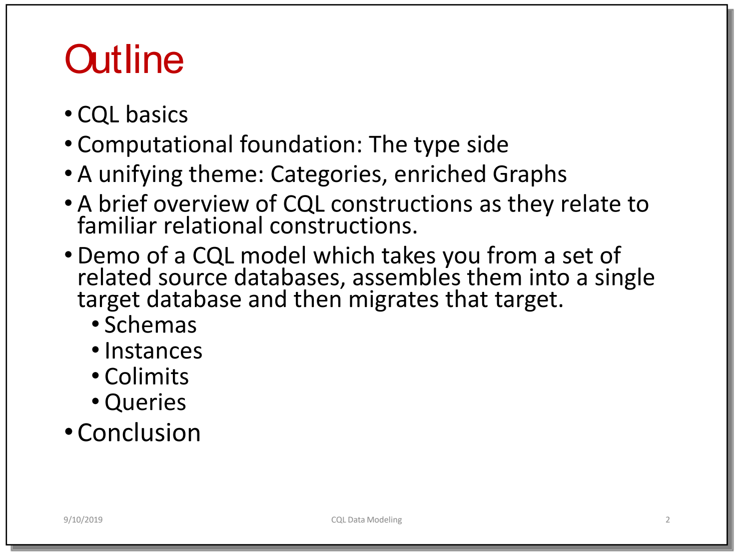
Slide 1

CQL: Databases Done Right

Peter Gates
Conexus.ai

$\Sigma \dashv \Delta \dashv \Pi$

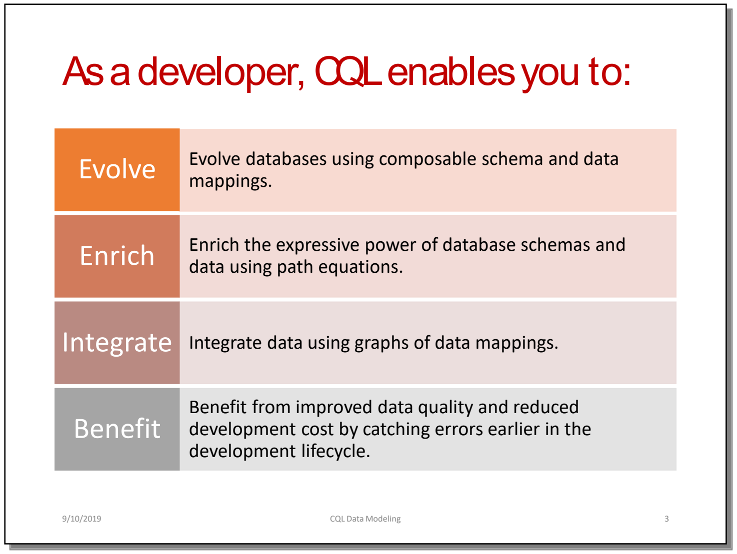
Slide 2



Here is an outline of the talk. We start with some user stories and a value proposition for CQL. We then provide a basic introduction to what CQL is and its conceptual and mathematical underpinnings.

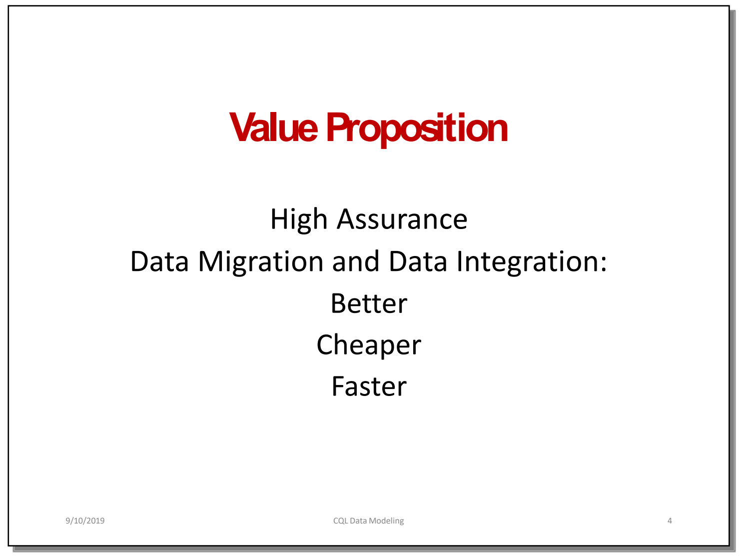
The body of the talk will involve a series of dives into some of the core features of CQL with screen shots from working executable files or CQL models. We will then wrap up with some conclusions regarding where we think CQL fits into the data and technology landscape.

Slide 3



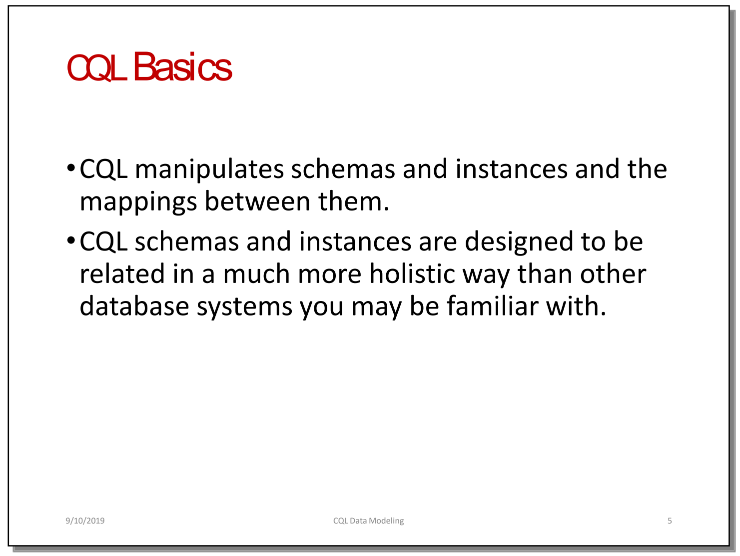
These are user stories. For the remaining part of this talk we will conduct a technical survey of specific CQL features to provide some grounding for these claims.

Slide 4



High assurance is achieved through the application of an imbedded theorem prover. This can be seen as intelligent assistance or IA. The theorem prover is able to catch errors that violate the underlying mathematics.

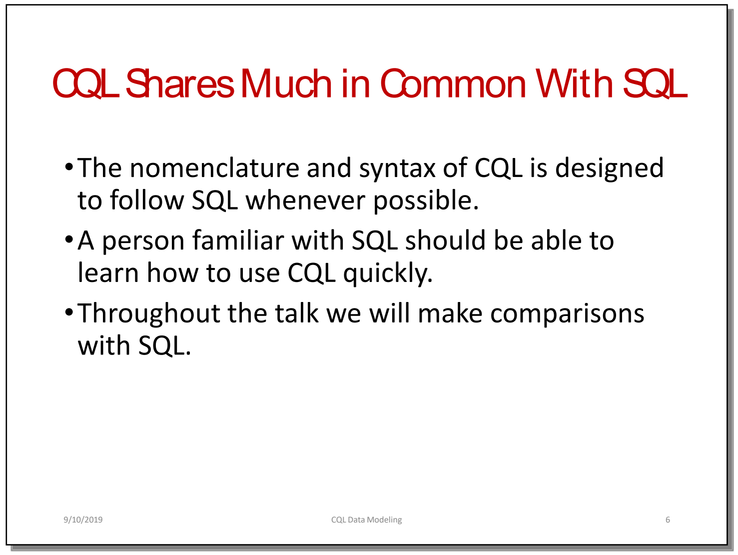
Slide 5



Schemas and instances are first class objects in CQL. What does that mean?

Mappings between schemas and instances treat those constructions as an encapsulated whole. What defines that whole?

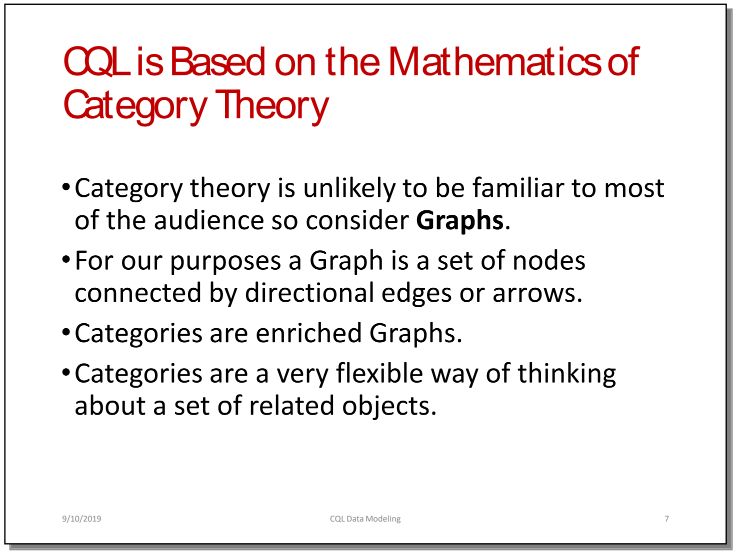
Slide 6



The name CQL suggests some connection with the tried and true Structured Query Language. CQL stands for Categorical Query Language. We will briefly discuss why later.

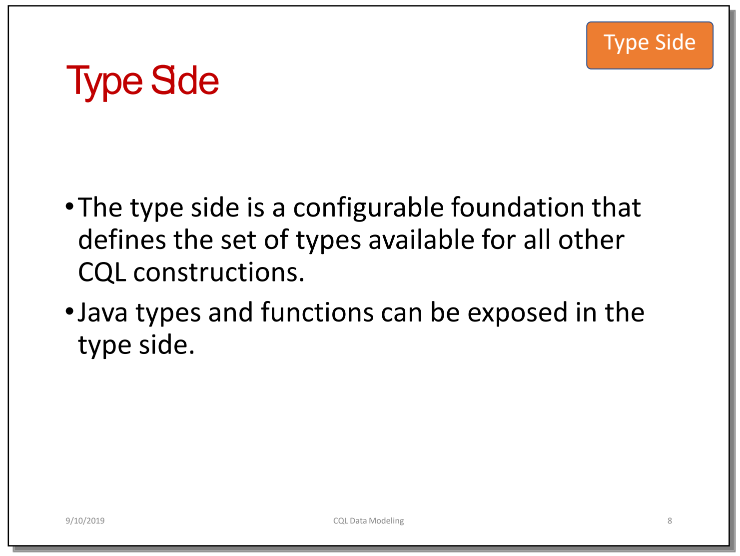
CQL syntax follows SQL syntax wherever possible and throughout the talk we will draw similarities between the two.

Slide 7



Let me ask the question: Does the audience feel comfortable with the concept of a directed graph?

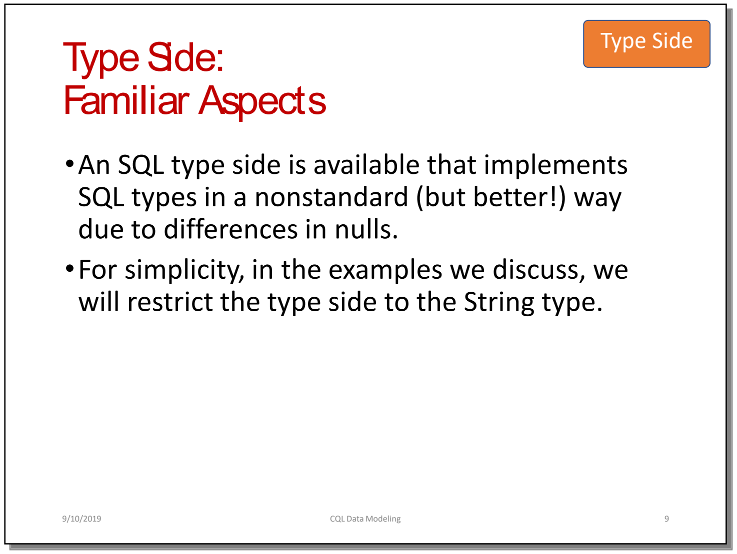
Slide 8



The type side is the foundation upon which all CQL constructions are based. You can roll your own type side from scratch if you understand how to construct multi-sorted algebraic theories. Alternatively, for your convenience, you can just expose whatever Java types and functions you would like to use in your database.

Better still: Next slide

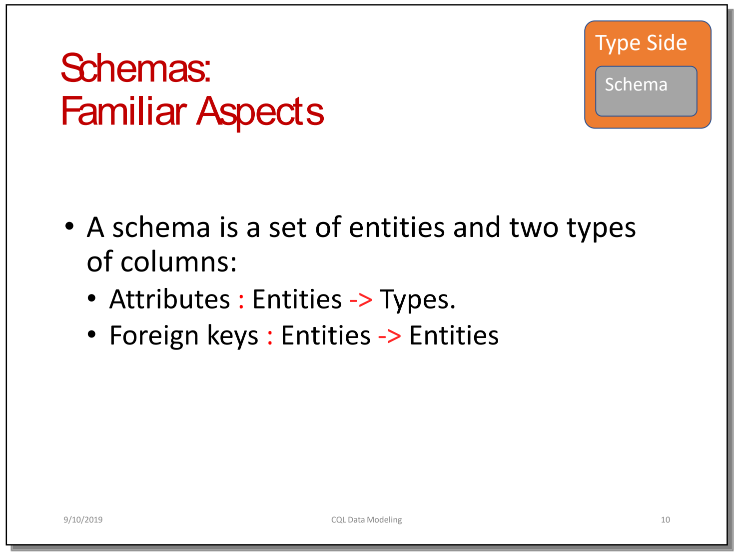
Slide 9



The CQL type side is analogous to the built in SQL types typical to other database systems. We will explore one important difference involving the way null values are handled. CQL provides a database developer with a great deal of flexibility in defining the type side. In the examples we explore we will restrict the type side to the string type.

By the way, without digging too far into the details, the string type is a simple example of an algebra. Because an algebra is a well-defined mathematical concept, we can build other CQL constructions on the type side and apply an embedded automated theorem prover to verify the correctness of CQL constructions.

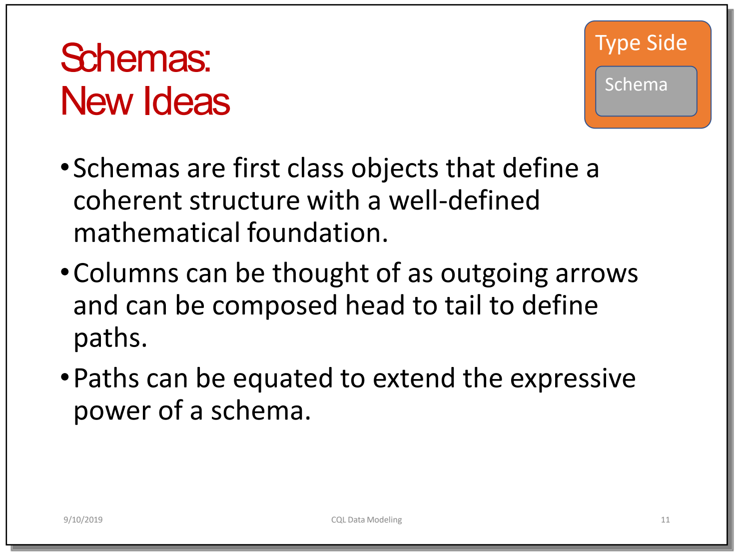
Slide 10



Notice there is something implied in this slide. Can anyone see what that is? I will give a hint. What is the connection between schemas and graphs?

There is an additional column not mentioned here that you might expect based on your familiarity with relational databases.

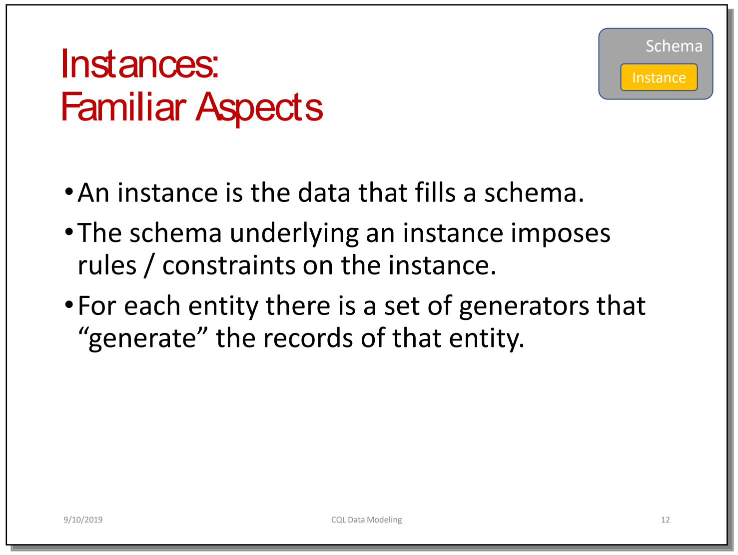
Slide 11



A schema is seen by the imbedded theorem prover as an intact whole with a well-defined structure. Think of a schema as a graph with some additional properties to include an identifier for each node and paths that can be used in equations.

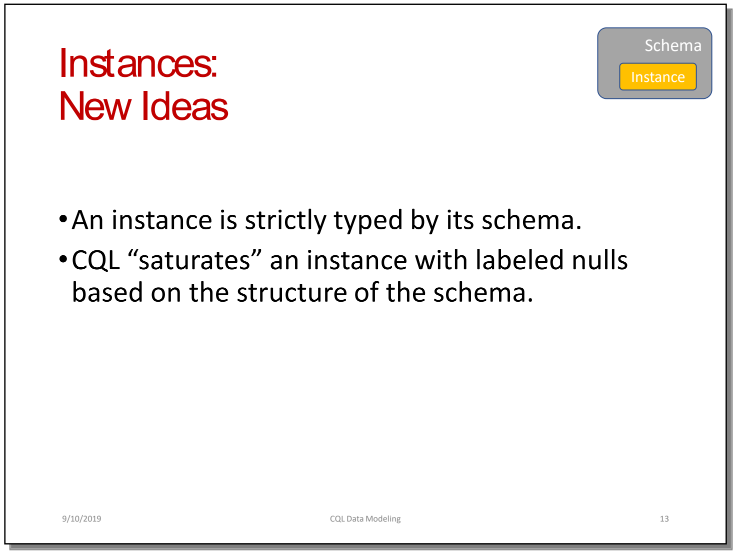
Question: It is worth mentioning that you cannot equate any two paths. What might you want to require of a pair of paths for them to be equated?

Slide 12



CQL instances share much in common with SQL data. One important difference is that CQL instances are graphs (categories) much like RDF graphs.

Slide 13

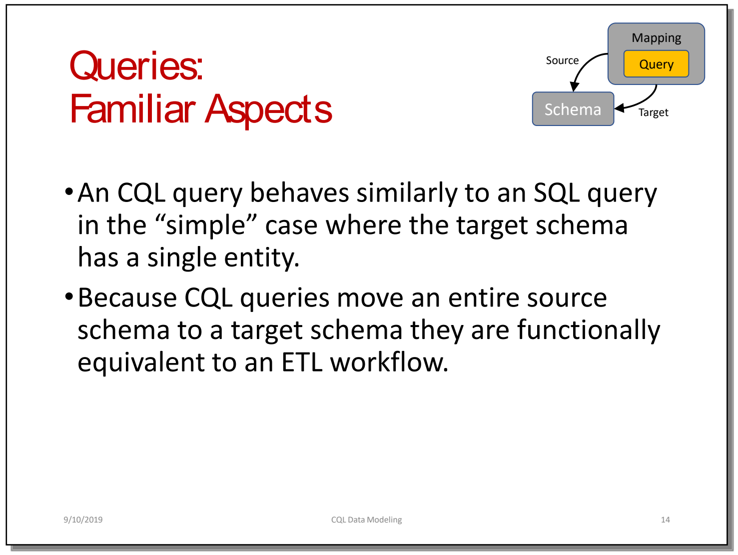


CQL instances differ from SQL instances in that one can define equations. We will not explore this feature in this presentation. The other difference is that null values are much more strongly typed than what you are used to with relational null values. A CQL null is typed by both its generator (row identifier) as well as the column in which it appears.

So for example, if we had an attribute that documented a patient’s (say John Doe) body weight (column), the null value would be typed by the generator associated with John Doe’s record and the body weight column. This would be interpreted as John Doe’s unknown body weight.

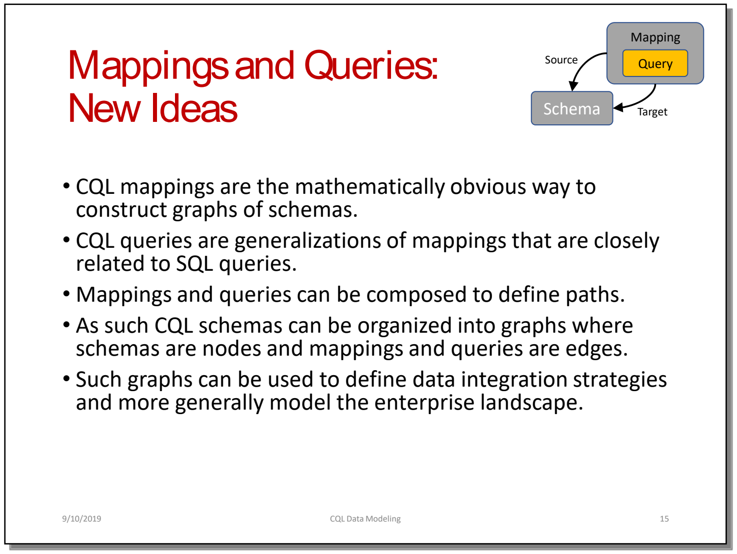
The advantage of this is that labeled nulls can be used in equations like any other value. For example we might not know either John Doe’s body weight or height, but we might know that if we take John Doe’s unknown weight in kg divided by his unknown height in meters squared, that equation involving labeled nulls is equal to a BMI of 26.

Slide 14



A CQL query constructs single entities in a way that is very similar to SQL. CQL differs in that the query construction has a separate SQL like block for each entity in the target schema. These blocks are interrelated through variables that rang over generators of the entities in the source schemas. Also each entity block includes a clause that constructs outgoing foreign keys of the target schema from generators associated with source entities. These two features provide a mechanism for mapping a source schema to a target schema that can be validated at compile time as preserving the data integrity of the source schema when mapped to the target schema.

Slide 15



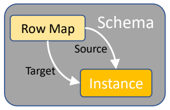
CQL queries and mappings are related constructions that define ways of migrating data from one schema to another. Mappings are roughly functional in that they are many to one. Queries extend mappings by being many to many mappings. So CQL mappings are to CQL queries as functions are to relations.

The consequence of this is that CQL mappings and queries can be used to define graphs (categories) whose nodes are databases and whose arrows are either mappings or queries. This defines an enterprise level picture of the data landscape.

A final point, we have now described schemas, instances and diagrams of related schemas as directed graphs each enriched with the additional structure of a category. This means we have three layers with which describe databases all of which are "categorified" and woven together into a mathematical rigorous tapestry.

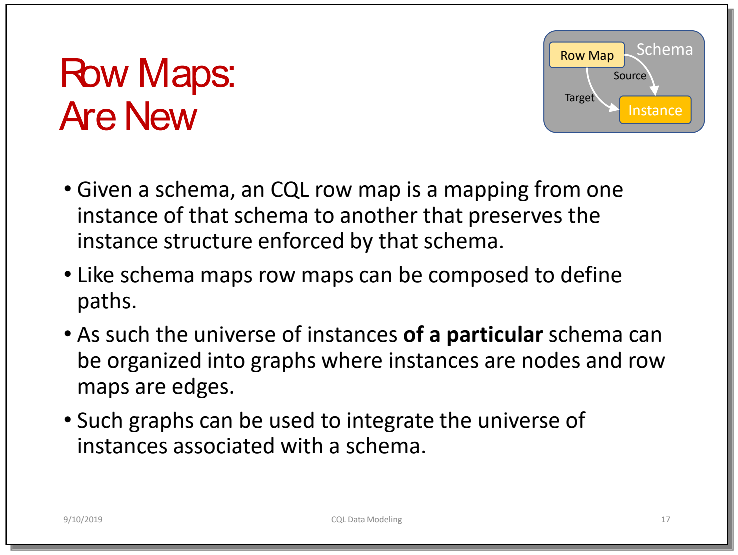
Slide 16

Row Maps:
That Which is Familiar



$$\varnothing$$

Slide 17



We will not discuss row maps in detail but mention them in passing as they are necessary to support the categorification of instance data. As already mentioned, instances are individually categories that look a lot like RDF graphs where the nodes are data elements and the arrows define how data elements are related.

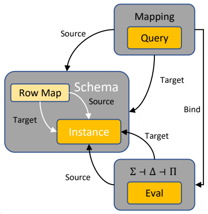
One can also consider a category whose nodes/objects are each an instance of some schema. If instances of a schema define objects what are the arrows that relate them. The answer is row maps. Row maps define the arrows between instances specific to a schema.

Slide 18

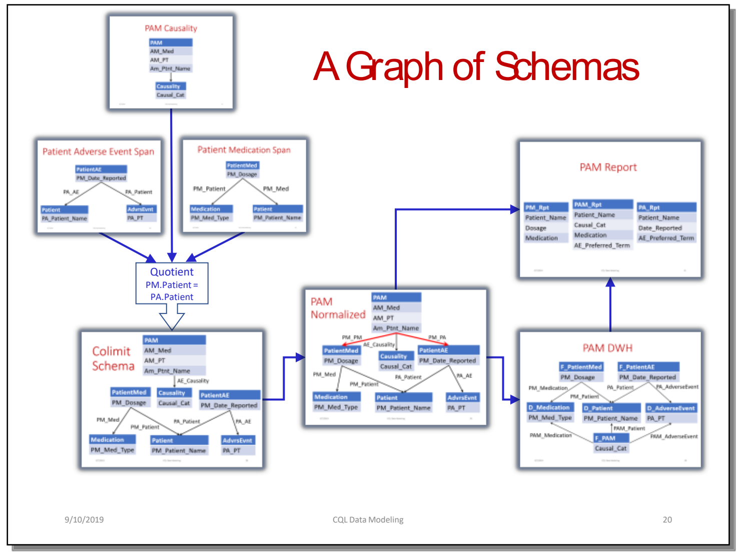
But We Have Data In Different Schemas!

Slide 19

Data Migration Demo

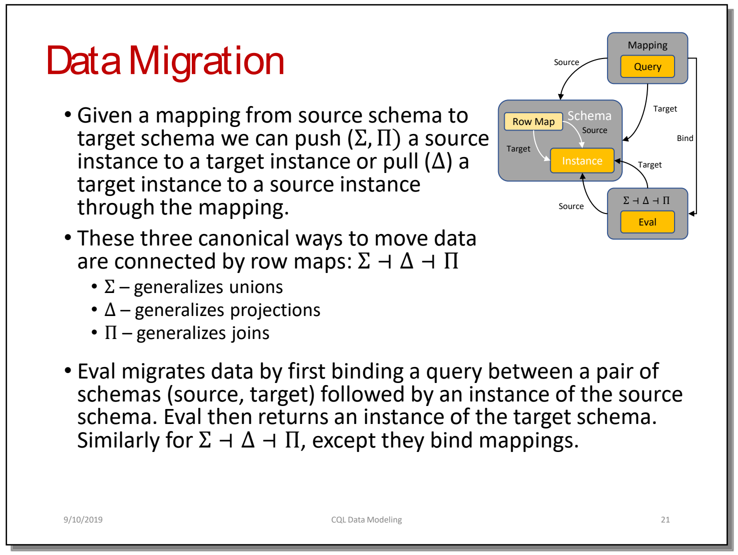


Slide 20



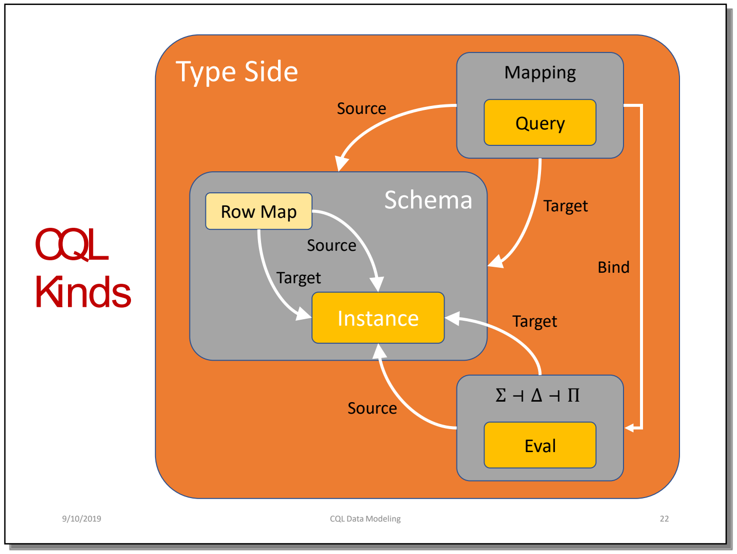
The diagram above is a schema level summary of the demo. The diagram includes the assembly of three source schemas into a single schema using the colimit quotient construction. The colimit schema in turn is then transformed several times illustrating various features of the query construction.

Slide 21



This slide is an overview of some of the technical features that are used to create the various ways CQL migrates data from one schema to another. The inset in the upper right-hand corner of the slide is a high-level picture of the CQL architecture.

Slide 22



Architectural summary of CQL kinds.

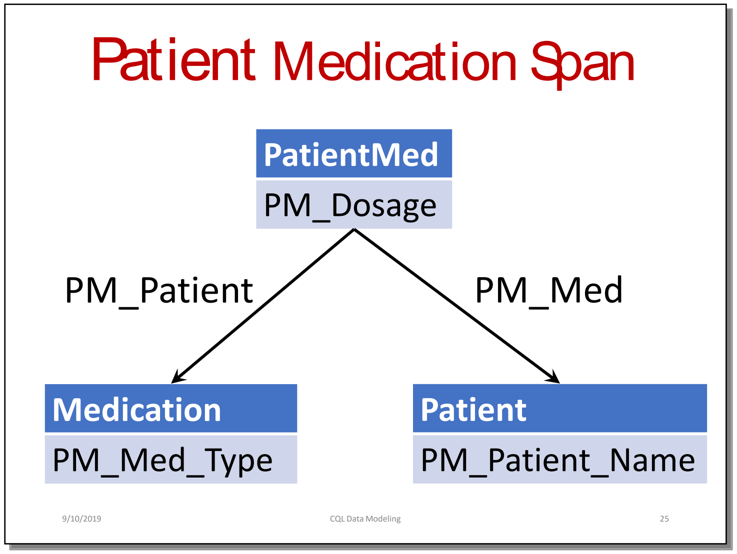
Slide 23

Conclusion

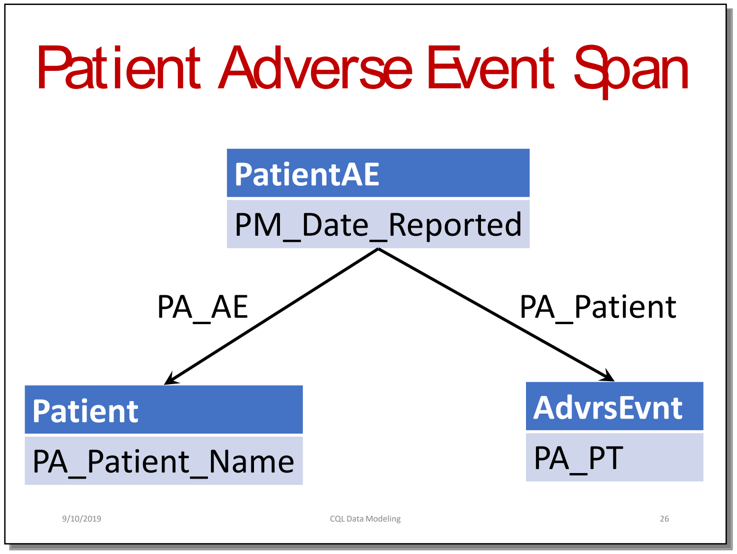
- Graphs of the universe of database instances with their maps between different databases are models of the enterprise.
- CQL's constructions from the type side foundation through schemas, instances, schema maps, row maps and enterprise maps provide a framework to bridge from business requirements to computational implementation.
- CQL is an intelligent assistant to a data architect.

Slide 24

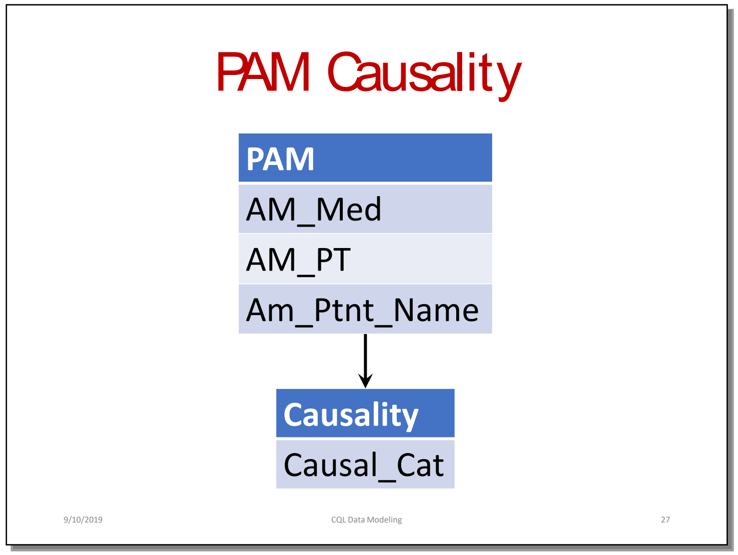
Slide 25


Patient Medication Span
PatientMed
PM_Dosage
PM_Patient
PM_Med
Medication
PM_Med_Type
Patient
PM_Patient_Name
9/10/2019
CQL Data Modeling
25

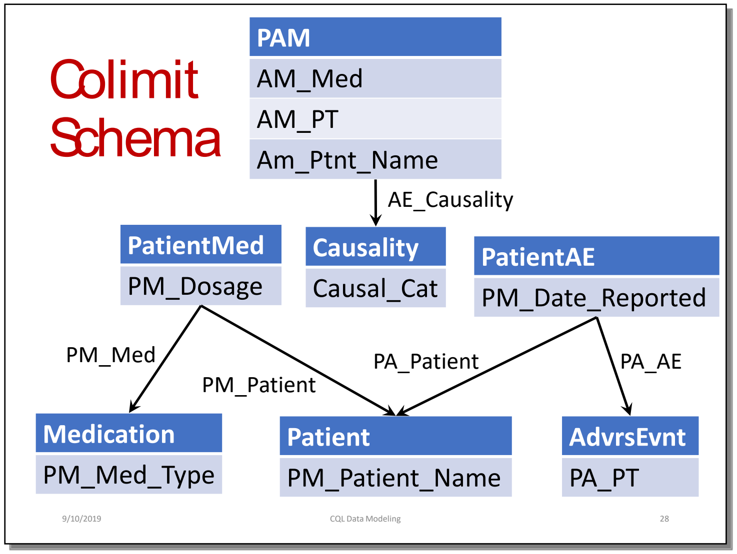
Slide 26


Patient Adverse Event Span
PatientAE
PM_Date_Reported
PA_AE
PA_Patient
Patient
PA_Patient_Name
AdvrsEvnt
PA_PT
9/10/2019
CQL Data Modeling
26

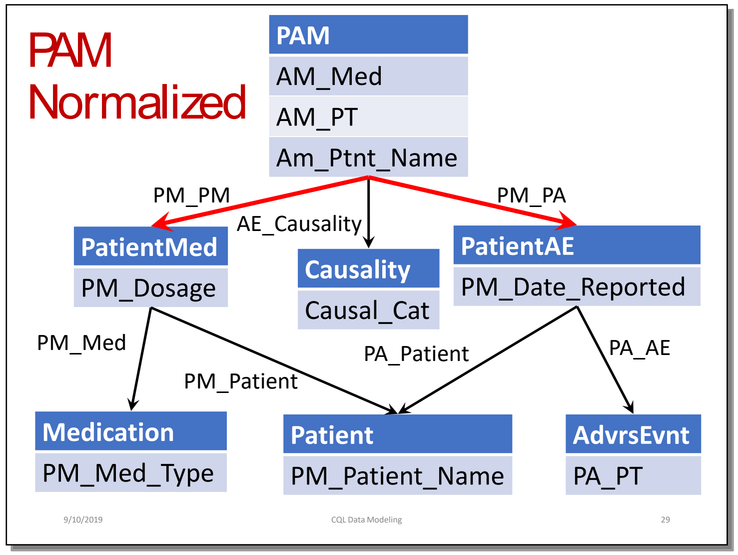
Slide 27


PAM Causality
PAM
AM_Med
AM_PT
Am_Ptnt_Name
Causality
Causal_Cat
9/10/2019
CQL Data Modeling
27

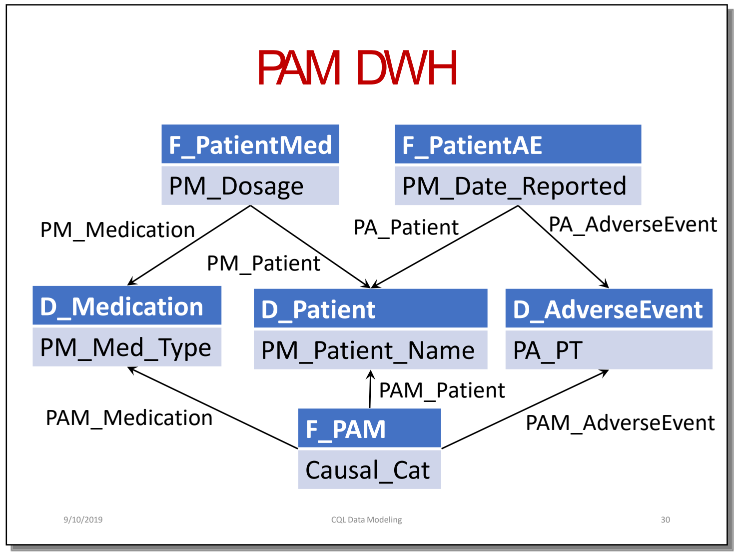
Slide 28


Colimit Schema
PAM
AM_Med
AM_PT
Am_Ptnt_Name
AE_Causality
PatientMed
PM_Dosage
Causality
Causal_Cat
PatientAE
PM_Date_Reported
PM_Med
PM_Patient
PA_Patient
PA_AE
Medication
PM_Med_Type
Patient
PM_Patient_Name
AdvrsEvnt
PA_PT
9/10/2019
CQL Data Modeling
28

Slide 29


PAM
Normalized
PAM
AM_Med
AM_PT
Am_Ptnt_Name
PM_PM
AE_Causality
PM_PA
PatientMed
PM_Dosage
Causality
Causal_Cat
PatientAE
PM_Date_Reported
PM_Med
PM_Patient
PA_Patient
PA_AE
Medication
PM_Med_Type
Patient
PM_Patient_Name
AdvrsEvnt
PA_PT
9/10/2019
CQL Data Modeling
29

Slide 30



PAM DWH
F_PatientMed
PM_Dosage
F_PatientAE
PM_Date_Reported
PM_Medication
PM_Patient
PA_Patient
PA_AdverseEvent
D_Medication
PM_Med_Type
D_Patient
PM_Patient_Name
D_AdverseEvent
PA_PT
PAM_Patient
PAM_Medication
PAM_AdverseEvent
F_PAM
Causal_Cat
9/10/2019
CQL Data Modeling
30

Slide 31

PAM Report

PM_Rpt
Patient_Name
Dosage
Medication

PAM_Rpt
Patient_Name
Causal_Cat
Medication
AE_Preferred_Term

PA_Rpt
Patient_Name
Date_Reported
AE_Preferred_Term