



SELECT queries

To retrieve data from a SQL database, we need to write **SELECT** statements, which are often colloquially referred to as *queries*. A query in itself is just a statement which declares what data we are looking for, where to find it in the database, and optionally, how to transform it before it is returned. It has a specific syntax though, which is what we are going to learn in the following exercises.

As we mentioned in the introduction, you can think of a table in SQL as a type of an entity (ie. Dogs), and each row in that table as a specific *instance* of that type (ie. A pug, a beagle, a different colored pug, etc). This means that the columns would then represent the common properties shared by all instances of that entity (ie. Color of fur, length of tail, etc).

And given a table of data, the most basic query we could write would be one that selects for a couple columns (properties) of the table with all the rows (instances).

Select query for a specific columns

```
SELECT column, another_column, ...  
FROM mytable;
```

The result of this query will be a two-dimensional set of rows and columns, effectively a copy of the table, but only with the columns that we requested.

If we want to retrieve absolutely all the columns of data from a table, we can then use the asterisk (*) shorthand in place of listing all the column names individually.

Select query for all columns

```
SELECT *  
FROM mytable;
```

This query, in particular, is really useful because it's a simple way to inspect a table by dumping all the data at once.

Exercise

We will be using a database with data about some of Pixar's classic movies for most of our exercises. This first exercise will only involve the **Movies** table, and the default query below currently shows all the properties of each movie. To continue onto the next lesson, alter the query to find the exact information we need for each task

Exercise 1 — Tasks

1. Find the **title** of each film
2. Find the **director** of each film
3. Find the **title** and **director** of each film
4. Find the **title** and **year** of each film
5. Find **all** the information about each film



Queries with constraints (Pt. 1)

Now we know how to select for specific columns of data from a table, but if you had a table with a hundred million rows of data, reading through all the rows would be inefficient and perhaps even impossible.

In order to filter certain results from being returned, we need to use a **WHERE** clause in the query. The clause is applied to each row of data by checking specific column values to determine whether it should be included in the results or not.

Select query with constraints

```
SELECT column, another_column, ...
FROM mytable
WHERE condition
      AND/OR another_condition
      AND/OR ...;
```

More complex clauses can be constructed by joining numerous **AND** or **OR** logical keywords (ie. num_wheels >= 4 AND doors <= 2). And below are some useful operators that you can use for numerical data (ie. integer or floating point):

Operator	Condition	SQL Example
=, !=, < <=, >, >=	Standard numerical operators	col_name != 4
BETWEEN ... AND ...	Number is within range of two values (inclusive)	col_name BETWEEN 1.5 AND 10.5
NOT BETWEEN ... AND ...	Number is not within range of two values (inclusive)	col_name NOT BETWEEN 1 AND 10
IN (...)	Number exists in a list	col_name IN (2, 4, 6)
NOT IN (...)	Number does not exist in a list	col_name NOT IN (1, 3, 5)

In addition to making the results more manageable to understand, writing clauses to constrain the set of rows returned also allows the query to run faster due to the reduction in unnecessary data being returned.

Did you know?

As you might have noticed by now, SQL doesn't *require* you to write the keywords all capitalized, but as a convention, it helps people distinguish SQL keywords from column and tables names, and makes the query easier to read.

Exercise

Using the right constraints, find the information we need from the **Movies** table for each task below.

Exercise -Tasks

1. Find the movie with a row **id** of 6
2. Find the movies released in the **years** between 2000 and 2010
3. Find the movies **not** released in the **years** between 2000 and 2010
4. Find the first 5 Pixar movies and their release **year**



Queries with constraints (Pt. 2)

When writing **WHERE** clauses with columns containing text data, SQL supports a number of useful operators to do things like case-insensitive string comparison and wildcard pattern matching. We show a few common text-data specific operators below:

Operator	Condition	Example
=	Case sensitive exact string comparison (notice the single equals)	col_name = "abc"
!= or <>	Case sensitive exact string inequality comparison	col_name != "abcd"
LIKE	Case insensitive exact string comparison	col_name LIKE "ABC"
NOT LIKE	Case insensitive exact string inequality comparison	col_name NOT LIKE "ABCD"
%	Used anywhere in a string to match a sequence of zero or more characters (only with LIKE or NOT LIKE)	col_name LIKE "%AT%" (matches "AT", "ATTIC", "CAT" or even "BATS")
_	Used anywhere in a string to match a single character (only with LIKE or NOT LIKE)	col_name LIKE "AN_" (matches "AND", but not "AN")
IN (...)	String exists in a list	col_name IN ("A", "B", "C")
NOT IN (...)	String does not exist in a list	col_name NOT IN ("D", "E", "F")

Did you know?

All strings must be quoted so that the query parser can distinguish words in the string from SQL keywords.

We should note that while most database implementations are quite efficient when using these operators, full-text search is best left to dedicated libraries like [Apache Lucene](#) or [Sphinx](#). These libraries are designed specifically to do full text search, and as a result are more efficient and can support a wider variety of search features including internationalization and advanced queries.

Exercise

Here's the definition of a query with a **WHERE** clause again, go ahead and try and write some queries with the operators above to limit the results to the information we need in the tasks below.

Exercise 3 – Tasks

1. Find all the Toy Story movies
2. Find all the movies directed by John Lasseter
3. Find all the movies (and director) not directed by John Lasseter
4. Find all the WALL-* movies