SQL tutorial

Aim:

Relational databases, select data from database using Structured Query Language (SQL)

Insight into how the arterial line data was extracted

Database system:

PostgreSQL

Tasks:

Find data of birth, find admission time

Notes:

list each variable in a separate column, list observations in rows

Relational databases:

A collection of tables which are linked together by shared keys. Organizing data across tables

* "**Database schema**": The model that defines the structure and relationships of the tables.
* "**Database query**": Data is extracted from relational databases using structured "queries".
* "**Primary key**": A primary key is a field that uniquely identifies each row in a table.
* "**Foreign key**": A foreign key is a field that refers to a primary key in another table.
* "**Normalisation**": The concept of structuring a database in a way that reduces data repetition and improves data integrity, usually by requiring one or more tables to be joined.
* "**Denormalisation**": The concept of structuring a database to improve readability, sometimes at the expense of data repetition and data integrity.
* "**Data type**": A term used to describe the behaviour of data and the possible values that it can hold (for example, integer, text, and date are all data types in PostgreSQL).

Commands:

##### Viewing columns and rows

SELECT \* FROM d\_items WHERE item\_id = 211; limit = 100

SELECT [columns] ## \* is a wildcard for all columns  
FROM [database\_name].[table\_name];

SET SEARCH\_PATH TO mimiciii;

SELECT subject\_id, gender, dob FROM patients

##### select subset of data

SELECT [columns]  
FROM [table\_name]  
WHERE [conditions];

SELECT \*   
FROM patients   
WHERE subject\_id = 109   
OR subject\_id = 117   
OR subject\_id = 127;

##More efficient

SELECT \*  
FROM patients  
WHERE subject\_id IN (109, 117, 127);

SELECT \*   
FROM patients   
WHERE subject\_id >= 109   
AND subject\_id <= 127;

##More efficient

WHERE subject\_id BETWEEN 109 AND 127; #between is inclusive

####Partial string matches

SELECT \*   
FROM icustays   
WHERE first\_careunit LIKE '%ICU%'; ## LIKE is to match text

## % regular expression all characters

WHERE first\_careunit LIKE ‘ICU%’;

LIKE ‘%ICU’;

### Two WHERE conditions

SELECT \* FROM icustays WHERE hadm\_id=170883 AND first\_careunit LIKE '%ICU%';

SELECT [columns]  
FROM [table\_name]  
WHERE [conditions]  
ORDER BY [columns] [ASC/DESC];

##### query multiple tables with SQL JOIN

Eg. INNER JOIN, LEFT JOIN, RIGHT JOIN

###INNER JOIN will only return rows where subject\_id  
###appears in both the patients table and the admissions table  
SELECT p.subject\_id, p.dob, a.hadm\_id, a.admittime  
FROM patients p  
INNER JOIN admissions a  
ON p.subject\_id = a.subject\_id  
ORDER BY subject\_id, hadm\_id;

##### Join three tables

SELECT \*

FROM admissions a

INNER JOIN icustays i

ON a.subject\_id = i.subject\_id

INNER JOIN patients p

ON a.subject\_id = p.subject\_id

SELECT p.subject\_id, p.dob  
FROM patients p  
LEFT JOIN admissions a  
ON p.subject\_id = a.subject\_id  
AND a.admission\_type = 'ELECTIVE'; #longer list  
  
SELECT p.subject\_id, p.dob  
FROM patients p  
LEFT JOIN admissions a  
ON p.subject\_id = a.subject\_id  
WHERE a.admission\_type = 'ELECTIVE'; #shorter list #show overlap only

##### Performing operations on columns

SELECT icustay\_id, round(los)  
FROM icustays;

SELECT icustay\_id, round(los) AS los\_integer\_days  
FROM icustays;

### reference for more operations

<https://www.postgresql.org/docs/9.5/static/functions-math.html>

SELECT subject\_id, admittime, deathtime, deathtime - admittime AS length\_of\_stay

FROM admissions

WHERE deathtime IS NOT NULL;

## Here we have introduced another concept: the IS NOT NULL clause which checks that the value is not null (a "null" is an empty value, and represents missing data).

###### Temporary tables to help manage queries

WITH patient\_dates AS (  
SELECT subject\_id, admittime, deathtime  
, deathtime - admittime AS length\_of\_stay  
FROM admissions  
WHERE deathtime IS NOT NULL  
)  
SELECT \*  
FROM patient\_dates;

## we begin by dropping any existing views with the same name  
DROP MATERIALIZED VIEW IF EXISTS patient\_dates\_view;  
CREATE MATERIALIZED VIEW patient\_dates\_view AS  
SELECT subject\_id, admittime, deathtime  
, deathtime - admittime AS length\_of\_stay  
FROM admissions  
WHERE deathtime IS NOT NULL;

### example of complicated SQL

<https://github.com/MIT-LCP/mimic-code/blob/master/sepsis/angus.sql>

|  |  |
| --- | --- |
|  | organ\_diag\_group as ( |
|  | SELECT subject\_id, hadm\_id, |
|  | CASE |
|  | -- Acute Organ Dysfunction Diagnosis Codes |
|  | WHEN substring(icd9\_code,1,3) IN ('458','293','570','584') THEN 1 |
|  | WHEN substring(icd9\_code,1,4) IN ('7855','3483','3481', |
|  | '2874','2875','2869','2866','5734') THEN 1 |
|  | ELSE 0 END AS organ\_dysfunction, |
|  | -- Explicit diagnosis of severe sepsis or septic shock |
|  | CASE |
|  | WHEN substring(icd9\_code,1,5) IN ('99592','78552') THEN 1 |
|  | ELSE 0 END AS explicit\_sepsis |
|  | FROM MIMICIII.DIAGNOSES\_ICD), |

##### CASE statement for if/else logic

##eg. Group length of ICU stay(los) into short, medium and long

-- Use if/else logic to categorise length of stay  
-- into 'short', 'medium', and 'long'  
SELECT subject\_id, hadm\_id, icustay\_id, los,  
    CASE WHEN los < 2 THEN 'short'  
         WHEN los >=2 AND los < 7 THEN 'medium'  
         WHEN los >=7 THEN 'long'  
         ELSE NULL END AS los\_group  
FROM icustays;

### Aggregate functions

## an aggregate value across multiple rows (eg. Number, average, maximum)

## COUNT(), MAX(), SUM() and AVG()

SELECT count(\*)  
FROM icustays;

SELECT MAX(los)  
FROM icustays;

### GROUP BY

-- find the maximum length of stay in the ICU  
-- for each patient  
SELECT subject\_id, MAX(los)  
FROM icustays  
GROUP BY subject\_id;

## HAVING

-- find the maximum length of stay in the ICU  
-- for each patient  
-- where the maximum length of stay is < 10 days  
SELECT subject\_id, MAX(los)  
FROM icustays  
GROUP BY subject\_id  
HAVING MAX(los) <= 10;

Write a query to select all of the heart rate measurements in the chartevents table. Use the GROUP BY keyword to find the maximum heart rate for each patient.

SELECT subject\_id, MAX(valuenum)

FROM chartevents

WHERE itemid IN (211, 220045)

GROUP BY subject\_id

HAVING MAX(valuenum) <= 140

##### Window functions

-- find the order of admissions to the ICU for a patient  
SELECT subject\_id, icustay\_id, intime,   
    RANK() OVER (PARTITION BY subject\_id ORDER BY intime)  
FROM icustays;

WITH icustayorder AS (  
    SELECT subject\_id, icustay\_id, intime,   
        RANK() OVER (PARTITION BY subject\_id ORDER BY intime)  
    FROM icustays  
)  
SELECT \*  
FROM icustayorder

WHERE rank = 1;

WITH icustayorder AS (SELECT subject\_id, icustay\_id, intime, outtime, outtime-intime ,RANK () OVER (PARTITION BY subject\_id ORDER BY intime) FROM icustays) SELECT \* FROM icustayorder WHERE rank = 1;

##### Multiple temporary views

## more than one inline view

WITH serv as (  
SELECT subject\_id, hadm\_id, transfertime, prev\_service, curr\_service  
FROM services  
)  
, icu as  
(  
SELECT subject\_id, hadm\_id, icustay\_id, intime, outtime  
FROM icustays  
)  
SELECT icu.subject\_id, icu.hadm\_id, icu.icustay\_id, icu.intime, icu.outtime  
, serv.transfertime, serv.prev\_service, serv.curr\_service  
FROM icu  
INNER JOIN serv  
ON icu.hadm\_id = serv.hadm\_id

## Count how many rows ended up with the above query

###Exercise 5 script play:

-- only take the first service at “services” table

WITH serv AS (  
    SELECT subject\_id, hadm\_id, transfertime, prev\_service, curr\_service,   
        RANK() OVER (PARTITION BY subject\_id ORDER BY transfertime)  
    FROM services  
),

serv\_rank1 AS (  
 SELECT \*  
 FROM serv

WHERE rank = 1

), icu as

(  
SELECT subject\_id, hadm\_id, icustay\_id, intime, outtime  
FROM icustays  
)  
SELECT icu.subject\_id, icu.hadm\_id, icu.icustay\_id, icu.intime, icu.outtime  
, serv\_rank1.transfertime, serv\_rank1.prev\_service, serv\_rank1.curr\_service  
FROM icu  
INNER JOIN serv\_rank1  
ON icu.hadm\_id = serv\_rank1.hadm\_id

INNER JOIN vs OUTER JOIN

Inner join:

An inner join using either of the equivalent queries gives the intersection of the two tables, i.e. the two rows they have in common.

(gives the intersection of the two tables)

select \* from a INNER JOIN b on a.a = b.b;

select a.\*,b.\* from a,b where a.a = b.b;

Left outer join:

A left outer join will give all rows in A, plus any common rows in B.

select \* from a LEFT OUTER JOIN b on a.a = b.b;

select a.\*,b.\* from a,b where a.a = b.b(+);

a | b

--+-----

1 | null

2 | null

3 | 3

4 | 4

Right outer join:

A right outer join will give all rows in B, plus any common rows in A.

select \* from a RIGHT OUTER JOIN b on a.a = b.b;

select a.\*,b.\* from a,b where a.a(+) = b.b;

a | b

-----+----

3 | 3

4 | 4

null | 5

null | 6

Full outer join

A full outer join will give you the union of A and B, i.e. all the rows in A and all the rows in B. If something in A doesn't have a corresponding datum in B, then the B portion is null, and vice versa.

select \* from a FULL OUTER JOIN b on a.a = b.b;

a | b

-----+-----

1 | null

2 | null

3 | 3

4 | 4

null | 6

null | 5

