Appendix B
SQL Reference

SQL stands for Structured Query Language, this is a standard language used to communicate with a variety of different database formats. SQL can be used in iNETstore to display any data that you have in your database tables. This can come in handy to display a listing of your Categories or Items or other database contents. SQL can be used on any iNETstore server interpreted page, *.ehtml and in iNETstore template files (located in the template directory).

SQL Tags

To use a SQL statement within iNETstore you need to use the following iNETstore SQL tags:

[DBxBEGIN_SQL "statement"]	This is the beginning tag for a SQL statement, indicates to the inetstore server that the following text will be part of a SQL statement. It also includes the actual SQL statement.
[DBxSQL_field]	A SQL statement usually asks for data from the database, this field is used to display the data that was retrieved from the database by the SQL statement. In this tag, the word field is the name of the field from the database you wish to display (eg. [DBxSQL_name]).
[DBxEND_SQL]	This indicates that the SQL statement is now complete and that the data retrieved by the statement is no longer necessary.

PURGEIFNORESULTS

If there is a possibility that there may be no results for your SQL query, you can prevent the text between the SQL_Begin and the SQL_end from being displayed by using a PURGEIFNORESULTS SQL tag.
[DBxBEGIN_PURGEIFNORESULTS_SQL]
NOTE: If you use this tag in a page you must continue to use this tag for subsequent SQL statements. If you have used [DBxBEGIN_PURGEIFNORESULTS_SQL], you cannot use [DBxBEGIN_SQL] on the same page. If you do, iNETstore will generate an error.

SQL Statements

As previously mentioned SQL is a standard database query language, due to the huge variation in formats between the many ODBC compliant databases throughout the world it differs quite a lot.
However there are some basic statements that are compatible with all databases, it is advisable to stick to basic commands whilst using iNETstore. More advanced SQL can be used if you have a good understanding of SQL. There are many useful sources of information on SQL on the web and in published form.

A SQL statement is made up of several distinct parts:

The SELECT Clause

This is where we tell iNETstore what data it is we want to retrieve from the database. The SELECT section starts with the word SELECT (in uppercase) and is followed by the names of the field you wish to retrieve separated by commas. A wildcard can also be used to retrieve all fields from the database table.

Examples:

SELECT name	This indicates that we want to retrieve the contents of the field titled 'name'.
SELECT name, address, phone	This indicates that we want to retrieve the fields titled 'name', 'address' and 'phone' from the database table.
SELECT *	This indicates that we want to retrieve all data in all fields from the database table. (* is a wildcard).

The FROM Clause

The next section of the SQL statement indicates which database table we want to retrieve the aforementioned data from, it is the FROM section. It consists of the word FROM (in uppercase) followed by the name of the table.

SELECT name, address FROM SYSmystore	This indicates that we want to retrieve the name and address field from the table titled 'SYSmystore'.
SELECT * FROM ITMshop	This indicates that we want to retrieve all data from the table 'ITMshop'.

The WHERE Condition

The next section of the SQL statement is the WHERE condition, this allows us to select data from certain records from the aforementioned table only, for instances where you don't want all the data from the database table. The WHERE condition consists of the word WHERE (in uppercase) and the condition. The condition is made up of a statement that sets a condition for a certain field to meet.

Examples:

WHERE suburb = 'Blacktown'	This would only retrieve data from the records where the the contents of the filed titled 'suburb' were Blacktown. A string of text in your condition must be surrounded by single quote marks (').
WHERE name = 's*'	By using a wildcard this condition indicates that we want to retrieve data from the records where the contents of the 'name' field start with the letter 's'.
WHERE age = 16	This condition would retrieve data from only the records where the contents of the age column were equal to 16. Numeric data does not need to be surrounded by quote marks.
WHERE age > 16	This condition would retrieve data from only the records where the contents of the age column were greater than 16.
WHERE age <> 16	This condition would retrieve data from any record where the contents of the age field are less than or greater than (not equal to) 16.
WHERE expiry > '8/22/00'	This would retrieve data from any record where the contents of the field 'expiry' were greater than the date '8/22/00'. Date data types must be surrounded by apostrophe (') symbols, this will only work if the data type of the expiry field is DATETIME format. NOTE: databases use US DateTime formats only. Avoid using = in a WHERE clause using a datetime field.
WHERE suburb = 'Blacktown' AND age = 16	Two conditions can be combined, in this case we only want data from records with Blacktown in the suburb field and 16 in the age field.

The ORDER BY Clause

The final section of a SQL statement is the ORDER BY clause, this allows us to specify the order in which we want the data retrieved returned to us. This clause consists of the words ORDER BY (in uppercase) followed by the field we want the results ordered by. NOTE: you can only order by fields that have been retrieved (specified in the SELECT clause above).
EXAMPLES:

ORDER BY name	This would return the results in alphabetical order of the field titles 'name'.
ORDER BY age	This would order the results in numerical order of the field titled 'age'.
By using the additional argument of DESC we can ask that the order of the results be reversed.	
ORDER BY age DESC	This would return the results in reverse (descending) numerical order of age. That is highest to lowest.
ORDER BY name DESC	This would return the results in reverse alphabetical order of the name field.

DISTINCT

Sometimes you may want to select data from a field that contains several multiple entries, but you only want to display each unique entry once. For example you may have a field that contains the brand of each item, several of the items may contain the same brand name. If you wanted to create a list of all the brand names a standard SQL SELECT statement would return every entry, by using DISTINCT you can ensure that you don't get any repeat entries.
Example:

ITEM	SELECT brand FROM ...	SELECT DISTINCT brand FROM ...
BRAND	Victor	Victor
Lawnmower	Makita	Makita
Victor	Riobi	Riobi
Brushcutter	Victor	
Makita		
Chainsaw		
Riobi		
Vacuum		
Victor		

Some Examples

[DBxBEGIN_SQL "SELECT name, phone FROM USRmystore WHERE age < 16 ORDER BY name"]

The above statement would select the name and phone data from the database table 'USRmystore', where the age is less than 16. Now that we have selected the data we need to display it.

The phone number for [DBxSQL_name] is [DBxSQL_phone].

This would substitute the data for name and Phone in the above line of text.

Then we need to close the SQL commands.
[DBxEND_SQL]

So we put it all together:

[DBxBEGIN_SQL "SELECT name, phone FROM USRmystore WHERE age < 16 ORDER BY name"]

The phone number for [DBxSQL_name] is [DBxSQL_phone].
[DBxEND_SQL]

Placing the above code in any iNETstore *.ehtml page will display a line of text for each record, similar to:

The phone number for John Smith is 02 9955 2233.
The phone number for Peter Brown is 02 9812 3946.
The phone number for Joe Jones is 02 9651 2233.

SQL Arithmetic Functions

You can also retrieve calculations using SQL statements, such as Averages of numbers, Counts, Maximums, Minimums. This could be handy if you want to display how many items you have in a particular category or the maximum or average price of the items in your store.

Before you start to do calculations in your SQL statements you need to understand the AS condition in a SQL statement. The AS condition can be used in the SELECT statement to specify the name of the results of the SQL query. Unlike a normal SELECT statement, calculation results don't automatically have a defined name, so we need to use the AS condition to specify the name.

Examples:

SELECT name AS customername, phone FROM ...

This would change the name of the result to customername, so to display the result of this query we would use:

The phone number for [DBxSQL_customername] is [DBxSQL_phone]
SELECT name AS customername, phone AS customerphone FROM...

Similarly in this example we have specified a different result name for both the name and the phone fields. So the results would be displayed as:
The phone number for [DBxSQL_customername] is [DBxSQL_customerphone]

MAX

To find the Maximum price of all the items in your database you can use the MAX function, it consists of the word MAX followed immediately by the name of the field surrounded by standard brackets.

Example:
[DBxBEGIN_SQL "SELECT MAX(price) AS topprice FROM ITM[DBxDBNAME] "]

The top price in this store is: $[DBxSQL_topprice]
[DBxEND_SQL]

MIN

To find the Minimum price of all the items in your database you can use the MIN function, it consists of the word MIN followed immediately by the name of the field surrounded by standard brackets.

Example:
[DBxBEGIN_SQL "SELECT MIN(price) AS minprice FROM ITM[DBxDBNAME] "]
The top price in this store is: $[DBxSQL_minprice]
[DBxEND_SQL]

COUNT

To find the number of items in the entire store you can use the COUNT function, it consists of the word COUNT followed immediately by the name of the field surrounded by standard brackets.

Example:
[DBxBEGIN_SQL "SELECT COUNT(id) AS totalitems FROM ITM[DBxDBNAME] "]

There are [DBxSQL_totalitems] items in this store.
[DBxEND_SQL]

AVG

To find the average price of all the items in your store you can use the AVG function, it consists of the word AVG followed immediately by the name of the field surrounded by standard brackets.
Example:
[DBxBEGIN_SQL "SELECT AVG(price) AS avprice
FROM ITM[DBxDBNAME]"]

The average price in this store is: $[DBxSQL_avprice]
[DBxEND_SQL]

SUM

To find the Sum of a field (maybe stocklevel) in your database you can use the SUM function, it consists of the word SUM followed immediately by the name of the field surrounded by standard brackets.

Example:
[DBxBEGIN_SQL "SELECT SUM(stocklevel) AS stock FROM ITM[DBxDBNAME] "]
This store has [DBxSQL_stock] individual items.
[DBxEND_SQL]

If you want the sum of two numbers you can just use an equation inside brackets without the SUM.

Example:
[DBxBEGIN_SQL "SELECT (stocklevel * price) AS stock FROM ITM[DBxDBNAME] WHERE id = 4 "]
The total value of item #4 is [DBxSQL_stock] (stocklevel multiplied by price).
[DBxEND_SQL]

TOP

You can also select the TOP few records from a field, perhaps the top 3 prices, to do this we use TOP followed by a space then the number of fields you want to display then another space then the field you want to retrieve the data from. If you are using the TOP feature you must specify an ORDER BY field, this will be the field that the TOP is dependant upon. TOP doesn't necessarily mean the highest values, it really means the first few records and will always default to Ascending order - giving you the lowest values first. To retrieve the highest values you need to add DESC after the ORDER BY.

Example:
[DBxBEGIN_SQL "SELECT TOP 3 name, price FROM ITM[DBxDBNAME] ORDER BY price DESC"]

Three most expensive items in the store:
[DBxSQL_name] at $ [DBxSQL_price]
[DBxEND_SQL]

NOTE: If there are two records with the same value within the TOP 'n' (n= number specified by you) records both records will be returned even if it means more than 'n' results are returned.

NAME PRICE Wheelbarrow 200.00 Drill 150.00 Hammer 20.00 Saw 20.00 Screwdriver 5.00	SELECT TOP 3 price FROM ITM[DBxDBNAME] Wheelbarrow 200.00 Drill 150.00 Hammer 20.00 Saw 20.00

INSERT

INSERT can be used to add a new record to a table. Assume you want to create a new record in the system table with id = 3 and put 'Acme Corporation' into the 'name' field, you could use the following syntax:

[DBxBEGIN_SQL "INSERT INTO SYS[DBxDBNAME] (id,name) VALUES (3,'Acme Corporation')"]
[DBxEND_SQL]

UPDATE

It is also possible to use standard SQL commands to update data in the iNETstore database. The command you should use is called 'UPDATE'. For example, the following line could be used to update the name of a product with id = 5:

```
UPDATE ITM[DBxDBNAME] SET name = 'T-Shirt' WHERE id = 5
```

In iNETstore syntax, this would like as follows:

```
[DBxBEGIN_SQL "UPDATE ITM[DBxDBNAME] SET name = 'T-Shirt' WHERE id = 5"]
[DBxEND_SQL]
```

If you use SQL statements, make sure you close them correctly, using the [DBxEND_SQL] tag.