SQL Guide

SQL– short for *Structured Query Language* – is a widely used language in database applications. It's main use is in retrieving information, usually only information that is of particular interest (a query).

SQL can be used in two places in Ability:

Database	using the View SQL button in the query dialog (View/Query)
Database functions	most of the database linking functions support SQL. For example DBSQL(). These allow SQL to be used in Write or Spreadsheet to retrieve data.

Ultimately, Ability uses SQL for all its database operations – for example creating and deleting tables and indexes. These types of operation are not covered here.

This guide covers five main types of SQL statements:

Selecting records	selecting all records from a table, selecting by column and by row
Aggregate and group selection	summary information on tables
Crosstabulations	crosstabs (or contingency tables)
Relational joins	linking two tables together
Editing tables	updating, adding and deleting records

If you are new to SQL, begin with Selecting records , since this topic covers ground required for later topics.

Selecting records - the basics

Consider the following example table of employee information:

EmployeeID	FirstName	LastName	Department	Age	JoinDate
1001	Phil	Roach	TECH	24	23/1/87
1002	Chris	England	TECH	36	1/10/86
1003	Andreas	Smith	SALES	25	18/6/90
1004	Jim	Smith	ADMIN	30	10/3/92
1005	Julia	Allan	SALES	25	26/9/91

To select the entire table, the SQL command is:

SELECT EmployeeID, FirstName, LastName, Department, Age, JoinDate FROM Employee;

This would return every row and column from the table. The two SQL keywords are SELECT and FROM. Note that every SQL statement ends with a semi-colon ";".

A shorter alternative expression is:

SELECT * FROM Employee;

The asterisk simply stands for all, or every, column.

To select only names and departments, use:

SELECT FirstName, LastName, Department FROM Employee;

Note that you can re-arrange the order of the columns by supplying the field list in a different order. For example:

SELECT Department, FirstName, LastName FROM Employee;

This would give the following results table (or query):

Department	FirstName	LastName
TECH	Phil	Roach
TECH	Chris	England
SALES	Andreas	Smith
ADMIN	Jim	Smith
SALES	Julia	Allan

If your table or field names include spaces, you'll need to surround them with square brackets as follows:

SELECT [field number 1] FROM [My test table];

See also:

Calculated fields and renaming column titles

Applying a sort order

Selecting top records only

Applying a condition

Numeric comparisons

Date comparisons

Text comparisons

Wildcards

Selecting blank entries

Other types of SQL statements

Calculated fields and renaming column titles

You can rename column titles using the AS keyword. For example:

SELECT Department AS Dept, FirstName, LastName FROM Employee;

This simply shortens the column header for the returned results table. More usefully, you can name computed fields. Using the Employee table as an example:

SELECT Department, FirstName + " " + LastName AS Name FROM Employee;

Produces the following:

Department	Name
TECH	Phil Roach
TECH	Chris England
SALES	Andreas Smith
ADMIN	Jim Smith
SALES	Julia Allan

Note that the space in between the FirstName and LastName is required to properly format the result.

Numeric field types can also be used to create calculated columns. Here's some examples:

SELECT Age + 2 AS [Age in two years] FROM Employee;

SELECT Age * 2 AS DoubleAge FROM Employee;

See also:

Next topic - applying a sort order

Selecting records - the basics

Other types of SQL statements

Applying a sort order

To sort the Employee table by LastName, add the keywords ORDER BY:

SELECT Department, FirstName, LastName FROM Employee ORDER BY LastName;

To make sure "Andreas" appear before "Jim", sort on both FirstName and LastName:

SELECT Department, FirstName, LastName FROM Employee ORDER BY LastName, FirstName;

To reverse the order, add the keyword for descend as follows:

SELECT Department, FirstName, LastName FROM Employee ORDER BY LastName DESC, FirstName;

See also:

Next topic - Selecting top records only

Selecting records - the basics

Other types of SQL statements

Selecting top records only

You can select the top *n* records or the top *n* percent of the Employee table instead of the whole table. For example:

SELECT TOP 3 * FROM Employee;

SELECT TOP 20 PERCENT * FROM Employee ORDER BY Age;

The first example returns the first 3 records only and the second example the youngest 20% of employees. To return the oldest 20%, order the table in reverse:

SELECT TOP 20 PERCENT * FROM Employee ORDER BY Age DESC;

See also:

Next topic - Applying a condition

Selecting records - the basics

Other types of SQL statements

Applying a condition

Rather than selecting all the records from the Employee table, you can select exactly which records you'd like to work with by applying a condition. To do this, you use the WHERE command.

For example, to work with only those employees who are in the sales department:

SELECT Department, FirstName, LastName FROM Employee WHERE Department = "SALES";

The results table would look like this:

Department	FirstName	LastName
SALES	Andreas	Smith
SALES	Julia	Allan

Here's another example:

SELECT FirstName, LastName FROM Employee WHERE Age >= 30 ORDER BY LastName;

This returns a sorted list of employees over the age of 29.

The part of the SQL statement following the WHERE key word and preceding the ORDER BY keywords is called a condition and the '>=' is called the operator. Here's a list of all the operators you can use with Ability:

Operator	Meaning
=	Exactly Equal
!=	Not Equal
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To
LIKE	Partial match
IS NULL	Contains no data
AND	Must match both conditions
OR	Match either condition
NOT	Reverses logic
IN	Matches one of a list
BETWEEN	Lies in-between two values

By using the operators together you can always define a condition to return the records you want.

See also:

Next topic - Numeric comparisons

Selecting records - the basics

Other types of SQL statements

Numeric comparisons

Using the Employee table, here are some examples of queries using numeric comparisons:

SELECT FirstName, LastName, Age FROM Employee WHERE Age = 24 OR Age = 25;

SELECT FirstName, LastName, Age FROM Employee WHERE Age >= 24 AND Age <= 25;

SELECT FirstName, LastName, Age FROM Employee WHERE Age BETWEEN 24 AND 25;

SELECT FirstName, LastName, Age FROM Employee WHERE Age IN (24, 25);

All these produce the same results:

FirstName	LastName	Age
Phil	Roach	24
Andreas	Smith	25
Julia	Allan	25

You can exclude the above records, that is return everyone else, by reversing the logic with the NOT operator:

SELECT FirstName, LastName, Age FROM Employee WHERE Age NOT BETWEEN 24 AND 25;

SELECT FirstName, LastName, Age FROM Employee WHERE Age NOT IN (24, 25);

If you are chaining a series of conditions together, take care to use the parenthesis to denote the order the conditions are evaluated.

See also:

Next topic - Date comparisons

Selecting records - the basics

Other types of SQL statements

Date comparisons

These work in the same way as numeric comparisons (a date is just a number, starting at 0 for 1/1/1900), except you should enclose the dates in a pair of #'s.

For example using the Employee table, to see who has joined since 1990:

SELECT LastName, JoinDate FROM Employee WHERE JoinDate > #1/1/90#;

See also:

Next topic - Text comparisons

Selecting records - the basics

Other types of SQL statements

Text comparisons

Text comparisons are similar to numeric comparisons. For example, using the Employee table:

```
SELECT FirstName, LastName FROM Employee WHERE FirstName > "Jim";
```

The comparison is made alphabetically on the first letter, then the second letter and so on. The above would return "Julia" and "Phil" but not "Andreas", "Chris" or "Jim". Note that all text comparisons must appear between quotes.

See also:

Next topic - Wildcards

Selecting records - the basics

Other types of SQL statements

Wildcards

Wildcards allow one or more characters to be ignored in a text comparison.

For example, using the Employee table:

SELECT FirstName, LastName FROM Employee WHERE FirstName = "Julia";

returns an exact match only - if there was a "Julian" in the table, it would not be returned by the above SELECT.

To get around this, we can use the LIKE operator in conjunction with an asterisk "*" (the wildcard):

SELECT FirstName, LastName FROM Employee WHERE FirstName LIKE "J*";

This produces a match on any FirstName beginning with the letter "J", in this case "Julia" and "Jim" would be returned.

To find any first name containing the letter "h" use the asterisk twice:

SELECT FirstName, LastName FROM Employee WHERE FirstName LIKE "*h*";

This returns "Phil" and "Chris".

Another wildcard is the question mark, "?". This can be used to replace a single, unknown letter in a condition:

SELECT FirstName, LastName FROM Employee WHERE FirstName LIKE "?h*";

This would match "Phil" and "Chris" again, whereas "??i*" would only match "Phil".

See also:

Next topic - Selecting blank entries

Selecting records - the basics

Other types of SQL statements

Selecting blank entries

You can use the operator IS NULL to find or exclude fields with no data. For example, using the Employee table:

SELECT Company, Phone, FROM AddressBook WHERE Company IS NULL;

This would return all none-companies from and address book. To reverse this, and select all companies from the address book, use:

SELECT Company, Phone, FROM AddressBook WHERE Company IS NOT NULL;

See also:

Selecting records - the basics

Other types of SQL statements

Groups and aggregate functions

You can produce summary statistics for a table using functions on fields with the SQL statement.

For example, using the Employee table:

SELECT COUNT(Department), SUM(Age), MIN(JoinDate), MAX(JoinDate) FROM Employee;

Produces a single row of summary information:

COUNT(Department)	SUM(Age)	MIN(JoinDate)	MAX(JoinDate)
5	147	1-Oct-1986	10-Mar-1992

You can use these functions in conjunction with the GROUP BY keywords to produce consolidated group statistics. For example:

SELECT Department, AVG(Age), COUNT(*) FROM Employee GROUP BY Department;

This time we can include Department in the select statement as this is the field we are grouping on. The results table lists, in each department, the average age and the total number of people.

Department	AVG(Age)	COUNT(*)
ADMIN	30	1
SALES	25	2
TECH	33.5	2

As normal select statements have an optional WHERE clause to specify which rows are included from the table, so GROUP statements have an optional HAVING clause.

For example, to show a list of departments with an average age over 30, use the following:

SELECT Department, AVG(Age), COUNT(*) FROM Employee GROUP BY Department HAVING AVG(Age) > 30;

Here's a complete list of aggregate functions you can use:

Function	Meaning
COUNT	Count the number of records
SUM	Total
MAX	Find the maximum value of field
MIN	Find the minimum value of field
AVG	Average
VAR	Sample variance
VARP	Population variance
STDEV	Sample standard deviation
STDEVP	Population standard deviation

See also:

Grouping on more than one level

Finding duplicate records

Other types of SQL statements

Grouping on more than one level

You can group by more than one field. For example, using the Employee table:

SELECT Department, Age, COUNT(*) FROM Employee GROUP BY Department, Age;

Produces the following table:

Department	Age	COUNT(*)
ADMIN	30	1
SALES	25	2
TECH	24	1
TECH	43	1

See also:

Groups and aggregate functions

Finding duplicate records

Other types of SQL statements

Finding duplicate records

You can use GROUP BY to display a list of duplicate records by counting the consolidated records. For example, using the Employee table:

```
SELECT LastName, COUNT(*) FROM Employee GROUP BY LastName HAVING COUNT(*) > 1;
```

As only the name "Smith" is duplicated in LastName, the results table looks like this:

LastName	COUNT(*)
Smith	2

As you can group on several fields, you can construct more complex de-duplication queries. Using the EmployeeSales table as an example, the following query picks out the first and last occurrence of duplicated sales region / employee combinations. It also calculates a count of how many duplications occur.

```
SELECT Employee, Region, COUNT(*) AS Total, FIRST(SalesID) AS [First Match],
LAST(SalesID) AS [Last Match] FROM EmployeeSales GROUP BY Employee, Region
HAVING COUNT(*) > 1;
```

This produces the following table:

Employee	Region	Total	First Match	Last Match
John	North	2	1	2
John	South	2	5	8
Richard	North	3	4	7
Susan	South	2	3	9

See also:

Groups and aggregate functions

Grouping on more than one level

Other types of SQL statements

Crosstabulations

Crosstabulations (or contingency tables, or crosstabs for short), allow you to compare the entries in one field with those of another. For example, suppose we have three employees John, Richard and Susan, who make sales in two regions, north and south. The sales are recorded in a table, EmployeeSales. Such a table may look like this:

SaleID	Employee	Region
1	John	North
2	John	North
3	Susan	South
4	Richard	North
5	John	South
6	Richard	North
7	Richard	North
8	John	South
9	Susan	South

Suppose you want a breakdown of sales by region, for each salesman. This can be done using a TRANSFORM statement:

TRANSFORM COUNT(Region) SELECT Employee FROM EmployeeSales GROUP BY Employee PIVOT Region;

This produces the following results table:

Employee	North	South
John	2	2
Richard	3	
Susan		2

For each employee, at total of sales in each region is calculated.

Let's look at the general form of this SQL statement. Given that you want to compare field_a against field_b, the SQL is really a standard GROUP select, wrapped in a TRANSFORM and PIVOT:

TRANSFORM COUNT(field_a) SELECT field_b FROM table GROUP BY field_b PIVOT field_a;

Suppose we wanted row totals. All we need to do is add a COUNT within the SELECT statement:

TRANSFORM COUNT(Region) SELECT Employee, COUNT(Employee) AS [Employee Total] FROM EmployeeSales GROUP BY Employee PIVOT Region;

producing the following table:

Emloyee	Employee Total	North	South
John	4	2	2
Richard	3	3	
Susan	2		2

Note that the employee count is given a new column title on the fly using the AS keyword.

As well as counting fields, other statistics can be used. Here's a complete list:

Aggregate Function
FIRST
LAST
COUNT
MAX
MIN

See also:

Restricting pivot field values

Other types of SQL statements

Restricting pivot field values

You can choose to select specific values of the pivot field. For example, using the EmployeeSales table we can choose to look at only those records in the north region:

> **TRANSFORM COUNT(Region) SELECT Employee FROM EmployeeSales GROUP BY Employee PIVOT Region IN ("North");**

This produces the following table:

Employee	North
John	2
Richard	3
Susan	

Note that the value list after the IN keyword must be in brackets. The general form of this statement is:

> **TRANSFORM COUNT(field_a) SELECT field_b FROM table GROUP BY field_b PIVOT field_a; IN ("value 1", "value 2", "value 3", …., "value n");**

See also:

Crosstabulations

Other types of SQL statements

Relational links and joins

Suppose that for each employee, you wanted to keep a record of holidays taken. Such a table might look like:

HolidayID	EmployeeID	StartDate	DaysHoliday
1	1002	1/2/97	5
2	1003	21/2/97	1
3	1002	24/2/97	3
4	1004	1/3/97	4

Note that this table has a reference - EmployeeID - to the Employee table . This is called a Foreign Key and implies some important rules that should (but are not always) abided by:

1. Each record in the Holiday table must contain a valid EmployeeID, that is you can't have an EmployeeID in the Holiday table that doesn't also have a matching entry in the Employee table.
2. Each EmployeeID listed in the Holiday table can only exist once in the Employee table. EmployeeID forms a unique index for the Employee table called a primary key. The resultant relation between the Employee table and the Holiday table is one-to-many, that each employee can have zero, one or more holidays.

These rules provide the basis for *referential integrity*, a goal of good database design that Ability will try to help you achieve.

To join the two tables, use the following statement:

SELECT Employee.*, Holiday.* FROM Employee INNER JOIN Holiday ON Employee.EmployeeID = Holiday.EmployeeID;

This selects all the fields from both tables. To select some of the fields, care has to be taken not to confuse fields from one table with another – for example, the field EmployeeID exists in both tables. To avoid conflicts, tag on the table name to each field in the following manner:

SELECT Employee.EmployeeID, Employee.FirstName, Employee.LastName, Holiday.HolidayID, Holiday.StartDate, Holiday.DaysHoliday FROM Employee INNER JOIN Holiday ON Employee.EmployeeID = Holiday.EmployeeID;

This produces the following results table:

EmployeeID	FirstName	LastName	HolidayID	StartDate	DaysHoliday
1002	Chris	England	1	1/2/97	5
1002	Chris	England	3	24/2/97	3
1003	Andreas	Smith	2	21/2/97	1
1004	Jim	Smith	4	1/3/97	4

See also:

Join types

Unmatched queries

Other types of SQL statements

Join types

There are three types of join:

INNER JOIN
LEFT JOIN (otherwise known as "left outer join")
RIGHT JOIN (otherwise known as "right outer join")

Inner joins are the most common types of join and only return records that match in both tables.

For example, joining the Holiday table and Employee table using:

```
SELECT Employee.*, Holiday.* FROM Employee INNER JOIN Holiday ON
Employee.EmployeeID = Holiday.EmployeeID;
```

produces the following results table:

EmployeeID	FirstName	LastName	HolidayID	StartDate	DaysHoliday
1002	Chris	England	1	1/2/97	5
1002	Chris	England	3	24/2/97	3
1003	Andreas	Smith	2	21/2/97	1
1004	Jim	Smith	4	1/3/97	4

Note that there no records here for EmployeeID 1001 or 1005 since there are no matching records in the Holiday table, that is, these employees have not taken any holiday to date.

If you wanted to include every employee, you'd use a LEFT JOIN as follows:

```
SELECT Employee.*, Holiday.* FROM Employee LEFT JOIN Holiday ON
Employee.EmployeeID = Holiday.EmployeeID;
```

Every record from the table "to the left" of the join statement is included.

Similarly, you can issue a RIGHT JOIN statement to include all records from the table to the right of the join statement:

```
SELECT Employee.*, Holiday.* FROM Employee RIGHT JOIN Holiday ON
Employee.EmployeeID = Holiday.EmployeeID;
```

Strictly speaking, this should return the same results table as the INNER JOIN, since it makes no sense to assign holidays to non-existent employees. However, this can sometimes happen, especially with "historical" data – data imported from a system not set-up to obey referential integrity.

See also:

Relational links and joins

Unmatched queries

Other types of SQL statements

Unmatched queries

LEFT and RIGHT JOINS are useful for finding unmatched records. For example, To produce a list of all employees who have not taken holidays, using the Employee and Holiday tables:

> **SELECT Employee.* FROM Employee LEFT JOIN Holiday ON Employee.EmployeeID = Holiday.EmployeeID WHERE Holiday.EmployeeID IS NULL;**

To produce a list of holidays that have not been assigned to any employee, if any exist:

> **SELECT Holiday.* FROM Employee RIGHT JOIN Holiday ON Employee.EmployeeID = Holiday.EmployeeID WHERE Employee.EmployeeID IS NULL;**

See also:

Relational links and joins

Join types

Other types of SQL statements

Editing records

SQL statements can make changes to one, many or all records, so all editing type SQL commands need careful application.

Ability supports three type of these SQL commands:

Update	modify records
Insert	add records or tables
Delete	remove records

See also:

Other types of SQL statements

Updating records

The SQL statement UPDATE can be used to edit or update fields in one, many or all records in a table. These types of queries don't produce results tables – you have to issue a further SELECT statement to view the results. Here are some examples, using the Employee table :

UPDATE Employee SET Department="MARKETING";

Note that this replaces the Department field for *every* record. To limit the scope of the update, use the WHERE clause:

UPDATE Employee SET Department="MARKETING" WHERE EmployeeID = 1005;

This performs an update to a single record. In the case where EmployeeID is a primary key, this form of update is guaranteed to modify a single record (at most).

To update several fields at once, list them after the SET keyword. For example, if an employee changed department and got married at the same time:

UPDATE Employee SET Department="MARKETING", LastName = "Pallister" WHERE EmployeeID = 1005;

See also:

Adding records

Deleting Records

Other types of SQL statements

Adding records

To add a record to a table, use the INSERT command and specify at least some field information. For example, to add a new employee to the Employee table, we'd at least need to assign a new EmployeeID:

INSERT INTO Employee (EmployeeID) VALUES (1006);

To create a new record and fill in the field details at the same time, list the fields after the table, and each value in order after the VALUES keyword:

INSERT INTO Employee (EmployeeID, FirstName, LastName, Department, Age, JoinDate) VALUES (1007, "Joel", "Coleman", "ADMIN", 49, #1/5/97#);

Note that you surround each new value with quotes, unless the field is of a numeric type where the quotes are omitted or date where the hash "#" sign is used instead. Also note that both the field list and the values list are enclosed with brackets.

See also:

Adding multiple records

Creating tables from existing data

Updating records

Deleting Records

Other types of SQL statements

Adding multiple records

You can use the INSERT command to add many records from one table to another. For example, if there was a table called NewEmployee, with an identical structure (field list) to Employee, you could add all the records from NewEmployee to Employee using the following statement:

> **INSERT INTO Employee SELECT * FROM NewEmployee;**

If the two tables are not identical, or you only want to append certain fields, list the fields for both source and destination table. For example, you want to add records to the Employee table from an old Employee table containing different field names:

> **INSERT INTO Employee (EmployeeID, FirstName, LastName) SELECT (EmpID, Fname, Lname) FROM OldEmployee;**

Note the SELECT part of the INSERT statement follows all the rules for general SELECT statements described above. For example, you can specify which records to append using the WHERE clause:

> **INSERT INTO Employee (EmployeeID, FirstName, LastName) SELECT (EmpID, Fname, Lname) FROM OldEmployee WHERE JoinDate < "1/1/80";**

See also:

Adding records

Creating tables from existing data

Updating records

Deleting Records

Other types of SQL statements

Creating tables from existing data

The combination of SELECT and INTO can create new tables based on existing information. For example, to create a table of employees in the sales department from the Employee table, use the following statement:

SELECT * INTO Sales FROM Employee WHERE Department = "SALES";

To create a table with just names, use:

SELECT Employee.FirstName, Employee.LastName INTO Sales FROM Employee WHERE Department = "SALES";

To create a complete copy of a table, just omit the WHERE clause:

SELECT * INTO EmpCopy FROM Employee;

See also:

Adding records

Adding multiple records

Updating records

Deleting Records

Other types of SQL statements

Deleting Records

The DELTE command allows you to drop records, permanently, from a table. There's no "undo" for this operation, so make sure you have adequate back-ups in case something goes wrong. A good safety measure is to use a SELECT statement first to see what records will be deleted.

For example, using the Holiday table, if we are no longer interested in holidays from the 1980's, we could check which records fell into this category by:

SELECT * FROM Holiday WHERE StartDate < "1/1/90";

Then delete them with:

DELETE * FROM Holiday WHERE StartDate < "1/1/90";

To drop all the records from a table:

DELETE * FROM Holiday;

Please use with care!

See also:

Adding records

Updating records

Other types of SQL statements

Employee table

The example employee data:

EmployeeID	FirstName	LastName	Department	Age	JoinDate
1001	Phil	Roach	TECH	24	23/1/87
1002	Chris	England	TECH	36	1/10/86
1003	Andreas	Smith	SALES	25	18/6/90
1004	Jim	Smith	ADMIN	30	10/3/92
1005	Julia	Allan	SALES	25	26/9/91

EmployeeSales table

The example employee sales data:

SaleID	Employee	Region
1	John	North
2	John	North
3	Susan	South
4	Richard	North
5	John	South
6	Richard	North
7	Richard	North
8	John	South
9	Susan	South

Holiday table

The example holiday data:

HolidayID	EmployeeID	StartDate	DaysHoliday
1	1002	1/2/97	5
2	1003	21/2/97	1
3	1002	24/2/97	3
4	1004	1/3/97	4