SQL Essentials

Mark McIlroy

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.

2 Instructions

A test environment is located at

www.blueskytechnology.com.au/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.

Customer_ID	First_Name	Surname	Date_Of_Birth	Postcode
0000001	Stephen	Adjei	5/06/1988	4235
0000002	Sammy	Adams	26/12/1983	5432
0000003	Linda	Larigue	21/04/1976	2342
0000004	Sina	Siva	24/07/1983	4342
0000005	Vangie	Robinson	5/12/1988	5432
0000006	Christiana	Majola	3/01/1973	2345
0000007	Olya	Ayinoko	22/07/1968	9464
0000008	David	Kellyn	14/08/1982	1242
0000009	Olusegun	Aby	11/09/1982	4344
0000010	Maulesh	Amoah	20/01/1965	5342
0000011	Raj	Lucas	1/01/1973	6543
0000012	Emmanuel	Crenshaw	6/04/1987	3456
0000013	Jessica	Adesina	3/05/1977	2356
0000014	Selina	Amoah	16/01/1982	5423
0000015	Ademola	karmakar	11/03/1971	3474

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

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

Customer Table

Customer_ID	First_Name	Surname	Date_Of_Birth
1	Stephen	Adjei	5/06/1988
2	Sammy	Adams	26/12/1983
3	Linda	Larigue	21/04/1976
4	Sina	Siva	24/07/1983

Extracted Data

Transaction_ID	Transaction_Date	Customer_ID	First_Name	Surname	Transaction_Code	Product_Code	Items	Amount
0000001	6/01/2008	0000001	Stephen	Adjei	PURCHASE	PCDE12	1	723.12
0000002	12/01/2008	0000001	Stephen	Adjei	PURCHASE	PCDE27	1	324.12
0000003	4/01/2008	0000002	Sammy	Adams	PURCHASE	PCDE12	3	623.23
0000004	12/01/2008	0000002	Sammy	Adams	CANCELLATION	PCDE27	1	123.34
0000005	12/01/2008	0000002	Sammy	Adams	PURCHASE	PCDE12	2	423.23
0000006	5/01/2008	0000003	Linda	Larigue	PURCHASE	PCDE12	2	153.24
0000007	12/01/2008	0000004	Sina	Siva	REFUND	PCDE43	1	233.22
0000008	21/01/2008	0000004	Sina	Siva	PURCHASE	PCDE43	1	823.11

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 data bases.

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

surname	*first_name*	*date_of_birth*
Adjei	Stephen	1988-06-05
Adams	Sammy	1983-12-26
Larigue	Linda	1976-04-21
Patel	Sabina	1973-07-24
Robinson	Vangie	1988-12-05
Majola	Christiana	1973-01-03
Ayinoko	Olya	1968-07-22
Kellyn	David	1982-08-14
Aby	Olusegun	1982-09-11

Amoah	Maulesh	1965-01-20
Lucas	Raj	1973-01-01
Crenshaw	Emmanuel	1987-04-06
Adesina	Jessica	1977-05-03
Amoah	Selina	1982-01-26
Karmakar	Ademola	1971-03-11

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

```
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.

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

Result

Query Results

full_name
Stephen Adjei
Sammy Adams
Linda Larigue
Sabina Patel
Vangie Robinson
Christiana Majola
Olya Ayinoko
David Kellyn
Olusegun Aby
Maulesh Amoah
Christopher Lucas
Emmanuel Crenshaw
Jessica Adesina
Selina Amoah
Ademola Karmakar

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,

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

Result

transaction_id	*trans_date*	*customer_id*	*transaction_code*	*product_code*	*items*	*amount*
1	2008-01-06 00:00:00	1	PURCHASE	PCDE12	1	723.12
5	2008-01-12 00:00:00	2	PURCHASE	PCDE12	2	423.23
6	2008-01-12 00:00:00	2	PURCHASE	PCDE12	2	423.23
7	2008-01-05 00:00:00	3	PURCHASE	PCDE12	2	153.24

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.

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

Result

transaction_id	*trans_date*	*customer_id*	*transaction_code*	*product_code*	*items*	*amount*
1	2008-01-06 00:00:0	1	PURCHASE	PCDE12	1	723.12
3	2008-01-04 00:00:0	2	PURCHASE	PCDE12	3	623.23
5	2008-01-12 00:00:0	2	PURCHASE	PCDE12	2	423.23
6	2008-01-12 00:00:0	2	PURCHASE	PCDE12	2	423.23
7	2008-01-05 00:00:0	3	PURCHASE	PCDE12	2	153.24
8	2008-01-12 00:00:0	4	REFUND	PCDE43	1	233.22
9	2008-01-21 00:00:0	4	PURCHASE	PCDE43	1	823.11

‘like’ operator

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

For example

```
select
        *
from
        customer
where
        surname like ‘A%’
```

This selects all customers with surnames starting with ‘A’

Result

customer_id	*first_name*	*surname*	*date_of_birth*	*postcode*
1	Stephen	Adjei	1988-06-05	4235
2	Sammy	Adams	1983-12-26	5432
7	Olya	Ayinoko	1968-07-22	9464
9	Olusegun	Aby	1982-09-11	4344
10	Maulesh	Amoah	1965-01-20	5342
13	Jessica	Adesina	1977-05-03	2356
14	Selina	Amoah	1982-01-26	5423

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

```
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

transaction_id	*trans_date*	*customer_id*	*transaction_code*	*product_code*	*items*	*amount*
2	2008-01-12 00:00:00	1	PURCHASE	PCDE27	1	324.12
1	2008-01-06 00:00:00	1	PURCHASE	PCDE12	1	723.12
4	2008-01-12 00:00:00	2	CANCELLATION	PCDE27	1	425.54
5	2008-01-12 00:00:00	2	PURCHASE	PCDE12	2	423.23
6	2008-01-12 00:00:00	2	PURCHASE	PCDE12	2	423.23
3	2008-01-04 00:00:00	2	PURCHASE	PCDE12	3	623.23
7	2008-01-05 00:00:00	3	PURCHASE	PCDE12	2	153.24
9	2008-01-21 00:00:00	4	PURCHASE	PCDE43	1	823.11
8	2008-01-12 00:00:00	4	REFUND	PCDE43	1	233.22

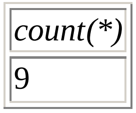
7 Counting rows

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

For example

```
	select
		count(*)
	from
		transactions
```

Result

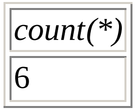


count(*)
9

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



count(*)
6

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

Advanced Topics

8 Summing totals

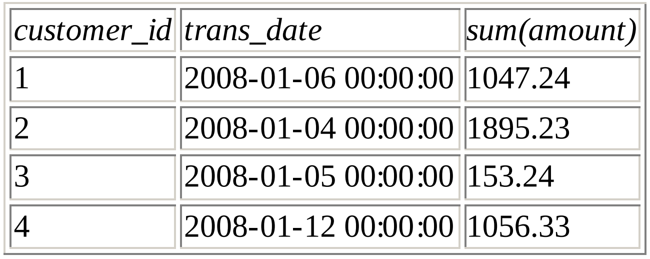
A ‘group by’ clause can be used to calculate totals, averages and counts of records.

‘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



customer_id	*trans_date*	*sum(amount)*
1	2008-01-06 00:00:00	1047.24
2	2008-01-04 00:00:00	1895.23
3	2008-01-05 00:00:00	153.24
4	2008-01-12 00:00:00	1056.33

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

```
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

```
select
        customer_id,
        count(*)            as transaction_count,
        sum(amount)         as transaction_total
from
        transactions
group by
        customer_id;
```

Result

customer_id	*transaction_count*	*transaction_total*
1	2	1047.24
2	4	1895.23
3	1	153.24
4	2	1056.33

This query will return one row for each customer who has transaction records, with the following columns:

customer_id	The customer number
count(*)	The number of transactions
sum(amount)	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

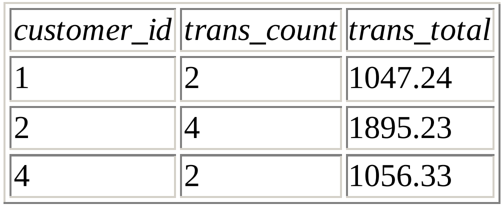
A 'having' clause can be added to a 'group by' clause when aggregate functions are used.

For example

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

```
        customer_id
    having
        sum(amount) > 200
```

Result



customer_id	*trans_count*	*trans_total*
1	2	1047.24
2	4	1895.23
4	2	1056.33

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 * n * p *…

Where m, n, p,… 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

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

Result

customer_id	*trans_date*	*postcode*
1	2008-01-06 00:00:00	4235
1	2008-01-12 00:00:00	4235
2	2008-01-04 00:00:00	5432
2	2008-01-12 00:00:00	5432
2	2008-01-12 00:00:00	5432
2	2008-01-12 00:00:00	5432
3	2008-01-05 00:00:00	2342
4	2008-01-12 00:00:00	4342
4	2008-01-21 00:00:00	4342

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’

```
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

```
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

customer_id	*trans_date*	*postcode*	*product*	*currency*
1	2008-01-06 00:00:00	4235	PCDE12	US Dollars
1	2008-01-12 00:00:00	4235	PCDE27	Hong Kong Dollars
2	2008-01-04 00:00:00	5432	PCDE12	US Dollars
2	2008-01-12 00:00:00	5432	PCDE27	Hong Kong Dollars
2	2008-01-12 00:00:00	5432	PCDE12	US Dollars
2	2008-01-12 00:00:00	5432	PCDE12	US Dollars
3	2008-01-05 00:00:00	2342	PCDE12	US Dollars
4	2008-01-12 00:00:00	4342	PCDE43	Japanese Yen
4	2008-01-21 00:00:00	4342	PCDE43	Japanese Yen

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

Joins 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

```
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

```
…
```

```
…
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

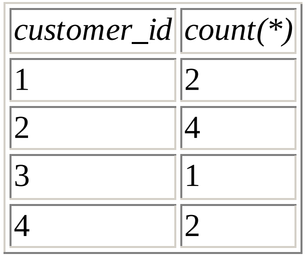
customer_id
1
2
3
4

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



customer_id	*count(*)*
1	2
2	4
3	1
4	2

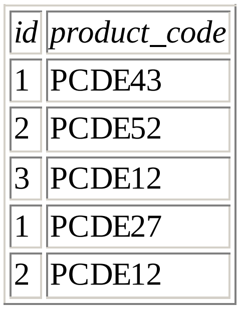
12 Union

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

Result



id	*product_code*
1	PCDE43
2	PCDE52
3	PCDE12
1	PCDE27
2	PCDE12

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

```
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 accidently 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

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

For example

```
        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

customer_id	*amount*
1	723.12
1	324.12
2	623.23
2	425.54
2	423.23
2	423.23
3	153.24
4	233.22
4	823.11

16 Appendix A

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’
>
> 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.

17 Appendix B

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

Appendix C

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
```

18 Appendix D

Test data 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