WBS CODING SCHOOL

MySQL basics cheat sheet

Data Retrieval
- SELECT
- FROM

Joining tables
- LEFT JOIN, RIGHT JOIN
  - LEFT JOIN
  - RIGHT JOIN
- JOIN, INNER JOIN, CROSS JOIN
- ON, USING

Sorting
- ORDER BY

Filtering
- WHERE
- HAVING
- LIMIT
- CASE… WHEN…
  - AND, OR
  - LIKE
  - IN
  - IS NULL
  - =, !=, <>, >, >=, <, <=
  - BETWEEN… AND…
  - DISTINCT

Grouping
- GROUP BY

Aggregates: AVG(), COUNT(), MIN(), MAX(), SUM(), ROUND()
- DATEDIFF()

Aliases

Wildcards
- *
- %
- _

Comments
- Single line comments
- Multi-line comments

;

Written order vs order of execution

The MySQL keywords are written in uppercase by convention. It is not, however, required. Because MySQL is case-insensitive, you can use either lowercase or uppercase in your MySQL statement.

Data Retrieval

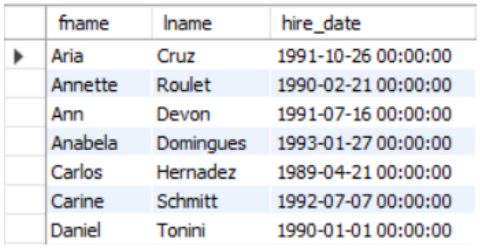
SELECT

You can use the SELECT statement to select data from one or more tables. Each element you place in the SELECT statement forms a column of the output table. Elements are separated by a comma (,).

FROM

Generally, SQL databases are composed of multiple tables. FROM is used to state which table you would like to select the information from.

```
SELECT
        fname,
        lname,
        hire_date
FROM
        Employee;
```



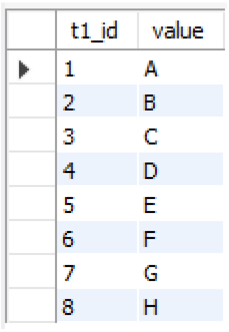
fname	lname	hire_date
Aria	Cruz	1991-10-26 00:00:00
Annette	Roulet	1990-02-21 00:00:00
Ann	Devon	1991-07-16 00:00:00
Anabela	Domingues	1993-01-27 00:00:00
Carlos	Hernadez	1989-04-21 00:00:00
Carine	Schmitt	1992-07-07 00:00:00
Daniel	Tonini	1990-01-01 00:00:00

Joining tables

Two words of caution before we proceed. Firstly, we are using MySQL, there are other types of SQL such as PostgreSQL, Oracle, MariaDB etc… The succeeding rules for joins apply only to MySQL, please read the appropriate documentation if you use a different type of SQL in the future. Secondly, there is a lot of bad information on the internet about joins where people confuse how joins work across different types of SQL (there is no OUTER JOIN in MySQL). Here we have taken the information straight from the official MySQL documentation, and used clear examples to highlight how joins function in MySQL.

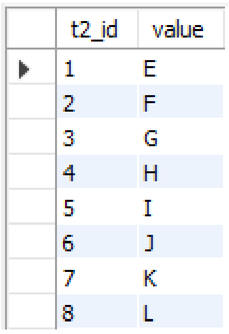
To help us explain how joins work we will use the two tables shown below.

```
SELECT *
FROM table_1;
```



t1_id	value
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H

```
SELECT *
FROM table_2;
```



t2_id	value
1	E
2	F
3	G
4	H
5	I
6	J
7	K
8	L

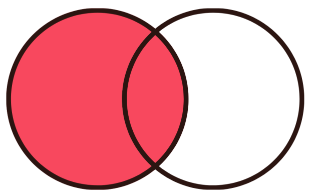
Table_1 values are from A-H. Table_2 values are from E-L. The points where these tables have the same values (E, F, G, H) will exemplify how JOINs work

LEFT JOIN, RIGHT JOIN

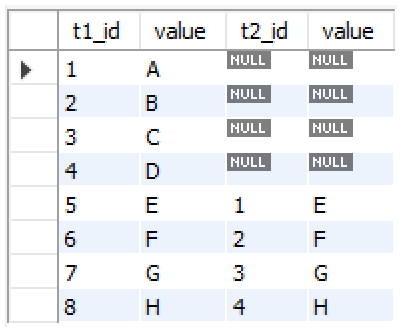
LEFT JOIN and RIGHT JOIN are very similar. They perform the same function but in the opposite direction.

LEFT JOIN

A left join selects all data from the left table, regardless of whether or not there are matching rows in the right table. And selects only those rows from the right table that have a matching row in the left table.



```
SELECT
          *
FROM
          table_1
LEFT JOIN
          table_2
ON
          table_1.value = table_2.value;
```

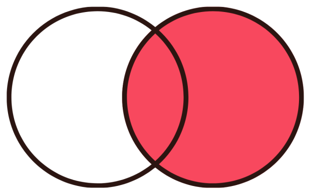


t1_id	value	t2_id	value
1	A	NULL	NULL
2	B	NULL	NULL
3	C	NULL	NULL
4	D	NULL	NULL
5	E	1	E
6	F	2	F
7	G	3	G
8	H	4	H

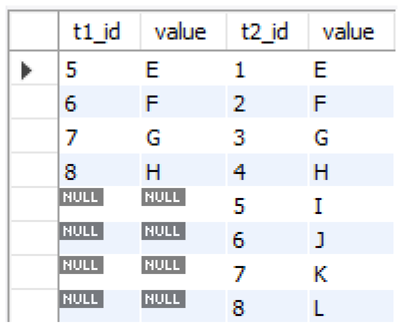
As you can see from the example above, all of the rows from table_1 were kept. And only the rows from table_2, where table_2 value is the same as table_1 value, were kept. Those rows where table_2 value did not have an equivalent table_1 value were lost.

RIGHT JOIN

A right join selects all data from the right table, regardless of whether or not there are matching rows in the left table. And selects only those rows from the left table that have a matching row in the right table.



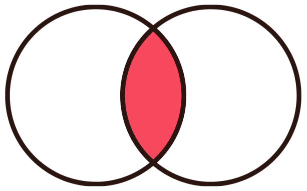
```
SELECT
          *
FROM
          table_1
RIGHT JOIN
          table_2
ON
          table_1.value = table_2.value;
```



t1_id	value	t2_id	value
5	E	1	E
6	F	2	F
7	G	3	G
8	H	4	H
NULL	NULL	5	I
NULL	NULL	6	J
NULL	NULL	7	K
NULL	NULL	8	L

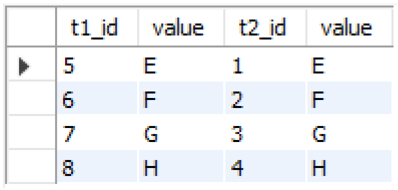
As you can see from the example above, all of the rows from table_2 were kept. And only the rows from table_1, where table_1 value is the same as table_2 value, were kept. Those rows where table_1 value did not have an equivalent table_2 value were lost.

JOIN, INNER JOIN, CROSS JOIN



These three options perform the same join. They are syntactically equivalent. They select only the data from both tables that have a matching row in the other table.

```
SELECT
        *
FROM
        table_1
JOIN
        table_2
ON
        table_1.value = table_2.value;
```



t1_id	value	t2_id	value
5	E	1	E
6	F	2	F
7	G	3	G
8	H	4	H

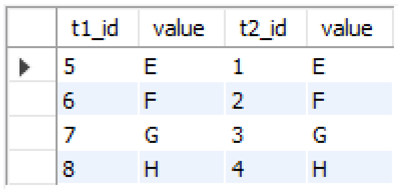
As you can see from the example above, only the rows from table_1 and table_2, where table_1 value and table_2 value are the same, were kept. Those rows where the other table did not have an equivalent value were lost.

If we replaced the word JOIN with either INNER JOIN, or CROSS JOIN we would end up with the same table as a result.

ON, USING

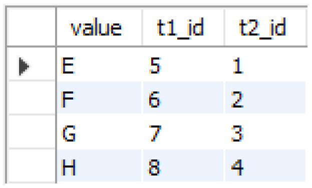
Both of these keywords can be used to say which columns you would like to join the tables on. USING requires the columns to have the same name. If the columns you are joining have different names, you must use ON.

```
SELECT
        *
FROM
        table_1
JOIN
        table_2
ON
        table_1.value = table_2.value;
```



t1_id	value	t2_id	value
5	E	1	E
6	F	2	F
7	G	3	G
8	H	4	H

```
SELECT
        *
FROM
        table_1
JOIN
        table_2
USING
        (value);
```



	value	t1_id	t2_id
▶	E	5	1
	F	6	2
	G	7	3
	H	8	4

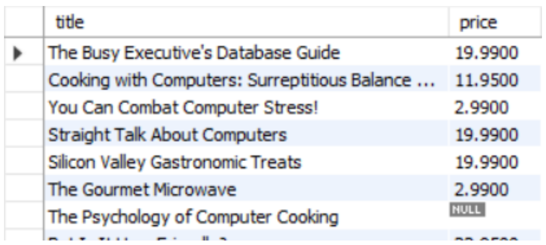
Notice that when we use the keyword USING, the two columns titled 'value' merge into one. Whereas, when we use the keyword ON, the two 'value' columns stay separate. Depending on how your original tables look, and how you wish your results to be presented, will determine whether you decide to use ON or USING.

Sorting

ORDER BY

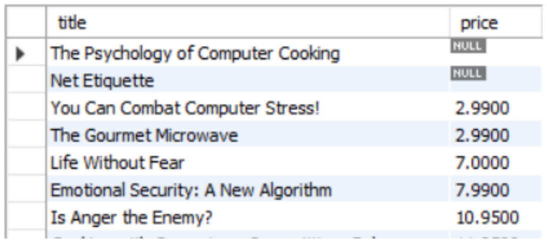
The order of rows in the result set is unspecified when you use the SELECT statement. The ORDER BY clause is used to sort the rows.

```
SELECT
        title,
        price
FROM
        titles;
```



	title	price
▶	The Busy Executive's Database Guide	19.9900
	Cooking with Computers: Surreptitious Balance ...	11.9500
	You Can Combat Computer Stress!	2.9900
	Straight Talk About Computers	19.9900
	Silicon Valley Gastronomic Treats	19.9900
	The Gourmet Microwave	2.9900
	The Psychology of Computer Cooking	NULL

```
SELECT
        title,
        price
FROM
        titles
ORDER BY
        price;
```



	title	price
▶	The Psychology of Computer Cooking	NULL
	Net Etiquette	NULL
	You Can Combat Computer Stress!	2.9900
	The Gourmet Microwave	2.9900
	Life Without Fear	7.0000
	Emotional Security: A New Algorithm	7.9900
	Is Anger the Enemy?	10.9500

ASC: ascending - 1, 2, 3, 4, 5 - or - A, B, C, D, E
DESC: descending - 5, 4, 3, 2, 1 - or - E, D, C, B, A

If you write neither ASC nor DESC, then MySQL will use the default option of ASC. Which means that both the options below produce the same output.

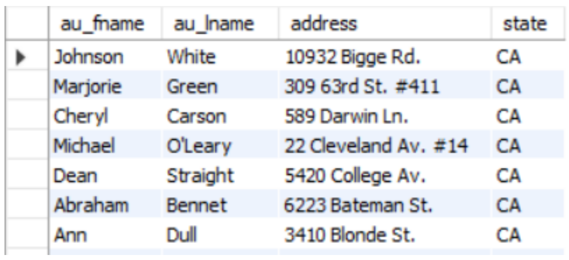
```
ORDER BY price ASC
ORDER BY price
```

Filtering

WHERE

You can provide a filtering condition for the rows returned using the WHERE clause. The condition must be Boolean, so the TRUE results (those that meet our condition) are included, and the FALSE results are excluded.

```
SELECT
        au_fname,
        au_lname,
        address,
        state
FROM
        authors
WHERE
        state = 'CA';
```

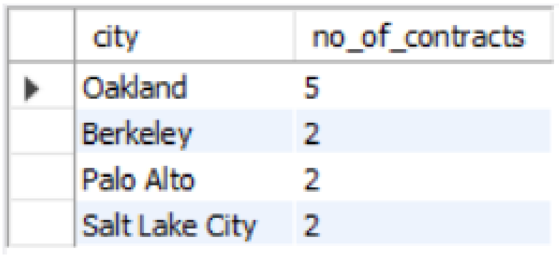


au_fname	au_lname	address	state
Johnson	White	10932 Bigge Rd.	CA
Marjorie	Green	309 63rd St. #411	CA
Cheryl	Carson	589 Darwin Ln.	CA
Michael	O'Leary	22 Cleveland Av. #14	CA
Dean	Straight	5420 College Av.	CA
Abraham	Bennet	6223 Bateman St.	CA
Ann	Dull	3410 Blonde St.	CA

HAVING

To set a filter condition for groups of rows (GROUP BY) or aggregates (SUM, AVG, etc…), use the HAVING clause.

```
SELECT
        city,
        COUNT(contract) AS
no_of_contracts
FROM
        authors
GROUP BY
        city
HAVING
        no_of_contracts >= 2;
```

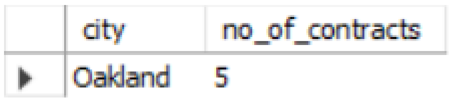


city	no_of_contracts
Oakland	5
Berkeley	2
Palo Alto	2
Salt Lake City	2

LIMIT

To limit the number of rows returned by the SELECT statement, use the LIMIT clause.

```
SELECT
        city,
        COUNT(contract) AS no_of_contracts
FROM
        authors
GROUP BY
        city
HAVING
        no_of_contracts >= 2
LIMIT 1;
```

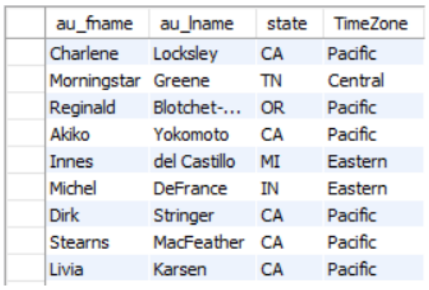


city	no_of_contracts
Oakland	5

CASE… WHEN…

A CASE statement is a way of adding an extra column to a dataframe, which can be useful to describe or help divide the data. As we are adding an extra column, CASE statements form part of the SELECT statement and come before the FROM statement.

```
SELECT
        au_fname,
        au_lname,
        state,
        CASE
                WHEN state IN ('CA', 'OR') THEN 'Pacific'
                WHEN state IN ('KS', 'UT') THEN 'Mountain'
                WHEN state IN ('TN') THEN 'Central'
                WHEN state IN ('MI', 'IN', 'MD') THEN 'Eastern'
                ELSE 'Unknown'
        END AS TimeZone
FROM authors;
```



au_fname	au_lname	state	TimeZone
Charlene	Locksley	CA	Pacific
Morningstar	Greene	TN	Central
Reginald	Blotchet-...	OR	Pacific
Akiko	Yokomoto	CA	Pacific
Innes	del Castillo	MI	Eastern
Michel	DeFrance	IN	Eastern
Dirk	Stringer	CA	Pacific
Stearns	MacFeather	CA	Pacific
Livia	Karsen	CA	Pacific

AND, OR

It is possible to place more than one condition in a WHERE or HAVING clause. These conditions can be daisy chained together with the operators AND, or OR.

An AND operator means that for a row to be returned it must satisfy all the conditions.

```
WHERE
        state = 'CA'
        AND city = 'Berkeley'
```

This clause will return only the rows that contain the city ‘Berkeley’ and the state ‘CA’

An OR operator means that for a row to be returned it must satisfy only one condition.

```
WHERE
        state = 'CA'
        OR state = 'UT'
```

This clause will return all of the rows that have state ‘CA’ and all of the rows that have state ‘UT’

LIKE

A LIKE operator is a case insensitive search, it ignores whether a letter is capitalised or not. It also accepts regular expressions if you wish to search for patterns as opposed to exact strings.

```
WHERE
        fname LIKE 'Toulo'
```

This clause will return any rows where fname is equal to Toulo, toulo, toUlo, or any other combination of capital and lowercase letter

IN

If any value in a list matches a value, the IN operator returns TRUE.

```
WHERE
        state IN ('CA', 'UT')
```

This clause will return any row where state is either ‘CA’ or ‘UT’

IS NULL

If a value is NULL, the IS NULL operator returns TRUE.

```
WHERE
        address IS NULL
```

This clause will return any row where address is null (i.e. has no value)

=, !=, <>, >, >=, <, <=

=	Equal to	!=	Not equal to	>	More than	>=	More than or equal to
		<>	Not equal to	<	Less than	<=	Less than or equal to

```
WHERE
        price >= 50
```

This clause will return any row where the price is 50 or higher

BETWEEN… AND…

This operator gives you a range of values between the two given values. It is the same as saying greater than or equal to and less than or equal to.

```
WHERE
        price BETWEEN 1 AND 5
```

This clause will return any row where price is 1, 2, 3, 4, or 5

DISTINCT

While DISTINCT filters the content, it is implemented in a slightly different way to all of the above operators. This operator appears in the SELECT statement and is used to remove duplicate rows: each value will only appear once (similar to a GROUP BY). It will affect all columns in the SELECT clause.

```
SELECT
        DISTINCT city
FROM
        authors;
```

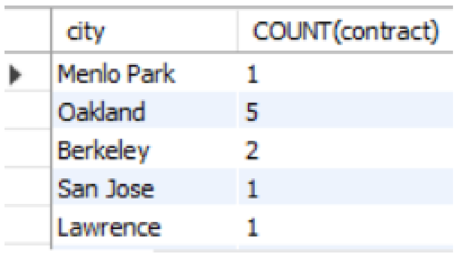
This statement will return each city only once, even if the city appears on multiple rows as multiple authors live there

Grouping

GROUP BY

For each given group, the GROUP BY clause returns one row.

```
SELECT
        city,
        COUNT(contract)
FROM
        authors
GROUP BY
        city;
```



city	COUNT(contract)
Menlo Park	1
Oakland	5
Berkeley	2
San Jose	1
Lawrence	1

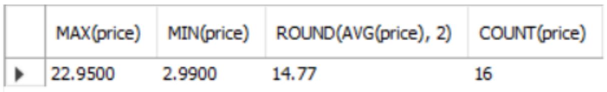
P.s. With aggregate functions, the GROUP BY clause is frequently used.

Aggregates

An aggregate function performs a calculation on multiple values and returns a single value.

AVG()	Average of all non-null values	MAX()	Highest value in the column	SUM()	Summation of all value in the column
COUNT()	The total number of rows in the column	MIN()	Lowest value in the column	ROUND()	Rounds to a number of decimal places

```
SELECT
        MAX(price),
        MIN(price),
        ROUND(AVG(price)),2)
        COUNT(price)
FROM
        titles;
```



MAX(price)	MIN(price)	ROUND(AVG(price), 2)	COUNT(price)
22.9500	2.9900	14.77	16

DATEDIFF()

The DATEDIFF function takes two inputs, each of which can be any valid date or date-time value. When DATETIME or TIMESTAMP values are passed to the DATEDIFF function, the function just calculates the date sections and ignores the time parts.

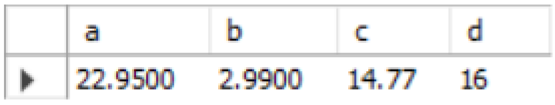
```
SELECT
        DATEDIFF('2022-01-01',
        '2022-01-09');
```

This statement will return 8. As there are 8 days difference between the dates

Aliases

It is possible to give both columns and tables aliases. This can make our lives easier by shortening a long name or by giving something a more suitable name.

```
SELECT
        MAX(price) AS a,
        MIN(price) b,
        AVG(price) AS c,
        COUNT(price) d
FROM
```



a	b	c	d
22.9500	2.9900	14.77	16

```
        titles AS df;
```

The AS keyword is optional, you can omit it.

It's worth noting that a column alias can't be used in the WHERE clause. The reason for this is that MySQL examines the WHERE clause before the SELECT clause: it therefore won't know of the alias. Please see the order of operations below to see in what order SQL runs your statements.

Wildcards

*

The asterisk (*) which is the shorthand for all columns

```
SELECT
        *
FROM
        authors;
```

This statement will return all of the columns from the authors table

%

The percentage symbol (%) represents zero or more characters

```
WHERE
        last_name LIKE '%son'
```

This clause will return any row where the last name ends in the three letters 'son'. So this would include 'Abramson', 'Lawson', 'Axelson' etc

_

The underscore represents a single character

```
WHERE
        age LIKE '4_'
```

This clause will return any row where the age begins with the number 4 and is succeeded by any other character. So this would include 40, 45, 49, and 4B

Comments

Comments are used to explain the code. They are not executed by the function

Single line comments

If you are only making a short note then a single line comment is perfect. They are denoted by 2 dashes (--)

```
– example of a single line comment
```

Multi-line comments

When you need more room for your notes, or to comment out a few lines of code so that they don't run, it's best to use a multi line comment. They open with a forward slash followed by a star (/*) and close with a star followed by a forward slash (*/)

```
/* This is a multiline comment.
None of this code will be executed */
```

;

The semicolon is not essential. It indicates the end of a statement. If you have more than one statement, you must use a semicolon to separate them so each one can be executed separately.

Written order vs order of execution

Many people believe MySQL operates from top to bottom like other programming languages, but it does not - MySQL is designed to be read by humans. Because of this, many scripts that appear to be rational can cause errors. For example, using an alias in a WHERE clause or MAX(price) in a HAVING clause sounds sensible. If you look at the sequence of execution, you'll notice that WHERE and HAVING are evaluated before the SELECT query.

Written order	Execution order
SELECT	FROM
FROM	WHERE
WHERE	GROUP BY
GROUP BY	HAVING
HAVING	SELECT
ORDER BY	ORDER BY
LIMIT	LIMIT