[[[REMINDERS ON POINTS FOR INTRODUCTION]]] <==

Problems of current approaches to combining databases with statistical signal processing

- Statistics and predictive analytics involve a tedious process where data are moved out of databases into statistical processing software (eg R) and then stored back.
- None of the cornerstones of the success of SQL systems is available: No logical/physical separation, no declarative queries and no automatic optimization of storage and query processing.
- Spatiotemporal sensor data are qualitatively different in a way that prevents the direct use of SQL, which has been tuned for discrete precise data: Unlike ERP and CRM business use cases, where the database data constitute the world, sensor data are mere measurements of the real world - potentially imprecise. Models (continuous functions over space and time) are the actual representation of the world and predict values at any coordinate. Query languages and database modeling has to facilitate the use of predicted values.
- Sensor data are unnecessarilly big in their raw form. Yet signal processing research has known for long that data can be split into signal and the often unimportant noise, leading to high effective lossless compression.

The benefits of a model-aware database

- Plato brings models in the database. Models are continous functions that predict the quantity of interest at any coordinate. A model introduces knowledge of the external physical reality.
- Well-developed theory and set of models and model generators that effectively capture many real world cases.
- Holds promise to solve the problems above. Plato enables declarative queries and automatic optimization of storage and query processing.

Research Challenges

- Specify declarative query languages that process models.
- The database algorithms can work efficiently directly on the models. Specify a logically/physically-separated architecture where an optimizer is aware of the specifics of the model representation and chooses query processing algorithms accordingly. For example, consider two models represented by their Fourier transform and a query that asks for their correlation. It is most efficient to compute directly on the frequency domain rather than bringing back to time domain.
- Signal processing community provides a wide variety of model generators, with various guarantees on the loss information. Plato must semiautomate the choice of the appropriate model generator. This requires a classification of model generators with respect to their loss of information properties. More importantly, different queries may pose different needs of accuracy and of what constitues information loss. The semiautomation must take into account query workload. The query answering must utilize the best model for the problem. Granularity of queries can also play a role: Can you avoid computing the entire model and instead compute on the fly only the parts of the model that are of interest to the query? Which models are best fo such?
- How do you incrementally maintain the model as the data change? Tradeoff between speed of convergence and computational resources used. Again, query workload must specify.

Use cases

0.1 Prior work on model-aware databases

[[[to Yannis K: Please check and add to the representations I make about the prior work.]]] <==

Works from ML and statistical signal processing communities [[[for Yoav]]] <==

MauveDB Argued for the value of direct SQL processing. MauveDB uses models to predict values on a developer-specified grid.

- Plato argues that models need no be discretized in the coordinates' grid. A fully virtual approach, where the model is perceived as a continous function, is easier for the developer and more opportune for the optimizer. For example, consider two models represented by their Fourier transform and a query that asks for their correlation. It is most efficient to compute directly on the frequency domain rather than bringing back to time domain.
- Did not investigate the connections between (a) choosing models and query workload and (b) query guarantees given chosen models.

Function Db (MIT)

- Showed that in the case of models captured by polynomials better compression and faster processing is achieved by working directly on polynomials.

Aberer [[[for Yannis K: It seems he promised a lot and did a little. What do we write about this?]]] <==

Approximate query processing

1 Plato: An extensible model-aware database

The proposed Plato database aims to solve the discussed problems by introducing models as first class citizens. This section provides the basic definitions, architecture (Figure **??**) and examples.

Sensor data measurements lead into conventional SQL tables, which we will refer to as *measurements tables*. We will regularly name space and time attributes as X, Y, Z and T. Their values will be denoted by x, y, z, t. We will denote the respective types by caligraphic letters $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$ and $\mathcal{T}$.

EXAMPLE 1.1 In our running example, a tuple (s, t, q) of a measurements table with schema `temp_meas(Sensor, T, Temp)` (i.e., a table with attributes `Sensor`, `T` and `Temp`) means that the temperature sensor s provided at time t the temperature measurement m. The `Sensor` attribute is a foreign key to a table `temp_sensor(ID, X, Y, Installed_on, ...)` that provides the latitude, longitude and other data about each sensor.[1] □

Plato's models are inferred from the measurements and external knowledge of the domain. They are mathematical representations of the world, which predict a quantity of interest (e.g., temperature). In the basic case, a *model* is a continuous function $f : \bar{\mathcal{D}} \mapsto \mathcal{Q}$. For each vector $\bar{d}$ in the (generally multidimensional) spatiotemporal domain $\bar{\mathcal{D}}$ the function f returns a quantity $q \in \mathcal{Q}$. The domain $\bar{\mathcal{D}}$ could be a sphere of (x, y, t) points around a center (x_0, y_0, t_0), a square of (x, y) points, etc.

Plato treats the function signature $\bar{\mathcal{D}} \mapsto \mathcal{Q}$ as a database type. Any table may have an attribute (or more attributes) whose values are models conforming to a particular type, i.e., particular function signature. In that sense, a model operates as a database User Defined Function (UDF). We will refer to a table that has model attributes as a *model table*.

Technically, a model is the output of a *model generator* function and, respectively, a model table is the output of a view definition that involves model generators. A model (or a model table for that matter) may be virtual in the sense that SQL view definitions are virtual, i.e., no actual computation is performed until a query uses them. However, in practice, the administrator is motivated to materialize models (and the respective model tables) in order to benefit from the data compression that models enable. Indeed, the administrator has to choose between multiple model generators, which come with different compression levels and corresponding guarantees regarding information preservation. Indeed, a major research issue, discussed in Section **??**, is the semiautomation of the choice of model generators, which should take into consideration the query/analytics workload.

EXAMPLE 1.2 A tuple (s, f) of the table `temp_models(Sensor, F:t->temp)` means that the function f describes the real world temperatures, which sensor s was observing. The signature `t -> float` indicates that the functions/values of `F` input time and output floating points. The table `temp_models` is defined by the view definition:

```
DEFINE MATERIZALIED VIEW sensor_models AS
SELECT sensor, arma_t_nr(G)
FROM temp_meas GROUP BY sensor AS G(T, Temp)
```

The view definition groups the temperature measurements sensor, and then the measurements `G` of each sensor are provided to the model generator function `arma_t_nr`, whose single argument must be a table of time-float tuples. The generator infers an ARMA model that captures the underlying temperature signal. The `nr` is a characterization, applicable to multiple generators, on how to separate signal from noise. Note that the temporal ARMA and `nr` choices are just one among the many model generator options.

In a modification of the running example, the administrator chooses to capture temperature in a single model `full_model` that predicts the temperature based on `x`, `y` and `t` and is defined as follows:

```
DEFINE MATERIALIZED VIEW full_model AS
arma_xyt_nr(SELECT X, Y, T, Temp
```

[1] The data administrator can decide what is the best schema for the measurements data. For example, instead of a single table carrying all the temperature sensor measurements, the administrator may choose one table for each sensor.

```
FROM temp_meas, tempsensor WHERE Sensor = ID)
```
The reasons for doing so could be: (a) The individual sensor models may be heavily correlated based on the `X` and `Y` sensor coordinates. In such case, a single model can have a much more compact representation than the collection of individual sensor models. (b) The analytics may need temperature predictions for specific locations, which do not coincide with any individual sensor. □

Vector quantities In the general case, a model's quantity is a vector (rather than a scalar) from the domain $\bar{\mathcal{Q}}$.

Probabilistic model tables In the general case a model returns probability distributions of the quantities, rather than absolute values. Therefore a model is a function $f : \bar{\mathcal{D}} \mapsto \mathcal{H}_{\bar{\mathcal{Q}}}$, where $\mathcal{H}_{\bar{\mathcal{Q}}}$ is the space of probability distributions over the quantity domain $\bar{\mathcal{Q}}$. The probability distributions capture the certainty of the predicted values. For example, the model of Example 1.2 can easily benefit from returning probability distributions of the predicted temperatures. A key use case is that the probability distributions indicate how certain one is about the confidence intervals of the predicted quantities.

Viewing models as infinitely-sized tables Notice that a model $f : \bar{\mathcal{D}} \mapsto \mathcal{H}_{\bar{\mathcal{Q}}}$ can be also perceived as a table $R_f(\bar{D}, \bar{Q}, H)$, where the table has an infinite number of tuples, the coordinates $\bar{D}$ form a key and the attribute H provides the value of the probability distribution at the specific coordinates. Correspondingly, the models that are the values of a table's attribute can be perceived as infinite size nested tables.[2]

1.1 Queries and Optimized Query Evaluation

A key success factor of database systems has been the declarative SQL query language where the user/developer expresses the desired result of his analysis without having to specify the algorithm that computes this result and without having to refer to the data structures where the data are stored. The database discovers the optimal plan to compute the result, making best use of the available data structures.

In the spirit of declarative programming, Plato offers two categories of SQL extensions that enable model-related computations, while Plato utilizes model-specific rewritings to optimize the computation.

Unlike conventional query optimization, where the rewritings produce equivalent expressions, opportune rewritings in model-based databases are not guaranteed to produce queries with identical results. Rather, in many importnt cases the results are guaranteed to be equivalent under common assumptions of statistical signal processing. In other cases, they are guaranteed to be equivalent within certain error bounds. Plato queries allow the user to specify the requested guarantee by providing appropriate parameters.

The first category utilizes various functions that input models and output statistic measures or models. The second category allows variables that range over the infinitely-many coordinate points.

2 Opportunities in Data Compression

As is well known from signal processing, relatively few model classes, such as ARMA models, Fourier and wavelet-based models and Singular Value Decomposition (SVD) based models capture very well various scenarios. The key function of the raw data administrator is to choose the appropriate *model generator* and feed it with the appropriate parameters that will dictate compression, accuracy. The important lesson from signal processing, which we will discuss in Section 2, is that high compression can be achieved with minimal or even no loss of accuracy.

[2] Extending SQL to nested tables has been a well-studied topic in the database management field and recently it is featured in multiple "Big Data" databases that feature semistructured models.