Other books by the author

Introduction to the Stockmarket

A guide to writing Excel formulas and VBA macros

The Wise Investor

Introduction to Computer Science

To download copies of these books please refer to the author’s personal website below.

www.markmcilroy.com

© Mark McIlroy 2016. All rights reserved.

ISBN 978-1492345831
Contents
1. Prerequisites ................................................................. 4
2. Instructions ..................................................................... 4
3. Joining tables .................................................................. 6
4. SQL .................................................................................. 8
5. Single-table operations .................................................... 9
  5.1 Selecting columns ....................................................... 9
  5.2 Selecting rows ............................................................. 12
6. Sorting result tables ........................................................ 14
7. Counting rows .................................................................. 15
8. Summing totals ............................................................... 16
9. Cartesian Joins ............................................................... 20
10. Retrieving data from multiple tables ................................. 21
11. Distinct values .............................................................. 25
12. Union .............................................................................. 27
13. Subqueries ....................................................................... 28
14. Updating data ............................................................... 29
  14.1 Inserting records ....................................................... 29
  14.2 Updating records ....................................................... 29
  14.3 Deleting records ....................................................... 30
15. NULL values ................................................................. 31
16. About the Author .......................................................... 32
17. Appendix A – Implementation variations .......................... 33
18. Appendix B – Summary of operators ............................... 34
19. Appendix C – Other statements ....................................... 35
20. Appendix D – Test environment ....................................... 36
1. Prerequisites

This book assumes a familiarity with relational data tables.

Readers should have access to a query environment that allows viewing data tables and running SQL queries against a database or data warehouse.

An online test environment is provided for readers of this book.

2. Instructions

A test environment is located at

www.markmcilroy.com/test_env/sql_test.php

You can try out the examples from the book in this environment.

The order of the notes is not significant.

Each of the queries listed in the notes is executable in the test environment.

Relational data tables

Relational data is stored in tables which can be represented in tabular format.

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>First_Name</th>
<th>Surname</th>
<th>Date_Of_Birth</th>
<th>Postcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000001</td>
<td>Stephen</td>
<td>Adjei</td>
<td>5/06/1988</td>
<td>4235</td>
</tr>
<tr>
<td>0000002</td>
<td>Sammy</td>
<td>Adams</td>
<td>26/12/1983</td>
<td>5432</td>
</tr>
<tr>
<td>0000003</td>
<td>Linda</td>
<td>Larigue</td>
<td>21/04/1976</td>
<td>2342</td>
</tr>
<tr>
<td>0000004</td>
<td>Sina</td>
<td>Siva</td>
<td>24/07/1983</td>
<td>4342</td>
</tr>
<tr>
<td>0000005</td>
<td>Vangie</td>
<td>Robinson</td>
<td>5/12/1988</td>
<td>5432</td>
</tr>
<tr>
<td>0000006</td>
<td>Christiana</td>
<td>Majola</td>
<td>3/01/1973</td>
<td>2345</td>
</tr>
<tr>
<td>0000007</td>
<td>Olya</td>
<td>Ayinoko</td>
<td>22/07/1968</td>
<td>9464</td>
</tr>
<tr>
<td>0000008</td>
<td>David</td>
<td>Kellyn</td>
<td>14/08/1982</td>
<td>1242</td>
</tr>
<tr>
<td>0000009</td>
<td>Olusegun</td>
<td>Aby</td>
<td>11/09/1982</td>
<td>4344</td>
</tr>
<tr>
<td>0000010</td>
<td>Maulesh</td>
<td>Amoah</td>
<td>20/01/1965</td>
<td>5342</td>
</tr>
<tr>
<td>0000011</td>
<td>Raj</td>
<td>Lucas</td>
<td>1/01/1973</td>
<td>6543</td>
</tr>
<tr>
<td>0000012</td>
<td>Emmanuel</td>
<td>Crenshaw</td>
<td>6/04/1987</td>
<td>3456</td>
</tr>
<tr>
<td>0000013</td>
<td>Jessica</td>
<td>Adesina</td>
<td>3/05/1977</td>
<td>2356</td>
</tr>
<tr>
<td>0000014</td>
<td>Selina</td>
<td>Amoah</td>
<td>16/01/1982</td>
<td>5423</td>
</tr>
<tr>
<td>0000015</td>
<td>Ademola</td>
<td>karmakar</td>
<td>11/03/1971</td>
<td>3474</td>
</tr>
</tbody>
</table>
Databases are typically used to store information such as customer records, transactions, accounting records and so on.

Data processing is used extensively in the government and corporate business sectors.
3. Joining tables

Processing typically requires that information be retrieved from several different tables.

This is done using a process known as a ‘relational join’.

In the following example, details are extracted from the customer table and the transaction table.

**Transaction Table**

<table>
<thead>
<tr>
<th>Transaction_ID</th>
<th>Transaction_Date</th>
<th>Customer_ID</th>
<th>Transaction_Code</th>
<th>Product_Code</th>
<th>Items</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000001</td>
<td>6/01/2008</td>
<td>0000001</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>1</td>
<td>723.12</td>
</tr>
<tr>
<td>0000002</td>
<td>12/01/2008</td>
<td>0000001</td>
<td>PURCHASE</td>
<td>PCDE27</td>
<td>1</td>
<td>324.12</td>
</tr>
<tr>
<td>0000003</td>
<td>4/01/2008</td>
<td>0000002</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>3</td>
<td>623.23</td>
</tr>
<tr>
<td>0000004</td>
<td>12/01/2008</td>
<td>0000002</td>
<td>CANCELLATION</td>
<td>PCDE27</td>
<td>1</td>
<td>123.34</td>
</tr>
<tr>
<td>0000005</td>
<td>12/01/2008</td>
<td>0000002</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>423.23</td>
</tr>
<tr>
<td>0000006</td>
<td>5/01/2008</td>
<td>0000003</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>153.24</td>
</tr>
<tr>
<td>0000007</td>
<td>12/01/2008</td>
<td>0000004</td>
<td>REFUND</td>
<td>PCDE43</td>
<td>1</td>
<td>233.22</td>
</tr>
<tr>
<td>0000008</td>
<td>21/01/2008</td>
<td>0000004</td>
<td>PURCHASE</td>
<td>PCDE43</td>
<td>1</td>
<td>823.11</td>
</tr>
</tbody>
</table>

**Customer Table**

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>First_Name</th>
<th>Surname</th>
<th>Date_Of_Birth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stephen</td>
<td>Adjei</td>
<td>5/06/1988</td>
</tr>
<tr>
<td>2</td>
<td>Sammy</td>
<td>Adams</td>
<td>26/12/1983</td>
</tr>
<tr>
<td>3</td>
<td>Linda</td>
<td>Larigue</td>
<td>21/04/1976</td>
</tr>
<tr>
<td>4</td>
<td>Sina</td>
<td>Siva</td>
<td>24/07/1983</td>
</tr>
</tbody>
</table>
**Extracted Data**

<table>
<thead>
<tr>
<th>Trans_ID</th>
<th>Trans_Date</th>
<th>Cust_ID</th>
<th>First_Name</th>
<th>Surname</th>
<th>Transaction_Code</th>
<th>Prod_Code</th>
<th>Items</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000001</td>
<td>6/01/2008</td>
<td>0000001</td>
<td>Stephen</td>
<td>Adjei</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>1</td>
<td>723.12</td>
</tr>
<tr>
<td>0000002</td>
<td>12/01/2008</td>
<td>0000001</td>
<td>Stephen</td>
<td>Adjei</td>
<td>PURCHASE</td>
<td>PCDE27</td>
<td>1</td>
<td>324.12</td>
</tr>
<tr>
<td>0000003</td>
<td>4/01/2008</td>
<td>0000002</td>
<td>Sammy</td>
<td>Adams</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>3</td>
<td>623.23</td>
</tr>
<tr>
<td>0000004</td>
<td>12/01/2008</td>
<td>0000002</td>
<td>Sammy</td>
<td>Adams</td>
<td>CANCELLATION</td>
<td>PCDE27</td>
<td>1</td>
<td>123.34</td>
</tr>
<tr>
<td>0000005</td>
<td>12/01/2008</td>
<td>0000002</td>
<td>Sammy</td>
<td>Adams</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>423.23</td>
</tr>
<tr>
<td>0000006</td>
<td>5/01/2008</td>
<td>0000003</td>
<td>Linda</td>
<td>Larigue</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>153.24</td>
</tr>
<tr>
<td>0000007</td>
<td>12/01/2008</td>
<td>0000004</td>
<td>Sina</td>
<td>Siva</td>
<td>REFUND</td>
<td>PCDE43</td>
<td>1</td>
<td>233.22</td>
</tr>
<tr>
<td>0000008</td>
<td>21/01/2008</td>
<td>0000004</td>
<td>Sina</td>
<td>Siva</td>
<td>PURCHASE</td>
<td>PCDE43</td>
<td>1</td>
<td>823.11</td>
</tr>
</tbody>
</table>

Note that data from one table has been duplicated in the columns of the result table.

In this example, the surname and first-name of the customer appears beside each transaction relevant to that customer.

This is known as a ‘one-to-many’ join.
4. SQL

SQL, Structured Query Language, is a database query language that provides functions for sorting, filtering and totaling information that is stored in relational databases.

SQL is the basis for most query operations against large-scale data storage systems.
5. Single-table operations

Viewing a table can be done using a statement such as

```
select * from customer;
```

In this case the ‘*’ represents that all columns should be included in the result table.

5.1 Selecting columns

In many cases only a selection of columns is needed in the result set.

This is specified as follows.

```
select
    surname,
    first_name,
    date_of_birth
from
    customer;
```

Result

<table>
<thead>
<tr>
<th>surname</th>
<th>first_name</th>
<th>date_of_birth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjei</td>
<td>Stephen</td>
<td>1988-06-05</td>
</tr>
<tr>
<td>Adams</td>
<td>Sammy</td>
<td>1983-12-26</td>
</tr>
<tr>
<td>Larigue</td>
<td>Linda</td>
<td>1976-04-21</td>
</tr>
<tr>
<td>Patel</td>
<td>Sabina</td>
<td>1973-07-24</td>
</tr>
<tr>
<td>Robinson</td>
<td>Vangie</td>
<td>1988-12-05</td>
</tr>
<tr>
<td>Majola</td>
<td>Christiana</td>
<td>1973-01-03</td>
</tr>
<tr>
<td>Ayinoko</td>
<td>Olya</td>
<td>1968-07-22</td>
</tr>
<tr>
<td>Kellyn</td>
<td>David</td>
<td>1982-08-14</td>
</tr>
<tr>
<td>Aby</td>
<td>Olusegun</td>
<td>1982-09-11</td>
</tr>
</tbody>
</table>
The ‘from’ clause specifies the table that is used as the source of the data.

The ‘select’ column names indicate which columns to select.

**Column definitions**

In general the ‘select’ list will include a list of column names.

SQL also supports expressions in place of column names.

For example

```sql
select
    surname || ' ' || firstname as full_name,
from
    customer

select
    amount * items as value
from
    transactions
```

Expressions can include the concatenation operator `||` which combines two text values and mathematical operators.

Note: the `||` operator is the standard SQL operator for string concatenation.

Due to the variation of SQL used in the test environment, the function ‘concat()’ must be used instead.

```sql
select
    concat(first_name , ' ', surname) as full_name,
```
from
customer

Result

Query Results

<table>
<thead>
<tr>
<th>full_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stephen Adjei</td>
</tr>
<tr>
<td>Sammy Adams</td>
</tr>
<tr>
<td>Linda Larigue</td>
</tr>
<tr>
<td>Sabina Patel</td>
</tr>
<tr>
<td>Vangie Robinson</td>
</tr>
<tr>
<td>Christiana Majola</td>
</tr>
<tr>
<td>Olya Ayinoko</td>
</tr>
<tr>
<td>David Kellyn</td>
</tr>
<tr>
<td>Olusegun Aby</td>
</tr>
<tr>
<td>Maulesh Amoah</td>
</tr>
<tr>
<td>Christopher Lucas</td>
</tr>
<tr>
<td>Emmanuel Crenshaw</td>
</tr>
<tr>
<td>Jessica Adesina</td>
</tr>
<tr>
<td>Selina Amoah</td>
</tr>
<tr>
<td>Ademola Karmakar</td>
</tr>
</tbody>
</table>
5.2 Selecting rows

Generally rows are filtered so that only rows matching certain criteria are included in the result set or the table calculations.

For example,

```sql
select * from transactions
where trans_date > '2008-01-04' and product_code = 'PCDE12'
```

<table>
<thead>
<tr>
<th>trans_id</th>
<th>trans_date</th>
<th>cust_id</th>
<th>transaction_code</th>
<th>product_code</th>
<th>items</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2008-01-06 00:00:00</td>
<td>1</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>1</td>
<td>723.12</td>
</tr>
<tr>
<td>5</td>
<td>2008-01-12 00:00:00</td>
<td>2</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>423.23</td>
</tr>
<tr>
<td>6</td>
<td>2008-01-12 00:00:00</td>
<td>2</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>423.23</td>
</tr>
<tr>
<td>7</td>
<td>2008-01-05 00:00:00</td>
<td>3</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>153.24</td>
</tr>
</tbody>
</table>

Multiple conditions can be combined using ‘and’ and ‘or’.

Brackets should be used when the ‘where’ expression includes a combination of ‘and’ and ‘or’ expressions.

‘in’ operator

Comparisons can also be done with another table using the ‘in’ operator.

```sql
select * from transactions
where product_code in (select product_code from sample_list)
```
Result

<table>
<thead>
<tr>
<th>Trans_id</th>
<th>trans_date</th>
<th>Cust_id</th>
<th>transaction_code</th>
<th>product_code</th>
<th>items</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2008-01-06 00:00:00</td>
<td>1</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>1</td>
<td>723.12</td>
</tr>
<tr>
<td>3</td>
<td>2008-01-04 00:00:00</td>
<td>2</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>3</td>
<td>623.23</td>
</tr>
<tr>
<td>5</td>
<td>2008-01-12 00:00:00</td>
<td>2</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>423.23</td>
</tr>
<tr>
<td>6</td>
<td>2008-01-12 00:00:00</td>
<td>2</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>423.23</td>
</tr>
<tr>
<td>7</td>
<td>2008-01-05 00:00:00</td>
<td>3</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>153.24</td>
</tr>
<tr>
<td>8</td>
<td>2008-01-12 00:00:00</td>
<td>4</td>
<td>REFUND</td>
<td>PCDE43</td>
<td>1</td>
<td>233.22</td>
</tr>
<tr>
<td>9</td>
<td>2008-01-21 00:00:00</td>
<td>4</td>
<td>PURCHASE</td>
<td>PCDE43</td>
<td>1</td>
<td>823.11</td>
</tr>
</tbody>
</table>

'like' operator

The ‘like’ operator allows for a selection of records matching similar patterns.

For example

```sql
select *
from customer
where surname like 'A%'
```

This selects all customers with surnames starting with ‘A’

Result

<table>
<thead>
<tr>
<th>customer_id</th>
<th>first_name</th>
<th>surname</th>
<th>date_of_birth</th>
<th>postcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stephen</td>
<td>Adjei</td>
<td>1988-06-05</td>
<td>4235</td>
</tr>
<tr>
<td>2</td>
<td>Sammy</td>
<td>Adams</td>
<td>1983-12-26</td>
<td>5432</td>
</tr>
<tr>
<td>7</td>
<td>Olya</td>
<td>Ayinoko</td>
<td>1968-07-22</td>
<td>9464</td>
</tr>
<tr>
<td>9</td>
<td>Olusegun</td>
<td>Aby</td>
<td>1982-09-11</td>
<td>4344</td>
</tr>
<tr>
<td>10</td>
<td>Maulesh</td>
<td>Amoah</td>
<td>1965-01-20</td>
<td>5342</td>
</tr>
<tr>
<td>13</td>
<td>Jessica</td>
<td>Adesina</td>
<td>1977-05-03</td>
<td>2356</td>
</tr>
<tr>
<td>14</td>
<td>Selina</td>
<td>Amoah</td>
<td>1982-01-26</td>
<td>5423</td>
</tr>
</tbody>
</table>
6. Sorting result tables

Rows returned from an SQL query may be returned in a random order.

The ordering of the rows can be specified using an ‘order by’ clause

For example

```sql
select *
from transactions
order by customer_id, trans_date desc;
```

Rows can be sorted in descending order by adding ‘desc’ after the column name.

Result

<table>
<thead>
<tr>
<th>transaction_id</th>
<th>trans_date</th>
<th>customer_id</th>
<th>transaction_code</th>
<th>product_code</th>
<th>items</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2008-01-12 00:00:00</td>
<td>1</td>
<td>PURCHASE</td>
<td>PCDE27</td>
<td>1</td>
<td>324.12</td>
</tr>
<tr>
<td>1</td>
<td>2008-01-06 00:00:00</td>
<td>1</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>1</td>
<td>723.12</td>
</tr>
<tr>
<td>4</td>
<td>2008-01-12 00:00:00</td>
<td>2</td>
<td>CANCELLATION</td>
<td>PCDE27</td>
<td>1</td>
<td>425.54</td>
</tr>
<tr>
<td>5</td>
<td>2008-01-12 00:00:00</td>
<td>2</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>423.23</td>
</tr>
<tr>
<td>6</td>
<td>2008-01-12 00:00:00</td>
<td>2</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>423.23</td>
</tr>
<tr>
<td>3</td>
<td>2008-01-04 00:00:00</td>
<td>2</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>3</td>
<td>623.23</td>
</tr>
<tr>
<td>7</td>
<td>2008-01-05 00:00:00</td>
<td>3</td>
<td>PURCHASE</td>
<td>PCDE12</td>
<td>2</td>
<td>153.24</td>
</tr>
<tr>
<td>9</td>
<td>2008-01-21 00:00:00</td>
<td>4</td>
<td>PURCHASE</td>
<td>PCDE43</td>
<td>1</td>
<td>823.11</td>
</tr>
<tr>
<td>8</td>
<td>2008-01-12 00:00:00</td>
<td>4</td>
<td>REFUND</td>
<td>PCDE43</td>
<td>1</td>
<td>233.22</td>
</tr>
</tbody>
</table>
7. Counting rows

count(*) can be used to count the number of records in a table.

For example

    select
      count(*)
    from
      transactions

Result

<table>
<thead>
<tr>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
</tr>
</tbody>
</table>

A ‘where’ clause can be used to count a sub-set of the rows in the table

    select
      count(*)
    from
      transactions
    where
      trans_date > '2008-01-06'

Result

<table>
<thead>
<tr>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

Counting sets of records can be done using the ‘Group By’ clause.
8. Summing totals

A ‘group by’ clause can be used to calculate totals, averages and counts of records.

‘Group by’ is a complex use of SQL functionality and is not recommending for initial use.

Example

```
select
    customer_id,
    trans_date,
    sum(amount)
from
    transactions
group by
    customer_id;
```

Result

<table>
<thead>
<tr>
<th>customer_id</th>
<th>trans_date</th>
<th>sum(amount)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2008-01-06 00:00:00</td>
<td>1047.24</td>
</tr>
<tr>
<td>2</td>
<td>2008-01-04 00:00:00</td>
<td>1895.23</td>
</tr>
<tr>
<td>3</td>
<td>2008-01-05 00:00:00</td>
<td>153.24</td>
</tr>
<tr>
<td>4</td>
<td>2008-01-12 00:00:00</td>
<td>1056.33</td>
</tr>
</tbody>
</table>

SQL has an open syntax that will allow many combinations of queries to be written.

Only some combinations of clauses will produce meaningful results.

The following rules should be used to produce meaningful result sets when ‘group by’ is used.

1. Select the ‘group by’ columns.

One row will be produced in the result set for each combination of the ‘group by’ columns.

For example,
group by customer_id

Will produce one set of values for each customer.

group by customer_id, trans_date

Would produce a row for each date on which a customer transaction occurred.

2. Include each ‘group by’ column as a column in the ‘select’ section.

Example

```sql
select
customer_id
from
customer
group by
customer_id;
```

3. Select the aggregate functions

The aggregate functions include:

- `sum(column)` Sum the column values
- `min(column)` Select the minimum value
- `max(column)` Select the maximum value
- `avg(column)` Calculate the average value
- `count(*)` Count the number of rows
- `count(column)` Count the number of rows with non-NULL values

For example

```sql
select
customer_id,
count(*) as transaction_count,
sum(amount) as transaction_total
from
transactions
group by
customer_id;
```
This query will return one row for each customer who has transaction records, with the following columns:

<table>
<thead>
<tr>
<th>customer_id</th>
<th>transaction_count</th>
<th>transaction_total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1047.24</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1895.23</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>153.24</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1056.33</td>
</tr>
</tbody>
</table>

The customer number
The number of transactions
The total value of the transactions

When an expression is used in place of a column name, the naming of the result column is database-dependant.

In these cases it is preferable to name the result column.

Columns can also be renamed in the result set in this way.

4. Do not include additional columns in the 'select' column list

This may result in an undefined result set.

Aggregate functions in row selection

A ‘having’ clause can be added to a ‘group by’ clause when aggregate functions are used.

For example

```sql
select
    customer_id, count(*) as trans_count,
    sum(amount) as trans_total
from transactions
group by
```
customer_id
having
sum(amount) > 200

Result

<table>
<thead>
<tr>
<th>customer_id</th>
<th>trans_count</th>
<th>trans_total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1047.24</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1895.23</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1056.33</td>
</tr>
</tbody>
</table>
9. Cartesian Joins

A Cartesian join involves creating a result set containing all combinations of the records from the input tables.

This is usually unintended.

For example

```
select *
from customer, transaction
```

This would not be a meaningful result set, as transaction data would appear beside customer details of a customer unrelated to the transaction.

The lack of a ‘where’ or ‘join’ clause will result in all combinations of records being returned.

In cases where a Cartesian join is required, the number of records returned is $m \times n \times p \times \ldots$

Where $m, n, p, \ldots$ is the number of rows in the input tables.
10. Retrieving data from multiple tables

Most queries involve retrieving data from several input tables.

Tables must be connected using key fields.

These are generally columns such as customer number, product code, transaction date, etc.

Key fields identify a record, rather than being stored data such as amounts, text values, etc.

Joins may be specified in one of two ways.

**Join syntax**

```sql
select t.customer_id, t.trans_date, c.postcode
from transactions t
inner join customer c
on t.customer_id = c.customer_id;
```

**Result**

<table>
<thead>
<tr>
<th>customer_id</th>
<th>trans_date</th>
<th>postcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2008-01-06 00:00:00</td>
<td>4235</td>
</tr>
<tr>
<td>1</td>
<td>2008-01-12 00:00:00</td>
<td>4235</td>
</tr>
<tr>
<td>2</td>
<td>2008-01-04 00:00:00</td>
<td>5432</td>
</tr>
<tr>
<td>2</td>
<td>2008-01-12 00:00:00</td>
<td>5432</td>
</tr>
<tr>
<td>2</td>
<td>2008-01-12 00:00:00</td>
<td>5432</td>
</tr>
<tr>
<td>2</td>
<td>2008-01-12 00:00:00</td>
<td>5432</td>
</tr>
<tr>
<td>3</td>
<td>2008-01-05 00:00:00</td>
<td>2342</td>
</tr>
<tr>
<td>4</td>
<td>2008-01-12 00:00:00</td>
<td>4342</td>
</tr>
<tr>
<td>4</td>
<td>2008-01-21 00:00:00</td>
<td>4342</td>
</tr>
</tbody>
</table>
The join types are

**Inner join**  Only records with matching keys are returned

**Left join**  All records are returned from the first table, and matching records from the second table

**Right join**  All records are returned from the second table, and matching records from the first table

**Alias names**

Alias names do not affect the result of a query however they can be useful in expressing the query more simply.

**For example**

This query uses alias names ‘t’ and ‘c’

```sql
select
    t.customer_id,
    t.trans_date,
    c.postcode
from
    transactions as t
inner join
    customer as c
on
    t.customer_id = c.customer_id;
```

Alias names are necessary in the rare case in which an input table appears more than once in a ‘select’ statement.

Also, if a column name appears in more than one input table, then an alias name should be used to identify the relevant input table.

This problem typically results in an ‘ambiguous column name’ error.

Columns can also be specified using alias names in the format ‘a.*’.

This indicates that all columns from table ‘a’ should be included in the result set.

**More than two join tables.**
The following layout is recommended when more than two input tables are included in a join

```sql
select
t.customer_id,
t.trans_date,
c.postcode,
p.product_code as product,
cu.description as currency
from
transactions as t
inner join
customer as c
on
t.customer_id = c.customer_id
inner join
product as p
on
t.product_code = p.product_code
left join
currency as cu
on
cu.code = p.currency_code;
```

Result

<table>
<thead>
<tr>
<th>customer_id</th>
<th>trans_date</th>
<th>postcode</th>
<th>product</th>
<th>currency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2008-01-06 00:00:00</td>
<td>4235</td>
<td>PCDE12</td>
<td>US Dollars</td>
</tr>
<tr>
<td>1</td>
<td>2008-01-12 00:00:00</td>
<td>4235</td>
<td>PCDE27</td>
<td>Hong Kong Dollars</td>
</tr>
<tr>
<td>2</td>
<td>2008-01-04 00:00:00</td>
<td>5432</td>
<td>PCDE12</td>
<td>US Dollars</td>
</tr>
<tr>
<td>2</td>
<td>2008-01-12 00:00:00</td>
<td>5432</td>
<td>PCDE27</td>
<td>Hong Kong Dollars</td>
</tr>
<tr>
<td>2</td>
<td>2008-01-12 00:00:00</td>
<td>5432</td>
<td>PCDE12</td>
<td>US Dollars</td>
</tr>
<tr>
<td>3</td>
<td>2008-01-05 00:00:00</td>
<td>2342</td>
<td>PCDE12</td>
<td>US Dollars</td>
</tr>
<tr>
<td>4</td>
<td>2008-01-12 00:00:00</td>
<td>4342</td>
<td>PCDE43</td>
<td>Japanese Yen</td>
</tr>
<tr>
<td>4</td>
<td>2008-01-21 00:00:00</td>
<td>4342</td>
<td>PCDE43</td>
<td>Japanese Yen</td>
</tr>
</tbody>
</table>
In the case of left joins and right joins, the order of tables in the query may affect the result set. Each table is joined to the result of the previous joins. Field names in ‘on’ expressions should only refer to tables that are specified earlier in the join list.

**Where syntax**

Joints can also be specified by listing multiple tables in the ‘from’ clause, and matching the keys within the ‘where’ clause.

This syntax is equivalent to using ‘inner join’ on all the joined tables.

A ‘where’ format does not facilitate ‘left’ or ‘right’ joins

```sql
select
t.customer_id,
t.trans_date,
c.postcode,
p.product_code as product,
cu.description as currency
from
transactions as t,
customer as c,
product as p,
currency as cu
where
  t.customer_id = c.customer_id and
cu.code = p.currency_code and
t.product_code = p.product_code
```

**Multiple join fields**

In some cases records will be identified by a single value such as transaction_id, customer_number etc.

In other cases tables may be joined by a number or fields, such as product_class, product_subclass

In these cases use a syntax similar to the following

```sql
...
```
on
    a.product_class = b.product_class and
    a.product_subclass = b.product_subclass

11. Distinct values

Distinct values can be returned using the ‘distinct’ keyword or a ‘group by’ clause.

For example

    select distinct
        customer_id
    from
        transactions;

This query will return a list of the customer_id values that appear in the transaction table

Result

<table>
<thead>
<tr>
<th>customer_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

This function can be used with any query, but is most useful when there is a single result
column or a small number of result columns.

If a count of these values is required, the ‘group by’ syntax should be used

    select
        customer_id,
        count(*)
    from
        transactions
    group by
        customer_id;
**Result**

<table>
<thead>
<tr>
<th>customer_id</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
12. Union

The ‘union’ statement can be used to combine the results of two queries into a single result set.

```
select * from
    (select * from sample_list
    union all
    select * from sample_list_ext) a
```

‘union all’ combines the two result sets, while ‘union’ selects only the distinct records

<table>
<thead>
<tr>
<th>id</th>
<th>product_code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCDE43</td>
</tr>
<tr>
<td>2</td>
<td>PCDE52</td>
</tr>
<tr>
<td>3</td>
<td>PCDE12</td>
</tr>
<tr>
<td>1</td>
<td>PCDE27</td>
</tr>
<tr>
<td>2</td>
<td>PCDE12</td>
</tr>
</tbody>
</table>
13. Subqueries

An SQL query can be used in place of a table name.

The query should be placed within brackets, and used in place of a table name within another query.

For example

```sql
select count(*)
from transactions as t
inner join (
    select distinct product_code
    from product
) as p
on t.product_code = p.product_code
```

In this example a bracketed query has been used in place of a table name.

In this case, a count of records is calculated from customer records joined to product codes.

The statement within the brackets is equivalent to a table containing the same data.
14. Updating data

The following sections describe SQL statements for updating data.

In many cases it is not possible to recover data that is accidentally altered or deleted. Caution should be used when using these statements. For example

```
delete from transactions
```

Will delete all records from the database table ‘transactions’.

### 14.1 Inserting records

Individual rows can be inserted into a table using the following syntax

```
insert into currexchange (name, amount, exchdate)
values ('name1', 12.52, '2003-02-01')
```

Importing large quantities of records is dependant on the functions provided by the database environment.

### 14.2 Updating records

Tables can be updated using the following syntax.

```
update currexchange
set amount = 32.23
where exchdate = '2003-02-01'
```
Implementations vary in their ability to perform updates on views created by joining several tables.

**14.3 Deleting records**

Deletion takes the format

```sql
delete from table [where condition]
```

For example

```sql
delete from currexchange where exchdate < '2003-03-04'
```
15. **NULL values**

NULL values represent missing data.

This may indicate that a data item is not known, or is not relevant in that particular case.

Visual tools may display this result in several formats including NULL, (null), a blank field etc.

In ‘where’ expressions the following syntax should be used

```
select 
customer_id, 
amount
from transactions
where amount is not NULL
```

Result

<table>
<thead>
<tr>
<th>customer_id</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>723.12</td>
</tr>
<tr>
<td>1</td>
<td>324.12</td>
</tr>
<tr>
<td>2</td>
<td>623.23</td>
</tr>
<tr>
<td>2</td>
<td>425.54</td>
</tr>
<tr>
<td>2</td>
<td>423.23</td>
</tr>
<tr>
<td>2</td>
<td>423.23</td>
</tr>
<tr>
<td>3</td>
<td>153.24</td>
</tr>
<tr>
<td>4</td>
<td>233.22</td>
</tr>
<tr>
<td>4</td>
<td>823.11</td>
</tr>
</tbody>
</table>
16. **About the Author**

Mark Laurence McIlroy has an undergraduate degree in Computer Science and Applied Mathematics from Monash University.

He also has Masters degrees in Applied Finance and Financial Planning.

He has extensive experience consulting in the banking and government sectors in Australia in large SQL data warehouse environments.

After a long career in Information Technology in the Financial Services sector in Australia, Mark has now made a career change into Financial Planning.

Mark lives with his wife in Melbourne, Australia.

Further information and resources can be found on the author’s personal website, www.markmcilroy.com
17. Appendix A – Implementation variations

The SQL statements described here should be executable in most SQL environments.

Some differences may occur with issues such as specifying date constants.

For example

‘1990-04-12’

\texttt{to\_date( ‘01JUL2009’ )}

etc.

Major implementations frequently have added syntax which is not compatible across alternative implementations.

Examples include variations on join types such as OUTER JOIN, CROSS JOIN etc.
18. Appendix B – Summary of operators

Operators

Mathematical

* Multiplication
/ Division
+ Addition
- Subtraction

Relational

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

String

|| Concatenate

Aggregate

sum Sum
avg Average
count Count
min Minimum value
max Maximum value
stdev Standard Deviation
19. Appendix C – Other statements

Other Issues

SQL includes the following groups of statements.

These statements are not widely used as these functions are more easily performed using database administration tools.

Data Definition statements

Statements for creating tables and altering table formats

CREATE TABLE transactions (id INTEGER NOT NULL, transact_date DATE, amount DOUBLE, description VARCHAR(255), PRIMARY KEY ("id") )

Administration statements

Statements for creating user accounts and assigning security privileges.

GRANT SELECT ON TABLE1 TO USERNAME1

Descriptive statements

Statements for returning the information about the database, such as the list of tables.

SHOW TABLES
20. Appendix D – Test environment

A test environment is located at

www.markmcilroy.com/test_env/sql_test.php

You can try out the examples from the book in this environment.

Tables

currency id, code, description

currexchange name, amount, exchdate

customer customer_id, first_name, surname, date_of_birth, postcode

product id, product_code, currency_code, description

sample_list id, product_code

sample_list_ext id, product_code

transactions transaction_id, trans_date, customer_id, transaction_code, product_code, items, amount