SQL Joins Practice Solutions:

**Problem 1 Solution:**

SELECT

  name,

  title,

  publish\_year

FROM author

JOIN book

  ON author.id = book.author\_id;

**Problem 1 Explanation:**

**Solution explanation:** The query selects the name of the author, the book title, and its publishing year. This is data from the two tables: author and book. We are able to access both tables by using INNER JOIN. It returns only rows with matching values (values that satisfy the join condition) from both tables.

We first reference the table author in the FROM clause. Then we add the JOIN clause (which can also be written as INNER JOIN in SQL) and reference the table book.

The tables are joined on the common column. In this case, it's id from the table author and author\_id from the table book. We want to join the rows where these columns share the same value. We do that using the ON clause and specifying the column names. We also put the table name before each column so the database knows where to look. That’s primarily because there’s an id column in both tables, but we want the id column only from the author table. By referencing the table name, the database will know from which table we need that column.

**Problem 1 output:**

Here’s the output snapshot. We got all this data by joining two tables:

| **name** | **title** | **publish\_year** |
| --- | --- | --- |
| Marcella Cole | Gone With The Wolves | 2005 |
| Lisa Mullins | Companions And Officers | 1930 |
| Dennis Stokes | Blacksmith With Silver | 1984 |
| Randolph Vasquez | Faith Of Light | 1995 |
| Michael Rostkovsky | Warrior Of Wind | 2005 |
| … | … | … |

**Problem 2 Solution:**

|  |
| --- |
| SELECT    name,    title,    publish\_year  FROM author  JOIN book    ON author.id = book.author\_id  WHERE publish\_year > 2005; |

**Problem 2 explanation:**This exercise and its solution are almost the same as the previous one. This is reflected by the query selecting the same columns and joining the tables in the same way as earlier.

The difference is that the exercise now asks us to show **only** books published after 2005. This requires filtering the output; we do that using the WHERE clause.

WHERE is a clause that accepts conditions used to filter out the data. It is written after joining the tables. In our example, we filter by referencing the column publish\_year after WHERE and using the comparison operator ‘greater than’ (>) to find the years after 2005.

**Problem 2 output:**

The output shows only one book published after 2005.

| **name** | **title** | **publish\_year** |
| --- | --- | --- |
| Darlene Lyons | Temptations In Nature | 2007 |

**Problem 3 Solution:**

|  |
| --- |
| SELECT    book.title AS book\_title,    adaptation.title AS adaptation\_title,    book.publish\_year,    adaptation.release\_year  FROM book  JOIN adaptation    ON book.id = adaptation.book\_id  WHERE adaptation.release\_year - book.publish\_year <= 4    AND book.rating < adaptation.rating; |

**Problem 3 explanation:**Let’s start explaining the solution from the FROM and JOIN clauses. The columns we need to show are from the tables book and adaptation. We reference the first table in FROM and the second in JOIN.

In the ON clause, we equal the two book ID columns and specify the table of each column. This is the same as earlier, only with different table and column names.

Now, we need to select the required columns. The thing here is there’s a title column in both tables. To avoid ambiguity, a best practice is to reference the table name before each column in the SELECT.

**Note:** The above is mandatory only for ambiguous columns. However, it’s a good idea to do that with all columns; it improves code readability and the approach remains consistent.

After selecting the columns, we need to rename some of them. We do that using the keyword AS and writing a new column name afterward. That way, one title column becomes book\_title, the other becomes adaptation\_title. Giving aliases to the column names also helps get rid of ambiguity.

Now we need to filter the output. The first condition is that the adaptation had to be released four years or less after the book. We again use WHERE and simply deduct the book publish year from the adaptation release year. Then we say that the difference has to be less than or equal to (<=) 4.

We also need to add the second condition, where the book has a lower rating than the adaptation. It’s simple! The question implies that both the first *and* the second conditions have to be satisfied. The clue is in AND, a logical operator we use for adding the second condition. Here, it uses the ‘less than’ (<) operator to compare the two ratings.

**Problem 3 output:**

The output shows three book–adaptation pairs that satisfy the conditions.

| **book\_title** | **adaptation\_title** | **publish\_year** | **release\_year** |
| --- | --- | --- | --- |
| Memory Of Hope | Patrons And Bearers | 2000 | 2004 |
| Music At The Lake | Music At The Lake | 2004 | 2007 |
| Companion Of Tomorrow | Lighting Faith | 1949 | 1952 |

**Problem 4 Solution:**

|  |
| --- |
| SELECT    book.title,    adaptation.title,    adaptation.release\_year  FROM book  LEFT JOIN adaptation    ON book.id = adaptation.book\_id; |

**Problem 4 explanation:** We first select the required columns from the two tables. Then we join book (the left table) with adaptation(the right table) using LEFT JOIN. You see that the SQL join syntax is the same for INNER JOIN. The only thing that changes is the join keyword.

**Note:** SQL accepts both LEFT JOIN and LEFT OUTER JOIN. They are the same command.

**Problem 4 output:**

The output snapshot shows the required data, with some of the data shown as NULL. These are the books without the adaptation.

| **title** | **title-2** | **release\_year** |
| --- | --- | --- |
| Soulless girl | Gone With The Wolves: The Beginning | 2008 |
| Faith Of Light | Companions Of Tomorrow | 2001 |
| Warrior Of Wind | Homeless Warrior | 2008 |
| … | … | … |
| Guarding The Emperor | NULL | NULL |
| Blacksmith With Silver | NULL | NULL |
| … | … | … |

**Problem 5 Solution:**

|  |
| --- |
| SELECT    book.title,    publishing\_house,    adaptation.title,    adaptation.type  FROM book  LEFT JOIN adaptation    ON book.id = adaptation.book\_id  WHERE type = 'movie'    OR type IS NULL; |

**Problem 5 explanation:**

The question asks to show all the rows, even those without any adaptations. It’s possible that there are books without adaptations, so we use LEFT JOIN.

We first select the book title, its publishing house, its adaptation title, and its type.

Then we join book (the left table) with adaptation (the right table) using LEFT JOIN. We join the tables on the book ID. All the books that don’t satisfy the conditions will have NULLs as an adaptation title and type.

We filter data using WHERE. The first condition is that the adaptation type has to be a movie, so we equal the type column with a movie using the equal sign (=).  **Note:** When using text data in the WHERE condition, it must be enclosed in single quotes ('').

The second filtering condition is added using the logical operator OR. It says that the type can also be NULL if it’s not a movie. The exercise asks us to keep books with no adaptations in the results.

**Problem 5 output:**

Here’s the output snapshot. You can see that it shows only books adapted as movies or not adapted at all.

| **title** | **publishing\_house** | **title-2** | **type** |
| --- | --- | --- | --- |
| Soulless girl | Golden Albatros | Gone With The Wolves: The Beginning | movie |
| Faith Of Light | White Cloud Press | Companions Of Tomorrow | movie |
| Warrior Of Wind | Maverick | Homeless Warrior | movie |
| … | … | … | … |
| Guarding The Emperor | Flying Pen Media | NULL | NULL |
| Blacksmith With Silver | Diarmud Inc. | NULL | NULL |

**Problem 6 Solution:**

|  |
| --- |
| SELECT    book.title,    book\_review.review,    book\_review.author  FROM book\_review  RIGHT JOIN book    ON book.id = book\_review.book\_id; |

**Problem 6 explanation:**

We first select the required columns. Then we do as we’re told: join the tables using RIGHT JOIN. We join the tables on the book ID. The table book is the right table; we want all the data from it, regardless of the reviews.

As you can see, the syntax stays the same as in INNER JOIN and LEFT JOIN.

**Note:** SQL accepts both RIGHT JOIN and RIGHT OUTER JOIN.

**Problem 6 output:**

The query returns all the book titles, their reviews, and authors. Where there’s no review or author information, a NULL is shown.

| **title** | **review** | **author** |
| --- | --- | --- |
| Soulless girl | An incredible book | Sylvia Jones |
| Soulless girl | Great, although it has some flaws | Jessica Parker |
| … | … | … |
| Guarding The Emperor | NULL | NULL |
| Companions And Officers | NULL | NULL |
| Blacksmith With Silver | NULL | NULL |
| … | … | … |

**Problem 7 Solution:**

|  |
| --- |
| SELECT    title,    name  FROM book  FULL JOIN author    ON book.author\_id = author.id; |

**Problem 7 explanation:** The question requires showing all books, but also all authors – FULL JOIN is perfect for doing this elegantly.

We select the book title and the author's name. Next, we FULL JOIN the table book with the table author. The joining condition is that the author ID has to be the same in both tables. Again, the syntax is the same as in all the previous join types.

**Note:** SQL accepts both FULL JOIN and FULL OUTER JOIN.

**Problem 7 output:**

The output shows all the books and all the authors, whether the authors or books exist in both tables or not.

| **title** | **name** |
| --- | --- |
| Gone With The Wolves | Marcella Cole |
| Companions And Officers | Lisa Mullins |
| … | … |
| NULL | Daniel Branson |
| … | … |
| Weep Of The West | NULL |

**Problem 8 Solution:**

|  |
| --- |
| SELECT    p.name AS product,    d.name AS department  FROM department d  JOIN product p    ON d.id = p.department\_id  JOIN nutrition\_data nd    ON nd.product\_id = p.id  WHERE nd.calories < 150; |

**Problem 8 explanation:** The general principle of how you join the third (fourth, fifth…) table is that you simply add another JOIN. You can see how it’s done in this [article explaining multiple joins](https://learnsql.com/blog/how-to-join-3-tables-or-more-in-sql/). We’ll do it the same way here.

We first join the department table with the product table on the department ID using JOIN. But we also need the third table. To get the data from it, we just add another JOIN, which will join the product table with the nutrition\_data table. The syntax is the same as with the first join. In this case, the query joins the tables on the product ID.

Then we use WHERE to find products with fewer than 150 calories. We finally select the product and department names and rename the columns as per the exercise instructions.

**Note:**You probably noticed both selected columns have the same original name. And you also noticed we solved this ambiguity by putting some strange short table names in front of all the columns in the query. These shortened names are table aliases, which you give by simply writing them after the table name in FROM or JOIN. By giving aliases to the tables, you can shorten the tables’ names. Therefore, you don’t have to write their full names (sometimes they can be really long!), but the short aliases instead. This saves time and space.

**Problem 8 output:**

The output shows a list of the products and the department they belong to. It includes only those products with fewer than 150 calories.

| **product** | **department** |
| --- | --- |
| Apple | fruits |
| Avocado | fruits |
| Banana | fruits |
| Kiwi | fruits |
| Lemon | fruits |
| … | … |

**Problem 9 Solution:**

|  |
| --- |
| SELECT    prod.name AS producer\_name,    d.name AS department\_name,    p.name AS product\_name,    nd.carbohydrate  FROM product p  LEFT JOIN producer prod    ON prod.id = p.producer\_id  LEFT JOIN department d    ON d.id = p.department\_id  LEFT JOIN nutrition\_data nd    ON nd.product\_id = p.id; |

**Problem 9 explanation:** The query selects the required columns. Then it joins the table product with the table producer on the producer ID using LEFT JOIN. We choose this type of join because we have to include products without producer data.

Then we add another LEFT JOIN. This one adds the department table and joins it with the product table. Again, we choose LEFT JOIN because we need to show products that don’t have a department.

There’s also a third join! We simply add it to the chain of the previous joins. It’s again LEFT JOIN, as we add thenutrition\_data table and join it with the producttable.

This is an interesting topic to explore, so [here’s an article that explains multiple LEFT JOINs](https://learnsql.com/blog/how-to-left-join-multiple-tables/) to help you with it.

**Problem 9 output:**

The output shows all the products with their producer and department names and carbohydrate amounts:

| **producer\_name** | **department\_name** | **product\_name** | **carbohydrate** |
| --- | --- | --- | --- |
| BeHealthy | fruits | Kiwi | 20 |
| BeHealthy | vegetables | Broccoli | 8 |
| BeHealthy | meat | Chicken | NULL |
| BeHealthy | NULL | Grapefruit | 15 |
| HealthyFood Inc. | vegetables | Celery | 4 |
| … | … | … | … |

**Problem 10 Solution:**

|  |
| --- |
| SELECT    p.name AS product\_name,    p.price AS product\_price,    prod.name AS producer\_name,    d.name AS department\_name  FROM product p  FULL JOIN producer prod    ON p.producer\_id = prod.id  FULL JOIN department d    ON d.id = p.department\_id; |

**Problem 10 explanation:** This exercise requires using FULL JOIN, as we need all the data from the tables we’ll use: product, producer, and department.

The syntax is the same as in the previous examples. We just join the different tables (product and producer) on the producer ID and use a different type of join:  FULL JOIN.

The second FULL JOIN joins the producttable with the department table.

After selecting the required columns and renaming them, we get the following output.

**Problem 10 output:**

The solution shows all the data from the selected tables and columns:

| **product\_name** | **product\_price** | **producer\_name** | **department\_name** |
| --- | --- | --- | --- |
| Chicken | 5.5 | BeHealthy | meat |
| Broccoli | 2.5 | BeHealthy | vegetables |
| Kiwi | 0.3 | BeHealthy | fruits |
|  |  |  |  |
|  |  |  |  |
| Grapefruit | 0.5 | BeHealthy | NULL |
| Cucumber | 0.7 | HealthyFood Inc. | vegetables |
| … | … | … | … |

**Problem 11 Solution:**

|  |
| --- |
| SELECT    car.model,    car.brand,    car.final\_price  FROM car  JOIN charity\_auction ca    ON car.id = ca.car\_id  JOIN car car2    ON car.mileage > car2.mileage  WHERE car2.id = 4    AND final\_price IS NOT NULL; |

**Problem 11 explanation:** We select the car model, brand, and final price.

In the first JOIN, we join the car table with the charity\_auction table. The tables are joined where the car IDs are the same. This is our regular equi JOIN.

We add the second JOIN, which is a self-join. It adds the table car again, so we can filter the data using the non-equi join condition. The condition will return all the cars from the car table and all the cars from the car2 table with the lower mileage. This is a non-equi condition as it uses the ‘greater than’ ( > ) operator. The syntax is the same, but there’s > instead of = this time.

Finally, we need to filter data using WHERE. We’re not interested in comparing the mileage of all cars. We want to show the cars that have a mileage higher than the car with id = 4. This is what the first filtering condition does.

We add another filtering condition that says the final price shouldn’t be NULL, i.e., the car has to have been sold in the auction.

**Problem 11 output:**

The result shows two cars:

| **model** | **brand** | **final\_price** |
| --- | --- | --- |
| Sundry | Teiko | 50,000 |
| Emperor | Domus | 120,000 |

**Problem 12 Solution:**

|  |
| --- |
| SELECT    apprentice.name AS apprentice\_name,    master.name AS master\_name  FROM workshop\_workers apprentice  JOIN workshop\_workers master    ON apprentice.master\_id = master.id; |

**Problem 12 explanation:** Let’s start with explaining the self-join. The general principle is the same as with regular joins. We reference the table in FROM and give it an alias, apprentice. Then we use JOIN and reference the same table in it. This time, we give the table the alias master. We’re basically pretending that one table has the apprentice data and the other has the master data.

The tables are joined on the master ID from the apprentice table and the ID from the master table.

This example is a typical use of a self-join: the table has a column (master\_id) that references another column from the same table (id). Both columns show the worker’s ID. When there’s NULL in master\_id, it means that the worker doesn’t have a master. In other words, they are the master.

After self-joining, we simply select the required columns and rename them.

**Problem 12 output:**

The output shows all the apprentices and their direct supervisors.

| **apprentice\_name** | **master\_name** |
| --- | --- |
| Kate Brown | Mathew Conn |
| John Doe | Suzan Gregowitch |
| John Kowalsky | Joe Darrington |
| Peter Parker | Joe Darrington |
| Mary Smith | Mathew Conn |
| Carlos Bell | Suzan Gregowitch |
| Dennis Wright | Joe Darrington |