# Structured Query Language Select & Join Commands

By Farrokh Alemi version Tuesday, May 30, 2017

Edited by Nancy Freeborne on February 3rd 2017

## Introduction

 Data in electronic health records is in large number of tables, linked to each other through primary keys. Before any analysis can be done, data must be prepared in a matrix format so that all relevant variables are present in the same table. Many statistical books do not show how this could be done and thus leave the analyst at a disadvantage in handling data from electronic health records. These books do not teach use of Standard Query Language (SQL). This book could also be used without knowing SQL commands, perhaps the reader can skip sections that describe the application of the SQL command. We do not recommend it but it can be done. We take a different approach from most statistical books and believe that SQL and data preparation are important components of data analysis. An analyst who wants to handle data in electronic health records needs to know SQL. The book teaches SQL because it matters in statistical analysis of the data. Accurate statistical analysis requires careful data preparation. Statisticians who learn statistics without deep understanding of data preparation may remain confused about their data. It is akin to living your life not knowing your parents, where you came from, or for that matter who you are. You can live your life in a fog but why do so? Knowing the source of the data and its unique features can help the analyst have insights into anomalies in the data. Manipulating the data, preparing it for analysis, looking at the data in different ways with different definitions, gives the analyst intuitions and insights into the data.

SQL gets to what statisticians do with most of their time. Statisticians spend more time on preparing the data than actually conducting the analysis. Perhaps 80% of data analyst’s time is spent in preparing data. That is a large portion of a statistician’s day to day tasks. Ignoring it would significantly handicap the statistician. Knowing SQL helps with the bulk of what statistical analysts do. Training in it is essential.

Of course, we do not need the statistician to become a computer programmer. This, after all, is a course on statistics and not computer programming. Thankfully, SQL programming is relatively easy (there are few commands) and can be picked up quickly. This chapter exposes the reader to the most important SQL commands. These include select, group by, where, join and some key functions. These commands are for the most part sufficient to do everything in this book. These commands help the analyst do many different transformations of data. While there are few SQL commands, the complexity of the data makes use of these commands difficult. A great deal of practice, and many checks and balances within the code, are necessary to make sure that the commands are appropriately used and the transformed data are meaningful.

 Decisions made in preparing the data could radically change the findings. These decisions need to be made carefully and transparently. The statistical findings depend on these decisions and the analyst must make every attempt to communicate the details of these preparations to the manager. Managers and policymakers must assure that decisions made in preparing the data are well thought out or they will be viewing erroneous data findings and possibly making incorrect major strategic decisions. Some common errors in preparing data include the following:

1. Visits and encounters reported for deceased patients. For example, when a patient’s date of visit or date of death is miss-entered then it may look like dead-patients (zombies) are visiting the provider. Errors in entry of dates of events would skew results and thus cleaning up these errors are crucial.
2. Inconsistent data (e.g. pregnant males) must be identified and steps must be taken to resolve these inconsistencies.
3. Incongruous data, such as short patient stay in patients with a medication error, must be reviewed. When a medication error occurs, one would expect to see that these patients have longer hospital stays. If that is not the case, one should review the details of the case to see why not.
4. Steps should be taken to resolve missing information. Sometimes, missing information could be replaced with the most likely response; other times missing information could be used as a predictor. For example, if a diagnosis is not reported in the record then it is likely that the patient did not have it. At the same time, the reverse could be true. If an emergency room patient is missing key diagnoses, then it is likely that there was no time to diagnose the patient but the patient had the diagnoses. Several studies have found that missing diagnoses in emergency rooms increases risk of subsequent mortality suggesting that in these situations a missing diagnosis does not mean that the patient did not have it. Before proceeding with the analysis, missing values must be imputed. There are many different strategies for dealing with missing values and the rationale for each imputation should be examined.
5. Information is double counted because of errors in joining tables. A common error is when in process of joining tables, analysts use a non-unique primary key and duplicate sets of data.

In short, a great deal must be done before any data analysis commences.

## Linking Data in Multiple Files

 Data in electronic health records are often simultaneously present in multiple files. Patient information is in one table. Prescription data is in another. Data on diagnoses are often in an outpatient visit table. Hospital data are still in another table. An important first step in any data analysis is to pull various variables of interest into the same table. These efforts lead to a large, often sparse, table where all the variables are present but many have missing values. The reason for the potentially missing values is that for instance, Patient X could have a diagnosis and prescription data but not hospital data if he/she was never hospitalized. Patient Y could have a diagnosis, prescription data, and hospital data but missing some other data (for instance surgical procedure if he/she did not have any surgery). The procedure to pull the data together requires the use of Standard Query Language (SQL). An introduction to SQL is beyond the scope of this book but we discuss some SQL commands that are often used in merging the data so that the reader is familiar with the concept. Throughout this book and especially in this chapter, we give brief SQL commands for preparing the data, so the reader can familiarize themselves with the key terms used in SQL.

 There are different implementations of SQL. In this chapter, we use the Microsoft Access SQL. Access is widely available to managers. Other versions of SQL such as dynamic SQL or Microsoft Server SQL are also available. If the reader is familiar with the concept of code laid out here, he/she can also find the equivalent version of the code in a different language on the web. To assist the reader, We also use also provide code using Microsoft SQL server.

 In EHRs, data reside in tables. A table is a collection of fields, and fields are variables that provide information. One of the fields in the table is a *primary key* and referred to as ID for the table. A primary key is a unique number for each row of data in the table. All of the fields in the table provide information about this primary key. For example, we may have a table about the patient, which would include demographics and contact information, and a separate table about the visit. The primary key for the patient is a patient identifier such as medical record number. A primary key for the visit table is a visit ID. The fields in the patient table (e.g. address) are all about the patient; and the fields in the visit table (e.g. diagnoses) are all about the visit. The relationships among the tables are indicated through repeating the primary key of one table in another table. In these situations, the key is referred to as a *foreign key*. For example, in the visit table we indicate the patient by providing the patient ID. Database designers do not provide any other information about the patient, e.g. his/her address, in the visit table so as to have efficient databases. In other words, databases use as little information as they can to preserve space and to improve data analyses time.

 As an example assume that you need to prepare a database that contains three entities:  Patients, Providers and Encounters. For each of these three entities, we need to create separate tables.  Each table will describe the attributes of one of the three entities.  Each attribute will be a separated field. For simplicity assume that the patient attributes are assumed to be first name, last name, date of birth, address (street name, street number, city, State and zip code) and email.  First name is a string with maximum size of 20 characters. Last name is a string with maximum size 50 characters.  Street number and zip code are integer numbers with no decimals.  Date of birth is a date displayed in the format DD MM YY (e.g. 19 Jan 04).  The possible values for the State are: Maryland, Virginia, District of Columbia and other.  Patient's telephone number could be numbers read as text (especially if they include parentheses and other non-number characters) or number.  A patient ID (auto-number) is used as the primary key for the table.  Table 1 shows the first three rows of data in our hypothetical patient table. Note that two patients are shown to live in the same household and have same last names.

**Table 1:  Three Rows of Data for Example Patient Table**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **First Name** | **Last Name** | **Zip Code** | **City** | **State** | **Date of Birth** | **Email** | **Telephone of the Patient** |
| Larry | Kim | 22101 | Mclean | DC | 08-Jan-54 | email@test.edu | 703-9934226 |
| George | Smith | 22102 | McLean | Virginia | 09-Sep-60 | email@tes.com | (703) 8884545 |
| Jill | Smith | 22102 | McLean | Virginia | 01-Aug-89 | test@test.com | 703 993 4226 |

 The provider attributes are assumed to be first name (text of maximum size 20), last name (text of maximum size 50), whether they are board certified (a yes/no value), date of hire (displayed in format DD MM YY), telephone (text or number) and email (text of size 75). Employer's ID number should be the primary key for the table. Table 2 shows the first three rows of data for providers; note that one of the patients is also a provider.

**Table 2:  Three Rows of Data for Example Providers Table**

| **First Name** | **Last Name** | **Board Certified** | **Email** | **Telephone** | **Employee****ID** |
| --- | --- | --- | --- | --- | --- |
| Jim | Jones | Yes | jl@w.com | 3456714545 | 452310 |
| Jill | Smith | No | js@w.com | 3454561234 | 454545 |
| George | John | Yes | g@w.com | 3104561234 | 456734 |

 The encounter entity is assumed to have the following attributes:  patient ID, provider ID, diagnosis (text of maximum size 50), treatment (text of size 50) and date of encounter (date entered in the format DD MM YY).  Each encounter should have its own ID number.  Table 3 shows the first five rows of the encounter table.

**Table 3:  Five Records for Example Encounters Table**

| **ID** | **Patient ID** | **Provider ID** | **Date of Encounter** | **Diagnosis** | **Treatment** |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 452310 | 10-Jan-04 | Hypertension | Assessment |
| 2 | 1 | 452310 | 17-Jan-04 | Heart Failure | Monitoring |
| 3 | 2 | 452310 | 10-Jan-04 | Null | Assessment |
| 4 | 3 | 452310 | 10-Jan-04 | Hypertension | Assessment |
| 5 | 1 | 454545 | 10-Jan-04 | Asthma | Education |

Figure 1 shows the relationship between patient, encounter, and provider entities in our hypothetical electronic medical record. In the encounter table, we have two foreign keys: patient ID and provider ID. These foreign keys link the encounter table to the patient and provider tables.

**Figure 1: Example of Relationship among Three Hypothetical Tables**



Sometimes, when the relationships are complex, a *junction table* is created to keep track of the relationships. A junction table is a table about relationships. For example, a patient may have multiple street addresses. It does not make sense to provide more than one addresses in the patient table. A separate table is made for addresses and a foreign key in the patient table would point to the address table. A junction table may be inserted in between the patient and the address table clarifying the type of address.

## Making a Query

Databases contain a great deal of information in separate tables.  An analyst is often called upon to integrate these separate tables so as to find answers to specific questions.  This process is called querying a database.  SQL is the list of commands that can be used to query a database.  An example is provided below:

SELECT Claims.PatientID, LAST(ICD9.ICDDescription) AS LastOfICDDescription

FROM Claims INNER JOIN icd ON Claims.DiagnosisCode = ICD9.ICD9Codes

WHERE ICD9.ICDDescription LIKE "\*diabete\*"

GROUP BY Claims.PatientID ORDER BY Claims.PatientID;

In the above SQL code, all commands are in capital letters. The “SELECT” command identifies the variables that should be in the new, integrated, table. In specifying a field, you need to also indicate the source table. Thus “Claims.PatientID” indicates that “PatientID” comes from the “Claims” table. Fields can be included from all joined tables. In this fashion, a select query makes information in various tables available in one new table. In addition, the new variable can be calculated from more than one other variable.

The “FROM” command specifies which tables should be used. If the data are in more than one table then the tables must be joined. The “INNER JOIN” sub-command of FROM specifies the keys that should be matched in joining two tables. There are at least four different types of join with difference consequences.

1. One-to-One Join: This is the most common join. The fields in both tables must be exactly the same before the content of the tables are joined together. For example, one might have a table Claims that contains diagnoses codes. The meaning of these diagnostic codes might be available in a separate table called [Diagnosis Codes]. A join can select the text for the diagnostic code and combine it with the data in Claims table. A one-to-one join will lead to listing of all claims in which the diagnostic code has a corresponding text in diagnosis table. In a one-to-one join, if the description of the diagnosis code is missing in the Diagnosis Code table, then all corresponding claims will not show. It is important to be careful that as a consequence of a one to one join important data are not missed. For example, suppose that we have two tables described in Table 4, one containing description of diagnosis codes and another encounters that refer to diagnoses. The description table includes text describing the nature of the diagnosis. The encounter table includes no text and just IDs that can be used to connect to the description table. We can make one to one join between these two tables by joining “Diagnosis ID” in encounter table with “ID of Code” in the Description table. Joining these two tables will allow us to see a description for each diagnosis, except diagnosis 6, which is not in our description table. The last row for encounter will be dropped because there is no “Diagnosis ID” 6 in the description table. In one to one joins, care should be taken that important information are not lost because of mismatch in IDs.

**Table 4: Encounter and Description Tables**

|  |  |  |
| --- | --- | --- |
| **Description of Diagnosis Codes** |  | **Encounters** |
| **ID of Code** | **Diagnosis Code** | **Description** | **Hospital ID** | **Patient ID** | **Provider ID** | **Diagnosis ID** | **Hospital ID** |
| 1 | 410.05 | Acute myocardial infarction of anterolateral wall | 1 | 1001 | 12 | 1 | 1 |
| 2 | 250.00 | Diabetes mellitus without mention of complication | 1 | 123 | 240 | 5 | 2 |
| 3 | 250.01 |  | 1 | 150 | 2555 | 6 | 1 |
| 4 | 410.05 | Acute MI of anterolateral wall | 2 |  |
| 5 | 250.00 | Diabetes mellitus w/out mention of complication | 2 |

1. One-to-Many Join: The other two joins allow the field in one table to be always included and the field from in the other table included only when it matches. When the two fields do not match, the record is still kept but there will be a null value in place of the missing match. Following with the previous example, in a one-to-many join, we can display all claims and their corresponding text for diagnosis. Then if the diagnosis text is missing in the Diagnosis Code table, a missing value is entered. Claims data are still there but the description of the diagnosis will be null. Thus, using information in Table 4, the last row in the encounter table will remain in the analysis but all fields in the description table will be null for this row of data.
2. No Join (Cross Join): The last type of join occurs when two tables are present and not joined together through any field. In this circumstance, every record in one table is coupled with every record in another table. For example if one table has two records A and B, and another table has records 1, 2, 3, and 4. Then the consequence of having the two tables in a query but without a join is 8 records of all possible combinations of the two tables: A1, A2, A3 A4, B1, B2, B3, and B4.

## Expressions and Functions

There are a number of functions available in SQL commands that allow one to calculate the value of the new variable. These functions include *arithmetic operations* such as *add* or *divide*; *text operations* such as *concatenate;* and *date operations* such as *days since,* and *logical operations* such as *maximum* and *if*. In the select command you can also use the AS statement to rename the new field. SQL usually does not allow for space in a name; if there is space in a name one must put the name inside a bracket. Here is an example of an expression for computing a new field called “Diagnosis” using the “IIF” expression:

 IIF (ICD9!Description like “\*diabetes\*”, “Diabetes”, “Other) AS Diagnosis

This expression tells us that if the field “Description” in the table “ICD9” contains the word “diabetes”, the system will assign to the field “Diagnosis” the value “Diabetes,” otherwise it will assign to the field “Diagnosis” the value “Other.” Here is another expression:

 Date()-Claims.Date AS DayTillNow

This expression tells us that we should set the field “DaysTillNow” to be equal to the number of days of difference between today and the field called “Date” in the “Claims” table.  The expression “Date()” is a built-in function of Microsoft Access that provides today's date. The exact format and meaning of various SQL functions are available by doing a key word search on the Internet.

The “WHERE” command allows one to filter the data and select only a specific subset of the data. A “WHERE” command uses one, or more, criteria. The records or rows in a table are reduced to the rows that meet the criteria. For example, we might have a table of claims and want to restrict it to patients who had a claim of influenza. Here are some more examples of criteria: The criterion “>120” used after WHERE means that the field must be greater than 120. The criterion “> 6/12/05” means the field entry must be passed June 12th 2005. The criterion Not “\*ism” means that the field cannot have any text ending with the word suffix “ism.” The criteria “<Date()” means before today’s date. The criterion “Is Null” means that there are no data in the field. The criterion “Not Diabetes” means not matching exactly to the word diabetes, diabetic will be allowed as it does not match diabetes. The criterion Like "dia\*" matches any text that starts with “dia,” such as diabetes, dialog, diagram and so on. The criterion Between A and D will match any text starting with A, B and C. The “LIKE” criterion selects rows of data that contain a specific text. For example, LIKE “\*diabete\*” tells the software to select rows that contain the referenced text anywhere in the field. It will select “patient had diabetes” as well as “Diabetic patient”. The stars in the reference text indicate where wildcard letters can be.

Finally the GROUP BY command tells the software to summarize the values in a column. The command “GROUP BY Claims.PatientID” restricts the entries in the PatientID field to one value per ID. So, if there are multiple values of PatientID, e.g. if the same patient had multiple visits, this will restrict the analysis to one entry per patient. You would need to select the most recent visit, the average of the visits or some other way to summarize the multiple visits of the same patient. The various commands that can be used to summarize a field include:

1. Group by, where all records having the same value will be grouped into one.
2. Average, where in all records in the group are averaged
3. Standard deviation, where the standard deviation of all records in the same group are calculated.
4. Count, where all duplicate values in the group are counted.
5. Max and min functions, where the maximum or minimum value for the field in the group are selected. Maximum of a date will select the most recent value.
6. Last or first, where the last or first value in the field in the group are selected.

If you summarize one field in your query, all fields must be summarized. If you need to select a subset of the query, you can use the command "WHERE" in a “GROUP BY” query. In a "WHERE" command, only grouped records matching the criterion under the where field are included in the query. The WHERE command is executed before summarizing the data. If you wish to apply a criterion after summarizing the data, you can use the command “HAVING”.

For ease of use, Graphical Interfaces are available for specifying SQL code, for example Microsoft Access has a helpful graphical interface for doing so.