Appendix II

List of SQL Keywords by Chapter

Chapter 4 Keywords

```
SELECT, AS, FROM, ORDER BY, ASC, DESC, LIMIT
```

```
/*
This is a block comment. Block comments start with a forward slash followed
by the asterisk, then end with an asterisk and another forward slash. Block
comments should usually follow this format:

CREATED BY: <your name>
CREATED ON: <date>
DESCRIPTION: <Brief description of what your query does>
*/



-- This is an example of using a single-line comment:

SELECT -- Specifies what data or fields to retrieve from the database
   FirstName AS 'First Name', - These are field names
   LastName AS [Last Name], - The AS keyword renames the field
   Company AS Co - One-word aliases do not need single quotes or parentheses

FROM -- Specifies the table containing the desired data
   customers - Refers to the customers table

ORDER BY -- Specifies the output order; ascending (A-Z) is the default
   FirstName DESC - Typing DESC specifies descending (Z-A) order

LIMIT -- Limits results to a specific number
   10; - The semicolon is optional here
```

Chapter 5 Keywords

```
WHERE, CASE, WHEN, THEN, ELSE, END AS, DATE()
```

Operators in SQL are used within SQL clauses.

TYPES OF OPERATORS

fig. 145

COMPARISON	LOGICAL	ARITHMETIC
= Equal To	BETWEEN	+ Add
> Greater Than	IN	- Subtract
< Less Than	LIKE	/ Divide
>= Greater Than or Equal To	AND	* Multiply
<= Less Than or Equal To	OR	% Modulo
<> Not Equal To		

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
Total = 1.98 -- Only returns records where the field Total is equal to 1.98
ORDER BY
   InvoiceDate

CASE - This statement allows you to filter records by user-specified condi...
   WHEN -- Used with a case statement to specify a condition
   THEN -- Used with a case statement after WHEN to create a label for all...
   ELSE -- Used to specify every condition not covered by the WHEN/THEN...
   END AS -- Creates a new field for the labels created by the ELSE state...

SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total,
CASE -- Creates four conditions to display different price ranges for the...
   WHEN TOTAL < 2.00 THEN 'Baseline Purchase' - Condition 1
   WHEN TOTAL BETWEEN 2.00 AND 6.99 THEN 'Low Purchase'
   WHEN TOTAL BETWEEN 7.00 AND 15.00 THEN 'Target Purchase'
ELSE 'Top Performers' -- The ELSE keyword handles all other conditions not...
   END AS PurchaseType
FROM
   invoices
ORDER BY
   BillingCity
```

The single-line comments are abbreviated in the previous example for the sake of print. Single-line comments must always be on one line in the SQL browser or they will be mistaken for code and will result in errors.

fig. 146

	InvoiceDate	BillingAddress	BillingCity	Total	PurchaseType
1	2009-05-10 00:00:00	Lijnbaansgracht 120bg	Amsterdam	8.91	Target Purchase
2	2010-12-15 00:00:00	Lijnbaansgracht 120bg	Amsterdam	1.98	Baseline Purchase
3	2011-03-19 00:00:00	Lijnbaansgracht 120bg	Amsterdam	3.96	Low Purchase
...	...	...	...	...	...
71	2010-03-21 00:00:00	162 E Superior Street	Chicago	15.86	Top Performers

`DATE()` is the first function introduced in the book. It is introduced early so it can be used with the other keywords in chapter 5. More functions are introduced in chapter 7.

```
/*
The DATE() function removes any time code information
from data stored as DATETIME.
*/
SELECT
   InvoiceDate,
   DATE(InvoiceDate) AS [Results of DATE Function]
FROM
   invoices
ORDER BY
   InvoiceDate
```

fig. 147

	InvoiceDate	Results of DATE Function
1	2009-01-01 00:00:00	2009-01-01
2	2009-01-02 00:00:00	2009-01-02
3	2009-01-03 00:00:00	2009-01-03
4	2009-01-06 00:00:00	2009-01-06
5	2009-01-11 00:00:00	2009-01-11
...		

Chapter 6 Keywords

```
INNER JOIN, ON, LEFT OUTER JOIN, RIGHT OUTER JOIN,
IS, NOT
```

The `RIGHT JOIN` is not supported in SQLite but is supported in other RDBMS implementations.

INNER JOIN

```
SELECT
   i.InvoiceId, -- Alias notation specifies what table the field is from
   c.CustomerId,
   c.Name,
   c.Address,
   i.InvoiceDate,
   i.BillingAddress,
   i.Total
FROM
   invoices AS i
INNER JOIN
   customers AS c
ON i.CustomerId = c.CustomerId
```

LEFT OUTER JOIN

```
SELECT
   i.InvoiceId,
   c.CustomerId,
   c.Name,
   c.Address,
   i.InvoiceDate,
   i.BillingAddress,
   i.Total
FROM
   invoices AS i
LEFT OUTER JOIN
   customers AS c
ON
   i.CustomerId = c.CustomerId
```

RIGHT OUTER JOIN (Not Supported in SQLite)

```
SELECT
   i.InvoiceId,
   c.CustomerId,
   c.Name,
   c.Address,
   i.InvoiceDate,
   i.BillingAddress,
   i.Total
FROM
   invoices AS i
RIGHT OUTER JOIN -- Switch position of tables listed in query to create L...
    customers AS c
ON i.CustomerId = c.CustomerId

SELECT
   ar.ArtistId AS [ArtistId From Artists Table],
   al.ArtistId AS [ArtistId From Albums Table],
   ar.Name AS [Artist Name],
   al.Title AS [Album]
FROM
   artists AS ar
LEFT OUTER JOIN
   albums AS al
ON
   ar.ArtistId = al.ArtistId
WHERE
   al.ArtistId IS NULL - Can also use IS NOT
```

Chapter 7 Keywords

`GROUP BY, HAVING`

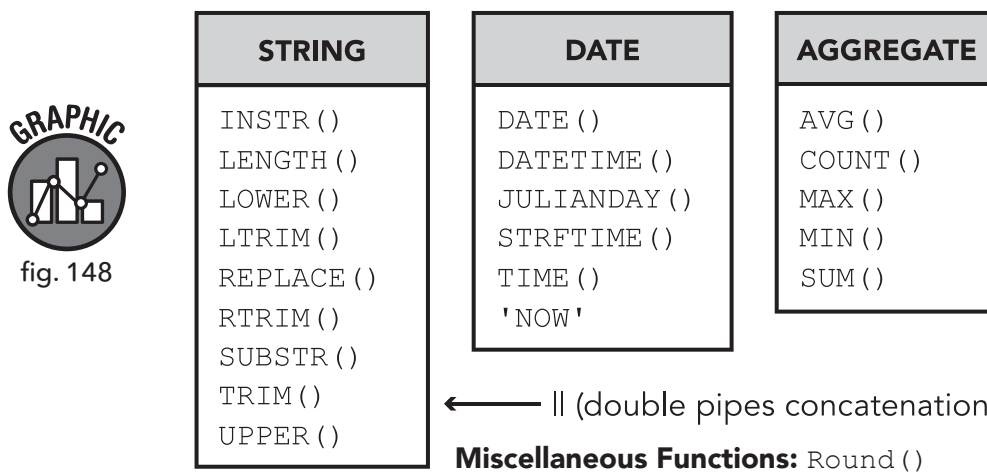

fig. 148

As stated in the chapter, there are many more functions recognized by SQLite than are included in this chapter. For a full list and further documentation on SQLite, visit https://www.sqlite.org/lang_corefunc.html

Chapter 8 Keywords

`DISTINCT`

The basic subquery:

fig. 149

```
1  SELECT
2      InvoiceDate,
3      BillingAddress,
4      BillingCity,
5      Total
6  FROM
7      invoices
8  WHERE Total <
9  (select
10     avg(Total)
11 from
12     invoices)
13 ORDER BY
14     Total DESC
15
```

OUTER QUERY

inner query

OUTER QUERY

fig. 150

TrackId
1
2
3
4
5
6
7
8
9
10
11
12
1984 rows returned in 11ms

The `DISTINCT` clause:

```
SELECT
      DISTINCT TrackId
FROM
      invoice_items
ORDER BY
      TrackId
```

Chapter 9 Keywords

```
CREATE VIEW, DROP VIEW
```

```
CREATE VIEW V_ViewName AS [Alias Name]

DROP VIEW V_ViewName
```

Chapter 10 Keywords

```
INSERT INTO, UPDATE, SET, DELETE
```

Data manipulation language (DML) can permanently alter a database. It is best to practice these commands in a sandbox space such as the sample database provided. Using DML on a live database with active customer data can have permanent deleterious effects.

```
INSERT INTO
artists (Name)
VALUES ('Bob Marley')


UPDATE
employees
SET PostalCode = '11202'
WHERE
   EmployeeId = 9


DELETE FROM
   employees
WHERE
   EmployeeId = 9
```