



## Preface

SQL (structured query language) is one of the main workhorse languages used today. Entire programs are written in it, for it, based on it... the list is endless. Although primarily used for database storage and retrieval, you will find that often times it is as a integral part of complex websites, tracking software, accounting software, and many other types of applications.

There are many different SQL database applications that exist on today's market. Most of the major vendor database applications you have heard of before: MS Access, Foxpro, MS SQL Server, Oracle, Informix. Each application has its own custom menus and methods of operation, but if you dug deep enough you would find that it eventually all breaks down into simple and complex SQL statements.

This primer has been purposely written to no specific database program. All the SQL queries used should be compatible with every major database system (please note: there may be some slight syntax differences in Oracle or Informix).



## Part 1. Using SELECT to retrieve specific information from tables

1. Retrieving all records in a table
  - a. All columns
  - b. Specific Columns
  
2. Using the WHERE clause
  - a. Basic where clause using =
  - b. Using NOT in a where clause
  - c. Using AND / OR in a where clause
  - d. Using IN / NOT IN
  - e. Using DISTINCT
  
3. Ordering, Counting, and Naming the Output
  - a. Listing selected data in a specific order
  - b. Counting values in a table
  - c. Assigning Names to your Output using AS

For the first part of this primer, we will be using the following three Almis 2.2 database tables: geog, license, occcodes

# SQL PRIMER

## Chapter 1: Retrieving all records in a table

### Lesson 1.1: Using the "Select-All"

**Explanation:** It is often useful to display every record in a table and all of its associated values. Sometimes you will see this referred to a "select-all" because you are viewing data for every column in the table. The general form for this statement is:

*Select \* from <table>*

---

**Activity:** Run the following select-all queries

select \* from geog

select \* form license

select \* from occcodes

---

# SQL PRIMER

**Chapter 1:** Retrieving all records in a table

**Lesson 1.2:** Specific columns

**Explanation:** Rather than displaying every bit of information inside a table, lets say that you only want to look at all the results for specific columns. If this is the case, then it is better to run a query that only returns those results. The general form for this statement is:

```
select <column name> from <table>
```

---

**Activity:** Run the following select-all queries

```
select stfips, area, areaname from geog
```

```
select lictitle from license
```

```
select codetype, codetitle from occcodes
```

---

# SQL PRIMER

## Chapter 2: Using the WHERE Clause

### Lesson 2.1: Basic where clause using =

**Explanation:** So far, our SELECT statements have returned every value for every record in the table regardless of their value. Most of the time however you only wish to see data that meets certain criteria. The WHERE clause lets us specify a constraint (called a predicate) that will separate the values we want to see from the values that we are not interested in. The general form for this statement is:

```
select <column names>  
  
from <table>  
  
where <column name> <operator> <value>
```

---

**Activity:** Run the following select-all queries using basic where clauses

```
select * from geog where stfips = '88'
```

```
select * from license where occcode = '32102'
```

```
select * from occcodes where codetype = '02'
```

---

## Chapter 2: Using the WHERE clause

### Lesson 2.2: Using NOT in a WHERE clause

**Explanation:** The use of not is a very simple concept – it will negate whatever predicate you use in the where statement, thus displaying opposite results of what you would see without it.

---

**Activity:** Run the following select-all queries using basic where clauses + NOT

```
select * from geog where NOT (stfips = '88')
```

```
select * from license where NOT (occcode = '32102')
```

```
select * from occcodes where NOT (codetype = '02')
```

---

## Chapter 2: Using the WHERE clause

### Lesson 2.3: Using AND / OR in a WHERE clause

**Explanation:** It is often useful to combine two or more predicates in your where clause to further refine the data retrieved from the table. Two ways to combine constraints are using AND and OR.

**Definition of AND:** If both predicates are true, then the result is returned.

**Definition of OR:** If either one of the predicates are true, then the result is returned.

---

**Activity:** Run the following select-all queries using AND / OR

```
select * from geog where stfips = '88' and areatype = '04'
```

```
select * from occcodes where codetype = '02' and stfips = '88'
```

```
select lictitle from license where occcode = '32102' and uniqueid = '140'
```

\*\*Question: why did the last License query return no results? \*\*

---

```
select * from geog where stfips = '88' or stfips = '37'
```

```
select * from license where lictitle = 'Lawyer' or lictitle = 'Dentist'
```

```
select * from occcodes where codetype = '02' or codetype = '04'
```

---

## Chapter 2: Using the WHERE clause

### Lesson 2.4: Using IN / NOT IN in a WHERE clause

**Explanation:** Another way to enhance your predicate constraint is with the use of IN and NOT IN. This function can be logically equivalent to constructing multiple OR statements together. The general form for this query is:

```
select <column names>
```

```
from <table>
```

```
where <column name> IN (or NOT IN) <values>
```

---

**Activity:** Run the following select-all queries using IN / NOT IN

```
select * from geog where stfips IN ('37','88')
```

```
select * from license where lictitle = 'Lawyer' or lictitle = 'Dentist'
```

```
select * from occcodes where codetype NOT IN ('02','04','08')
```

---

## Chapter 2: Using the WHERE clause

### Lesson 2.5: Using DISTINCT in a WHERE clause

**Explanation:** In many cases you will have several records of a table with the same value. Sometimes you may not be interested in the duplicates, but more interested in what values appear in the table column at all. The DISTINCT tag should be used to accomplish this. The general form of this query is:

```
select DISTINCT (<column name> )  
    from <table>
```

---

**Activity:** Run the following queries using DISTINCT

```
select distinct(stfips) from geog where areatype = '04'
```

```
select distinct (occcode) from license
```

```
select distinct codetitle from occcodes
```

---

## Chapter 3: Listing selected data in a specific order

### Lesson 3.1: Using ORDER BY

**Explanation:** When designing a query, you more than likely have an idea not only what you are looking for, how it should be displayed in. Keep in mind that there IS NO DEFAULT ORDER when running a select statement. Therefore, when designing any query that might be used more than once, you will almost always include an ORDER BY clause. The general form for this is:

```
select <column names>  
  
from <table>  
  
where <predicate>  
  
order by <column names>
```

---

**Activity:** Run the following queries using ORDER BY

```
select * from occcodes  
order by codetitle → (default)
```

```
select occcode, lictitle from license where area = '000088'  
order by occcode desc → (desc means descending order)
```

```
select * from geog  
order by areaname, stfips
```

---

## Chapter 3: Counting values in a table

### Lesson 3.2: Using COUNT

**Explanation:** Sometimes you might be in a situation where you may not need the data in the table, you just want to know how many records there are. An example of that type of question would be if someone asked how many states there were in the US. This type of a question is easily answered using the COUNT function.

---

**Activity:** Run the following queries using COUNT

```
select count(*) from license
```

```
select count(*) from occcodes where codetype = '02'
```

```
select count(distinct stfips) from geog
```

---

## Chapter 3: Assigning names to your output

### Lesson 3.3: Using AS

**Explanation:** When using complex where statements or aggregate functions (more on these later) in your constraint, it will sometime help to give names to the output that the query generates. This can be accomplished by using the AS keyword.

---

**Activity:** Run the following queries using AS

```
select count(*) as 'TheCount' from occcodes  
where codetype = '02'
```

```
select areaname AS 'list Of Counties in Generica' from geog  
where areatype = '04' and stfips = '88'
```

```
select licdesc AS 'What a dentist does' from license  
where lictitle = 'Dentist'
```

---



## Part 2. Retrieving data with operators, aggregates, and wildcards

4. Retrieving data by using ranges
  - d. Use of NULL
  - e. Use of Mathematical Operators
  
5. Using wildcards and searches with LIKE
  - a. Using the \*
  - b. Using the \_
  
6. Using aggregate functions
  - a. Defining aggregates
  - b. Using SUM, MAX, MIN

For the second part of this primer, we will be using the industry and the indprj Almis 2.2 database tables.

## Chapter 4: Retrieving data by ranges

### Lesson 4.1: Use of NULL

**Explanation:** When constructing tables, the data in each record must follow the strict guidelines that have been set by the design. Numeric values only belong in number fields, extremely large text only belongs in varchar fields, etc. However, there is one specific value that can be assigned anywhere – the NULL value. Null does NOT represent zero – null represents the complete absence of data for that field.

Any type of field can be assigned to a null value as long as it is allowed in the table design. For example, let's look at the industry table to examine NULL values. Then we will look at something more specific (notice carefully the syntax on the second statement).

---

**Activity:** View Industry and examine the nulls:

```
select * from industry
```

**Activity:** View Industry and examine all records with NULL firms values:

```
select * from industry where firms IS NULL
```

---

## Chapter 4: Retrieving data by ranges

### Lesson 4.2: Use of Mathematical Operators

**Explanation:** As in many other programming languages, SQL recognizes the most common syntax to perform basic mathematic functions:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Greater than	>
Less than	<
Equal to	=
Is not equal to	<> (or !=)

---

**Activity:** Suppose the data in the avgemp field is listed as 1000's of people. Write a query to view raw value of avgemp, as well as the stfips, area, and areatype.

```
select stfips, area, areatype, (avgemp * 1000) as RawData
from industry
```

**Activity:** Run the same query but now find the avgDailyWage (assume 5 days to a week)

```
select stfips, area, areatype, (avgemp * 1000) as RawData,
(avgkwage / 5) as avgDailyWage from industry
```

---

## Chapter 5: Use of Wildcards and searches with LIKE

**Lesson 5.1:** Use of \*

**Lesson 5.2:** Use of \_

**Explanation:** When searching for information, we often know only a piece of what we're after. In this case SQL provides the LIKE function (also can be used as NOT LIKE). The general form for this query is:

```
select <column name>
    from <table>
    where <column name> LIKE <value>
```

There are two wildcard characters that can be used – % or \_ . A % is used to represent any length & value of characters, where a \_ is used to represent any value for a single character.

---

**Activity:** Selecting records using %

```
select * from inddir where matintitle like 'Computer%'
```

```
select * from inddir where matintitle like '%Computer%'
```

**Activity:** Selecting records using \_

```
select distinct (indcode) from industry where indcode like '239_'
```

```
select distinct (indcode) from industry where indcode like '_3_1'
```

---

## Chapter 6: Using Aggregate functions

### Lesson 6.1: Definition of Aggregates and their use

**Explanation:** The SQL language contains several pre-defined functions that are usually referred to as "*aggregates*." Each one of these functions could be calculated with raw SQL statements, but since they are included as part of the language, it is much easier to use these functions instead.

MAX	Determine a column's maximum value
MIN	Determine a column's minimum value
SUM	Determine the sum of a column's values
AVG	Determine the average value of a column
COUNT	Count the number of rows that exist

Using aggregates in combination with complex where clauses, grouping methods, or joins (more on the latter two later) these simple aggregate functions can produce some of the most important data out of a SQL database.

# SQL PRIMER

**Chapter 6:** Use of Wildcards and searches with LIKE

**Lesson 6.2:** Use of Sum, Max, Min

**Explanation:** We have already defined what each of the aggregate functions do. Now it is time to run them together and see the results.

---

**Activity:** Run the following query first to display all area data for the given industry code:

```
select * from industry where periodtype = 2 and period = 01 and  
indcode = 6500
```

**Activity:** Run the following using Max, Min and SUM

```
select sum(avgemp) as 'AnnualAvgemp', max(avgemp) as  
'Highest Area', min (avgemp) as 'Lowest Area'  
from industry  
where periodtype = '02' and period = '01'  
and indcode = '6500' and areatype = '04'
```

\*\* Class should determine exactly what it is that we are running and why it may be important.\*\*



## Part 3. Adding, Modifying, and Deleting data

### 7. Deleting data from the database

- a. Use of DELETE

### 8. Creating and Destroying Tables

- a. Dropping a table
- b. Creating a table

### 9. Adding data to the database

- a. Using INSERT
- b. Use of UPDATE

For the third part of this primer, we will be using the **empdb** and **dislocat** Almis

2.2 database tables. In its complete form, dislocat appears as:

dislocated	dislocdesc
0	Total applicants
1	Dislocated
2	Not dislocated
9	Unknown

## Chapter 7: Deleting records in a table

### Lesson 7.1: Use of Delete

**Explanation:** Deleting records from a table is a very simple task. The general form of the query is:

```
delete from <table>  
where <predicate>
```

If you wish to delete all the records from a table, then do not add any predicate at all to the delete query. Please be aware this action is NOT REVERSIBLE – once you issue the delete, the data will be gone.

---

**Activity:** Delete all the records from the dislocat table

```
delete from dislocat
```

---

# SQL PRIMER

## Chapter 8: Creating and dropping tables

**Lesson 8.1:** Use of DROP TABLE

**Lesson 8.2:** Use of CREATE TABLE

**Explanation:** Deleting an entire table is also a very simple task. The general form of the query is

*DROP <table>.*

Creating a table, while it sounds fairly simple, can become complicated. The general form of the query is

*CREATE TABLE <table>*

*(<column descriptions>);*

**\*\*There is very little reason to use raw SQL to create tables, because every database administration tool has a *wizard* program to do this. Also, the SQL for a table with a large # of columns and restraints would be very difficult – empdb for example. It is simply included in the primer to help the class understand that SQL is used to perform this function.\*\***

---

**Activity:** None

---



## Chapter 9: Adding data to the database

### Lesson 9.1: Use of Insert

**Explanation:** Once a table is created, it will contain zero records (rows).

Since no records exist, you must create a new record for data to be stored in.

This is accomplished by using the INSERT tag in the following way:

```
INSERT INTO <table>(<column names>)  
VALUES<values>
```

If you are adding values to each column, then you do not need to specify the column names.

---

**Activity:** Insert 4 records into dislocat to return it to its previous condition (see above if necessary)

```
INSERT INTO dislocat  
VALUES ('0','Total Applicants')
```

\*\*the class should create the other 3 records needed

---

## Chapter 9: Adding data to the database

### Lesson 9.2: Use of Update

**Explanation:** It is important to understand the difference in UPDATE and INSERT. If you ever need to increase the # of records in a table, you must use the INSERT function. Otherwise, you can use UPDATE to change any existing value in a table record. The general form for the query is:

```
UPDATE <table>  
  
    SET <column name> = <value>  
  
    WHERE <predicate>
```

---

**Activity:** Use the UPDATE function to make the following changes in empdb:

city: Capital → Capital City  
city: Big City → Metropolis

```
UPDATE empdb  
  
    SET city1 = 'Capital City'  
  
    WHERE city1 = 'Capital'
```

\*\*the class should create the other records needed

---



## Part 5. Advanced Topics

### 10. Grouping

- a. Explanation of concept:
- b. Example: Calculate state data from county data

### 11. Subqueries

- a. Explanation of concept:
- b. Example: Calculate siccodes not present in industry

### 12. Joins

- a. Explanation of concept:
- b. Example: Report all 2001 private industry info requested

### 13. Other concepts

- a. Aliases
- b. Stored Procedures
- c. Views
- d. Triggers
- e. Privileges



## Chapter 10: Grouping

### Lesson 10.1: Use of GROUP BY

**Explanation:** Quite often (especially in ALMIS) the data that is collected can be organized into groups. This is done by using the GROUP BY clause. The general form for the query is:

```
select <column names>  
    from <table>  
    where <predicate>  
    group by <column names>
```

While sounding simple enough, using GROUP BY can become easily confusing. Every column of data that is returned by the select statement must have only one of these two properties:

1. Part of the GROUP BY clause
2. Calculated by using an Aggregate

So for example, let's suppose you have a table with 5 fields and you wish to return data from each of them. If you decide to group the data together with common values for the first 3 columns, then the data from the last two can only be pulled by one of the aggregate functions.

## Chapter 10: Grouping

### Lesson 10.2: Example of a GROUP BY statement

---

**Activity:** Create state industry data using only available county data

```
insert into industry
select
    stfips, '01' as areatype, '000088' as area, periodyear, periodtype,
    period, indcodty, indcode, ownership,
    max(firms) as firms,
    sum(estab) as estab,
    sum(avgemp) as avgemp,
    sum(mnth1emp) as mnth1emp,
    sum(mnth2emp) as mnth2emp,
    sum(mnth3emp) as mnth3emp,
    sum(topempav) as topempav,
    sum(totwage) as totwage,
    (sum(totwage) / sum(avgemp) / 52) as avgwk wage,
    sum(taxwage) as taxwage,
    sum(contrib) as contrib,
    '0' as suppress

from industry
where areatype = '04'
group by
    stfips, periodyear, periodtype, period,
    indcodty, indcode, ownership
```

---

## Chapter 11: Subqueries

**Lesson 11.1:** Use of subqueries

**Lesson 11.2:** Subquery example

**Explanation:** A “relational” database such as ALMIS consists entirely of tables. Every piece of info is stored in a record of some table; sometimes it is stored in multiple tables. We have repeatedly used SELECT to view data out of tables. You may have noticed though that the results that select return are also in a table. Sometimes you might need to select data out of the new “table” that you have created by the first SELECT query. This type of action is called a subquery.

---

**Activity:** Run a report that shows all of the siccodes that do not have industry data.

```
select code
      from indcodes
      where codetype = 05 and code not in
            (select distinct(indcode)
              from industry )
order by code
```

---

## Chapter 12: Joins

**Lesson 12.1:** Use of joins

**Lesson 12.2:** Join example

**Explanation:** So far every query we have run has been executed on one table at a time. However, part of the power of a relational database is being able to pull data from multiple sources at once. Any query that extracts data from more than one table must perform what is called a "join."

---

**Activity:** Run a report that shows the annual industry data for 2001 – private sector only. It should show each industry title, average employment, total wages, and the average weekly wage.

```
select indcodes.codetitle, avgemp, totwage, avgwk wage
from industry
      join indcodes on industry.indcode = indcodes.code
where areatype = '01' and ownership = '50'
```

---

*or it can be written as*

```
select indcodes.codetitle, industry.avgemp,
      industry.totwage, industry.avgwk wage
from industry, indcodes
where areatype = '01' and ownership = '50' and
      industry.indcode = indcodes.code
```

# SQL PRIMER

## Chapter 13: Other concepts and definitions

**Aliases:** When writing long queries using multiple tables, sometimes it is useful to give your tables alias names. This allows shorter keystrokes per query required.

Example: `select B.codetitle, A.avgemp, A.totwage, A.avgwkwage  
from industry A, indcodes B  
where areatype = '01' and ownership = '50' and A.indcode = B.code`

**Stored Procedures:** Just as an alias is a short hand notation for a table in a query, a stored procedure is a shorthand notation for a pre-determined set of instructions. Queries that are used repetitively (especially those by a website for example) are often written as stored procedures so they may be used easily.

**Views:** Views are best described as “virtual tables.” A view is not a stored table in the database, but rather instructions on how to create a temporary table for the user when needed. They become useful when trying to show multiple table data (especially to non-system admins).

**Triggers:** A trigger is a special type of stored procedure that is automatically executed when some predetermined action occurs. Triggers are used in Almis 2.2 to maintain indcodes and occcodes for example.

**Privileges:** This is the ability for a system admin of a relational database admin to be able to set which users can view/execute objects in the database.