Getting Started with SQL

TAMÁS BUDAVÁRI – AUGUST 3, 2016

Databases are easy: they consist of tables, which in turn have columns – just like the files with which you are probably used to working. The difference is that you don't have to think about how to read and write the files but can focus on the questions you try to answer. The following code snipets and exercises will teach you the basics of expressing your questions in the Structure Query Language, or SQL for short. SQL is a standard language and most of these commands will work on any relational database but there are minor differences in dialects.

The database you will be querying stores a collection of measurements of some (X,Y) quantities. There is a relation between them but the Y measurements are noisy. Our database is synthetic but carries several aspects of real measurements: separate instruments, multiple students and runs of measurements in different observational domains. The relevant tables are Data, Runs, Instruments and Users. The names should be suggestive of their contents. We will use an online service to access the data. To get started, visit the link at http://gwen1.pha.jhu.edu/sqlweb and try the queries below. The left panel shows the content of the database. If you click on Tables you'll see a list of them and you can dig deeper to find the column names.

The First Queries

To get all (X,Y) values from the table Data, you would type the following command into the top left panel and click Execute. But don't do it! First you should always think about what will happen.

```
-- never run queries like this
select X, Y
from Data
```

The table might have hundreds of millions of rows! Do you really want all that data dumped on you? Try the next set of commands to see their effects and understand how they work. Consider them as illustrations accompanying the lecture and ask questions if something is not clear!

```
-- have a quick peek
select top 5 X, Y
from Data
```

```
-- sorting to peek at the extremes
select top 3 X, Y
from Data
order by X
```

```
-- filtering with formulas and functions
select top 5 x
from Data
where 2*y between -sin(x) and x
order by RunID desc, y desc
```

You can comment your SQL code by using the following formats

```
-- single-line comments go after --

/* multi-line comments are
   like this one
*/
```

SAVE YOUR WORK!

The website you are using will not save your queries. If you would like to keep them for future reference, open a text editor to cut & paste the relevant lines into a file that you can regularly save.

Aggregation

Consider getting only the relevant information from the database and not everything. Run the following commands and see what the different constructs achieve:

```
-- counting
select COUNT(ID) as N, COUNT(RunID), COUNT(distinct RunID)
from Data

-- aggregates in general
select COUNT(id), SUM(x), AVG(y), STDEV(x-y)
from Data
where Y>0

-- grouping data
select RunID, MIN(x), MAX(x), MIN(y), MAX(Y)
from Data
group by RunID
order by AVG(X) desc
```

```sql
-- having: contraints on aggregates
select RunID, MIN(x), MAX(x), MIN(y), MAX(Y)
from Data
where X>0.2          -- filtering on the input
group by RunID
having MAX(Y) < 0    -- filtering on aggregate
order by AVG(X) desc

-- or
select RunID, MIN(x), MAX(x), MIN(y), MAX(Y)
from Data
where X>0.2
group by RunID
having COUNT(*) > 30
order by AVG(X) desc


-- number of measurements in each run
select RunID, COUNT(*)
from Data
group by RunID
order by 2 desc


-- rounding is easy
select top 100 X, ROUND(X*X,2) from Data


-- building a histogram
select ROUND(x,2) as X, COUNT(*) as N
from Data
group by ROUND(x,2)
order by 1


-- custom bin size using a variable and save the results
declare @bin float = 0.016
select ROUND(x/@bin,0)*@bin as X, COUNT(*) as Cts
into MyHistogram  -- name of new table
from Data
group by ROUND(x/@bin,0)*@bin
order by ROUND(x/@bin,0)*@bin
```

Asking Questions using SQL

You'll see that every question you have about the data will nicely translate to commands. For example, let's consider the following question: Who ran the first measurement? The following 3 select statements will get you an answer:

```
-- who ran the first measurement? execute these lines separately!
select top 1 RunID from Data order by ID -- 100
select UserID from Runs where RunID=100 -- 12
select * from Users where UserID=12
```

What did we really mean by "first" measurement? Any other definition to use?

Nested queries can combine multiple searches into one request. Run and analyze the following queries and their results:

```
-- nested queries
select UserID
from Runs
where RunID = (select top 1 RunID from Data order by ID)

-- doubly so
select * from Users
where UserID = (
  select UserID from Runs
  where RunID = (select top 1 RunID from Data order by ID)
)
-- whole set of runs using the 'in' keyword
select UserID
from Runs
where RunID in (select top 1 RunID
                from Data order by ID)
```

Combining tables is where relational database engines really shine. The terminology is "joining tables". Here are different implementations of similar questions. Make sure you understand these queries because they will be important:

```
-- inner join (old style)
select u.Name, r.RunID, r.Xmax - r.Xmin
from Runs r, Users u -- aliases are convenient
where r.UserID=u.UserID
-- risk of forgetting a constraint when many tables

-- inner join (preferred)
select u.Name, r.RunID, r.Xmax - r.Xmin
from Runs r
     join Users u on u.UserID=r.UserID
```

```
-- or explicitly
select u.Name, r.RunID, r.Xmax - r.Xmin
from Runs r
     inner join Users u on u.UserID=r.UserID

-- list of users with measurements
select distinct u.Name
from Runs r
     join Users u on u.UserID=r.UserID
```

A different kind of join is used less frequently to look at all combinations of rows in the specified tables. Compare the following queries to the previous ones and run them to see the differences in the results:

```
-- cross join (old style)
select u.Name, r.RunID, r.Xmax - r.Xmin
from Runs r, Users u
-- same as old-style inner join w/o contraint

-- cross join: explicitly
select u.Name, r.RunID, r.Xmax - r.Xmin
from Runs r cross join Users u

-- compare the size of the result set with these
select COUNT(*) from users
select COUNT(*) from runs
-- all combinations
```

The following new flavor of join can give you extra information when it's there but will return a special NULL value when there isn't any matching entry to indicate that. You can even use those special values to filter on and look for missing data. Analyze the following examples:

```
-- left outer join
select u.Name, r.RunID, r.Xmax - r.Xmin
from Users as u
     left join Runs as r
          on u.UserID=r.UserID

-- users without measurements
select u.Name, r.RunID, r.Xmax - r.Xmin
from Users as u
     left outer join Runs as r on u.UserID=r.UserID
where r.RunID is NULL

-- or
select u.Name, r.RunID, r.Xmax - r.Xmin
from Users as u
     left join Runs as r on u.UserID=r.UserID
where r.Xmax is null
```

SQL has set operators to combine results of separate queries. The example below finds users who don't appear to have any measurement runs associated with their identifier:

```
-- or with set operations
select UserID from Users
except
select UserID from Runs

-- join to get the names
select Name, UserID from Users
except
select u.Name, r.UserID
from Runs r join Users u on u.UserID=r.UserID

-- join with nested query
select u.*
from Users u join (
        select UserID from Users
        except
        select UserID from Runs
    ) as n -- alias required!
    on u.UserID=n.UserID

/* Keywords for set operations:

   union, union all, intersect,
   except/minus
*/
```

Exercises

The goal of this tutorial is to get you started with SQL, so you can translate your own questions to database requests. Based on what you learnt about the database and its contents, try to formulate the queries to address the following questions. Some of the open questions are intentionally vague. You will have to come up with better ones and write the queries that answer them…

1. List the lengths of the (X,Y) vectors in descending order for all measurements (in the Data table) with X and Y closer than 0.1
2. What is the histogram of Y if binned into intervals between integers?
3. What is the largest Y value measured by each student?
4. Who are the two busiest users?

- Who makes the best measurements?
- Which instrument is better?
- What is the relation between X and Y?

Hints: Separate instruments often perform differently. Also some students might be better than others at using these machines for obtaining the measurements.

Common Table Expressions or CTEs

One way to find “the 2 busiest users” is to count the entries in the Runs table (in the nested query below) and get the relevant names by joining with the Users table:

```
-- the busiest 2 users
select u.Name, b.N
from Users u
    join (
        select top 2 UserID, N=count(d.ID)
        from Runs r join Data d on d.RunID=r.RunID
        group by UserID
        order by 2 desc
    ) b
    on b.UserID=u.UserID
```

This query is hard to read as we have to first find the nested query and understand what it does before we can decipher the entire SQL command. Instead we can move the nested part to a CTE upfront:

```
-- common table expressions or CTEs
with BusyUsers(UserID, N) as
(
    select top 2 UserID, count(d.ID)
    from Runs r join Data d on d.RunID=r.RunID
    group by UserID
    order by 2 desc
)
select u.Name, b.N
from Users u join BusyUsers b on b.UserID=u.UserID
```

The CTE called BusyUser can be used as if it were a table in the second part of the command.

You can even write recursive CTEs as the simple counter here:

```
-- recursive queries with CTE
with n(i) as
(
    select 1
    union all
    select i+1 from n where i < 10
)
select i from n
```

With this advanced SQL construct you can create the Fibonacci series and calculate factorials or even traverse trees & graphs.

Data Management

Let's build a grading system for a class to learn about creating and managing data in tables. We'll talk about what these things mean as we go.

```
-- need table for grades and their score limits
create table Grades
(
    GradeID tinyint not null identity(1,1) primary key, -- automatic column
    Grade varchar(2) not null unique,  -- constraint to make grades unique
    ScoreLimit int not null, -- lower score limit for each grade
)
```

Primary keys (PKs) are unique and determine the data layout, so very fast to join on.

```
-- load some entries
insert Grades (Grade, ScoreLimit) values ('A+', 30)
insert Grades (ScoreLimit, Grade) values ( 0, 'F')

insert Grades values
      ('A',  27),
      ('A-', 25),
      ('B',  20),
      ('C',  15)

-- check on the grading order
select Grade from Grades order by ScoreLimit desc


-- exams
create table Exams
(
    UserID int not null,   -- Link to Users table
    Problem int not null,  -- 1-3
    Score int not null,    -- 0-10

    PRIMARY KEY (UserID, Problem) -- composite PK!
)


-- add data
insert into Exams (UserID, Problem, Score) values (12, 1, 10);
insert into Exams values (12, 2, 7);
insert Exams values (42, 2, 8); -- no such user

select * from Exams

-- remove all entries
-- truncate table Exams
```

```
-- remove certain rows: no such problem
delete Exams where Problem > 3
delete Exams where UserID > 100

-- remove entries non-existing users
delete e
from Exams e left join Users u on u.UserID=e.UserID
where u.Name is null

-- re-take a problem and get new score
update Exams
set Score = 10
where UserID=13
    and Problem=1

-- or using the name
update e set e.Score=13
from Exams e join Users u on u.UserID=e.UserID
where e.Problem=1 and u.Name like 'Hugo%'
;



-- what are the grades of the students?
with
UserScores (UserID, Total) as
(
    select UserID, SUM(Score)
    from Exams group by UserID
),
UserMax (UserID, Total, MaxLimit) as
(
    select UserID, Total, MAX(ScoreLimit)
    from UserScores join Grades
        on Total >= ScoreLimit
    group by UserID, Total
)
select u.Name, s.Total, g.Grade
from UserMax s
    join Users u on u.UserID = s.UserID
    join Grades g on g.ScoreLimit = s.MaxLimit
order by 1
```

Programming in SQL

SQL is a programming language with variables, functions, procedures, aggregates, triggers, etc. In fact we can write custom variants of everything, e.g., user-defined functions a.k.a. UDFs:

```
-- user-defined function or UDF: scalar
create function MaxDataX (@ymin float)
returns float
as
begin
      declare @x float = null;

      select top 1 @x = X
      from Data
      where Y > @ymin
      order by X desc

      return @x;
end
go


-- now try it
select dbo.MaxDataX(PI()/10)
select dbo.MaxDataX(99999)
go


-- table-valued UDF
CREATE FUNCTION dbo.MaxNDataX(@ymin float, @n int)
RETURNS @ret TABLE
(
    MaxX float not null
)
AS
BEGIN
    insert @ret
    select top (@n) X
    from Data
    where y > @ymin
    return;
END
go
-- drop function dbo.MaxNDataX


-- test it
select f.* from dbo.MaxNDataX(-2,5) as f
```

```
-- called them in place of a table and use them in joins, etc.
select MaxX, d.ID
from dbo.MaxNDataX(-3,10) x
      join Data d on ABS(MaxX - X) < 1e-6
go
/* tables are returned in memory by default */


-- inline functions will yield better performance
CREATE FUNCTION dbo.MaxNDataXinline(@ymin float, @n int)
RETURNS TABLE
AS
RETURN
(
      select top (@n) X as MaxX
      from Data
      where y > @ymin
)
go
-- drop function MaxNDataXinline

-- same calling syntax
select MaxX, d.ID
from dbo.MaxNDataXinline(-3,10) x join Data d on MaxX=X

-- apply to multiple rows
select d.ID, d.Y, f.MaxX
from Data d
      cross apply dbo.MaxNDataXinline(d.Y, 3) f
where d.Y > 5
```

Indexing Tables

To make searches faster we can build an index on any quantity in the table. To make searches and joins on RunID, we can do

```
create index ix_Data_RunID on Data
(
    RunID asc -- order by RunID
)
include -- optional but can make things faster
(
    X,  -- include columns to avoid going
    Y   -- back to the original table
)

-- remove the index
drop index ix_Data_RunID on Data
```

Transactions and Locking

Shared resources in parallel processing; Elevator problem; Mutual Exclusion (mutex); Dining philosophers;

Transactions have ACID properties (Jim Gray): Atomicity, Consistency, Isolation, Durability
Lock resources: Databases, Tables, Pages, Ranges, Rows, …

```
/*
  Programming transactions explicitly

  - keyword is TRANSACTION or TRAN for short
  - using try-catch to handle exceptions
*/
begin tran t1
begin try
    insert Instruments values ('New Instr')
    declare @i int = 1/0 -- generate divide-by-zero error
    commit tran t1
end try
begin catch
      select @@ERROR as ErrorCode -- print error code
      rollback tran t1
end catch
```

Exercises

1. Implement a user-defined function that returns the 3 largest Y measurements by a given student (UserID)
2. Save a table that contains all users (UserID) as well as the instruments (InsID) that they used; and store the special value NULL for the InsID, if no measurements were done
3. Write a SQL query using a recursive Common Table Expression to calculate the first 5 factorials