

STRUCTURED QUERY LANGUAGE (SQL) : SELECT

SQL is a standard language for storing, manipulating, and retrieving information from relational databases. This sheet covers standard functionality across all systems.

GENERAL FORMAT

SELECT {columns, aggregate functions, or subqueries}

FROM {tables or subqueries}

WHERE {filters based on columns or subqueries}

GROUP BY {columns to aggregate by}

HAVING {filters based on aggregates}

ORDER BY {specific columns or aggregate functions}

SELECT

Choose the columns, aggregates, and calculations that you would like shown in the results of your queries. Separate the columns using commas.

Select	Using	Example
All columns	*	SELECT *
Specific columns	column or table.column	SELECT firstname, students.age
Rename columns	column AS newname	SELECT lastname AS surname

The following aggregate functions are available, but require the use of the **GROUP BY** function to define the non-aggregate columns:

MIN(), MAX(), SUM(), AVG(), COUNT()

```
SELECT firstname, COUNT(firstname) FROM students GROUP BY firstname
```






Subqueries that return a single column and single row can be used by enclosing the entire subquery in brackets:

```
SELECT firstname, COUNT(firstname) / (SELECT COUNT (firstname) FROM students) FROM students GROUP BY firstname
```

Calculations can be created by using basic mathematical operators to add, subtract, multiply and / or divide any numerical columns.

FROM

Choose the tables that your columns are found in. Just like columns, use AS to alias (rename) your tables. If you are using multiple tables, you need to join them together using one or more columns..

Join Type	Illustration	Example
Left join		FROM table_a AS a LEFT JOIN table_b AS b ON a.key = b.key
Right join		FROM table_a AS a RIGHT JOIN table_b AS b ON a.key = b.key
Inner join		FROM table_a AS a INNER JOIN table_b AS b ON a.key = b.key
Full outer join		FROM table_a AS a FULL OUTER JOIN table_b AS b ON a.key = b.key
Cartesian join		FROM table_a CROSS JOIN table_b

Join using multiple columns by using **AND**, and use comparison and logical operators to add filtering conditions directly to the join.

WHERE

Filter the results of your query based on the initial columns by using a mix of comparison and logical operators. Use brackets to change the default precedence of logical operators.

LOGICAL OPERATORS

(Listed in order of precedence)

Operator	Example	Result
NOT	NOT a	TRUE if A is FALSE
OR	a OR b	TRUE if A or B are TRUE
AND	a AND b	TRUE if A and B are TRUE

COMPARISON OPERATORS

Comparison	Operator	Example
Equal to	=	WHERE age = 20
Not equal to	<>	WHERE age <> 20
Greater than	>	WHERE age > 20
Greater than or equal to	>=	WHERE age >= 20
Less than	<	WHERE age < 20
Less than or equal to	<=	WHERE age <= 20
Between (inclusive)	BETWEEN 1 AND 5	WHERE age BETWEEN 18 AND 22
In list of values or subquery	IN (1, 3, 5)	WHERE age IN (18, 19, 20, 21, 22)
Partial string matching	LIKE '%X%Y%Z%'	WHERE firstname LIKE 'Tom%'

GROUP BY

Choose the columns that aggregation will be performed across. (ie, all the non-aggregate columns) Separate the columns with commas:

```
SELECT city, province, count(studentnumber) FROM students GROUP BY city, province
```

HAVING

Filter the query using the results of aggregate functions by using comparison and logical operators.

```
HAVING AVG(grade) <= 3.5
```

ORDER BY

Reorder the results of your query by specifying columns and a sort direction (ASC for ascending order or DESC for descending order), separated by commas.

```
ORDER BY age ASC, AVG(grade) DESC
```

