

Database Theory and Applications for Biomedical Research and Practice

BMIN 502 / EPID 635
Week 9: More database implementation

John H. Holmes, PhD



Today we are continuing to explore the implementation of data bases on the MySQL platform.

Agenda for today

- Joins, revisited
- Complex queries
- The Entity-Attribute-Value model

We will look at a common type of data model, known as Entity-Attribute-Value. We will also briefly review joins as understanding these is essential to developing more complicated queries.

Joins, revisited

Now, let's revisit joins.

Relationships between tables

- **Join**
 - Association between a field in one table and its counterpart in another table
- **View (or Dynaset)**
 - Sets (virtual tables) created dynamically as the result of joining two or more tables

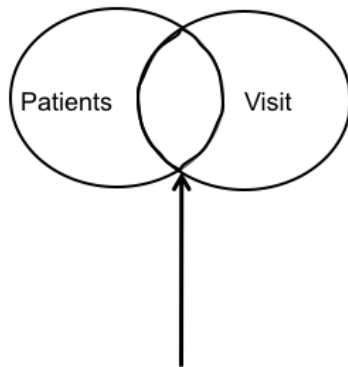
A join is an association or relationship between a field (or set of fields) on one table and the counterpart in another table. A join creates a view, or dynaset, which is a virtual table that is created on the fly as a result of executing SQL code with a join statement.

Types of joins

- **Equijoin (Inner join)**
 - Records are combined and added to dynaset only when values are equal for the join fields
- **Left outer join**
 - Records from “left-hand” table are added to dynaset even if none in “right-hand” table match
- **Right outer join**
 - Records from “right-hand” table are added to dynaset even if none in “left-hand” table match

There are three types of joins: Equijoin, Left outer join, and Right inner join. Please review how each works for a minute. The next three slides demonstrate them graphically.

INNER JOIN (or JOIN)



Result of JOIN

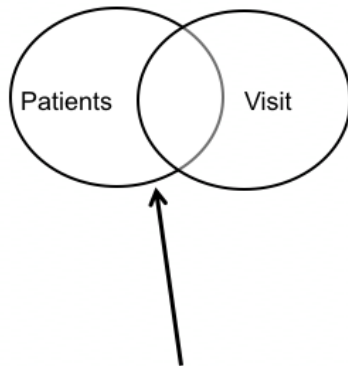
```
SELECT Patients.ID, Visit.VISITDATE
FROM Patients
JOIN Visit on Patients.ID = Visit.ID
```

Patients		Visit	
ID	NAME	ID	VISITDATE
1	Smith	1	1/1/95
2	Jones	1	1/2/95
3	Williams	1	1/7/95
4	Peterson	2	12/14/94
		2	12/22/94
		2	1/8/95
		2	1/9/95
		4	3/14/95
		5	4/1/95

Dynaset		
ID	NAME	VISITDATE
1	Smith	1/1/95
1	Smith	1/2/95
1	Smith	1/7/95
2	Jones	12/14/94
2	Jones	12/22/94
2	Jones	1/8/95
2	Jones	1/9/95
4	Peterson	3/14/95

Here is a basic join, which is the same as an equijoin. In this case, only those records that match the joining field(s) in the participating tables will be returned in the dynaset. Note that the dynaset does not include a record for Patient 3, nor a visit record for Patient 5. By the way, that visit record for Patient 5 is an orphan record, and should never be allowed in your databases. That's why, when setting up the model in the Workbench, you use the 1:M (or 1:1) relationship with the solid line, which enforces referential integrity, making orphan records impossible to enter or import.

LEFT JOIN



Result of LEFT JOIN

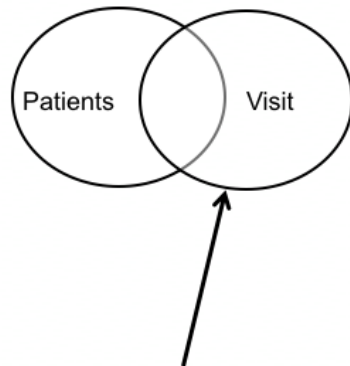
```
SELECT Patients.ID, Visit.VISITDATE
FROM Patients
LEFT JOIN Visit on Patients.ID = Visit.ID
```

Patients		Visit	
ID	NAME	ID	VISITDATE
1	Smith	1	1/1/95
2	Jones	1	1/2/95
3	Williams	1	1/7/95
4	Peterson	2	12/14/94
		2	12/22/94
		2	1/8/95
		2	1/9/95
		4	3/14/95
		5	4/1/95

Dynaset		
ID	NAME	VISITDATE
1	Smith	1/1/95
1	Smith	1/2/95
1	Smith	1/7/95
2	Jones	12/14/94
2	Jones	12/22/94
2	Jones	1/8/95
2	Jones	1/9/95
3	Williams	
4	Peterson	3/14/95

Here's a left join, the result of which will include all records from Patient and only those from Visit that match on the joining field, ID. Note that Patient 3 now has a record in the dynaset, but Patient 5 is not included. This is the most common type of join for those working in biomedical sciences. However, the other two joins can be very useful, especially in assessing the integrity and quality of the database.

RIGHT JOIN



Result of RIGHT JOIN

```
SELECT Patients.ID, Visit.VISITDATE
FROM Patients
RIGHT JOIN Visit on Patients.ID = Visit.ID
```

Patients		Visit	
ID	NAME	ID	VISITDATE
1	Smith	1	1/1/95
2	Jones	1	1/2/95
3	Williams	1	1/7/95
4	Peterson	2	12/14/94
		2	12/22/94
		2	1/8/95
		2	1/9/95
		4	3/14/95
		5	4/1/95

Dynaset		
ID	NAME	VISITDATE
1	Smith	1/1/95
1	Smith	1/2/95
1	Smith	1/7/95
2	Jones	12/14/94
2	Jones	12/22/94
2	Jones	1/8/95
2	Jones	1/9/95
4	Williams	3/14/95
5		4/1/95

And, finally, the right join. Note that the orphan record in Visit is now included in the dynaset, and Patient 3 is not included.

Complex queries

Now let's turn our attention to complex queries.

Complex queries

- Most commonly used to retrieve data from two or more tables
- Typically accomplished with JOINS
 - Left join of two tables:

```
SELECT Patients.ID, Admission.admitdate  
FROM Patients  
LEFT JOIN Admission on Patients.ID = Admission.ID
```

- Left join of three tables:

```
SELECT Patients.ID, Admission.admitdate, Radiology.normal  
FROM Patients  
LEFT JOIN Admission on Patients.ID = Admission.ID  
LEFT JOIN Radiology on Admission.ID = Radiology.ID
```

We often use joins as a way to effect complex queries, because these types of queries usually involve more than one, and sometimes many, tables. Take a minute and look at these two code examples. It might be helpful for you to sketch out an little entity-relationship model of each to see graphically how these work.

Let's try it!

Assignment 8: Exercise 1

Open up Assignment 8, and try Exercise 1.

Subqueries

- Also called nested queries
- You can nest any number of queries
- The inner-most query is executed first, then in succession each in turn to the outer-most
- Queries can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.
- A subquery is usually added within the WHERE Clause of another SQL SELECT statement

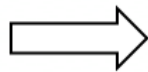
Subqueries are a bit more complex. You can nest queries within others, up to any level. In a nested query, the inner-most one is executed first, followed by the one at the next level up, and so on.

Example of a nested query

Patients		
patient_ID	dob	sex
1	10/1/1998	1
2	4/16/2004	2
3	3/7/1995	1
4	9/1/2000	1
5	11/15/1993	2

Admissions		
patient_ID	adm_date	hospital
1	10/1/1998	HUP
2	4/16/2004	PUPMC
3	3/7/1995	PAH
3	9/1/2000	PAH
4	11/15/1993	HUP
5	11/15/1993	PAH
5	8/1/1999	HUP
5	3/15/2003	PAH

```
SELECT patient_ID, dob FROM Patients WHERE  
patientID=(SELECT patientID FROM Admissions WHERE year(adm_date) >= 2000);
```



Patients	
patient_ID	dob
2	4/16/2004
3	3/7/1995
5	11/15/1993

Here's an example of a nested query, where we want to get the date of birth on patients who were admitted after 2000. Keeping in mind that the inner-most query is executed first, here's how it works:

Patients who were admitted later than 2000 are selected from the admissions table

For the patients who were selected in step 1, return their date of birth

Let's try it!

Assignment 8: Exercise 2

Try your hand at Exercise 2.

Functions

NB: x =a literal or a field

- **Numeric**
 - Performs arithmetic operations on a single value, but not across rows in a table
 - Example: `ROUND(x,d)` rounds x to d decimal places
- **String**
 - Example: `LENGTH(x)` returns the length of x
- **Date/Time**
 - Example: `DATE(x1, x2, INTERVAL y)` returns the difference between the later date ($x1$) and the more recent ($x2$), in the units expressed in y

Now we are going to take a look at a few functions. Again, take a look at Google, Muredach, and the MySQL documents that are on Canvas for a list of functions and what they do.

Let's try it!

Assignment 8: Exercise 3

Try your hand at Exercise 3.

Aggregate Functions

Perform across records

- Examples:
 - COUNT()
 - AVG()
 - STD()
 - MAX()
 - MIN()
 - SUM()
- Used with SELECT
 - SELECT COUNT(*) FROM Patients
 - Returns number of records in the Patient table

Aggregate functions are a special class- these perform an operation across records to provide some aggregate value, such as the average platelet count, or a count of records meeting some type of criteria in a WHERE statement.

Let's try it!

Assignment 8: Exercise 4

Try your hand at Exercise 4.

The Entity-Attribute-Value model

Now let's turn our attention to the Entity-Attribute-Value model

“Typical” relational tables

Patients		
patient_ID	dob	sex
1	10/1/1998	1
2	4/16/2004	2
3	3/7/1995	1
4	9/1/2000	1
5	11/15/1993	2

Admissions		
patient_ID	adm_date	hospital
1	10/1/1998	HUP
2	4/16/2004	PUPMC
3	3/7/1995	PAH
3	9/1/2000	PAH
4	11/15/1993	HUP
5	11/15/1993	PAH
5	8/1/1999	HUP
5	3/15/2003	PAH

- Tables = Entities
- Rows represent an instance of an entity
- Columns represent specific attributes

Here you see the “typical” structure that you’d find in your typical relational database. In this case, the tables are equivalent to entities, the rows represent an instance of an entity (and thus are often referred to as records), and the columns represent specific attributes or fields. This can be a very restrictive format, however.

However, sometimes this is a problem

- When you don't know how many columns to fix in a table
- When the data are likely to be sparse
- When you don't know how to model a relational database 😊

What if you don't know how many columns to fix in a table, or the data are likely to be sparse, or you don't enough information to model the database in the format you just saw on the previous slide? Here's an example of a situation where this could be a problem. Say you have a series of lab tests you want to capture, including hematology and chemistry. Here are some attributes you might want to capture:

- Hematology
 - Hemoglobin
 - Hematocrit
 - White blood cell count
 - Platelet count
- Chemistry
 - Blood glucose
 - Blood urea nitrogen
 - Serum creatinine
 - Potassium
 - Sodium

What if you later need to capture the attributes for the differential white cell count

for hematology, or add a series of chemistries that you hadn't accounted for? You would need to change the structure of a live database, and that's very dangerous. Or, what if a patient had only a hemoglobin and hematocrit (a very common subset of hematology tests)? Then you'd have a lot of holes in your data.

The Entity-Attribute-Value model

- Minimum of three columns in a single table!
 - Entity (table)
 - Attribute (name of the data element)
 - Value (the value of the attribute)
- Thus:

Entity	Attribute	Value
Patient	Gender	Female
Patient	BirthDate	3/10/1990
Lab	Hemoglobin	12.1
Lab	Hematocrit	35.7
Lab	Glucose	92
Lab	Potassium	4.3

So here's a solution: the Entity-Attribute-Value (or EAV) model. In this model, there are only three columns that are absolutely necessary, one for the Entity, one for the attribute, and one for the value of the attribute. Here's an example that solves the problem raised on the previous slide. It provides maximum flexibility in capturing data of all types, in that you would not need to know or anticipate the need for additional columns in a table. Take a few minutes to inspect this table.

Comparing traditional relational and EAV models

Relational	Patients		
	patient_ID	dob	sex
	1	10/1/1998	1
	2	4/16/2004	2
	3	3/7/1995	1
	4	9/1/2000	1
	5	11/15/1993	2
	Admissions		
	patient_ID	adm_date	hospital
	1	10/1/1998	HUP
	2	4/16/2004	PUPMC
	3	3/7/1995	PAH
	3	9/1/2000	PAH
	4	11/15/1993	HUP
	5	11/15/1993	PAH
	5	8/1/1999	HUP
	5	3/15/2003	PAH

EAV	Entity	Attribute	Value
	Patient	patientID	1
	Patient	dob	10/1/1998
	Patient	sex	1
	Admissions	adm_date	10/1/1998
	Admissions	hospital	HUP
	Admissions	adm_date	4/16/2004
	Admissions	hospital	PUPMC

Here is a comparison of the traditional relational model, in which there is a column for every attribute, and a record for every instance within a given table- with the EAV model, in which there is a row for each E-A-V triplet. See the difference?

An example of a table in EAV format

Patient ID	Age (years)	Gender	Event Type	Event Name	Event Code	Start Date	End Date	Relevant Data
1031926	31	FEMALE	Diagnostic Test	Platelet	Platelet	3/19/2012		Result: 251 THO/uL,
1031926	31	FEMALE	Diagnostic Test	nrRBCPOP	nrRBCPOP	3/19/2012		Result: Unit of Measure Not Available,
1031926	31	FEMALE	Diagnostic Test	% Neutro Auto	% Neutro Auto	3/19/2012		Result: 65 %,
1031926	31	FEMALE	Diagnostic Test	MCHC	MCHC	3/19/2012		Result: 34 g/dL,
1031926	31	FEMALE	Diagnostic Test	AMC	AMC	3/19/2012		Result: 0 THO/uL,
1031926	31	FEMALE	Diagnostic Test	Chloride	Chloride	3/19/2012		Result: 100 mmol/L,
1031926	31	FEMALE	Diagnostic Test	Alk Phos	Alk Phos	3/19/2012		Result: 47 U/L,
1031926	31	FEMALE	Diagnostic Test	% Mono Auto	% Mono Auto	3/19/2012		Result: 6 %,
1031926	31	FEMALE	Diagnostic Test	ALT	ALT	3/19/2012		Result: 13 U/L,

If you wanted to obtain mean platelet counts across all patients,
How would you do it?

Slide 24

Here is another example, this one drawn from PennOmics in the usual way you get data from the warehouse. In this case, there are extra columns for Patient ID, Age, Gender, Event Code, Start Date, and End Date. But the highlighted ones are the skeleton of the EAV model, where Event Type is the Entity, Event Name is the Attribute, and Relevant Data is the Value. This is typically how the data come from PennOmics or Penn Data Store, or the CHOP clinical data warehouse- or just about any electronic health record system.

How would you answer the question on the slide?

Here are the issues for this query

- There is a record for each component of a lab test
- You would need many joins to retrieve each platelet instance, because they are spread out in separate rows
- The values have to be parsed from the units of measurement

So here are things you should be thinking about. You can see how the EAV makes the data architect's life easy, because the structure of the table is so simple. But for the database administrator, and you as a user, it isn't so simple!! There is a way to address this in SQL, and the steps for doing this are provided on the next several slides.

Steps to success: 1

1. Import the table as-is into platelets_EAV

2. Create a new table:

```
CREATE TABLE platelets_new (  
    patient_ID INT NOT NULL,  
    start_date DATETIME,  
    event_name VARCHAR(30),  
    platelet_count FLOAT,  
    PRIMARY KEY (patient_ID, lab_date));
```

First, import the EAV table as-is into a schema in MySQL. Then, create a new table that will have the fields needed to address the question on Slide 24- specifically platelet_count.

Steps to success: 2

3. Convert and transfer the data:

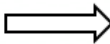
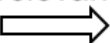
```
INSERT INTO platelets_new
(patient_ID, start_date, event_name, platelet_count)
SELECT patient_ID,
       start_date,
       event_name,
       CASE WHEN event_name='Platelet' THEN
           CAST(relevant_data) AS
           FLOAT( SUBSTR(relevant_data,8,LOCATE(relevant_data,' ',9)
                       IN relevant_data))
       FROM platelets_EAV
```

Now, you need to convert and transfer the EAV data into the new table you just created. Note that there are several new functions (CASE WHEN, CAST, FLOAT, and LOCATE). Look these up on Google or in Muredach for more information about how these are used and why you need them. But the basics of this code are that you need convert the contents of the “relevant_data” field to a platelet count that you can use for analysis.

How did SUBSTR and LOCATE work?

Patient ID	Age (years)	Gender	Event Type	Event Name	Event Code	Start Date	End Date	Relevant Data
1031926	31	FEMALE	Diagnostic Test	Platelet	Platelet	3/19/2012		Result: 251 THO/uL
1031926	31	FEMALE	Diagnostic Test	nrRBCPOP	nrRBCPOP	3/19/2012		Result: Unit of Measure Not Available,
1031926	31	FEMALE	Diagnostic Test	% Neutro Auto	% Neutro Auto	3/19/2012		Result: 65 %
1031926	31	FEMALE	Diagnostic Test	MCHC	MCHC	3/19/2012		Result: 34 g/dL
1031926	31	FEMALE	Diagnostic Test	AMC	AMC	3/19/2012		Result: 0 THO/uL
1031926	31	FEMALE	Diagnostic Test	Chloride	Chloride	3/19/2012		Result: 100 mmol/L
1031926	31	FEMALE	Diagnostic Test	Alk Phos	Alk Phos	3/19/2012		Result: 47 U/L
1031926	31	FEMALE	Diagnostic Test	% Mono Auto	% Mono Auto	3/19/2012		Result: 6 %
1031926	31	FEMALE	Diagnostic Test	ALT	ALT	3/19/2012		Result: 13 U/L

Result: 251 THO/uL
123456789012345678

1. SUBSTR(relevant_data,9,LOCATE(relevant_data,' ',9))
 251 THO/uL
2. SUBSTR(relevant_data,9,LOCATE(relevant_data,' ',9))
 251

Now that you've gotten a bit familiar with the functions in the SQL coder on the previous slide (you did look them up, right?), take a look at the mechanics of how they actually worked in our example. Hint: the reason the "9" is highlighted in red is because that will be the position of the Value you need, excluding the text "Result: ". We will want to use SUBSTR again, to strip off the units.

For Assignment 9, Part 1

1. Get the PennOmics data in EAV format .csv file from the Files folder on Canvas
2. The tasks for Assignment 9 are on Canvas
3. Try completing the assignment- it's not due until April 3, but we will discuss it in class on March 27

You should start on Part 1 of Assignment 9. It's not due until April 3, but we will discuss it in class on March 27, as it's a tricky assignment. Yes, there will be a Part 2, but we'll discuss that in class as well.