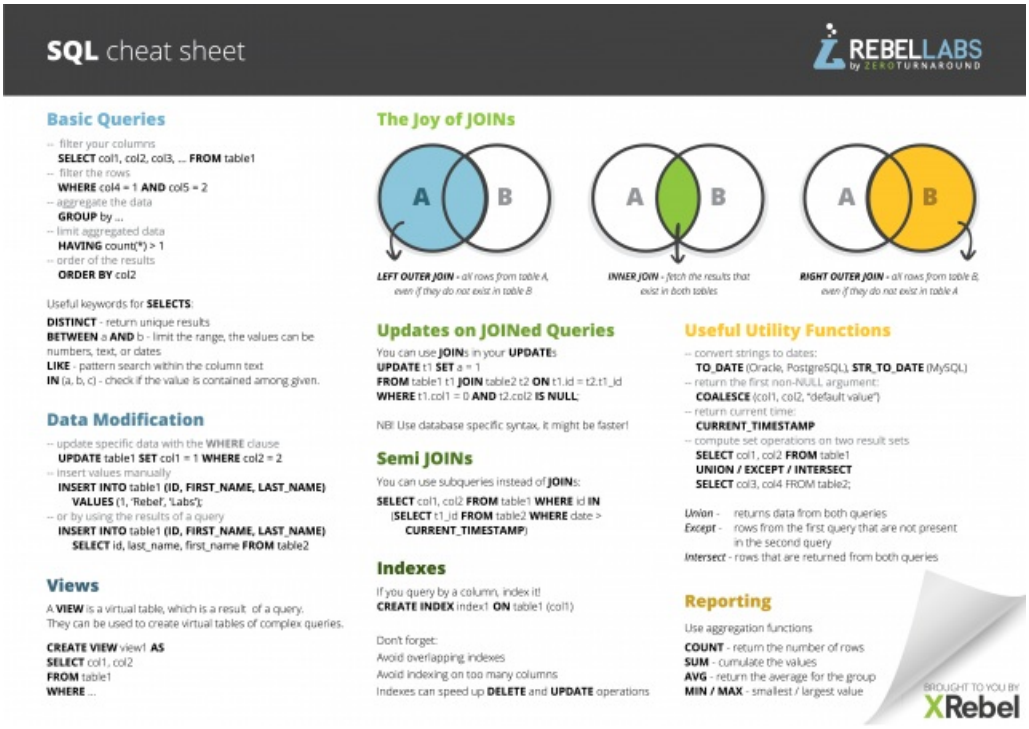
SQL cheat sheet

June 29, 2016 | Oleg Shelajev | 1 Comment



If there's one thing that almost no application can live without, it's the database. The pillar that holds the data, the ultimate source of the data conflict resolution, the storage that survives power outages. Working with a database correctly is the key to successful application design.

That is why we're dedicating this cheat sheets to the important topic of SQL. In the past we've released many other cheat sheets including:

- the best practices of Java 8,
- how to best use the Streams framework in Java 8,
- the most useful Git commands,
- the main commands you need to know for using Docker, and
- the guide to Java collections framework.

Now it's time to turn our attention to the crown jewel of the declarative programming languages: SQL. And while we realize that it's hard to fit everything you need to know about SQL on a single

A4 page, we've tried to incorporate some of the essential information you will need to reference again and again.

Download and print out this cheat sheet so you can use it whenever you need it. To get full the explanations and details of the content in the cheat sheet, continue reading this blog post!

GET THE SQL CHEAT SHEET!

I'd like to thank Lukas Eder for early feedback on this cheat sheet, including advice what to include and telling me that it's actually a good idea to make an SQL cheat sheet. Lukas is the author of jOOQ – the type safe DSL for Java. jOOQ is a simple way to integrate the SQL language into Java in a way that allows for developers to write safe and quality SQL fast and directly in Java. He knows what he's talking about, especially when it's about SQL. And I'm really happy that I had an opportunity to tap into that knowledge.

One language you can't go without

So your project has a database, how exciting. Not at all, actually; any developer knows that most of the consumer facing products use one. And chances are you're using a relational database, where the data is organized into tables of entities.

To communicate to a relational database, you'll use the structured query language, SQL.
SQL is a great language, designed to manage complex queries in a declarative form. When writing SQL, you will focus on the data you want to fetch, rather than how you want to fetch it.

The commands in SQL are called queries and there are two main types:

- Data definition queries – the statements that define the structure of your database, create tables, specify their keys, indexes, and so on.
- Data manipulation queries – what you typically think of when you talk about SQL: select, update, insert operations etc.

In this post we'll mostly look at the data manipulation queries and try to explain how you want to use them.

Basic SQL queries

Data manipulation queries are used to retrieve the data from the database, and create or update the records in it. To retrieve the data, you query it, starting your command with the SELECT clause.

When you think of your data in a relational database, you should think in terms of set theory – you have the description of the entities in a multidimensional space, where the columns in the tables correspond to the dimensions and the rows in the tables are the data points in that space.

When you use the SELECT query you essentially specify the projection of the data point onto a certain multidimensional space: which dimensions to include in the projection is set by the query.

Below is the typical example of an SQL SELECT query:

```
SELECT col1, col2, col3, ... FROM table1
WHERE col4 = 1 AND col5 = 2
GROUP BY …
HAVING count(*) > 1
ORDER BY col2
```

You tell the database which columns from which tables you want. Then you restrict which data points to include in this projection. That is done with the WHERE clause. The rows that do not satisfy the conditions you write in the WHERE clause are omitted from the result set.
On top of that, you can group the related rows using the GROUP BY clause, for example, you can aggregate the rows that belong to the same type, like the employees of the department.

You can enhance your SELECT statements with additional modifiers, for example, asking for DISTINCT entries will return you a results set with unique rows.

And there are several very useful predicates you can use in a WHERE clause. The most frequently used ones are:

- BETWEEN a AND b – filter the entries that do not belong to a range. This works well when you are dealing with dates, or numbers that represent something real, like a salary. It also works on text fields too.
- LIKE – wildcards in text. You can use the wildcard '%' to specify unknown characters in the beginning and at the end of the column, '_' for a single character.
- IN (a, b, c) – filter the values that do not belong to a given set. This particularly works well, when you provide types for the entries or statuses. This tends to be used when you use a subquery to fetch the data and then operate on that results set.

Data manipulation queries

Now we've covered the basics of retrieving data, the following commands will help you update the entries in your tables and create new rows.

To update a table, you use the UPDATE statement, and specify which columns you assign new values to.

```
UPDATE table1 SET col1 = 1 WHERE col2 = 2
```

This is pretty straightforward, just don't forget to limit the values you want to update. If you don't specify the WHERE clause, you will update all the rows in the table and that is probably not what you wanted to achieve.

Additionally, you can update the values dynamically using the data fetched by a SELECT. However, we omitted that from the cheat sheet because it's not a use case you typically find on the application side of the project.

To add new data into the tables, you'll need to use the INSERT statement. Its syntax is also pretty understandable from looking at the command. You just need to specify the data and map which values you want to assign to which columns in the table. Or you can batch insert the values that are returned by a SELECT query.

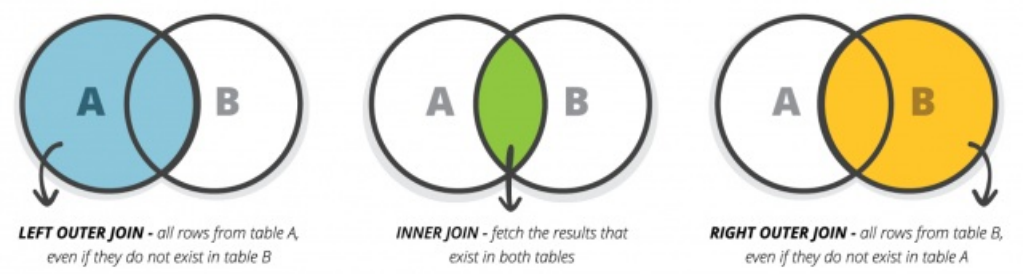
```
INSERT INTO table1
       (id, first_name, last_name)
VALUES (1, 'Rebel', 'Labs');

INSERT INTO table1 (id, first_name, last_name)
  SELECT id, last_name, first_name FROM table2
```

Armed with UPDATEs and INSERTs you are pretty dangerous to any database now, so don't forget to limit the update range with a WHERE clause. Moreover, typically, writing into a database requires a transaction, so if you start one and execute an UPDATE, review the results and commit the transaction.

Working with multiple tables: joins and subqueries

When you master simple queries, you're ready to make the database a really powerful ally. To do so, you need to fetch the data from several tables at the same time, relating the entries from one table to the corresponding rows in another.
This is where you'd use a JOIN. A JOIN clause is a part of a SELECT statement; it allows you to specify multiple tables for data retrieval.



The syntax of a JOIN query is the following:

```
SELECT … from TABLE table1 JOIN table2 ON table1.id = table2.t1_id
```

You just specify the tables to join and based on which columns to find what rows correspond to each other.

There are several types of JOINs, but here are three most frequently used.

- INNER JOIN – fetch the results that exist in both tables
- LEFT OUTER – fetch all rows from the table A, even if they do not exist in table B so that the result set will have half-populated wors.
- RIGHT OUTER – the opposite: fetch all rows from table B, even when the corresponding data in table A are absent.

When you don't know how to JOIN the tables correctly, you can use subqueries instead. A subquery is a SELECT query specified in the body of another:

```
SELECT col1, col2 FROM table1 WHERE id IN
  (SELECT t1_id FROM table2 WHERE date > CURRENT_TIMESTAMP)
```

Subqueries work best when combined with the IN clause in the outer select. Typically you'll fetch the ids that correspond to the entries you want in the subquery and process them on the outer level.

Besides just fetching the data, JOINs can make updating your data easier. Indeed, you can use the JOIN clause in the UPDATE query and filter which entries to update based on the data in the joined tables.

```
UPDATE t1 SET a = 1
FROM table1 t1 JOIN table2 t2 ON t1.id = t2.t1_id
WHERE t1.col1 = 0 and t2.col2 is NULL;
```

Even despite the restrictions (like you can only update one table this way, not both, and that the rows have to be unambiguously identifiable for this to work) this approach is quite useful to know about.

Also, make sure to check if your database supports a special syntax for using joins in the update statements, as it will likely improve your performance.

Since we mentioned performance, we should say that you should also care about how your code uses the database. While it's typical to monitor the performance in later stages of the development process, you can catch most mistakes early on, right when you're developing your application.

Use a tool like XRebel — the only performance tool for Java developers — and see how you can use it to find database related issues as early as possible in the short video below.

Error -118 when loading url https://player.vimeo.com/video/172445994

Useful SQL functions to remember

There is one more thing, on top of querying the values that sit in the database, you can specify the functions that will transform those values into something more useful. There are quite a few of these, and you can write your own utility functions. However, here's a taste of what you can expect from any database.

- **TO_DATE** – converts a string to date. SQL result set is typed, so if you need to use a **BETWEEN** clause, you'd need to convert your string dates to proper date types.
- **COALESCE** – return the first non-NULL results, use it like COALESCE(col1, 'default value') when querying from the columns that can contain NULLs.
- **CURRENT_TIMESTAMP** – returns the current time on the database server.
- **COUNT** – an aggregate function that returns the number of rows in the results set.
- **SUM** – an aggregate function to cumulate the values in the results set.
- **AVG** – an aggregate function to compute the mean average of the values in the results set.
- **MIN / MAX** – aggregate function to return the smallest / largest value among the results.

On top on that, you can use the set operations on the returned results. You can use UNION to append the results of one query to another:

```
SELECT col1, col2 FROM table1
UNION
SELECT col3, col4 FROM table2;
```

The UNION operation will not allow the duplicates, UNION ALL would append all the results even if there are duplicate rows. There is also the EXCEPT operation which will return the difference between the result sets and INTERSECT – for finding the intersections of the results.

Conclusion

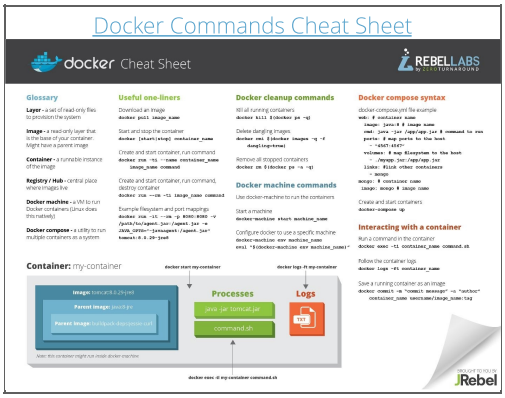
We've talked a lot about the basic SQL commands, including how different joins work and when to substitute them with a subquery. We also looked at a number of useful utility functions and how to perform set operations on the results of your queries.

Hopefully you liked this post and the cheat sheet that tries to give you the information about SQL on a single printable A4 sized piece of paper.

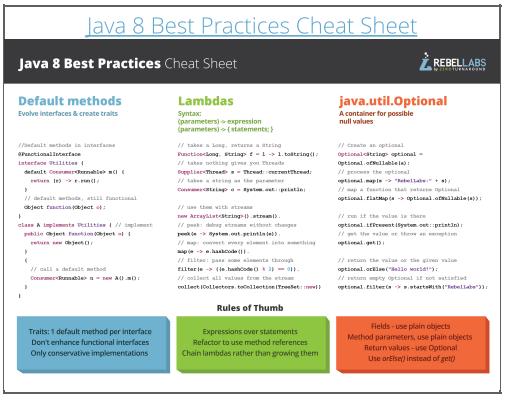
TAKE ME TO THE SQL CHEAT SHEET!

Read next:

Docker Commands Cheat Sheet



Java 8 Best Practices Cheat Sheet



OLEG SHELAJEV
Developer Advocate

Oleg Šelajev is an engineer, author, speaker, lecturer and advocate at ZeroTurnaround. He spends his time testing, coding, writing, giving conference talks, crafting blogposts and reports. He is pursuing a PhD on Dynamic System updates and code evolution. Oleg is a part-time lecturer at the University of Tartu and enjoys speaking and participating in Java/JVM development conferences such as JavaOne, JavaZone, JFokus and others. In his free time, Oleg plays chess at a semi-grandmaster level, loves puzzles and solving all kinds of problems.

cheat sheet, database, guide, RebelLabs, SQL

Company

- Careers
- Contact Us
- Our story

Resources

- JRebel White Paper (PDF)
- Devs Productivity Report
- IT Ops & DevOps Productivity Report
- Rocket powered Java development
- XRebel ROI White Paper