Querying a medical database

Databases and more

The purpose of the lecture

A medical data analyst- scientist should use all the information in hands to acquire valuable data for its own research or project which could literally affect the lives of people. He should be prepared to extract the information that is given, and make a valuable contribution, provide a solution finding new paths that could make a difference.

In order to achieve all the above a basic knowledge of how to handle one of the commonest databases is necessary. This lecture is going to introduce you to basic querying of SQL databases, through an example, a demo, of one of the most useful medical databases so far, the one called MIMIC.

What is actually MIMIC?

MIMIC stands for 'Medical Information Mart for Intensive Care' and as it is described by the people that contributed to it:

"… Is a large, freely-available database comprising deidentified health-related data associated with over 40,000 patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012". [1]

We are going to deal with the third version of this database MIMIC III. What we are actually going to query, is a demo version of the package, with fictitious numbers, so do not by alarmed if you come across with extremely big numbers. Our purpose is to understand and learn how to query, extract the info that we are seeking for.

It is a relational database consisting of 26 tables:

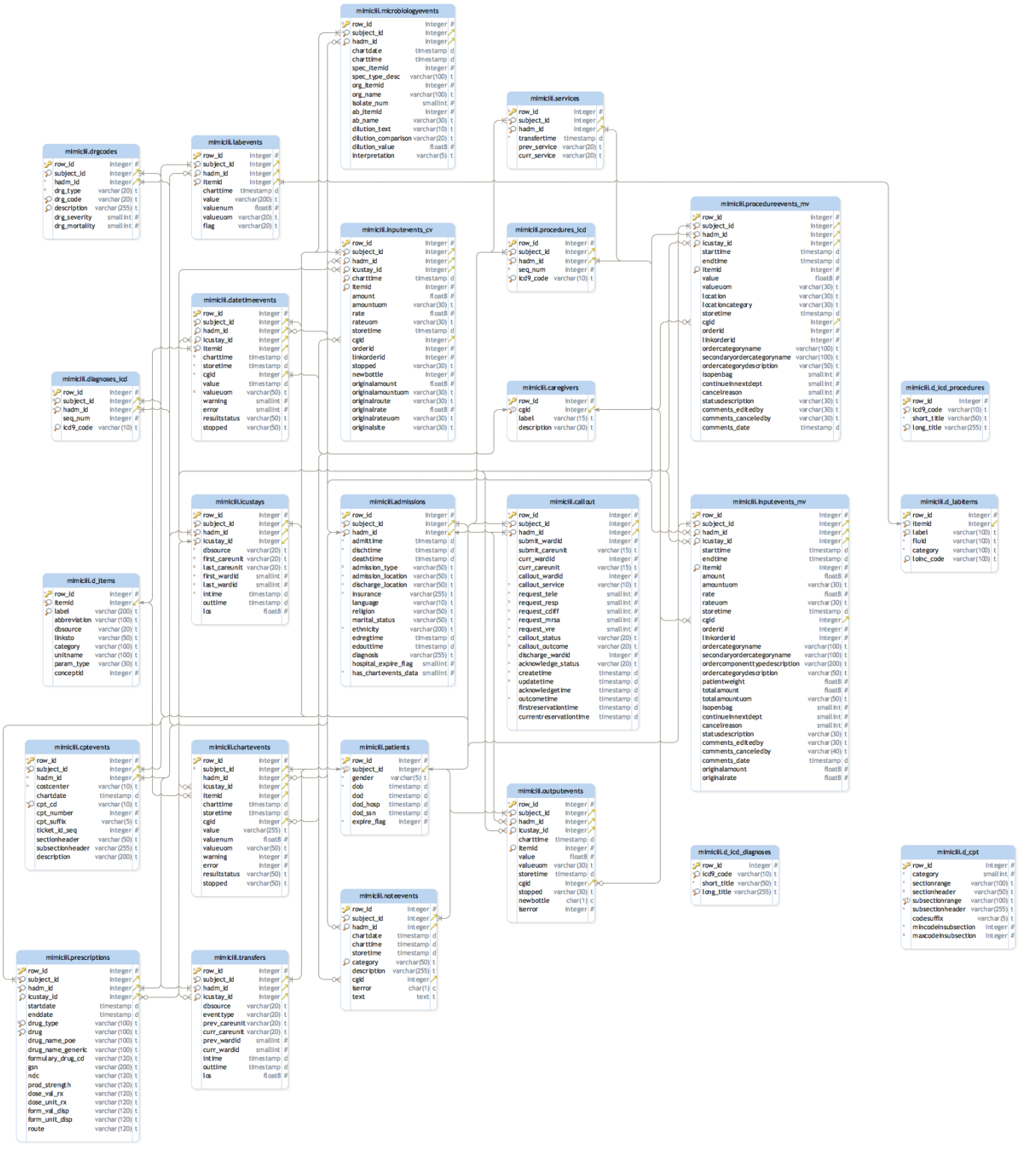


Figure 1: The structure of MIMIC III. All tables and relations of them.

Fonts are too small for you to read? Do not worry. I will include the image file for you and you can magnify it and see everything that you need to see.

You can find very thorough information of the tables and fields (the columns of the tables) consisting the database here:

https://mimic.mit.edu/docs/iii/tables/

The demo version that we are using follows the same pattern. We are going to deal with some of the tables, not all of them, but it is to your own initiative to find answers to your own questions by applying the knowledge that you are going to acquire from this lecture.

Installing the Jupyter Notebook

For the sake of the lecture a program called miniconda should be installed from here:

https://docs.conda.io/en/latest/miniconda.html

You can choose the installer according to your operating system, windows, linux, or Mac from the list and run it.

The second step is to install the Jypyter Notebook our guide for this lecture. It is a software in terms of Notebook that we can run commands of sql and also have comments for everything in just one Notebook. That will make our effort easier and the course more understandable.

To get Jupyter, we will use the conda package manager that was installed as part of Miniconda:

1. Open up command line interface and navigate to your directory. (In windows you can directly write in search 'Anaconda prompt' and open the command line)
2. Run: **conda install jupyter** in the command line to install Jupyter

   (You will need to press y to confirm the install).

After finishing with the installation we are ready to use it.

Just write down **jupyter notebook** in command line and voila!!! The notebook will appear in your browser. A quick start for jupyter can be found in this small video here: https://www.youtube.com/watch?v=A5YyoCKxEOU

Few things about SQL, SQLite and the MIMIC III demo

SQL

SQL stands for Structured Query Language. It is a standard programming language specifically designed for storing, retrieving, managing or manipulating the data inside a relational database management system (RDBMS).

It is the most widely-implemented database language and it is adopted by a lot of relational database systems you may have already heard:

1. Oracle
2. MySQL
3. Microsoft SQL Server
4. PostgreSQL
5. SQLite
6. Teradata
7. Microsoft Azure SQL Database

and more. Some features of the SQL standard are implemented differently in different database systems. In general though most of the commands that we are going to use in this lecture are the same for all of them with very small variations.

They are a lot of things that one can do using SQL programming language.

One can:

- create a database initially,
- create tables that we have already mentioned before and populate them with names, numbers, dates etc.
- update, modify them
- connect tables together to extract information from common fields
- extract useful info from database, by querying, calculating by using different functions applying in the fields of the database etc.

In our case, we are going to focus to the last two of them as we are already having a database in hand, and we are going to extract useful information by querying applying filters, aggregating fields and much more.

Why SQLite?

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

Due to these properties it is easy to install and maintain and that's way is quite popular. It is the most widely deployed SQL database engine in the world.

- SQLite does not require a separate server process or system to operate (serverless).
- SQLite comes with zero-configuration, which means no setup or administration needed.
- A complete SQLite database is stored in a single cross-platform disk file.
- SQLite is very small and light weight
- SQLite is self-contained, which means no external dependencies.

MIMIC III Demo and how we are going to use it through Jupyter and alternatives

A **mimic3.db** file is provided for you to download. It is actually the database demo provided by:

https://physionet.org/content/mimiciii-demo/1.4/

It contains 26 tables of data that we are going to explore through Jupyter notebook.

More information on how to install and apply it through Jupyter is given in Jupyter files that are available for the course.

As an alternative one can download and apply all queries from lecture using **sqlitebrowser.**

You can easily download and use it from the following site:

https://sqlitebrowser.org/

Installation and first commands

Jupiter Notebook

Few words before we continue:

Jupyter can integrate instructional text with area that you can run code. In our case SQL. We are taking advantage of it to make lecture more understandable. It runs through Python so minor extras in executing the commands are needed.

Basic shortcuts

Some basics for using the notebook are included to the table below.

Pressing button	Result
a	Add a cell above
b	Add a cell below
c	Copy a cell
v	Paste a cell
x	Cut a shell
dd	Delete a shell
Ctrl + z	Undo
Ctrl + Shift + z	Redo
Ctrl + Enter	Execute a command in a cell
Y	Change mode to code
m	Change mode to markdown

Installing SQL to Jupiter and loading the compiler

First thing first. What is initially needed is to install is the sql compiler. We can do it with the following command:

```
!pip install ipython-sql
```

Loading the compiler is the second thing we are trying and first from now on everytime we want to start working with a database. Load the SQL library.

```
%load_ext sql
```

Using Jupiter to run SQL

As Jupiter runs through Python, to execute a statement will require applying minimal amount of python code in our case:

%sql before each command
/ to continue a line of code to a new line.

But you will do this thing all the time in the future so do not worry it will come handy quite fast.

Getting to know our medical database

We finally reached the point to implement our target. SQL will be our guide and Jupiter our vessel to accomplish our target. As we need a lot of help at this point an alternative would be sqlitebrowser. Try to download and install it, if not yet done. It would definitely help you as you can directly see the whole tables in front of you

Connecting with the MIMIC III demo database

Let's try to download the database (download mimic3.db from elearning) and run it through our notebook.

```
%sql sqlite:////Your_Path_To_MIMIC3_demo_file/mimic3.db
```

Show the tables of the database

```
%sql SELECT name FROM sqlite_master WHERE type='table';
```

All info about tables and fields can be found here:

https://mimic.mit.edu/docs/iii/tables/

Show the columns and other information

This pragma returns one row for each column in the named table.

- Columns in the result set include the column name, data type, whether or not the column can be NULL, and the default value for the column.
- The "pk" column in the result set is zero for columns that are not part of the primary key, and is the index of the column in the primary key for columns that are part of the primary key.

To show information of a single table you can use the command:

```
%sql PRAGMA table_info(admissions)
```

The SELECT command and friends ...

The SQL SELECT statement is used to retrieve records from one or more tables in your SQL database. The records retrieved are known as a result set.

The skeleton of the command is:

```
SELECT columnname FROM tablename;
```

SQL syntax is case insensitive. (Which means that any way you write your command, or your table, capitals or small letters sql will understand and run your code). It is recommended though, in order not to get confused and to easily read and debug your code to use SQL keywords in capital letters and table or column names in lower case or as they look (defined) initially in your database.

```
%sql SELECT ethnicity FROM admissions;
```

In case we need to add extra columns to our select command, we separate each name of the column with a comma ',' except of the last name column.

E.g.

```
SELECT column1,column2,column3,...,columnn FROM table;
```

```
%sql SELECT ethnicity,admittime,diagnosis FROM admissions;
```

Exercise

Try to retrieve the date of birth **dob** and date of death **dod** from **patients** table.

```
%sql
```

Including all the columns

If we would like to include all the columns of the table we just use * instead of any name of a field (column)

```
SELECT * FROM table;
```

```
%sql SELECT * FROM icustays;
```

The DISTINCT command for removing duplicates

In case you need to retrieve unique rows and remove the duplicates you need to use the DISTINCT keyword just after the SELECT statement:

```
SELECT DISTINCT column1,... FROM table;
```

```
%sql SELECT DISTINCT ethnicity FROM admissions;
```

Exercise

Retrieve all the distinct cases from religion, marital status fields in admissions ta ble.

```
%sql
```

Sorting results using ORDER BY

ORDER BY statement allows to sort alphabetically the output of the query. The default is to order in ascending order **ASC**. You can always use **DESC** statement to reverse the order:

```
%sql SELECT DISTINCT subject_id,ethnicity FROM admissions ORDER BY ethn
icity;
```

And in descending order:

```
%sql SELECT DISTINCT subject_id,ethnicity FROM admissions ORDER BY ethn
icity DESC;
```

OR To limit your results, in case the query produces a lot of terms you can also use **LIMIT** and the number of the first terms to report e.g.

```
%sql SELECT DISTINCT subject_id,ethnicity FROM admissions ORDER BY ethn
icity LIMIT 10;
```

Sorting by multiple fields

Sorting can be extended to multiple fields using a specified order. For example you can see all the possible combinations of ethnicity types and their marital status using ORDER BY and including both fields:

```
%sql SELECT DISTINCT ethnicity,marital_status FROM admissions ORDER BY
ethnicity,marital_status;
```

The LIMIT command

To limit your results, in case the query produces a lot of terms you can also use **LIMIT** and the number of the first terms to report

```
%sql SELECT DISTINCT subject_id, ethnicity FROM admissions ORDER
BY ethnicity LIMIT 10;
```

Exercise

Find the types of caregivers in alphabedical order and print the first 10 of them, from caregivers table

```
%sql SELECT DISTINCT description FROM caregivers ORDER BY description L
IMIT 10;
```

ALIASES

You can rename the name of the column in your query outputs using the **AS** command. That makes your code more understandable and readable. To implement it just include it after the column you would like to change and right after the name of the column. AN **alias** is assigned (temporary name). You can also apply it to tables. The structure is as follows:

```
SELECT column1 AS newcolumn1,... FROM table1;
```

```
%sql SELECT DISTINCT subject_id AS PatientNo, dob AS DateOfBirth FROM p
atients;
```

Filtering results. The WHERE Statement

The WHERE statement is used to filter the results so as to extract specific rows that fulfill a specific target.

```
SELECT column1, column2, ... FROM table1 WHERE filter1 ;
```

It is followed by operators:

Operator	Description
AND, OR, NOT	Logical operators
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. In some versions of SQL may be written as !=

BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Let's start with an simple example to make things more clear. Filter the women in the **patients** table.

```
%sql SELECT * FROM patients WHERE gender = 'F' ;
```

As there are only two categories, alternative queries could be:

```
%sql SELECT * FROM patients WHERE gender <> 'M' ;
```

Or:

```
%sql SELECT * FROM patients WHERE gender IS NOT 'M' ;
```

Exercise

Select the patients with row_id between 10000 and 40000

%sql SELECT * FROM admissions Where row_id BETWEEN 12258 AND 40000;

LIKE statement

You can use LIKE to find a pattern, word or part of words using % and _. These are called wildcards. e.g.

Wildcard	Explanation
%	Allows you to match any string of any length (including zero length)
_	Allows you to match on a single character

To better understand and use it appropriately take a look of the following table:

LIKE Operator	Description
WHERE drug LIKE 'a%'	Finds any values that start with "a"
WHERE drug LIKE '%a'	Finds any values that end with "a"
WHERE drug LIKE '%or%'	Finds any values that have "or" in any position
WHERE drug LIKE '_r%'	Finds any values that have "r" in the second position
WHERE drug LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length

WHERE drug LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE drug LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Exercise

Find all information given from drug column which start which have the word Magnesium in the **prescriptions** table and are not Magnesium Sulfate.

```
%sql SELECT * FROM prescriptions WHERE drug LIKE '%Magnesium%' AND drug
<> 'Magnesium Sulfate';
```

IN statement

IN can be used instead of multiple **OR** statements. E.g.

```
%sql SELECT * FROM prescriptions WHERE drug IN ('Magnesium Oxide','Magn
esium Citrate','Magnesium Sulfate');
```

```
%sql SELECT * FROM prescriptions WHERE drug IS 'Magnesium Oxide' OR 'Ma
gnesium Citrate' OR 'Magnesium Sulfate';
```

Exercise

Try doing it using OR statement

```
%sql
```

Summarizing the data

Aggregate functions for summarizing your result

We can use functions in SQL to run calculations that give useful information and summarize your data. The most common ones are given in the table below:

Function	What it returns
COUNT()	Counts how many rows are in a particular column
SUM()	Adds together all the values in a particular column
MIN()	Returns the lowest value in a particular column
MAX()	Returns the highest value in a particular column
AVG()	Calculates the average of a group of selected values

The best way to learn is by an example. Let's figure out how many patients we have in patients database:

```
%sql SELECT COUNT(subject_id) AS NUMBER_OF_PATIENTS FROM patients;
# You can also use * inside the COUNT function instead of subject_id, C
OUNT(*)
```

Have in mind that:

When a column is included in a COUNT function, null values are ignored in the count. That is not the case when * is used.

Exercise

Find the minimum the maximum so as the average time of hospital stay in days. (Hint: Make use of the los column of icustays)

```
%sql SELECT MAX(los) AS MAX_STAY, MIN(los) AS MIN_STAY, AVG(los) AS Ave
rage_Stay FROM icustays;
```

Arithmetical Operations

We can perform arithmetical operations

Operator	Meaning	Operates on
+	Addition	Numeric value
-	Substraction	Numeric value
*****	Multiplication	Numeric value
/	Devision	Numeric value
%	Division Remainder	Numeric value

- An example: 13 % 5 = 3 because the remainder of 13 divided by 5 is 3

```
%sql SELECT (row_id - subject_id) AS test_difference FROM admissions LIMIT 10;
```

Calculating time, and time differences

As it is easily seen in the database, we have TIMESTAMP datatypes, in our database.

In order to calculate time, or time differences we can use some of the functions of SQLITE in:

https://www.sqlite.org/lang_datefunc.html

In our case, the best choice is JULIANDAY() function.

It gives the number of days since Nov 24, 4714 BC 12:00pm Greenwich time in the Gregorian calendar:

https://www.techonthenet.com/sqlite/functions/julianday.php

Just a reminder. JULIANDAY() can be used only by SQLITE. If we had a different implementation e.g. MySQL we have different case.

The following example calculates the lifespan of the patients in years rounding to 2 digits.

SQLite function ROUND() takes two arguments:

- X, the number which will be rounded
- Y, A number indicating up to how many decimal places N will be rounded.

```
%sql SELECT ROUND((JULIANDAY(dod)-JULIANDAY(dob))/365.24,2) AS AGE_OF_D
EATH FROM patients;
```

Exercise

Find the days, by subject_id of hospital stay of the patients that did not pass away in hospital, by descending time.

```
%sql SELECT subject_id,ROUND((JULIANDAY(dischtime)-  \ JULIANDAY(admitt
ime)),2) AS HIME FROM admissions WHERE \ hospital_expire_flag =0 ORDER
BY HIME DESC;
```

The GROUP BY Statement

The AGGREGATE functions become more usful when used in conjuction with the GROUP BY Statement. It is used to summarize multiple subsets of the data in the same query.

The GROUP BY clause comes after the FROM clause of the SELECT statement. In case a statement contains a WHERE clause, the GROUP BY clause must come after the WHERE clause.

```
SELECT columnname, AGR_FUNCTION1, AGR_FUNCTION2 FROM tablename WHERE
conditions_provided GROUP BY columnname;
```

Let's count the number of male and female in **patients** table.

```
%sql SELECT gender, COUNT(gender) AS TOTAL_NUMBER FROM patients GROUP B
Y gender;
```

Exercise

- Calculate the number of doses per drug from prescriptions table
- Order by maximum number of doses
- Limit to the first 10

```
%sql SELECT drug, COUNT(drug) AS number_of_doses FROM prescriptions
\GROUP BY drug ORDER BY number_of_doses DESC LIMIT 10
```

Grouping by multiple fields

Grouping can be extended to multiple fields using a specified order. For example you can identify the numbers of deaths in terms of type of entrance using GROUP BY and including both fields (**admission_type**,**hospital_expire_flag**):

```
%sql SELECT admission_type, hospital_expire_flag, COUNT(subject_id) as
number FROM admissions GROUP BY admission_type,hospital_expire_flag
```

The HAVING clause

HAVING clause is used to query subsets of aggregated groups (in a GROUP BY clause) just as WHERE is used to query subsets of rows. The expression that follows a HAVING clause has to be applicable to these groups as WHERE is applicable to each row of data in a column.

e.g. Let's filter from the exercise before the drugs that more than 100 of doses where provided to the patients.

```
%sql SELECT drug, COUNT(drug) AS number_of_doses FROM prescriptions GRO
UP BY drug HAVING number_of_doses>100 ORDER BY 2 DESC;
```

Exercise

SELECT the patients that have more than one **admissions** in the hospital by descending order.

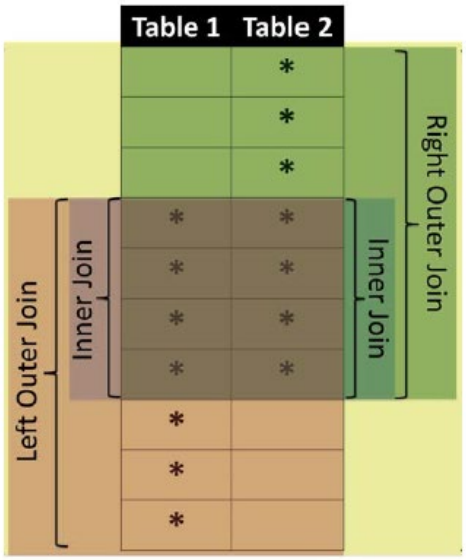
```
%sql SELECT subject_id, COUNT(subject_id) AS number_of_admissions FROM
admissions \
GROUP BY subject_id HAVING number_of_admissions > 1 ORDER BY number_of_
admissions DESC;
```

Joining and Subquerying

What is join? How are the tables joining?

So far we have used a single table to extract information, count, add, calculate averages and so on. In real life we usually need to query multiple tables which have relationships. But what type of relationship this could it be? Tables in relational databases are linked through primary keys and sometimes other fields that are common to multiple tables, as it is in our database.

An execution of a JOIN is used to combine rows from two or more tables, based on a related column between them.



INNER JOIN

INNER JOIN selects records that have matching values in both tables

The syntax is as follows:

```
SELECT table1.columnX, table2.columnY FROM table1 INNER JOIN table2 ON
table1.column_name=table2.column_name
```

In case columnX and columnY appear only to one table there is no need to specify the table name within the definition of the column.

Let's give an example to make things clearer. Take a look of the **icustays** table. There are (among others) the fields **subject_id** and **intime** where are representing the patient id and the time the person entered the hospital. Now take a look at **patients** table. It also include **subject_id** and also a field **dob** (date of birth). Let's try to inner join these two tables to find out the age of the common patients.

```
%sql SELECT ie.subject_id, \
    ROUND((julianday(ie.intime) - julianday(pat.dob))/365.242, 0) AS ag
e \
FROM icustays ie \
INNER JOIN patients pat \
ON ie.subject_id = pat.subject_id;
```

LEFT JOIN

In few words:

- A LEFT JOIN performs a join starting with the left table.
- Then, any matching records from the right table will be included.
- Rows without a match return columns with NULL values.

- We use a LEFT JOIN when we want every row from the first table, regardless of whether there is a matching row from the second table.
- This is similar to saying, **'Return all the data from the first table no matter what'**

Let's see an example to make things clearer:

I am going to use **patients** and **callout** tables.

Callout provides information when a patient was READY for discharge from the ICU, and when the patient was actually discharged from the ICU.

I am using callout as it includes a subset of the patients of **patients** table.

```
%sql SELECT patients.subject_id,curr_careunit \
FROM callout LEFT JOIN  patients \
ON patients.subject_id = callout.subject_id
```

RIGHT JOIN

It is exactly opposite of the LEFT JOIN. It is not supported from SQLite.

Subqueries

They are also called inner queries or nested ones. They are queries embedded within the context of another query. The output is included as part of the queries that surround it. They can be used in SELECT WHERE and FROM clauses. We are using it because:

- It can separate each logical part of a statement, so as to easily find any mistake in long and complicated queries.
- It is sometimes faster that JOINS.
- It is the most logical way to retrieve information.

Subqueries Rules

- ORDER BY phrases can not be used in subqueries.
- Subqueries in SELECT and WHERE clauses that return more that one row must be used in combination with operators that are designed explicitely to handle multiple values as IN operator. Otherwise subqueries in SELECT or WHERE can output no more than one row.

Learn by example

Let's try to find the overall average of drugs used per person using the **prescriptions** table.

- First let's try to find the number of drugs each person took.

```
%sql SELECT subject_id, COUNT(drug) AS DrugCounter \
FROM prescriptions GROUP BY subject_id
```

- To find the average let's enclose it to a outer query asking the AVG from DrugCounter:

```
%sql SELECT AVG(DrugCounter) AS AVGCOUNT FROM \
( \
SELECT subject_id, COUNT(drug) AS DrugCounter \
FROM prescriptions GROUP BY subject_id \
);
```

Exercise

Find the subject_id of patients that diagnosed with Atherosclerosis **d_icd_diagnoses** and **diagnoses_icd**

```
%sql SELECT DISTINCT subject_id FROM diagnoses_icd WHERE icd9_code IN \
(SELECT icd9_code AS DIABETES_NUM FROM d_icd_diagnoses WHERE long_title
LIKE '%Atherosclerosis%')
```

Revisional Exercises

Patients

1. Question

Retrieve the patients' religions distribution. Order from highest to lowest number of patients in a group.

```
%sql SELECT religion,COUNT(DISTINCT subject_id) AS NUM_PEOPLE FROM admi
ssions GROUP BY religion ORDER BY NUM_PEOPLE DESC;
```

2. Question

Retrieve all admission and discharge dates so as the diagnosis of one patient: subject_id = 44083 from **admissions** table

```
%sql SELECT subject_id,admittime,dischtime,diagnosis FROM admissions WH
ERE subject_id = 44083;
```

3. Question

Calculate the minimum, maximum, average of the patients' age. Round the results to one decimal.

```
%sql SELECT ROUND(AVG(AGE),1),ROUND(MAX(AGE),1),ROUND(MIN(AGE),1) FROM
\
(SELECT subject_id,(JULIANDAY(dod)-JULIANDAY(dob))/365.24 AS AGE FROM p
atients);
```

Diagnosis

4. Question

Find the total number of all different diagnoses codes. (table **diagnoses_icd**)

```
%sql SELECT COUNT(DISTINCT icd9_code) AS NUM_OF_CODES FROM diagnoses_ic
d;
```

5. Question

Find the codes of the 10 most often diagnoses according to the number of admissions

```
%sql SELECT icd9_code,COUNT(row_id) AS NUM_OF_ADMISSIONS FROM diagnoses
_icd \
GROUP BY icd9_code ORDER BY NUM_OF_ADMISSIONS DESC LIMIT 10;
```

6. Question

Display the short title instead of the icd9_code in the previous query.

```
%sql SELECT diag.icd9_code,COUNT(diag.row_id) AS NUM_OF_ADMISSIONS, cod
e_diag.short_title \
FROM diagnoses_icd as diag LEFT JOIN d_icd_diagnoses as code_diag ON \
diag.icd9_code = code_diag.icd9_code \
GROUP BY diag.icd9_code ORDER BY NUM_OF_ADMISSIONS DESC LIMIT 10;
```

7. Question

Find all patients and their date of birth who have had any diagnosis that includes 'diabetes' anywhere in the diagnosis long title in Tables:

diagnoses_icd, d_icd_diagnoses and patients

Steps:

- Find first the from dicd_diagnoses all the codes that have to do with diabetes
- Then do a connection of subject_id and these codes with table diagnoses_icd
- Finally use the subject_id's from them to collect information for the relative patients.

```
%sql SELECT subject_id,dob FROM patients WHERE subject_id IN \
(SELECT DISTINCT subject_id FROM diagnoses_icd WHERE icd9_code IN \
(SELECT icd9_code AS DIABETES_NUM FROM d_icd_diagnoses WHERE long_title
LIKE '%Diabetes%'))
```

Mortality

8. Question

Find the diagnosis of the people who have died in hospital from **admissions** table and group them in descending order.

```
%sql SELECT diagnosis, COUNT(diagnosis) AS DIAG_COUNT FROM admissions \
WHERE hospital_expire_flag = 1 GROUP BY diagnosis ORDER BY DIAG_COUNT D
ESC
```

9. Question

Find the gender of the people that died. (Use join with **patients** table)

```
%sql SELECT ad.subject_id,gender FROM admissions AS ad LEFT JOIN patien
ts AS pat ON ad.subject_id = pat.subject_id \
WHERE hospital_expire_flag = 1
```

10. Question

Write a query to find the total number of men and women who have died.

```
%sql SELECT gender,COUNT(gender) as G_count FROM admissions AS ad LEFT
JOIN patients as pat ON ad.subject_id = pat.subject_id \
WHERE hospital_expire_flag = 1 GROUP BY gender
```

11. Question

Write a query to find gender and the age of men and women who have died.

```
%sql SELECT ad.subject_id,gender,ROUND((JULIANDAY(dod)-JULIANDAY(dob))/
365.24,0) AS AGE FROM admissions as ad LEFT JOIN patients as pat \
ON ad.subject_id = pat.subject_id \
WHERE hospital_expire_flag = 1
```

12. Question

Find the subject_id's and diagnosis of people that died in the hospital from 1-1-2130 to 31-12-2149

Hint: use strftime() function

```
%sql SELECT subject_id,diagnosis,strftime('%Y',deathtime) as YEAR_OF_DE
ATH FROM admissions \
WHERE hospital_expire_flag = 1 AND deathtime BETWEEN '2130-01-01' AND '
2149-12-31'
```

Lab

13. Question

Find all the microbiological results for the person with subject_id =10126 which belong to the category of Chemistry. Use tables **labevents** and **d_labitems**

```
%sql SELECT * FROM labevents INNER JOIN d_labitems \
ON labevents.itemid = d_labitems.itemid \
WHERE subject_id = 10126 AND Category = 'Chemistry'\
ORDER BY charttime DESC;
```

14. Question

Find the people with abnormal results in 'LDL' tests.

```
%sql SELECT DISTINCT subject_id FROM labevents INNER JOIN d_labitems \
ON labevents.itemid = d_labitems.itemid \
WHERE label LIKE '%LDL%' AND flag = 'abnormal';
```

15. Question

Which of them is connected with Atherosclerosis of any type? Hint: use your prior knowledge of finding a long_title.

```
%sql SELECT DISTINCT subject_id FROM labevents INNER JOIN d_labitems \
ON labevents.itemid = d_labitems.itemid \
WHERE label LIKE '%LDL%' AND flag = 'abnormal' AND subject_id IN \
(SELECT DISTINCT subject_id FROM diagnoses_icd WHERE icd9_code IN \
```

```
(SELECT icd9_code FROM d_icd_diagnoses WHERE long_title LIKE '%Atherosc
lerosis%'))
```

Procedures

We are going to use the following tables: d_icd_procedures (codes-name of procedure) and procedures_icd (the codes only for procedures-patients) procedureevents_mv

16. Question

Find all the patients that underwent Arterial catheterization

```
%sql SELECT DISTINCT subject_id FROM procedures_icd WHERE icd9_code IN
\
(SELECT icd9_code AS catheterization_NUM FROM d_icd_procedures WHERE lo
ng_title LIKE '%Arterial catheterization%')
```

17. Question

Find the average time of their stay in hospital.

Contents