**SQL Summary – Version 10 for Ms Access. Numbered for ease of reference. Grade 10 and 11**

|  |  |
| --- | --- |
| **Structure of the language** | **SQL operators** |
| **SELECT <field(s)>****FROM <table(s)>****WHERE condition(s)****ORDER BY expression** |

|  |  |
| --- | --- |
| **Arithmetic****Comparison****Logical****Concatenation****Special** | **+ - \* / MOD ^****< <= > >= = (equal) <> (not equal)****AND, OR, NOT****&** (joining text)**Is NULL** or **is NOT NULL, LIKE, BETWEEN val1 and Val2, IN ( , , , )** |

 |

**SELECT ALL fields using the wildcard**

1 SELECT \* FROM tablename . . . “ \* “ wildcard selects everything – all fields.

“SELECT” and “FROM” are the only compulsory keywords

**SELECT. What you are looking for — fields, the answer to a calculation, a random number, the date, the time**

2 SELECT field1, field2, field3 FROM tablename . . . the separator is the comma

3 SELECT field, (Math + Science), date( ), now( ) – dateOfTest, FROM tblLearnerTests

**ORDER BY – ASC or DESC. ASC is default – smallest first, largest last. A first, Z last.**

4 SELECT \* FROM tblName ORDER BY LastName

5 SELECT \* FROM tblName ORDER BY price DESC . . . the highest price is first

6 SELECT \* FROM tblInventory ORDER BY price, make . . . cheapest to most expensive in alphabetical order by make

7 SELECT \* FROM tblLearners ORDER BY dateOfBirth . . . the “bigger” the date, the younger the person is

8 SELECT \* FROM tblInvoices ORDER BY ((Now()-invoicedate)/365.25)

**LIKE – When you are searching for something . . . but you only have an idea, or a pattern. You only use LIKE with the wildcard**

9 SELECT \* FROM tblName WHERE lastName LIKE “W\*” . . . starts with a “W”

10 SELECT \* FROM tblName WHERE lastName LIKE “\*van\*” . . . . has “van” in the middle

Do not use LIKE when you know exactly what you are looking for.

**THE BEST . . . THE 5 BEST . . . THE WORST . . . THE HIGHEST . . . THE MOST (EXPENSIVE) (CHEAPEST) . . . EARLIEST . . . LATEST**

11 SELECT **TOP 1** \* FROM tblName ORDER BY ASC / SELECT **TOP 5** \* FROM ORDER BY DESC.

12 SELECT TOP 20 \* FROM tblPrintLogs ORDER BY TotalColourPages DESC

13 SELECT TOP 1 surname FROM tblPrintLogs ORDER BY HOUR(date) DESC

14 SELECT TOP 1 lastName, firstName FROM tblPrintLogs WHERE email LIKE “\*student\*” ORDER BY cost DESC

**SELECT ONLY CERTAIN FIELDS AND THEN SORT BY A SPECIFIC FIELD**

15 SELECT field1, field2, field3, fieldN FROM tblName ORDER BY Age

**SELECT VALUES THAT ARE UNIQUE**

16 SELECT DISTINCT fields FROM tblName ORDER BY . . . . e.g. SELECT DISTINCT LastName FROM tblStudents

**PERFORM ARITHMETIC ON THE SELECTED FIELDS AND THEN NAME THIS “NEW” FIELD (called an Alias)**

17 SELECT fields calculation FROM ORDER BY . . . . e.g. SELECT Name, (Math + Science + IT) AS [Total marks] FROM tblStudents

**USE ARITHMETIC FUNCTION TO FORMAT A RESULT e.g. rounding to 1 decimal place**

18 SELECT fields arithmetic function(calculation) . . . .e.g. SELECT Name ROUND((Math + IT) /2, 1) AS [Average Mark] FROM tblStudents

19 SELECT fields calculations (MOD) AS alias . . . . . e.g. SELECT totalPages MOD 4 AS [Total

**STRING MANIPULATION TO EXTRACT A PORTION OF A LONGER STRING e.g. finding a person’s initial**

20 SELECT string manipulation (field) & field & field . . . . e.g. SELECT LEFT(firstName,1)

**EXTRACT A PORTION OF A DATE - YEAR, MONTH, DAY FROM A FIELD THAT HAS THE DATA TYPE OF “Date”**

21 SELECT date or time(field) . . . e.g. SELECT YEAR(DOB) . . . . SELECT MONTH(DOB) . . . .

**PERFORMING AGGREGATE FUNCTIONS ON ALL THE VALUES IN A PARTICULAR FIELD; YEILDS A SINGLE VALUE.**

**Aggregate functions;** MAX(), MIN(), AVG(), SUM(), COUNT()

22 SELECT aggregate function(field) . . . e.g. SELECT MIN(SizeKB) . . . . SELECT AVG(TotalPages) . . . . SELECT SUM(Cost)

**NOTE:** Aggregate functions only return a value – no other details.

**WHERE - COMPARING VALUES IN A PARTICULAR FIELD TO A PARTICULAR CONDITION e.g. larger than 5**

23 SELECT fields WHERE condition = < > <= >= <> . . . . e.g. SELECT surname FROM PrintLogs WHERE TotalPages > 5 . . . .

25 SELECT Name FROM tblStudents WHERE Address1 IS NULL

26 SELECT \* FROM tblName WHERE ModelNumber LIKE “XC-450??” – wildcard ? for a single character

**DETERMINING AGE AND DISPLAYING AGE FROM DATE OF BIRTH**

27 SELECT Name, YEAR(NOW ( ) ) – YEAR(DOB) AS Age. This version is **not** accurate

28 SELECT name, ROUND(((NOW( ) - DOB)/365.25),1) AS [Age] FROM tblLearners. This version is accurate

**COMPARING AGES TO A PARTICULAR CONDITION e.g. larger and equal to 30**

29 SELECT fields FROM WHERE calculation condition . . . e.g. SELECT fields WHERE YEAR(NOW()) – YEAR(DOB) <= 30

**COMPARING AGES TO MORE THAN ONE CONDITION e.g. larger than March but smaller than June**

30 SELECT fields WHERE condition AND condition . . . e.g. SELECT fields WHERE MONTH(DOB) BETWEEN 3 AND 6.

**DETERMINGING THE AVERAGE AGE FROM DATE OF BIRTH**

31 SELECT Name, AVG( NOW( ) ) – YEAR(DOB)) AS [Average Age]. Discouraged as the age may not be exact

32 SELECT Name, AVG(NOW( ) - DOB)/365.25)AS {Average Age]. This is accurate.

**PERFORMING A SIMPLE CALCULATION ON FIELDS IN THE SAME RECORD**

33 SELECT (TotalPages \* Copies) . . . . . SELECT (Maths + Science + IT) . . . .

SELECT(price \* 1.05) . . . this increases the price by 5%

38 **Arithmetic function**: INT( ), ROUND( ). Formats the single parameter within the brackets. ROUND rounds up or down. INT does not round.

 ROUND(((Now()-invoicedate)/365.25),3) . . . this rounds to 3 decimal places. Take care with the round brackets.

**RND( seed goes here)** – Generating a random number

A) Using a random number seed within the brackets generates a number between zero and 1.
B) Then you need to multiple it by 10, 100 or 1000 to get a real value bigger than one.
C) Then you need to add one to avoid generating a zero.
D) Then you need INT to truncate the real to an inte.g.er e.g. 0.87678902 becomes 87.678902 becomes 88.678902 becomes 88

E.g. SELECT INT (RND(riderID) \* 100 + 1) AS [Random number] FROM tblRiders. Here the “rider id” is used as the random number seed.

39 **Random numbers:** RND(Upper bound – Lower bound ) + Lower bound. To generate numbers between interval e.g. between 32 and 78 – see below

SELECT RND((riderID) + (78 - 32 ) + 78) AS [Random number] FROM tblRiders

40 **Comparison operators:** < > >= <= <> (not equal to)

41 **String manipulation:** LEFT(x), RIGHT(x), MID(x, y), LEN(field ), & - concatenation operator in Ms Access. NOTE: SQL counts from 1, not zero.

42 **Date and time:** DATE, TIME, NOW, YEAR, MONTH, DAY, TIME, HOUR, MINUTE.
NOW( ) yields the date and time from the PC. DATE( ) yields date from the PC. TIME( ) yields time from the PC.

Example: YEAR(dob) will yield the year on its own. MONTH(dob) will yield the month on its own

43 **Aggregate functions:** MAX(), MIN(), AVG(), SUM(), COUNT(). Considers the whole and returns a single result (value)

e.g. it adds all the values in a field when you SUM. Most useful when used in conjunction with GROUP BY
They do not return any details from a specific record. Example: If you need maximum and minimum values *with details* use
“TOP 1” in conjunction with “ORDER BY”.

COUNT does not count a record that has a NULL value in the specified field. The other aggregate functions ignore NULL values e.g. SUM

44 **Compound conditions**: NOT, AND, OR

 WHERE town = “Johannesburg” AND maritalstatus = 1 AND gender = 1

45 **More conditions**: BETWEEN . . AND, IN and NOT IN, LIKE, NULL

 SELECT \* FROM tblDetails WHERE town IN (“Johannesburg”, “Pretoria”, “Midrand”)

 SELECT \* FROM tblDetails WHERE town NOT IN (“Johannesburg”, “Pretoria”, “Midrand”)

 SELECT \* FROM tblTournament WHERE score BETWEEN 100 AND 200

46 **Quotes.**

Regular quotes for string data (“ “ or ‘ ‘).

Hash symbols for date/time . #2019/05/23 9:33:00#.

Boolean – no quotes.

NOTE: SQLite does not use # # for dates but instead uses ‘ ‘

**Queries that alter data in a table (insert records, delete records or edit existing records) (table structure is not altered)**

**INSERT . . . INTO . . . . VALUES**

**UPDATE . . . . SET**

**DELETE**

47 INSERT – adds a new record to a table and populates all the fields (when autonumber is not the primary key)

INSERT INTO tablename VALUES (field1Data, field2Data, field2Data) – no field names.

NOTE: The VALUES, the order of the values, and the datatypes match the table structure exactly

48 INSERT - adds a new record, specified fields . . . (when autonumber is the primary key)

INSERT INTO tablename (fieldTitle1, fieldTitle2, fieldTitle3) VALUES (field1Data, field2Data, field2Data)

E.g. INSERT INTO tblname (name, DOB, gender, grade, boarder) VALUES (‘Lynn’, #02 Feb 2000#, ‘F’, 10, false)

49 UPDATE – all . . . (the whole table, and all its records are given a new value e.g. the school gets a new name – everybody is affected.

UPDATE tablename SET field1 = value1, field2 = value2, fieldN = valueN

50 UPDATE – updates a record that matches a condition

UPDATE tablename SET field1 = value1, field2 = value2, fieldN = valueN WHERE condition

E.g. UPDATE PrintLogs SET FirstName = “Henrietta” WHERE Surname = “Bates” AND FirstName = “Henry”.

51 DELETE – all . . . (Deletes all the records in the table and cannot be undone in Ms Access. The table structure is not affected)

DELETE \* FROM tablename

52 DELETE – those that match a condition . . . (NOTE: This delete SQL command cannot be undone Ms Access)

DELETE FROM tablename WHERE fieldname = value

E.g. DELETE FROM tblStudents WHERE studentID = 38 (Use the primary key value, not the person’s name)

**More examples – SELECT**

1. SELECT \*
2. SELECT name, re.g.ion
3. SELECT name, area/population . . . (area divided by population which gives the population density)
4. SELECT ROUND(area/population, 2) . . . as above rounded to 2 decimal places
5. SELECT LENGTH(name)
6. SELECT name, LEFT(name,1)
7. SELECT name FROM tblWorldStats WHERE population > (SELECT population FROM tblWorldStats WHERE name = “Russia”)
	1. Note: A query within a query – the second SELECT must only return one value or the comparison operator cannot work.
8. SELECT INT(genderMale / totalEnrolement \* 100) . . . Percentage of male students enrolled in a colle.g.e rounded down.
9. SELECT ROUND(genderMale / totalEnrolement \* 100, 2) . . . Percentage of male students enrolled in a colle.g.e rounded to 2 decimal places
10. SELECT ROUND(RND(SizeKb) \* 5,0) . . . whole random numbers from 1 to 5 inclusive.
11. SELECT LastName & “ “ & LEFT(firstName,1) AS LastNameInit . . . last name concatenated with initial.

**More examples – WHERE**

1. WHERE area = 1000000
2. WHERE country = ‘Germany’
3. WHERE country IN (‘United Kingdom’, ‘Europe’, ‘Asia’)
4. WHERE MONTH(DOB) IN (1,2,3) . . . finds people born in the first quarter of the year
5. WHERE name LIKE “Al\*”
6. WHERE name LIKE “Al\*” OR “El\*”
7. WHERE length(name) > 10 AND region = “Europe”
8. WHERE area < 500000 AND population > 1000000
9. WHERE area BETWEEN 1000000 AND 2000000
10. WHERE nationality = ‘England” AND goalsScored BETWEEN 40 AND 50
11. WHERE subject = ‘English’ AND YEAR(publication) BETWEEN 2000 AND 2015
12. WHERE LastName BETWEEN ‘A’ AND ‘M’
13. WHERE DOB BETWEEN #01/01/2010# AND #31/12/2010#
14. WHERE MONTH(DOB) = 12 AND DAY(DOB) = 25 . . . . finds people born on Christmas Day.
15. WHERE name NOT LIKE “ \* \* ” . . . space in the middle. Names made of two separate words would not be selected.

RESOURCE: www.sqlzoo.net - useful teach, example and quiz website. Note: Does not focus on Ms Access SQL implementation – small differences

NOTES
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_