.

```sql
SELECT
    mv.id,
    mv.title,
    bo.domestic_sales,
    bo.international_sales
FROM
    movies AS mv
    INNER JOIN boxoffice AS bo
        ON mv.id = bo.movie_id;
```

2. Show the sales numbers for each movie that did better internationally rather than domestically

```sql
SELECT
    mv.id,
    mv.title,
    bo.domestic_sales,
    bo.international_sales
FROM
    movies AS mv
    INNER JOIN boxoffice AS bo
        ON mv.id = bo.movie_id
WHERE
    bo.international_sales > bo.domestic_sales;
```

3. List all the movies by their ratings in descending order.

```sql
SELECT
    mv.id,
    mv.title,
    bo.rating
FROM
    movies AS mv
    INNER JOIN boxoffice AS bo
        ON mv.id = bo.movie_id
ORDER BY
    bo.rating DESC;
```

7. `OUTER JOIN`s

### Tables

`buildings`

| `building_name` | `capacity` |
|---|---|
| 1e | 24 |
| 1w | 32 |
| 2e | 16 |
| 2w | 20 |

`employees`

| `role` | `name` | `building` | `years_employed` |
|---|---|---|---|
| Engineer | Becky A. | 1e | 4 |
| Engineer | Dan B. | 1e | 2 |
| Engineer | Sharon F. | 1e | 6 |
| Engineer | Dan M. | 1e | 4 |
| Engineer | Malcom S. | 1e | 1 |
| Artist | Tylar S. | 2w | 2 |
| Artist | Sherman D. | 2w | 8 |
| Artist | Jakob J. | 2w | 6 |
| Artist | Lillia A. | 2w | 7 |
| Artist | Brandon J. | 2w | 7 |
| Manager | Scott K. | 1e | 9 |
| Manager | Shirlee M. | 1e | 3 |
| Manager | Daria O. | 2w | 6 |

### Tasks

1. Find the list of all buildings that have employees.

```
SELECT DISTINCT
    b.building_name
FROM
    buildings AS b
    INNER JOIN employees AS e
        ON b.building_name = e.building;
```

2. Find the list of all buildings and their capacity.

```
SELECT
    b.building_name,
    b.capacity
FROM
    buildings AS b;
```

3. List all buildings and the distinct employee roles in each building (including empty buildings).

```
SELECT DISTINCT
    b.building_name,
    e.role
FROM
    buildings AS b
    LEFT JOIN employees AS e
        ON b.building_name = e.building;
```

## 8. A short note on `NULL`s

### Tables

`buildings` ↑

`employees`

| `role` | `name` | `building` | `years_employed` |
|---|---|---|---|
| Engineer | Becky A. | 1e | 4 |
| Engineer | Dan B. | 1e | 2 |
| Engineer | Sharon F. | 1e | 6 |
| Engineer | Dan M. | 1e | 4 |
| Engineer | Malcom S. | 1e | 1 |
| Artist | Tylar S. | 2w | 2 |
| Artist | Sherman D. | 2w | 8 |
| Artist | Jakob J. | 2w | 6 |
| Artist | Lillia A. | 2w | 7 |
| Artist | Brandon J. | 2w | 7 |
| Manager | Scott K. | 1e | 9 |
| Manager | Shirlee M. | 1e | 3 |
| Manager | Daria O. | 2w | 6 |
| Engineer | Yancy I. |  | 0 |
| Artist | Oliver P. |  | 0 |

### Tasks

1. Find the name and role of all employees who have not been assigned to a building.

```
SELECT
    name,
    role
FROM
    employees
WHERE
    building IS NULL;
```

Note: Remember to use `IS` and `IS NOT` when comparing values to `NULL`. According to the SQL standard, the `NULL` value is not equal to any other value—even another `NULL` value! Further, `NULL` “ripples” through expressions: Any expression that incorporates a `NULL` value, but which doesn’t check for and handle that value correctly, will also have the value `NULL`.

Some languages do allow for a “relaxed” syntax, supporting comparisons with `NULL` using the `=` and `!=` operators, but these are non-standard and should not be relied upon, in general.

2. Find the names of the buildings that hold no employees.

```
SELECT
    b.building_name
FROM
    buildings AS b
    LEFT JOIN employees AS e
        ON b.building_name = e.building
WHERE
    e.building IS NULL;
```

## 9. Queries with expressions

### Tables

`movies`

| `id` | `title` | `director` | `year` | `length_minutes` |
|---|---|---|---|---|
| 1 | Toy Story | John Lasseter | 1995 | 81 |
| 2 | A Bug's Life | John Lasseter | 1998 | 95 |
| 3 | Toy Story 2 | John Lasseter | 1999 | 93 |
| 4 | Monsters, Inc. | Pete Docter | 2001 | 92 |
| 5 | Finding Nemo | Andrew Stanton | 2003 | 107 |
| 6 | The Incredibles | Brad Bird | 2004 | 116 |
| 7 | Cars | John Lasseter | 2006 | 117 |
| 8 | Ratatouille | Brad Bird | 2007 | 115 |
| 9 | WALL-E | Andrew Stanton | 2008 | 104 |
| 10 | Up | Pete Docter | 2009 | 101 |
| 11 | Toy Story 3 | Lee Unkrich | 2010 | 103 |
| 12 | Cars 2 | John Lasseter | 2011 | 120 |
| 13 | Brave | Brenda Chapman | 2012 | 102 |
| 14 | Monsters University | Dan Scanlon | 2013 | 110 |

`boxoffice`

| `movie_id` | `rating` | `domestic_sales` | `international_sales` |
|---|---|---|---|
| 5 | 8.2 | 380843261 | 555900000 |
| 14 | 7.4 | 268492764 | 475066843 |
| 8 | 8 | 206445654 | 417277164 |
| 12 | 6.4 | 191452396 | 368400000 |
| 3 | 7.9 | 245852179 | 239163000 |
| 6 | 8 | 261441092 | 370001000 |
| 9 | 8.5 | 223808164 | 297503696 |
| 11 | 8.4 | 415004880 | 648167031 |
| 1 | 8.3 | 191796233 | 170162503 |
| 7 | 7.2 | 244082982 | 217900167 |
| 10 | 8.3 | 293004164 | 438338580 |
| 4 | 8.1 | 289916256 | 272900000 |
| 2 | 7.2 | 162798565 | 200600000 |
| 13 | 7.2 | 237283207 | 301700000 |

## Tasks

1. List all movies and their combined sales in **millions** of dollars.

```sql
SELECT
    mv.title,
    (bo.domestic_sales + bo.international_sales) / 1000000 AS combined_sales
FROM
    movies AS mv
    INNER JOIN boxoffice AS bo
        ON mv.id = bo.movie_id;
```

   Note: For anything other than quick-and-dirty exploratory code, it's a good idea to define aliases for computed columns.

2. List all movies and their ratings in percent.

```sql
SELECT
    mv.title,
    bo.rating * 10 AS rating_percent
FROM
    movies AS mv
    INNER JOIN boxoffice AS bo
        ON mv.id = bo.movie_id;
```

3. List all movies that were released on even number years.

```sql
SELECT
    title
FROM
    movies
WHERE
    year % 2 = 0;
```

## 10. Queries with aggregates (Pt. 1)

### Table

`employees` ↑

### Tasks

1. Find the longest time that an employee has been at the studio.

```sql
SELECT
    MAX(years_employed) AS max_years
FROM
    employees;
```

2. For each role, find the average number of years employed by employees in that role.

```sql
SELECT
    role,
    AVG(years_employed) AS avg_years
FROM
    employees
GROUP BY
    role;
```

Note: In general, when using aggregate functions with a `GROUP BY` clause, the column list should only include columns specified in the `GROUP BY` clause, and aggregate functions of other columns. (In some cases, including columns other than these will cause syntax others; in most others, the values for these columns will be meaningless.)

3. Find the total number of employee years worked in each building.

```sql
SELECT
    building,
    SUM(years_employed) AS total_years
FROM
    employees
GROUP BY
    building;
```

## 11. Queries with aggregates (Pt. 2)

### Table

`employees` ↑

### Tasks

1. Find the number of Artists in the studio (without a `HAVING` clause).

```
SELECT
    COUNT(*) AS artist_count
FROM
    employees
WHERE
    role = 'Artist';
```

Note: In general, any column (or computed expression) can be used as the argument to the `COUNT()` aggregate function. However, if the value of the specified column (or expression) is `NULL` in any of the rows selected by the query criteria, the corresponding rows will not be included in the count. To ensure that *all* rows are included in the count, `COUNT(*)` or `COUNT(1)` is generally used.

2. Find the number of Employees of each role in the studio.

```
SELECT
    role,
    COUNT(*) AS role_count
FROM
    employees
GROUP BY
    role;
```

3. Find the total number of years employed by all Engineers.

```
SELECT
    SUM(years_employed) AS total_engineer_years
FROM
    employees
WHERE
    role = 'Engineer';
```

## 12. Order of execution of a query

### Tables

`movies` ↑

`boxoffice` ↑

### Tasks

1. Find the number of movies each director has directed.

```sql
SELECT
    director,
    COUNT(*) AS movie_count
FROM
    movies
GROUP BY
    director;
```

2. Find the total domestic and international sales that can be attributed to each director.

```sql
SELECT
    mv.director,
    SUM(bo.domestic_sales + bo.international_sales) AS total_sales
FROM
    movies AS mv
    INNER JOIN boxoffice AS bo
        ON mv.id = bo.movie_id
GROUP BY
    mv.director;
```

## 13. Inserting rows

### Tables

`movies` ↑

`boxoffice` ↑

### Tasks

1. Add the studio’s new production, **Toy Story 4** to the list of movies (you can use any director).

   ```
   INSERT INTO movies
       (title, director, year, length_minutes)
   VALUES
       ('Toy Story 4', 'Josh Cooley', 2019, 100);
   ```

   (This `id` column of the record added by this `INSERT` is 15.)

   Note: Although the one-column-per-line approach to column lists helps with producing maintainable code, it can lead to simple statements that span far more lines than is useful or necessary—especially in `INSERT` statements. Partly because of this, these `INSERT` snippets employ a commonly used format that is collapsed into fewer lines.

2. Toy Story 4 has been released to critical acclaim! It had a rating of **8.7**, and made **340 million domestically** and **270 million internationally**. Add the record to the `boxoffice` table.

   ```
   INSERT INTO boxoffice
       (movie_id, rating, domestic_sales, international_sales)
   VALUES
       (15, 8.7, 340000000, 270000000);
   ```

## 14. Updating rows

### Table

`movies`

| `id` | `title` | `director` | `year` | `length_minutes` |
|---|---|---|---|---|
| 1 | Toy Story | John Lasseter | 1995 | 81 |
| 2 | A Bug’s Life | El Directore | 1998 | 95 |
| 3 | Toy Story 2 | John Lasseter | 1998 | 93 |
| 4 | Monsters, Inc. | Pete Docter | 2001 | 92 |
| 5 | Finding Nemo | Andrew Stanton | 2003 | 107 |
| 6 | The Incredibles | Brad Bird | 2004 | 116 |
| 7 | Cars | John Lasseter | 2006 | 117 |
| 8 | Ratatouille | Brad Bird | 2007 | 115 |
| 9 | WALL-E | Andrew Stanton | 2008 | 104 |
| 10 | Up | Pete Docter | 2009 | 101 |
| 11 | Toy Story 8 | El Directore | 2010 | 103 |
| 12 | Cars 2 | John Lasseter | 2011 | 120 |
| 13 | Brave | Brenda Chapman | 2012 | 102 |
| 14 | Monsters University | Dan Scanlon | 2013 | 110 |

### Tasks

Use `UPDATE` statements to correct the following issues:

1. The director for A Bug’s Life is incorrect, it was actually directed by **John Lasseter**.

```sql
UPDATE
    movies
SET
    director = 'John Lasseter'
WHERE
    id = 2; -- We might instead user WHERE title = 'A Bug's Life'
```

2. The year that Toy Story 2 was released is incorrect, it was actually released in **1999**.

```
UPDATE
    movies
SET
    year = 1999
WHERE
    id = 3; -- We might instead user WHERE title = 'Toy Story 2'
```

3. Both the title and director for Toy Story 8 are incorrect! The title should be “Toy Story 3” and it was directed by **Lee Unkrich**.

```
UPDATE
    movies
SET
    title = 'Toy Story 3',
    director = 'Lee Unkrich'
WHERE
    id = 11;
```

## 15. Deleting rows

### Table

`movies` ↑

### Tasks

1. This database is getting too big, lets remove all movies that were released **before** 2005.

```
DELETE FROM
    movies
WHERE
    year < 2005;
```

2. Andrew Stanton has also left the studio, so please remove all movies directed by him.

```
DELETE FROM
    movies
WHERE
    director = 'Andrew Stanton';
```

## 16. Creating tables

### Task

Create a new table named `database` with the following columns:

- `name` A string (text) describing the name of the database
- `version` A number (floating point) of the latest version of this database
- `download_count` An integer count of the number of times this database was downloaded

This table has no constraints.

```sql
CREATE TABLE database (
    name TEXT,
    version FLOAT,
    download_count INTEGER
);
```

## 17. Altering tables

### Table

`movies` ↑

### Tasks

1. Add a column named `aspect_ratio` with a `FLOAT` data type to store the aspect-ratio each movie was released in.

   ```
   ALTER TABLE movies
   ADD COLUMN aspect_ratio FLOAT;
   ```

   Note: When a column is added to an existing table, any rows already in the table will take the `DEFAULT` value for that column. If a `DEFAULT` is not declared, then the value `NULL` will be used. If there is no `DEFAULT`, and if the column is constrained to be `NOT NULL`, the `ALTER TABLE` operation will fail. In this case, since a `DEFAULT` value has not been set for `aspect_ratio`, and since it has not been declared with a `NOT NULL` constraint, existing rows will have the value `NULL` in the `aspect_ratio` column.

2. Add another column named `language` with a `TEXT` data type to store the language that the movie was released in. Ensure that the default for this language is **English**.

   ```
   ALTER TABLE movies
   ADD COLUMN language TEXT DEFAULT "English";
   ```

## 18. Dropping tables

### Tables

`movies` ↑

`boxoffice` ↑

### Tasks

1. We've sadly reached the end of our lessons, lets clean up by removing the `movies` table.

   ```
   DROP TABLE IF EXISTS movies;
   ```

2. And drop the `boxoffice` table as well.

   ```
   DROP TABLE IF EXISTS boxoffice;
   ```

Note: In a real-world application, we almost certainly would have defined a `FOREIGN KEY` constraint in the `boxoffice` table, referencing the `movies` table (since every row in `boxoffice` references a row in `movies`). This would dictate that we either `DROP` the `boxoffice` table before we `DROP` the `movies` table, or that we `DROP` the `FOREIGN KEY` constraint first.