SQL Intermediate

- DISTINCT
- CASE
- GROUP BY
- HAVING
- DATE

SQL CLAUSES Wrap UP

- SELECT
- ORDER BY
- LIMIT
- WHERE
- Comparison Operators
- IN
- LIKE
- BETWEEN
- IS NULL/NOT NULL
- DISTINCT
- CASE
- GROUP BY
- Aggregation Functions (MIN, MAX, SUM, AVG, COUNT)
- HAVING
- DATE()

SQL Intermediate: DISTINCT

```
SELECT DISTINCT select_list
FROM table;
```

- The DISTINCT clause allows you to remove the duplicate rows in the result set.
    - SELECT DISTINCT (*)
    - SELECT DISTINCT(column1)
    - SELECT COUNT (DISTINCT column1)
    - SELECT COUNT (DISTINCT *)

SQL Intermediate: CASE

- CASE expression evaluates a list of conditions and returns an expression based on the result of the evaluation.
- he CASE expression is similar to the IF-THEN-ELSE statement in other programming languages.

```
CASE case_expression
     WHEN when_expression_1 THEN result_1
     WHEN when_expression_2 THEN result_2
     ...
     [ ELSE result_else ]
END
```

SQL Intermediate: CASE

- In case no case_expression matches the when_expression, the CASE expression returns the result_else in the ELSE clause. If you omit the ELSE clause, the CASE expression returns NULL.

SQL Intermediate: GROUP BY

- The GROUP BY clause a selected group of rows into summary rows by values of one or more columns.
- For each group, you can apply an aggregate function such as MIN, MAX, SUM, COUNT, or AVG to provide more information about each group.

```
SELECT
    column_1,
    aggregate_function(column_2)
FROM
    table
GROUP BY
    column_1,
    column_2;
```

SQL Intermediate: GROUP BY

- The GROUP BY clause comes after the FROM clause of the SELECT statement. In case a statement contains a WHERE clause, the GROUP BY clause must come after the WHERE clause.
- Following the GROUP BY clause is a column or a list of comma-separated columns used to specify the group.

SQL Intermediate

- DISTINCT
- CASE
- GROUP BY
- DATE
- HAVING

SQL Intermediate: DATE, DATETIME

Time Strings

A time string can be in any of the following formats −

Sr.No.	Time String	Example
1	YYYY-MM-DD	2010-12-30
2	YYYY-MM-DD HH:MM	2010-12-30 12:10
3	YYYY-MM-DD HH:MM:SS.SSS	2010-12-30 12:10:04.100
4	MM-DD-YYYY HH:MM	30-12-2010 12:10
5	HH:MM	12:10
6	YYYY-MM-DD**T**HH:MM	2010-12-30 12:10
7	HH:MM:SS	12:10:01
8	YYYYMMDD HHMMSS	20101230 121001
9	now	2013-05-07

SQL Intermediate: DATE, DATETIME

- The date and time functions use a subset of IS0-8601 date and time formats.
- The date() function returns the date in this format: YYYY-MM-DD. The time() function returns the time as HH:MM:SS.
- The datetime() function returns "YYYY-MM-DD HH:MM:SS".
- The julianday() function returns the Julian day - the number of days since noon in Greenwich on November 24, 4714 B.C. (Proleptic Gregorian calendar).
- **The strftime()** routine returns the date formatted according to the format string specified as the first argument

SQL Intermediate: DATE, DATETIME

- SQLite does not support built-in date and/or time storage class. Instead, it leverages some built-in date and time functions to use other storage classes such as TEXT, REAL, or INTEGER for storing the date and time values.

Function	Equivalent strftime()
date(...)	strftime('%Y-%m-%d', ...)
time(...)	strftime('%H:%M:%S', ...)
datetime(...)	strftime('%Y-%m-%d %H:%M:%S', ...)
julianday(...)	strftime('%J', ...)

SQL Intermediate: DATE, DATETIME

%d	day of month: 00
%f	fractional seconds: SS.SSS
%H	hour: 00-24
%j	day of year: 001-366
%J	Julian day number
%m	month: 01-12
%M	minute: 00-59
%s	seconds since 1970-01-01
%S	seconds: 00-59
%w	day of week 0-6 with Sunday==0
%W	week of year: 00-53
%Y	year: 0000-9999
%%	%

SQL Intermediate: HAVING

- The HAVING clause specifies a search condition for a group.
- You often use the HAVING clause with the GROUP BY clause. The GROUP BY clause groups a set of rows into a set of summary rows or groups. Then the HAVING clause filters groups based on a specified condition.
- If you use the HAVING clause, you must include the GROUP BY clause; otherwise, you will get the following error:

```
Error: a GROUP BY clause is required before HAVING
```

- **Note:** that the HAVING clause is applied after GROUP BY clause, whereas the WHERE clause is applied before the GROUP BY clause.

SQL Intermediate: SUBQUERY - chain type

- Subquery - chain type: SELECT statement inside another SELECT statement

```
SELECT column1, column 2
FROM (
        SELECT columnA, columnB
        FROM table )
```

SQL Intermediate: SUBQUERY - WITH

- Subquery - WITH: creates a table in the query memory

WITH ***tablename*** AS

(SELECT column1

FROM table)

SELECT column1

FROM ***tablename***

SQL Intermediate: CREATE

- CREATE clause is used to create and record new table
- If the table already exists it should be deleted using DROP first before it can be created again with a same name.

```
--deleting the table
DROP TABLE newcust;

-- creating new table
CREATE TABLE newcust AS
SELECT
    *
FROM
    Customers
```