ΑΥΤΟΜΑΤΗ ΚΑΤΑΣΚΕΥΗ ΣΕΙΡΩΝ OLAP ΕΡΩΤΗΣΕΩΝ ΜΕ ΣΧΟΛΙΑΣΜΟ ΣΕ ΚΕΙΜΕΝΟ ΚΑΙ ΗΧΟ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από τον

Δημήτριο Γκεσούλη

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Ιούνιος 2013

DEDICATION

This thesis is dedicated to my family for supporting me all the way since the beginning of my studies.

ACKNOWLEDGMENTS

CONTENTS

iv

LIST OF TABLES

LIST OF FIGURES

ΠΕΡΙΛΗΨΗ

Γκεσούλης Δημήτριος του Γεωργίου και της Μαρίνας. MSc, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ελλάδα. Ιούνιος, 2013. Αυτόματη κατασκευή σειρών OLAP ερωτήσεων με σχολιασμό σε κείμενο και ήχο.

Στην παρούσα διατριβή, εξετάζουμε πώς μπορούμε να εκμεταλλευτούμε την ύπαρξη ενός σχήματος άστρου(star schema), προκειμένου να απαντηθούν OLAP ερωτήματα των χρηστών με CineCube ταινίες. Η δική μας μέθοδος, που εφαρμόζεται σε ένα πραγματικό σύστημα, περιλαμβάνει τα παρακάτω βήματα. Ο χρήστης υποβάλλει ένα ερώτημα μέσω ενός υπάρχουν σχήματος άστρου. Λαμβάνοντας αυτό το ερώτημα ως είσοδο, το σύστημα αναπαράγει ένα σύνολο από ερωτήματα που συμπληρώνουν το περιεχόμενο των πληροφοριών του αρχικού ερωτήματος, και εκτελεί αυτά. Στη συνέχεια, το σύστημα οπτικοποιεί τα αποτελέσματα του κάθε ερωτήματος και συνοδεύει την παρουσίαση τους με κείμενο το οποίο σχολιάζει τα σημαντικά μέρη των αποτελεσμάτων. Επιπλέον, μέσω ενός συστήματος μετατροπής κειμένου σε ήχο, το σύστημα μας παράγει αυτόματα ήχου για το κείμενο που δημιουργούμε. Κάθε συνδυασμός της απεικόνισης, του κειμένου και του ήχου αποτελεί ουσιαστικά μία cube movie, το οποίο γίνεται μια παρουσίαση του PowerPoint και επιστρέφεται στον χρήστη.

Επιβλέπων Καθηγητής: Πάνος Βασιλειάδης

EXTENDED ABSTRACT IN ENGLISH

Gkesoulis, Dimitrios. MSc, Department of Computer Science and Engineering, University of Ioannina, Greece. June, 2013. Automatic construction of OLAP query sequences with text and audio commentaries.

In this thesis, we investigate how we can exploit the existence of a star schema in order to answer user OLAP queries with *CineCube* movies. Our method, implemented in an actual system, includes the following steps. The user submits a query over an underlying star schema. Taking this query as input, the system comes up with a set of queries complementing the information content of the original query, and executes them. Then, the system visualizes the query results and accompanies this presentation with a text commenting on the result highlights. Moreover, via a text-to-speech conversion the system automatically produces audio for the constructed text. Each combination of visualization, text and audio practically constitutes a cube movie, which is wrapped as a PowerPoint presentation and returned to the user.

Thesis Supervisor: Panos Vassiliadis

CHAPTER 1. INTRODUCTION

Can we answer user queries with movies? Why should query results be treated simply as sets of tuples returned by the DBMS as if they would be visualized in an orange CRT of the 70's? So far, database systems assume their work is done once results are produced, effectively prohibiting even well-educated end-users to work with them. Can we do something better?

In this paper, we make a first attempt towards showing that *it is possible to produce query results that are (a) properly visualized, (b) textually exploitable, i.e., enriched with an automatically extracted text that comments on the result, (c) vocally enriched, i.e., enriched with audio that allows the user not only to see, but also hear. Moreover, we provide an extensible method to accompany a query result with results of complementary queries that allow the qualitative assessment of its information content.* Interestingly, *a meaningful sequence of related queries that provide context and depth to the original query, "dressed" with the appropriate visualization and sound, ends up to be nothing else but a* <u>movie</u> *where cubes star.*

Assumptions. In this paper we make a realistic assumption that empowers us with the ability to address the challenge in a clear setting. We assume the existence of a star schema with clean, reconciled hierarchies of reference data; we also assume that the end users are interested in working with OLAP queries over these data. We exploit the star schema in order to generate complementary queries automatically.

The movie's parts and their extension. Much like movies, we organize our stories in acts, with each act including several episodes all serving the same purpose. Our method involves *two extensibility mechanisms, (i) one concerning the generation of*

complementary queries that contextualize the original result and give insight and (ii) another concerning the automatic identification of interesting information within the results of each query. We are further exploit the outcome of the latter mechanism, as it is the main means via which we accompany results with automatically generated text (which in turn, is then fed to text-to-speech conversion in order to generate audio).

Low technical barrier. An important goal of this paper is to demonstrate that *the technical barrier for someone who would be interested to conduct research on this problem is low*. Existing API's for the construction of PowerPoint presentations [APOI] and for text to speech conversion [MARY] allow us to produce a pptx programmatically: each query can have a slide where its result is neatly visualized; the slide's notes can contain the text explaining the result and the slide's audio can be produced via text-to-speech conversion.

Contribution & call to arms. The individual parts of the method are not the core contribution of the paper; however, it is their principled and extensible bundling in a single, extensible tool that creates a research opportunity and an actual contribution. The fundamental message carried from this paper is that it is feasible (and we have done it) to drastically change the way users interact with business intelligence tools via simple programmatic APIs. Moreover, we can systematically expand this research ground by plugging in more and more techniques both from existing and foreseeable research results in the areas of text commenting, query recommendation and data visualization.
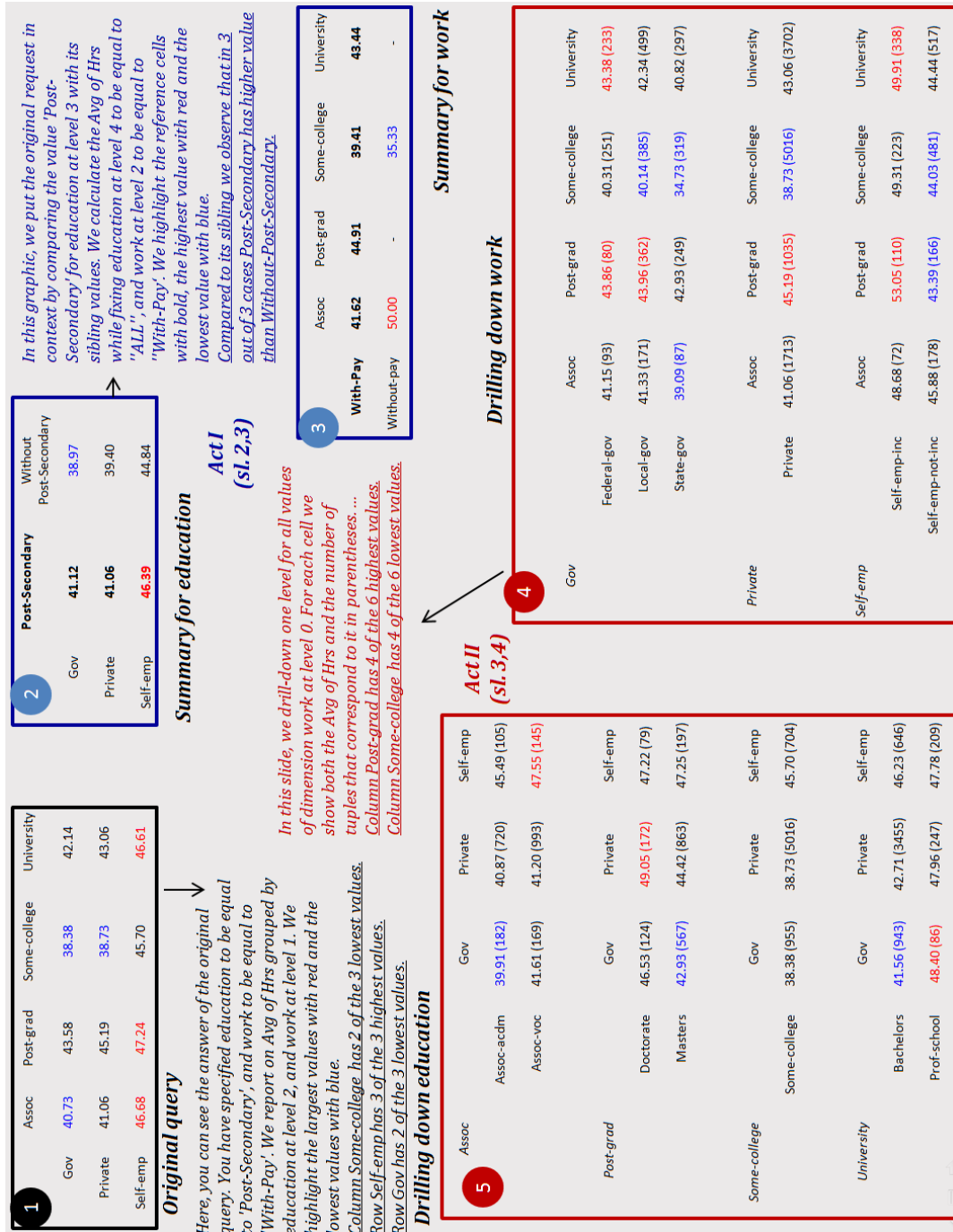
Original query

	Assoc	Post-grad	Some-college	University
Gov	40.73	43.58	38.38	42.14
Private	41.06	45.19	38.73	43.06
Self-emp	46.68	47.24	45.70	46.61

Here, you can see the answer of the original query. You have specified education to be equal to 'Post-Secondary', and work to be equal to 'With-Pay'. We report on Avg of Hrs grouped by education at level 2, and work at level 1. We highlight the largest values with red and the lowest values with blue.
Column Some-college has 2 of the 3 lowest values.
Row Self-emp has 3 of the 3 highest values.
Row Gov has 2 of the 3 lowest values.

Drilling down education

		Gov	Private	Self-emp
Assoc	Assoc-acdm	39.91 (182)	40.87 (720)	45.49 (105)
	Assoc-voc	41.61 (169)	41.20 (993)	47.55 (145)
Post-grad	Doctorate	46.53 (124)	49.05 (172)	47.22 (79)
	Masters	42.93 (567)	44.42 (863)	47.25 (197)
Some-college	Some-college	38.38 (955)	38.73 (5016)	45.70 (704)
University	Bachelors	41.56 (943)	42.71 (3455)	46.23 (646)
	Prof-school	48.40 (86)	47.96 (247)	47.78 (209)

Summary for education

Act I (sl. 2,3)

	Post-Secondary	Without Post-Secondary
Gov	41.12	38.97
Private	41.06	39.40
Self-emp	46.39	44.84

In this graphic, we put the original request in context by comparing the value 'Post-Secondary' for education at level 3 with its sibling values. We calculate the Avg of Hrs while fixing education at level 4 to be equal to "ALL", and work at level 2 to be equal to "With-Pay". We highlight the reference cells with bold, the highest value with red and the lowest value with blue.
Compared to its sibling we observe that in 3 out of 3 cases Post-Secondary has higher value than Without-Post-Secondary.

Summary for work

	Assoc	Post-grad	Some-college	University
With-Pay	41.62	44.91	39.41	43.44
Without-pay	50.00	-	35.33	-

Drilling down work

Act II (sl. 3,4)

		Assoc	Post-grad	Some-college	University
Gov	Federal-gov	41.15 (93)	43.86 (80)	40.31 (251)	43.38 (233)
	Local-gov	41.33 (171)	43.96 (362)	40.14 (385)	42.34 (499)
	State-gov	39.09 (87)	42.93 (249)	34.73 (319)	40.82 (297)
Private	Private	41.06 (1713)	45.19 (1035)	38.73 (5016)	43.06 (3702)
Self-emp	Self-emp-inc	48.68 (72)	53.05 (110)	49.31 (223)	49.91 (338)
	Self-emp-not-inc	45.88 (178)	43.39 (166)	44.03 (481)	44.44 (517)

In this slide, we drill-down one level for all values of dimension work at level 0. For each cell we show both the Avg of Hrs and the number of tuples that correspond to it in parentheses. ...
Column Post-grad has 4 of the 6 highest values.
Column Some-college has 4 of the 6 lowest values.

Figure 1.1. An excerpt of a CineCube story over the Adult data set

1.1. Constructing a CineCube Story

A really useful characteristic of cubes is that *dimensions* provide a *context* for facts [JePT10]. This is especially important if combined with the fact that dimension values come in *hierarchies*; therefore, every single fact can be simultaneously placed in multiple hierarchically structured contexts, providing thus the ability to analyze sets of cats from multiple perspectives. At the same time, hierarchies allow the comparison of their members with (a) *ancestors*, (b) *descendants* and (c) *siblings* (children of the same parent). Assume a basic, detailed cube C defined (a) over a set of dimensions $D = \{D_1, \ldots, D_n\}$ and (b) over a measure M. A query Q in our context exploits the multidimensionality of the cube space and can be considered as a quintuple $Q=(C,D,\Sigma,\Gamma,\gamma(M))$ where:

Σ is a conjunction of dimensional restrictions of the form $D_i.L_j = value_i$ — i.e., constraints that focus the context of the query to certain dimensional values.

Γ is a set of grouper dimensional level (practically comprising the GROUP BY attribute set in a SQL query), over which the information will ultimately be grouped.

$\gamma(M)$ is an aggregate function applied to the measure of the cube; again, we restrict ourselves to a single measure.

Given a query Q and its result $Q.RS$, we can make a short story by seeking for answers to the following questions:

0. A first assessment of the current state of affairs. Practically, this requirement refers to the execution of the original query.

1. Put the state in Context. Are the results of $\gamma(M)$ good? What does "good" mean in this case? Typically, we would expect to compare the result of the query Q to the results of similar queries over siblings of the values that appear in the filter list Σ.

2. Analysis of why things are this way. Given a certain cuboid that is the result of a query, we would like to provide some more insight on the presented results; one way to achieve this is to show the breakdown of the contributions of the detailed values to the overall, aggregate value. Practically speaking, this involves drilling-down for each of the involved groupers and presenting the analysis of the internal breakdown for each of the groupers.

Clearly, this set of complementary queries that a story comprises is extensible; existing and novel results in query recommendation (see Section 5) can be

progressively plugged in our method in order to produce more informative CineCube movies.

1.2. Running Example

To demonstrate our approach we use an example from the well known Adult (a.k.a census income) dataset referring to data from 1994 USA census. There are 7 dimensions (*Age, Native Country, Education*, *Occupation*, *Marital status*, *Work class*, and *Race*) in the data set and a single measure, *Hours per Week*. We will use a uniform terminology to refer to the dimensions' levels, (L_0, L_1, ..). Also, the ragged dimensions are complemented with values identical to their parent, to make them balanced and fit to the model of [VaSk00].

We start with an original query where the user has fixed *Education* to 'Post-Secondary' (at level L_3), and *Work* to 'With-Pay' (at level L_2) and requests the *Avg* of *HrsPerWeek* grouped by *Education* at level 2, and *Work* at level 1. We arrange the presentation of the result in columns (*Education*) and rows (*Work*). In Fig. 1, in slide with the indication ❶, one can also see the actual presentation as a 2D matrix, the visualization interventions (highlighting high and low values with color) and the text accompanying the visual presentation. The text is (a) part of the slide's notes (so that the user can reuse it) and (b) orally voiced as an audio file accompanying the slide. The slide's text is delivered via a set of *highlight extraction* methods that search the 2D matrix for prominent features (high and low values, rows or columns dominating some of these indicatory values, etc).

Once the originally query has been answered, we move on to put it in context. *Act I* of the CineCube movie, including slides ❷ and ❸ (dressed in blue color), performs the following analysis: since there is a selection condition with two atoms (*Education.L3*='Post-Secondary' and *Work.L2*='With-Pay'), we compare each of the defining values with its sibling. So, slide ❷ presents a comparison between the siblings of 'Post-Secondary' at level L_3 of *Education* (specifically, the single value 'W/O post secondary'). The analysis shows that in 3 out of 3 cases people with Post-Secondary education work more (see Fig. 1 at top right for the respective text). Similarly, in slide ❸, we relax the constraint on *Work* and compare the value 'With-

Pay' with its siblings at level L_2 of *Work* (again the single value 'W/O Pay). The results are inconclusive; for lack of space we omit the respective text from Fig. 1. In both these cases, we did two things: (a) we took a single atomic formula from the selection condition of the original query and replaced it by fixing the defining value to the parent of the original value, and (b) we put the grouping level to the level of the replaced value.
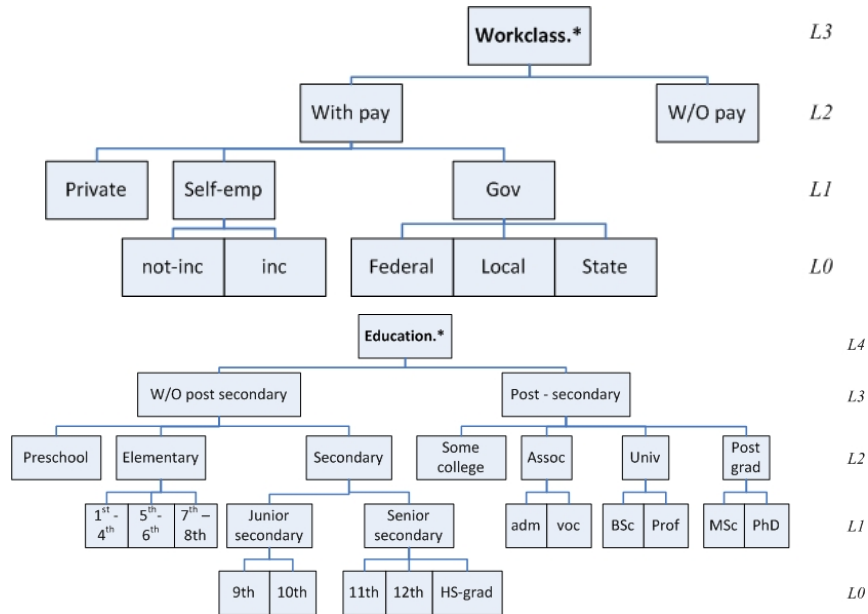

Figure 1.2. Dimensions Workclass and Education

Then, we detail the results of the original query in *Act II* of the CineCube movie. In slides ❹ and ❺ (dressed in red color) we present the results of drilling-down one level per grouper value. Observe slide ❹ as an example (slide ❺ is similar): for each of the values in the rows of the original query (at level L_1 of dimension *Work*) we drill-down one level (at level L_0 that is) and group-by accordingly. For each aggregated cell of the result we also show the number of detailed tuples that correspond to it, in parentheses. The text is constructed similarly with the previous act and includes a discussion of trends for high and low values along columns and rows.
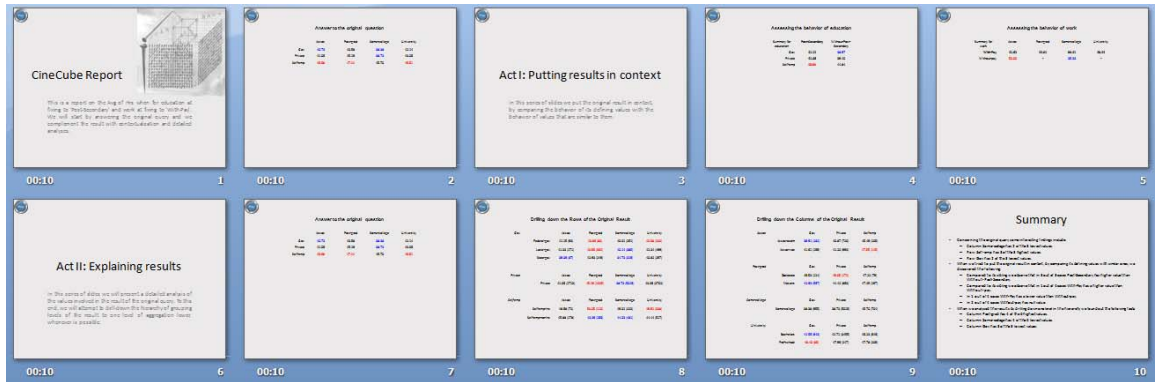
Figure 1.3. A snapshot of the internal structure of the CineCube movie

In the actual presentation that we generate, the set of information-carrying slides is also enriched with transition slides among the acts, explaining the intuition behind them as well as with a summary of the key highlights in the end (see Fig. 3).

1.3. List of Contributions

In this paper we provide the following contributions:

- Our main result is the introduction of a method that allows the generation of a CineCube movie, over an OLAP database, with a simple user query as starting point. Specifically, we can detail the individual contributions of our method as follows.

- We demonstrate how to complement the original query with additional queries that allow the contextualization and analysis of the original result. Moreover, we demonstrate an extensible method for searching for interesting findings in their results. Both these tasks are fully automated, by taking into advantage the value hierarchies of OLAP cubes. At the same time, they provide two points of extensibility of our method, both with respect to the complementary results and with respect to the highlight detection within these results.

- We demonstrate how to automate the generation of text describing the aforementioned highlight findings (by accompanying each type of highlight with a template text) and how to convert this text to audio (via publicly available text-to-speech conversion software).

 − Finally, we show that all the above can be packaged with small programming effort in a PowerPoint presentation, practically presenting a small movie to the user.

1.4. Roadmap

In Chapter 2 we explain the low technical barrier of the method and we discuss our method's internals. In Chapter 3 we show experimental results. In Chapter 4 we discuss related work. We conclude with a presentation of open issues in Chapter 5.

CHAPTER 2. AUTOMATING CINCECUBE CONSTRUCTION

2.1. Internal Structure of the CineCube Movie

2.2. Formal Background

2.3. Act I: Putting Things in Context – or "How good is the original cube compared to its siblings?"

2.4. Act II: Explaining Variation – or "Drilling into the breakdown of the original result"

2.5. Highlights and Text

2.6. Employed Technologies

2.7. Creation of CineCubes

2.1. Internal Structure of the CineCube Movie

A typical movie story is structured in approximately 3 acts: the first providing contextualization for the characters as well as the incident that sets the story on the move, the second where the protagonists and the rest of the roles build up their actions and reactions and the third where the resolution of the film is taking place. Each act is composed of sequences of scenes: each scene involves a change in the status of the plot (typically oscillating this status in order to keep viewers interested) and a sequence drives a subset of the plot to a major status update [McKe97].

We follow this traditional structure of a movie in our effort. We are clearly avoiding the temptation to automate a 90' movie; on the contrary, we wish to keep the story short and limited, as we anticipate users will explore several CineCube stories before gathering their results and discoveries from exploring the data. We organize *Acts* in *Episodes*: each episode practically corresponds to a pptx slide (although, we can

envision extensions to other formats -- e.g., it could be a section in a document). This *result-based structure* of the CineCube movie is accompanied by a *procedural-based structure*, with a set of classes that actually get the job done. Here, we introduce the *two extensibility mechanisms* that allow our method to be extensible to all sorts of algorithms for extra results and discoveries. *There are two "dimensions" of extensibility: (i) what kind of query results (episodes) we collect from the database, and, (ii) how we automatically discover important findings within these results.*
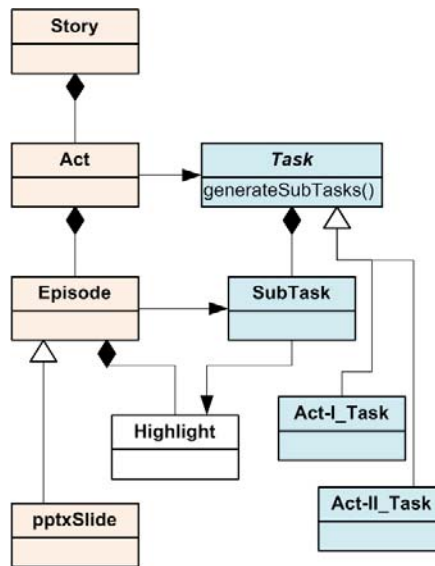

Figure 2.1. Extensibility mechanism for CineCubes

The first extensibility mechanism concerns the generation of queries (and slides) within each *Act*. The abstract class *Task* is the generator of the queries of each *Act*: therefore, we materialize it differently for each kind of *Act* (here we have two such materializations, for Act I and Act II). The crux of the approach is that each episode comes with (typically one, but sometimes more) queries in its background; therefore, each Act generates *SubTask*s, with each *Subtask* carrying and being responsible for the execution of a query that gathers the data (that are ultimately visualized in the main part of the slide). An *Episode* can have several *SubTask*s to compute its contents. Since each *SubTask* carries its own query depending on the *Act/Task,* the above mechanism is extensible by appropriately constructing the method *generateSubTasks()* for each materialization of *Act*.

The second extensibility mechanism concerns the determination of key findings, or *Highlight*s within each *Episode*. We fundamentally consider the presentation of results as a 2D matrix on the screen[1]; to this end, we have structured several methods that scan a 2D matrix and isolate interesting cells (top-k max or top-k min values, domination of a class of values by a column or row, etc). Class *Highlight* is a point of extensibility where methods for result extraction can be added to search for more results within the answer of a query.

There are several other classes that accompany the above core of the method which are omitted from this discussion for lack of space. These classes concern the management of cubes and their relationship with a relational database, the construction of the text, the derivation of the audio for the constructed text and so on.

2.2. Formal Background

Our method operates on top of a simple hypercube model for OLAP, expressed via a star schema in terms of relational representation. Hypercubes, commonly referred to as *cubes* – a term that we will adopt henceforth -- are very popular with end-users due to their simplicity and usability. A cube is a structured group of cells, each defined with respect to fixed set of *dimensions* and containing measurable quantities, or *measures*. The dimensions act as coordinates for the cell and the measures as contents; for example, a cell must state that with the respect to the coordinates `[City=Athens, Year=2013]` we have data for the following measured quantities `[AmtSales=10, Revenue=100]`. A cube organizes its cells along specific dimensions (here: Geography and Time) offering thus a multidimensional view of the data to the user. Then, the user can perform statistical analyses of the data by focusing on specific subsets of the cube and aggregating data at various level of detail. Each dimension offers a *hierarchy* of aggregation levels, or *levels*, constructed via

---

[1] Of course, other forms of visualization can *accompany* the result; however, it is our conviction that *the actual data should definitely be part of the answer* [Tuft97].

relationships which we call ancestor relationships. Each level is more detailed than its ancestors (here: the dimension Time has a hierarchy Year, Month, Day, with Year being an ancestor of both Month and Day and Month being an ancestor of Day). The representation of a cube along with its dimensions in a relational database is typically performed via a star schema that include (a) fact tables, referring to cubes at the lowest level of detail and (b) dimension tables, storing the hierarchies of the dimension values and levels.

In a nutshell, the logical layer involves (a) *dimensions* defined as lattices of dimension *levels*, (b) *ancestor functions* (in the form of $\text{anc}^{L_2;L_1}$) mapping values between related levels of a dimension, (c) *detailed data sets*, practically modeling fact tables at the lowest granule of information for all their dimensions and (d) *cubes*, defined as aggregations over detailed data sets.

Formally, we strictly follow the logical cube model of [VaSk00], accurately summarized in [Man+05] as follows:

Four countable pairwise disjoint infinite sets exist: a set of *level names* (or simply *levels*) $\mathsf{U}_{\mathcal{L}}$, a set of *measure* names (or simply *measures*) $\mathsf{U}_{\mathcal{M}}$, a set of *dimension* names (or simply *dimensions*) $\mathsf{U}_{\mathcal{D}}$ and a set of *cube* names (or simply *cubes*) $\mathsf{U}_{\mathcal{C}}$. The set of *attributes* U is defined as $\mathsf{U}=\mathsf{U}_{\mathcal{L}}\cup\mathsf{U}_{\mathcal{M}}$. For each $A\in\mathsf{U}_{\mathcal{L}}$, we define a countable totally ordered set $\text{dom}(A)$, the domain of A, which is isomorphic to the integers. Similarly, for each $A\in\mathsf{U}_{\mathcal{M}}$, we define an infinite set $\text{dom}(A)$, the domain of A, which is isomorphic to the real numbers. We can impose the usual comparison operators to all the values participating to totally ordered domains $\{<, >, \leq, \geq\}$.

A *dimension* D is a lattice $(\mathsf{L}, \prec)$ such that:

- $\mathsf{L}=(\mathsf{L_1},…,\mathsf{L_n})$, is a finite subset of $\mathsf{U}_{\mathcal{L}}$.
- $\text{dom}(L_i)\cap\text{dom}(L_j)= \varnothing$ for every i≠j.
    - $\prec$ is a partial order defined among the levels of L.
- The highest level of the hierarchy is the level $\mathsf{D.ALL}$ with a domain of a single value, namely $\mathsf{'D.all'}$.

Each path in the dimension lattice, beginning from its upper bound and ending in its lower bound is called a *dimension path*.

A family of functions $\text{anc}^{L_2;L_1}$ is defined, satisfying the following conditions:

For each pair of levels L_1 and L_2 such that $L_1 \prec L_2$ the function $\text{anc}^{L_2;L_1}$ maps each element of $\text{dom}(L_1)$ to an element of $\text{dom}(L_2)$.

Given levels L_1, L_2 and L_3 such that $L_1 \prec L_2 \prec L_3$, the function $\text{anc}^{L_3;L_1}$ equals to the composition $\text{anc}^{L_2;L_1} \circ \text{anc}^{L_3;L_2}$. This implies that:

- $\text{anc}^{L_1;L_1}(x) = x$.

- if $y = \text{anc}^{L_2;L_1}(x)$ and $z = \text{anc}^{L_3;L_2}(y)$, then $z = \text{anc}^{L_3;L_1}(x)$.

- for each pair of levels L_1 and L_2 such that $L_1 \prec L_2$ the function $\text{anc}^{L_2;L_1}$ is monotone (preserves the ordering of values). In other words:

- $\forall x,y \in \text{dom}(L_1): x < y \Rightarrow \text{anc}^{L_2;L_1}(x) \leq \text{anc}^{L_2;L_1}(y), L_1 \prec L_2$

A *schema* S is a finite subset of U. Normally, we will represent a schema as divided in two parts: $S = [D_1.L_1, \ldots, D_n.L_n, A_1, \ldots, A_m]$, where:

- $(L_1, \ldots, L_n)$ are levels from a dimension set $D = (D_1, \ldots, D_n)$ and level L_i comes from dimension D_i, for $1 \leq i \leq n$.

- $(A_1, \ldots, A_m)$ are attributes, i.e. measures and levels.

A *detailed schema* S^0 is a schema whose levels are the lowest in the respective dimensions. When we refer to a level L as the *lowest* in the dimension, it means that there does not exist any other level L', such that $L' \prec L$.

A *tuple* t over a schema $S = [L_1, \ldots, L_n, A_1, \ldots, A_m]$ is a total and injective mapping from S to $\text{dom}(L_1) \times \ldots \times \text{dom}(L_n) \times \text{dom}(A_1) \times \ldots \times \text{dom}(A_m)$, such that $t[X] \in \text{dom}(X)$ for each $X \in S$.

A *data set* DS over a schema $S = [L_1, \ldots, L_n, A_1, \ldots, A_m]$ is a finite set of tuples over S such that:

- $\forall\ t_1, t_2 \in DS, t_1[L_1, \ldots, L_n] = t_2[L_1, \ldots, L_n] \Rightarrow t_1 = t_2$.

- for no strict subset $X \subset \{L_1, \ldots, L_n\}$, the previous also holds.

In other words, $A_1, \ldots, A_m$ are functionally dependent (in the relational sense) on levels $(L_1, \ldots, L_n)$ of schema S. A *detailed data set* DS^0 is a data set over a detailed schema S. A *star schema* (D, S^0) is a couple comprising a finite set of dimensions D and a detailed schema S^0 defined over (a subset of) these dimensions.

An atom is `true`, `false` (with obvious semantics) or an expression of the form $x \partial y$, where x and y can be one of the following: (a) a level L_1 (i.e. not a measure); (b) a value l; (c) an expression of the form $\text{anc}^{L_2;L_1}(L_1)$ where $L_1 \prec L_2$; (d) an expression of the form $\text{anc}^{L_2;L_1}(l)$ where $L_1 \prec L_2$ and $l \in \text{dom}(L_1)$. If x and y are levels then they should belong to isomorphic dimensions. ∂ is an operator from the set ($>$, $<$, $=$, $\geq$, $\leq$, $\neq$).

A *selection condition* ϕ is a formula involving atoms and the logical connectives $\wedge$, $\vee$ and $\neg$. A selection condition is always applied to a data set such that all the level names occurring in the selection condition – either in the form (a) or (c) – belong to the schema of the data set. Let `DS` be a data set over schema `S`. The expression $\phi(\text{DS})$ is a set of tuples **X** belonging to `DS` such that when, for all the occurrences of level names in ϕ, we substitute the respective level values of every $x \in$ **X**, the formula ϕ becomes true. A *detailed selection condition* ϕ^0 is a selection condition where all participating levels are the detailed levels of their dimensions.

A *primary cube* `c` (over the schema $[L_1, \ldots, L_n, M_1, \ldots, M_m]$), is an expression of the form:

`c=(DS`^{0}`,`φ`,[L`$_1$`,…,L`$_n$`,M`$_1$`,…,M`$_m$`],[agg`$_1$`(M`$^{0;1}$`),…,agg`$_m$`(M`$^{0;m}$`)]), where:`

- `DS`^{0} is a detailed data set over the schema $S=[L^{0;1}, \ldots, L^{0;n}, M^{0;1}, \ldots, M^{0;k}], m \leq k$.
- ϕ is a detailed selection condition.
- `M`$_1$`, …, M`$_m$ are measures.
- $L^{0;i}$ and `L`$_i$ are levels such that $L^{0;i} \prec L_i$, $1 \leq i \leq n$.
- $\text{agg}_i \in \{\text{sum}, \text{min}, \text{max}, \text{count}\}$, $1 \leq i \leq m$.

The semantics of a primary cube in terms of SQL over a star schema are:

```
SELECT L₁,…,Lₙ, agg₁(M⁰;¹),…,aggₘ(M⁰;ᵐ)
FROM DS⁰ INNER JOIN D₁ … INNER JOIN Dₙ
WHERE ɸ
GROUP BY L₁,…,Lₙ
```

The user can submit *cube queries* to the system. A cube query specifies (a) the (basic) cube over which it is imposed, (b) the selection condition that isolates the records that

qualify for further processing, (c) the aggregator levels, that determine the level of coarseness for the result, and (d) a list of aggregations over the measures of the underlying cube that accompany the aggregator levels in the final result.

Clearly, there is a variety of choices for the expressiveness of each of these constituents, and thus, the query class of a method is determined by the combination of these choices. In our approach, we make the following assumptions for the query class of the supported cube queries:

- – We work with cube queries that involve a single measure.
- – We assume strictly two aggregator levels for the result; this allows a straightforward tabular representation of the result in a 2D screen.
- – We assume that the selection condition is defined as the conjunction of a set of atomic formulae, *one per dimension*, each of which is of the form `L = v`, with `L` being a dimension level and `v` being a value in the domain of this level.

In the rest of our deliberations, we will assume that the users submit to the system *Cube Queries* that we denote as:

`q=(DS`^{0}`,φ`$_1$` ∧ … ∧ φ`$_k$`,[L`$_α$`,L`$_β$`],agg(M))`

The results of a cube query of this form can be visualized in tabular format with the values of L_a as rows and the values of L_b as columns. Expanding the method for more than two dimensions (via the typical nesting of dimensions in rows and columns) is part of future work. Also, although, there are several other ways that we can employ to visualize results, like for example scatter plots on a 2D space or bar charts with multiple data series, we would like to stress once again that any such visualization methods are *complementary* to the actual data.

2.3. Act I: Putting Things in Context – or "How good is the original cube compared to its siblings?"

In this subsection, we deal with the first of the acts. The main purpose of the first act is to provide a context for the original query. So, we compare the marginal aggregate results of the original query to the results of "sibling" queries that use "similar" values in their selection conditions (to be explained right next).

Method. We assume an original query and we want to compare its results with similar queries. We define a sibling query as a query with a single difference to the

original: instead of an atomic selection formula $L_i=v_i$, the sibling query contains a formula of the form `Lᵢ ∈ childen(parent(vᵢ))`.

Formally, given an original query

`q = (DS⁰,ϕ₁ ∧ … ϕₓ ∧ … ∧ ϕₖ,[Lₐ,Lᵦ],agg(M)), ϕᵢ:Lᵢ=vᵢ, i=1,...,k`

a new query q^s is a *sibling query* if is of the form

`qˢ = (DS⁰,ϕ₁ ∧ …ϕ*;ˣ∧ … ∧ ϕₖ,[Lₐ,Lᵦ],agg(M)), ϕᵢ:Lᵢ=vᵢ, i=1,...,x-1,x+1,...,k,ϕ*;ˣ:Lₓ₊₁=ancᴸˣ⁺¹;ᴸˣ(v)`

Naturally, if `q` originally has `k` atomic selections, it also has `k` sibling queries.

To compare the results of the original query to the ones of its siblings, one would need to lay out all the k sibling queries on the same screen and visually inspect their differences. This becomes too hard to exploit as k increases – in fact, even with a very small k (e.g., k=2) it can be too hard to be able to visually compare the results. So we, need to resort to auxiliary comparisons that provide the context needed. To this end, we introduce two *marginal sibling queries*, one for each aggregator. Each time, we keep one of the two aggregators, and the other becomes L_x. If we combine this with the fact that the new selection condition $\phi^{*;x}$ restricts L_x to the siblings of the original value v, then the resulting 2D matrix has one of the original aggregators in one of its two dimensions and the siblings of v on the other. This way, the marginal values of the original query on one of the two aggregators are compared to the respective marginal values of the siblings.

Formally, given an original query

`q = (DS⁰,ϕ₁ ∧ … ϕₓ ∧ … ∧ ϕₖ,[Lₐ,Lᵦ],agg(M)), ϕᵢ:Lᵢ=vᵢ, i=1,...,k`

its two *marginal sibling queries* are

`qˢ;ᵃ = (DS⁰,ϕ₁ ∧ …ϕ*;ˣ∧ … ∧ ϕₖ,[Lₐ,Lₓ],agg(M)), ϕᵢ:Lᵢ=vᵢ,`
`i=1,...,x-1,x+1,...,k,ϕ*;ˣ:Lₓ₊₁=ancᴸˣ⁺¹;ᴸˣ(v)`

`qˢ;ᵇ = (DS⁰,ϕ₁ ∧ …ϕ*;ˣ∧ … ∧ ϕₖ,[Lₓ,Lᵦ],agg(M)), ϕᵢ:Lᵢ=vᵢ,`
`i=1,...,x-1,x+1,...,k,ϕ*;ˣ:Lₓ₊₁=ancᴸˣ⁺¹;ᴸˣ(v)`

Example. The original query is expressed as:

`q=(DS⁰,W.L₂='With-Pay'    ∧    E.L₃='Post-Sec',    [W.L₁,E.L₂],`
`avg(Hrs)),`

In the reference example, slides ❷ and ❸ involve the two marginal subqueries – see for example the former with the selection set to `parent('With-Pay')` and the grouping to the level of `'With-Pay'`(i.e., L_3):

q^2=(DS^0,W.L$_2$='With-Pay' $\wedge$ E.L$_4$='ALL', [W.L$_1$,E.L$_3$], avg(Hrs))

2.4. Act II: Explaining Variation – or "Drilling into the breakdown of the original result"

The purpose of Act II is to help the user understand why the situation is as observed in the original query. In order to shed some more light to what is happening, we drill in the details of the cells of the original result in order to inspect the internals of the aggregated measures of the original query.

Assume a cube query

q = (DS^0,ϕ_1 $\wedge$ … $\wedge$ ϕ_k,[L$_\alpha$,L$_\beta$],agg(M)), ϕ_i:L$_i$=v$_i$, i=1,...,k

and its result, visualized as a 2D matrix. Then, each selected cell is characterized by the following cube query:

q^c = (DS^0,ϕ_1 $\wedge$ … $\wedge$ ϕ_k $\wedge$ ϕ_c,[L$_\alpha$,L$_\beta$],agg(M)), ϕ_i:L$_i$=v$_i$, i=1,...,k,

ϕ_c:L$_\alpha$=v$^{c;a}$ $\wedge$ L$_\beta$=v$^{c;b}$

For every selected cell c, we can generate two *explanatory drill in queries*, one per grouper level.

q$^{c;\alpha}$ = (DS^0, ϕ_1 $\wedge$ … $\wedge$ ϕ_k $\wedge$ ϕ_c,[L$_{\alpha-1}$],agg(M)),

q$^{c;\beta}$ = (DS^0, ϕ_1 $\wedge$ … $\wedge$ ϕ_k $\wedge$ ϕ_c,[L$_{\beta-1}$],agg(M))

Then, for each of the two grouper dimensions we create a slide. In each of these slides we have one query for each of the values that appear in the original result for this dimension.

Example. Observe slide ❹ where we drill-down for values Gov, Private and Self-emp via the explanatory drill in queries for dimension *Work*.

q^{gov}=(DS^0,W.L$_2$='With-Pay' $\wedge$ W.L$_1$='Gov' $\wedge$ E.L$_3$='Post-Sec', [W.L$_0$,E.L$_2$], avg(Hrs))

q^{prv}=(DS^0,W.L$_2$='With-Pay' $\wedge$ W.L$_1$='Private'$\wedge$ E.L$_3$='Post-Sec', [W.L$_0$,E.L$_2$], avg(Hrs))

$q^{s\text{-}e}=(DS^0,W.L_2=\text{'With-Pay'} \quad \wedge \quad W.L_1=\text{'s-e'} \quad \wedge \quad E.L_3=\text{'Post-Sec'}, [W.L_0,E.L_2], avg(Hrs))$

Observe that due to the fact that this is the special case where selection conditions involve grouper values at finer levels of detail, we have completely removed the atomic formula of the dimension that we drill-down ($W.L_2=\text{'With-Pay'}$).

2.5. Highlights and Text

As already mentioned, the extraction of highlights is orthogonal to the query that creates the results of a slide. Once the results of the query are computed and organized in a 2D matrix, we utilize a palette of highlight extraction methods that take a 2D matrix as input and produce important findings as output. This way, (a) we can reuse highlight extraction methods to all the query results, independently of the Act or the query that has been executed, and, (b) we can gracefully extend the palette of highlight extraction methods with more results. We have implemented a small number of highlight extraction methods for the moment that include the highlighting of the top and bottom quartile of values in a matrix, the absence of values from a row or column, the domination of a quartile by a row or a column (i.e., when all the values of a quartile appear in a certain row or column), the identification of min and max values, etc. Clearly, there is a vast area of enriching this palette (trend analysis, correlations, relative relationships of rows and columns, to name just a few); however, implementing the full spectrum of such techniques can be done with diligence as part of future work. We utilize a dedicated Highlight Manager class to extract Highlights.

Text is constructed by a Text Manager that customizes the text per Act, by plugging values to a template that comes with each act. Compare the following excerpt with the text of slide ❹ in Fig. 1.

In this slide, we drill-down one level for all values of dimension `<dim>` at level `<l>`. For each cell we show both the `<agg>` of `<measure>` and the number of tuples that correspond to it.

2.6. Employed Technologies

One of the major goals of this paper is to highlight how we can automatically construct a CineCube presentation that includes result visualization, text and audio. In this subsection, we explain the main technologies via which our PowerPoint presentations are programmatically constructed.

Apache POI [APOI] is a Java API that provides several libraries to create and modify Microsoft Word, PowerPoint and Excel files. MS Office files obey the Office Open XML standards (OOXML) and Microsoft's OLE 2 Compound Document format (OLE2). More specifically, XSLF is the Java implementation of the PowerPoint 2007 OOXML (.pptx) file format in POI.

The automatic manipulation of .pptx files is relatively simple for simple tasks. See the following excerpt for creating a file and a slide:

```
XMLSlideShow ss = new XMLSlideShow();
XSLFSlideMaster sm = ss.getSlideMasters()[0];
XSLFSlide sl= ss.createSlide
(sm.getLayout(SlideLayout.TITLE_AND_CONTENT));
XSLFTable t = sl.createTable();
t.addRow().addCell().setText("added a cell"); …
```

As we will discuss later, we automate the construction of text that characterizes each slide. We add the text for each slide that we create as a slide's note. At the same time, the existence of text can help us create a narrative as audio. We use the API provided by MARY [MARY], which is an open-source, multilingual Text-to-Speech Synthesis (TTS) platform written in Java and allows to generate one audio file per slide, simply by providing the notes of the slide as input to a method call.

```
MaryInterface m = new LocalMaryInterface();
m.setVoice("cmu-slt-hsmm");
AudioInputStream audio = m.generateAudio("Hello");
AudioSystem.write(audio,    audioFileFormat.Type.WAVE,    new
File("myWav.wav")); …
```

Naturally, there are several nuts and bolts to fine tune. *However, the main lesson learned here is that the packaging of the results of our method, one by one as slides in a presentation is attainable with neat programming facilities, already available in the Web*.

2.7. Creation of CineCubes

Having explained all the individual steps, we now move on to discuss the overall process for creating a CineCube movie. In its current configuration, a CineCube movie includes three kinds of acts: the *Introductory Act* (including the introductory slide), three *Operational Act*s including the act involving the original query and the two acts for the management of complementary queries, and a *Summary Act* with a summary slide with all the important highlights of the previous three acts.

Overall the method includes the following steps:

1. Construct Introductory Act

2. For all the Operational Acts, execute the *Construct Operational Act* algorithm that calculates the Act's contents (result visualization, highlights, text and audio)

3. Construct Summary Act in the end
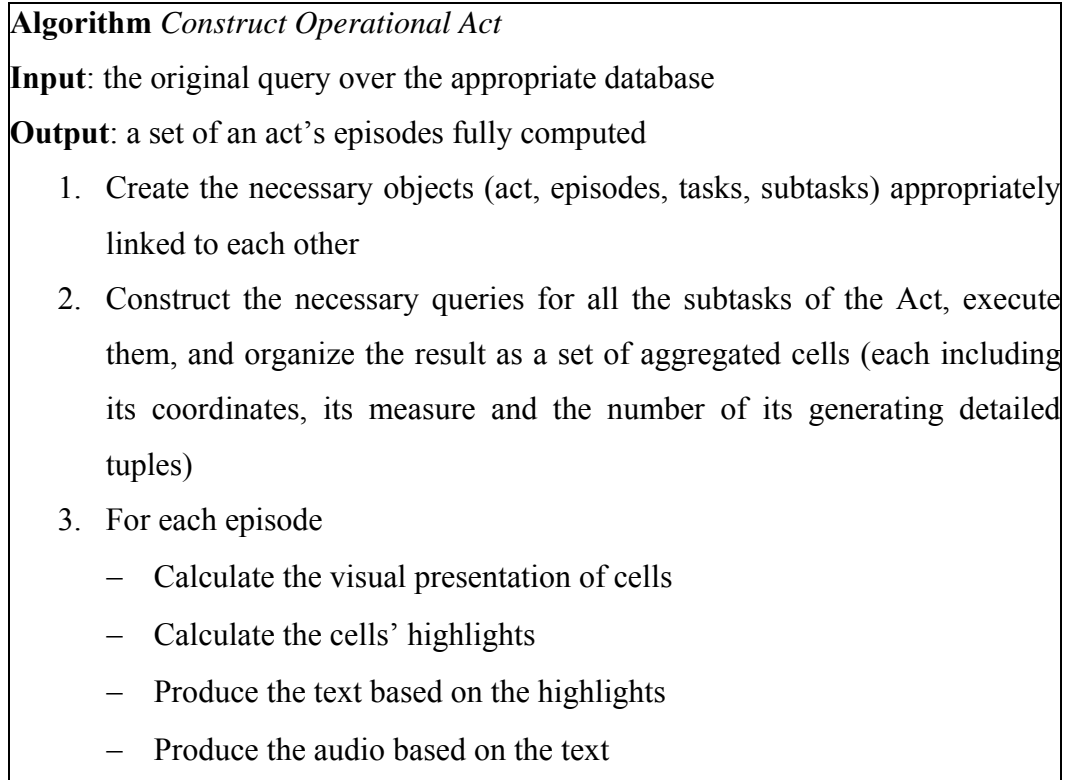
4. Wrap-up the Acts in a PowerPoint movie

```
Algorithm Construct Operational Act

Input: the original query over the appropriate database

Output: a set of an act's episodes fully computed

    1.  Create the necessary objects (act, episodes, tasks, subtasks) appropriately
        linked to each other

    2.  Construct the necessary queries for all the subtasks of the Act, execute
        them, and organize the result as a set of aggregated cells (each including
        its coordinates, its measure and the number of its generating detailed
        tuples)

    3.  For each episode
        –  Calculate the visual presentation of cells
        –  Calculate the cells' highlights
        –  Produce the text based on the highlights
        –  Produce the audio based on the text
```

Figure 2.2. Constructing an Operational Act

The computation of the contents and presentation of the *Operational Acts* is outlined in the Algorithm of Figure 5. Here, we would like to stress the extensibility aspect again: depending on the *Act* (and more specifically, on its operational *Task* counterpart), the queries of the subtasks are specialized per slide. Moreover, highlights, text and audio are produced via dedicated manager classes (not shown in Fig. 4 for lack of space).

The *Summary Act* is simply a slide with the text of the highlights copied to it, organized per act. However, the *Wrapping-up* Act introduces a few programmatic tasks worth mentioning here. Basically, for every episode we create a slide, with its title and contents (i.e., the 2D tables or the text, depending on the type of slide). This can be done straightforwardly with the programming facilities provided by the Apache POI. Unfortunately, though, POI does not support the management of notes, where we actually store the text of each slide and audio. To deliver a presentation in the form that we wish to have it, we proceed as follows: (i) we unzip the pptx in a temporary folder (remember: each MS Office file is actually a zipped folder with a rigid structure, within which, XML and media files are located in a principled

fashion); (ii) create appropriate files for the notes in the `ppt/notes/` folder, along with the necessary links that link them to their slide, (iii) do the same for audio at the `ppt/media` folder and (iv) zip the folder again to a .pptx file.

On the following pages we depict the result of our method of the query we use as example in this chapter. The result is given as a table where the left column is the produced slide and the right column has the notes of this slide.

Table 2.1 Result Slideshow for Example Query
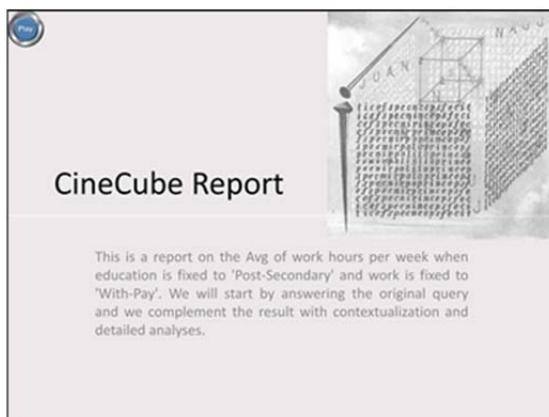
This is a report one Avg of work hours per week when education is fixed to 'Post- Secondary' and work is fixed to 'With-Pay'. We will start by answering the original query and we complement the result with contextualization and detailed analyses.

Here, you can see the answer of the original query. You have specified education to be equal to 'Post-Secondary', and work to be equal to 'With-Pay'. We report on Avg of work hours per week grouped by education at level 2, and work at level 1.

You can observe the results in this table. We highlight the largest values with red and the lowest values with blue color.

Column Some-college has 2 of the 3 lowest values. Row Self-emp has 3 of the 3 highest values.
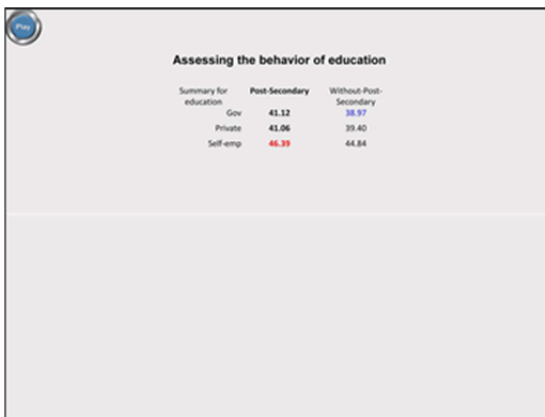
Row Gov has 2 of the 3 lowest values.

Act I: Putting results in context
In this series of slides we put the original result in context, by comparing the behavior of its defining values with the behavior of values that are similar to them.

In this graphic, we put the original request in context by comparing the value 'Post-Secondary' for education at level 3 with its sibling values. We highlight the reference cells with bold, the highest value with red and the lowest value with blue color. We calculate the Avg of work hours per week while fixing education at level 4 to be equal to
"ALL", and work at level 2 to be equal to "With-Pay".
Compared to its sibling we observe that in 3 out of 3 cases Post-Secondary has higher value than Without-Post-Secondary.

In this graphic, we put the original request in context by comparing the value 'With-Pay' for work at level 2 with its sibling values. We highlight the reference cells with bold, the highest value with red and the lowest value with blue color. We calculate the Avg of work hours per week while fixing education at level 3 to be equal to "Post-Secondary", and work at level 3 to be equal to "ALL".

Compared to its sibling we observe that in 1 out of 4 cases With-Pay has a higher value than Without-pay.

In 1 out of 4 cases With-Pay has a lower value than Without-pay. In 2 out of 4 cases Without-pay has null value.

Assessing the behavior of work

Summary for work	Assoc	Post-grad	Some-college	University
With-Pay	41.62	44.91	39.41	43.44
Without-pay	50.00	-	35.33	-

Act II: Explaining results

In this series of slides we will present a detailed analysis of the values involved in the result of the original query. To this end, we drill-down the hierarchy of grouping levels of the result to one level of aggregation lower, whenever this is possible.

Act II: Explaining results

In this series of slides we will present a detailed analysis of the values involved in the result of the original query. To this end, we drill-down the hierarchy of grouping levels of the result to one level of aggregation lower, whenever this is possible.

In this slide we remind you the result of the original query. Now we are going to explain the internal breakdown of this result by drilling down its grouper dimensions. In the first of the following two slides we will drill-in dimension work at level 1. Then we will drill-in dimension education at level 2.

Answer to the original question

	Assoc	Post-grad	Some-college	University
Gov	40.73	43.58	38.38	42.14
Private	41.06	45.19	38.73	43.06
Self-emp	46.68	47.24	45.70	46.61

In this slide, we expand dimension work by drilling down from level 1 to level 0. For each cell we show both the Avg of work hours per week and the number of tuples that correspond to it in parentheses. We highlight the 6 lowest values in blue and the 6 largest in red color.

Some interesting findings include:

Column Post-grad has 4 of the 6 highest values.

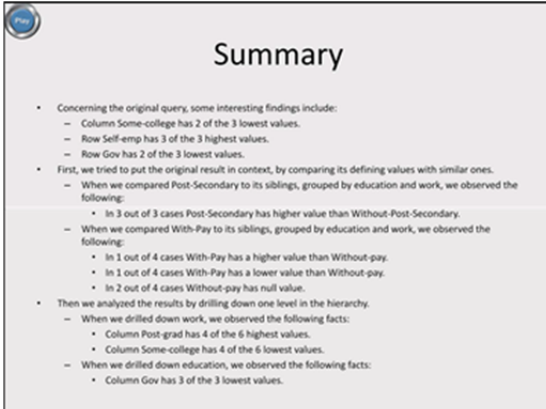Column Some-college has 4 of the 6 lowest values.

In this slide, we expand dimension education by drilling down from level 2 to level 1. For each cell we show both the Avg of work hours per week and the number of tuples that correspond to it in parentheses. We highlight the 3 lowest values in blue and the 3 largest in red color.

Some interesting findings include:

Column Gov has 3 of the 3 lowest values.

In this slide we summarize our findings.

Concerning the original query, some interesting findings include:

Column Some-college has 2 of the 3 lowest values.

Row Self-emp has 3 of the 3 highest values.

Row Gov has 2 of the 3 lowest values.

First, we tried to put the original result in context, by comparing its defining values with similar ones.

When we compared Post-Secondary to its siblings, grouped by education and work, we observed the following:

In 3 out of 3 cases Post-Secondary has higher value than Without-Post-Secondary. When we compared With-Pay to its siblings, grouped by education and work, we observed the following:

In 1 out of 4 cases With-Pay has a higher value than Without-pay. In 1 out of 4 cases With-Pay has a lower value than Without-pay. In 2 out of 4 cases Without-pay has null value.

Then we analyzed the results by drilling down one level in the hierarchy. When we drilled down work, we observed the following facts:

Column Post-grad has 4 of the 6 highest values. Column Some-college has 4 of the 6 lowest values.

When we drilled down education, we observed the following facts:

Column Gov has 3 of the 3 lowest values.

CHAPTER 3. EXPERIMENTS

3.1. Experimental Setup

We have experimented with the Adult (a.k.a census income) dataset referring to data from 1994 USA census. The dataset in its cleansed version (after uncertain and NULL values are removed) comprises 30162 tuples of the 1994 USA census. There are 8 dimensions (Age, Native Country, Education, Occupation, Marital status, Work class, Gender, and Race) in the data set and a single measure, Hours per Week. The hierarchies for the fields *Education*, *Occupation*, *Marital status*, *Work class*, and *Race* are depicted in Figure 3.1-Figure 3.5. Attribute *Age* is organized in years, 5-year intervals, 10-years intervals, 20-year intervals and *. Attributes *Gender* and *Salary* were not used due to their very small domain of values (Salary has only two values, higher or lower than 50K). The levels of hierarchy Native Country, except the level 0, are depicted in Figure 3.6.

We have experimented with the Adult data set by assessing the time needed for generating a presentation for different kinds of original queries. All experiments have taken place in a conventional PC running Windows 7 over an Intel Core Duo CPU at 2.50GHz, and with 3GB main memory.
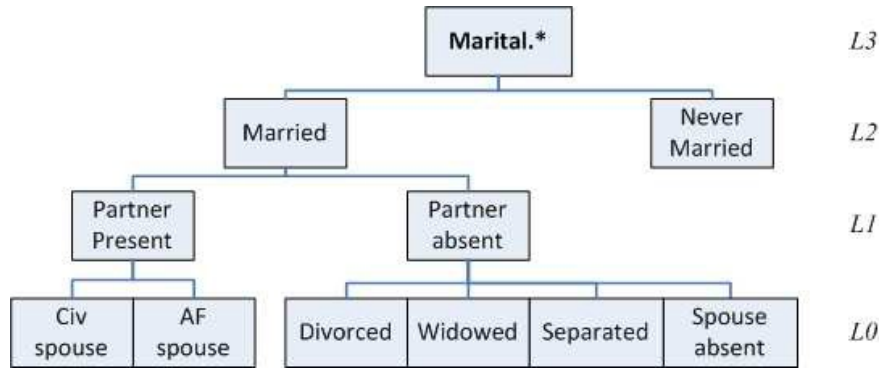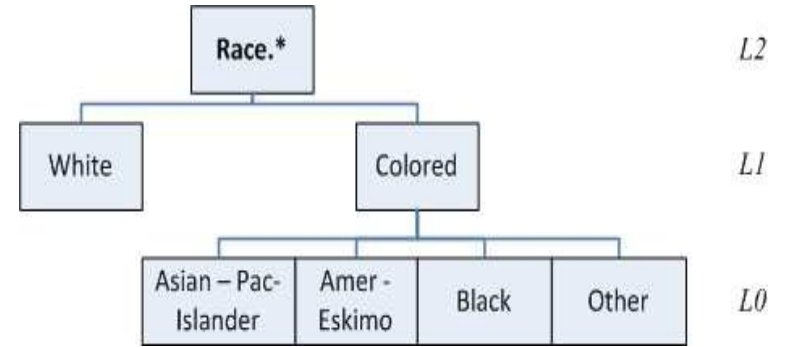
Figure 3.1 The hierarchy for the QI dimension *Marital Status*
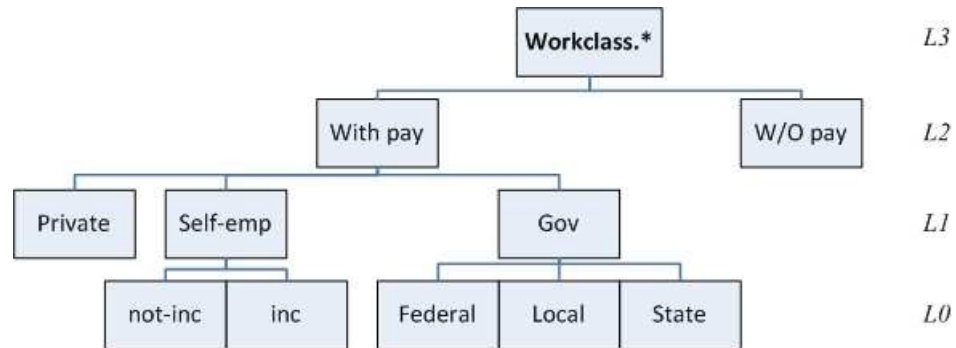

Figure 3.2 The hierarchy for the QI dimension Race

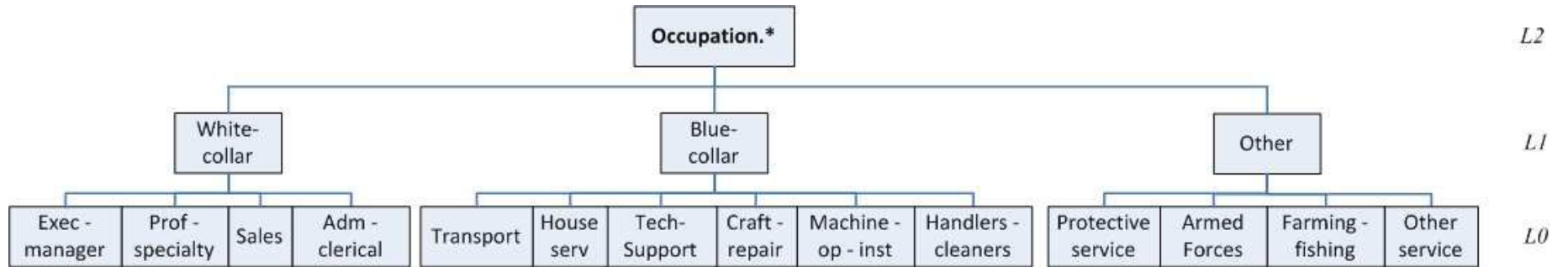Figure 3.3 The hierarchy for the QI dimension Work class

Figure 3.4 The hierarchy for the QI dimension *Occupations*
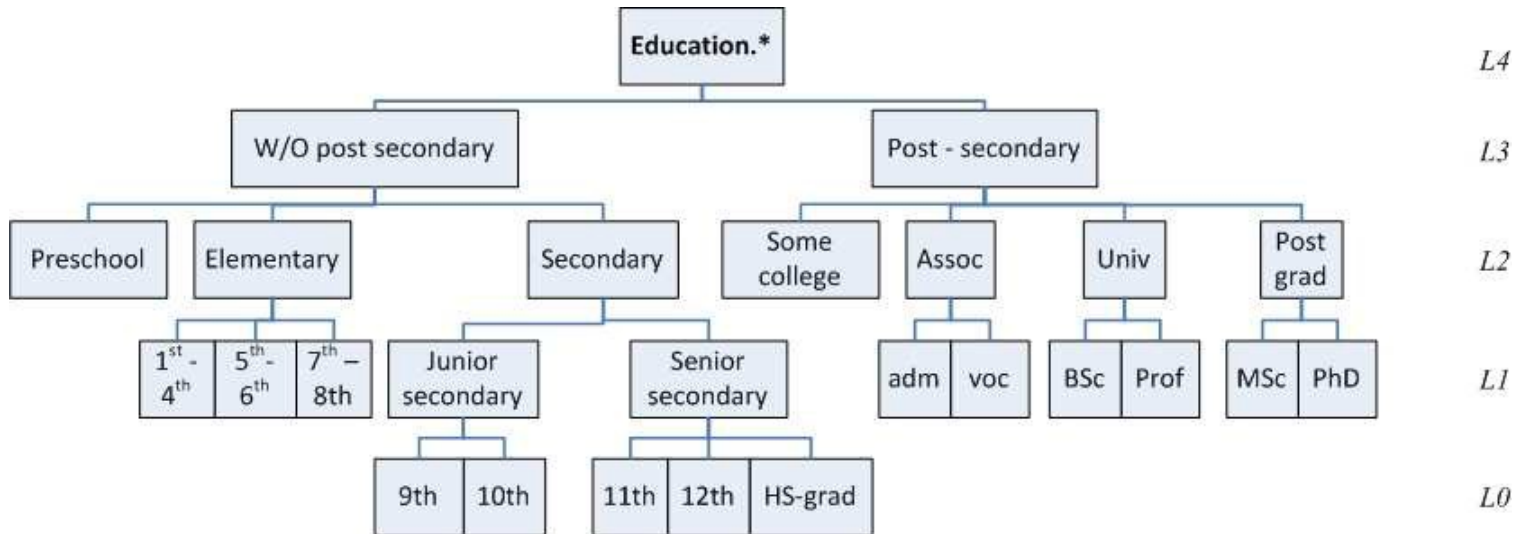
Figure 3.5 The hierarchy for the QI dimension *Education*
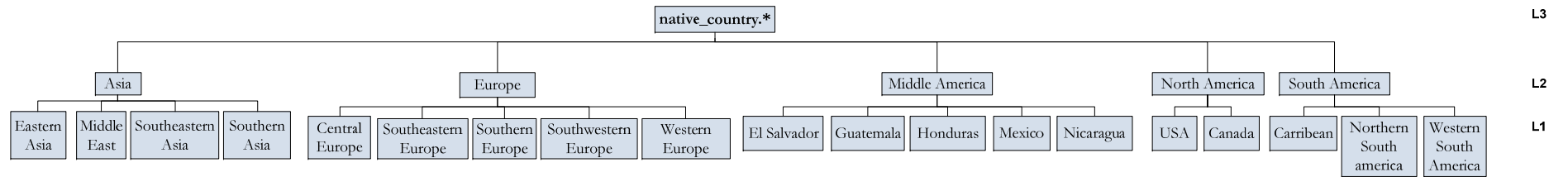
Figure 3.6 The hierarchy for the QI dimension Native Country

3.2. Detailed Findings

We have measured the time needed to perform each part of the method. For each slide, we have measured the time spent to perform the following tasks:

i. *Produce Cube Query*: in this part we count the time we needed to produce a Cube Query from the original query.

ii. *Produce SQL Query*: in this part we count the time needed to convert a Cube Query to SQL query.

iii. *Execute SQL Query:* in this part we measure the time to perform the query to the database and take the result back.

iv. *Tabular Creation,* in this part we measure the time needed to format the result of (iii) to a pivot table.

v. *Highlight Creation,* in this part we count the time needed to calculate the highlights on pivot table (such as row domination, largest values etc).

vi. *Color Table Creation*: in this part we count the time needed to add color to each cell of pivot table.

vii. *Combine Slide*: this action is performed only on Act II. It helps us to put in one slide more than one pivot table. We measure the time which took this action to complete.

viii. *Text Creation:* in this part we calculate the time needed to produce the slide's text from the calculated highlights.

ix. *Audio Creation:* in this part we calculate the time needed the MARYTTS to create the audio file from text which we had produce at previous step.

x. *Put in PPTX*: in this part we count the time needed to wrap up the above to a slideshow presentation.

In our analysis, we can group the three first parts to *Result Generation* and the four next grouped to *Highlight Generation & Visualization* as shown in the Tables 3.1-3.4. In these tables we can observe the time in milliseconds which takes to construct each part per slide when we have two, three, four or five atomic selection conditions in the WHERE clause of original query. Also, observe that the number of slides in Act I is increasing as the number of atomic selections in the WHERE clause increases.

Instead, the slides of Act II remain in all cases constant, which is consistent to that we described in Chapter 2.

Table 3.1 Time Breakdown (msec) for the Method's Parts when we Have 2 Atomic Selection in Where Clause

#slide	Result Generation			Highlight Generation & Visualization				Text Creation	Audio Creation	Put in PPTX
	Produce Cube Query	Produce SQL	Execution SQL Query	Tabular Creation	Highlight Creation	Color Table Creation	Combine Slide			
1								0.15	3746.10	119.72
2		0.10	140.27	0.42	0.38	0.10		0.13	7810.45	62.79
3	0.01		0.56					0.01	3117.66	9.81
4	0.01	0.04	167.56	0.15	0.21	0.05		0.17	8079.25	18.46
5	0.01	0.04	136.43	0.09	0.16	0.07		0.15	9655.91	29.87
6	0.01		0.55					0.01	4174.00	17.79
7		0.10	140.27	0.21	0.27	0.07		0.10	4384.05	24.79
8	0.09	0.12	141.01	0.35	0.60	0.15	0.11	0.13	6476.65	36.78
9	0.10	0.24	441.48	0.28	0.52	0.16	0.08	0.10	5626.54	34.54
10								0.38	18392.60	23.70
SUM	0.24	0.63	1168.13	1.49	2.14	0.59	0.19	1.32	71463.21	378.25

ACT I { slides 3, 4, 5

ACT II { slides 6, 7, 8, 9

Table 3.2 Time Breakdown (msec) for the Method's Parts when we Have 3 Atomic Selection in Where Clause

#slide	Result Generation			Highlight Generation & Visualization				Text Creation	Audio Creation	Put in PPTX
	Produce Cube Query	Produce SQL	Execution SQL Query	Creation Tabular	Highlight Creation	Color Table Creation	Combine Slide			
1								0.06	4240.10	19.14
2		0.05	72.44	0.12	0.23	0.07		0.10	8352.19	20.25
3	0.01		0.55						3050.50	8.33
4	0.01	0.02	62.99	0.09	0.13	0.05		0.13	10134.32	17.20
5	0.02	0.02	45.98	0.04	0.09	0.03		0.12	6702.00	34.78
6	0.01	0.02	193.64	0.07	0.16	0.04		0.13	10597.90	24.30
7	0.02	0.03	156.20	0.11	0.24	0.04		0.13	11613.28	22.75
8	0.01		0.55						4076.01	9.67
9		0.05	72.44	0.12	0.24	0.06		0.08	4302.92	16.65
10	0.07	0.07	127.82	0.19	0.49	0.14	0.04	0.10	7477.84	37.25
11	0.10	0.09	148.21	0.18	0.39	0.17	0.05	0.09	6281.38	46.24
12								0.46	27805.81	29.34
SUM	0.24	0.35	880.82	0.93	1.98	0.60	0.09	1.42	104634.27	285.89

ACT I: slides 3–7
ACT II: slides 8–11

Table 3.3 Time Breakdown (msec) for the Method's Parts When we Have 4 Atomic Selection in Where Clause

#slide	Result Generation			Highlight Generation & Visualization				Text Creation	Audio Creation	Put in PPTX
	Produce Cube Query	Produce SQL	Execution SQL Query	Tabular Creation	Highlight Creation	Color Table Creation	Combine Slide			
1								0.11	4919.52	20.84
2		0.05	196.35	0.12	0.24	0.05		0.10	9037.19	87.04
3	0.01		0.56						3069.75	8.72
4	0.01	0.03	153.83	0.10	0.11	0.04		0.13	10572.53	15.96
5	0.02	0.03	150.55	0.06	0.12	0.03		0.14	10637.11	17.52
6	0.01	0.03	164.40	0.05	0.10	0.02		0.13	10118.57	15.82
7	0.01	0.03	148.05	0.04	0.07	0.02		0.12	7847.23	13.90
8	0.01	0.03	191.04	0.09	0.18	0.03		0.13	13676.05	20.66
9	0.02	0.03	190.89	0.11	0.25	0.04		0.14	13725.84	23.32
10	0.01		0.59					0.01	4077.62	8.82
11		0.05	196.35	0.14	0.19	0.06		0.07	4293.18	30.57
12	0.08	0.08	320.42	0.19	0.41	0.10	0.03	0.10	7138.29	104.50
13	0.11	0.11	550.12	0.18	0.34	0.11	0.04	0.09	6435.41	34.51
14								0.52	39455.92	50.57
SUM	0.27	0.49	2263.15	1.08	2.02	0.50	0.07	1.80	145004.20	452.74

ACT I { slides 3–9

ACT II { slides 10–13

Table 3.4 Time Breakdown (msec) for the Method's Parts When we Have 5 Atomic Selection in Where Clause

| #slide | Result Generation | | | Highlight Generation & Visualization | | | | Text Creation | Audio Creation | Put in PPTX |
	Produce Cube Query	Produce SQL	Execution SQL Query	Tabular Creation	Highlight Creation	Color Table Creation	Combine Slide			
1								0.08	5572.83	19.26
2		0.05	185.64	0.12	0.18	0.05		0.11	9391.24	20.72
3	0.01		0.55						3029.61	8.29
4	0.01	0.03	146.16	0.11	0.16	0.04		0.15	12057.95	18.40
5	0.02	0.04	144.97	0.09	0.19	0.03		0.15	12213.87	20.07
6	0.01	0.03	92.73	0.04	0.08	0.02		0.14	8272.26	14.02
7	0.01	0.03	109.92	0.05	0.11	0.02		0.15	11574.43	16.65
8	0.01	0.03	93.90	0.04	0.09	0.03		0.14	10984.57	16.30
9	0.02	0.04	91.95	0.06	0.12	0.02		0.15	11080.51	97.39
10	0.01	0.03	167.20	0.05	0.11	0.02		0.14	10078.00	17.12
11	0.02	0.04	134.58	0.06	0.14	0.02		0.20	10078.14	17.45
12	0.01		0.62						4149.39	9.10
13		0.05	185.64	0.12	0.18	0.05		0.10	4550.45	16.54
14	0.09	0.10	269.41	0.18	0.36	0.11	0.03	0.11	7033.72	98.90
15	0.10	0.13	339.49	0.17	0.32	0.12	0.04	0.09	6391.55	32.48
16								0.63	42750.07	37.86
SUM	0.31	0.60	1962.77	1.11	2.02	0.54	0.07	2.35	169208.59	460.55

ACT I: slides 3–11
ACT II: slides 12–15

3.3. Analysis of Results per Task

We have measured the time needed to perform each part of the method. We varied the number of atomic selection conditions within the WHERE clause and measure the time needed per step of the method (measured in millisecond). As the number of selection conditions rises, each time we have two extra slides at Act I (the number of slides of each try is depicted in parentheses at the header of Table 3.5). Clearly, the audio generation dominates the entire process, being several orders of magnitude larger than anything else and presenting a clear case for improvement. As the number of slides slowly increases, the number of texts generated slowly increases too. Concerning every other part of the process, we see that query generation and execution takes up two orders of magnitude more than the other two tasks; therefore, being prudent with the number of slides (and thus, executed queries) is also necessary – esp., if someone would decide to exclude audio generation from the process. A very interesting observation is also that, so far, both text creation and highlight extraction are extremely fast, and thus, provide the potential for enrichment with more algorithms that try to find interesting highlights and create representative textual descriptions for them.

Table 3.5 Time breakdown (msec) for the method's parts

	# atomic selections in WHERE clause			
	2 (10 sl.)	3 (12 sl.)	4 (14 sl.)	5 (16 sl.)
Result Generation	1169,00	881,40	2263,91	1963,68
Highlight Extraction & Visualization	4,41	3,60	3,67	3,74
Text Creation	1,32	1,42	1,80	2,35
Audio Creation	71463,21	104634,27	145004.20	169208.59
Put in PPTX	378.24	285.89	452.74	460.55

`

3.4. Analysis of Results per Act

We have measured the time needed to produce each Act of the story (measured in millisecond). For more details for the number of slides in each Act and the times per slide please refer to Table 3.1-3.4. As the number of selection conditions rises, each time we have two extra slides at Act I (the number of slides of each try is depicted in parentheses at the header of Table 3.6). Clearly, we can observe that the time of each Act is increasing as the number of atomic selection condition increases. Also, the construction of Act I in three of four cases takes more time than the construction of the others. In the case when we have two atomic selection conditions the construction of Act II takes about 90 msec more. In addition, the time of creation Act II practically stable between different numbers of atomic selection condition in WHERE clause.

Table 3.6 Time breakdown (msec) per Act

	# atomic selections in WHERE clause			
	2 (10 sl.)	3 (12 sl.)	4 (14 sl.)	5 (16 sl.)
Intro Act	3746.99	4240.22	4919.71	5572.97
Original Act	7955.17	8425.59	9234.47	9577.76
Act I	21160.78	42562.10	70653.22	90359.89
Act II	21250.44	22419.34	22819.94	22738.88
Summary Act	18393.10	27806.35	39456.52	42750.78

In the Figure 3.7, observe that as the number of slides increases (2 extra slides each time) Act I increases with significant rate and similarly for the Summary Act. Both these effects are due to the text and audio generation. Moreover, the increasing time of Act I can be explain better from the details data in Tables 3.1-3.4 where we can observe that the cost per slide of Act I for two selection conditions is about 11000 msec.
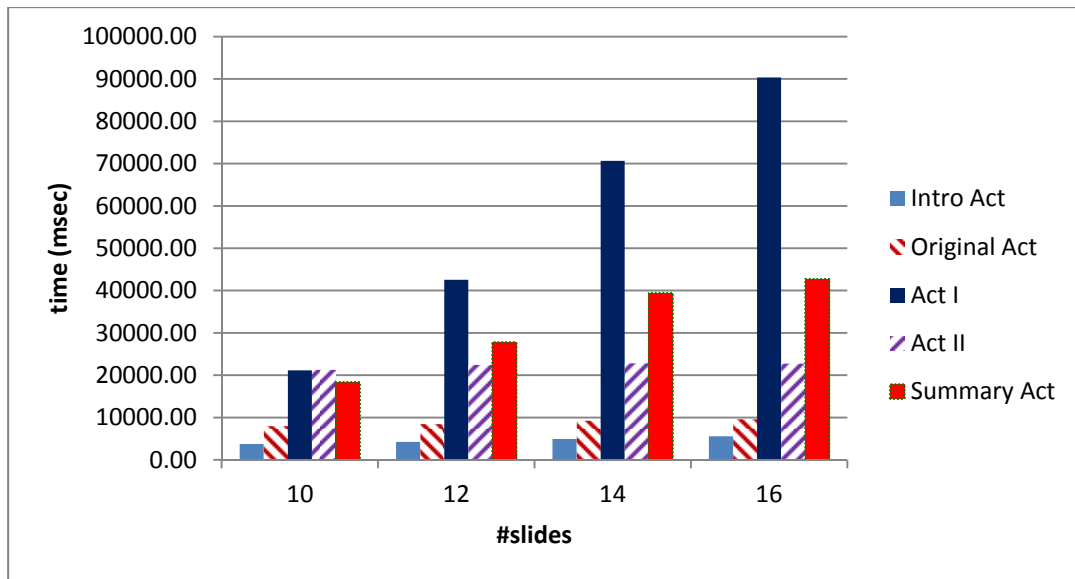
Figure 3.7 Bar chart of Time breakdown (msec) per Act

Also in Figure 3.7, we can observe that the Summary Act needed more time than the ACT II in three of four cases. This happens because the Summary Act, as we described in Chapter 2, has all the highlights of the story, which means all the text for these highlights. In addition, to complete this Act the text must be also converted to sound and once again the text to speech API dominates the time of our result.

CHAPTER 4. RELATED WORK

In this Chapter, we discuss related work around the topic of our discourse. Specifically, research pertaining to our work can be identified in the fields of query recommendation, advanced OLAP operators and text synthesis from query results. We present each of these categories in the following.

4.1. Query Recommendations

The first that relates to our work is the area of query recommendation. Roughly speaking, the general theme of this area revolves around the situation where the user has submitted a query to the system and the system suggests one or more related queries to the user as a guide that helps him continue his search. The suggestion can be based on the user's profile, history of queries, history of other users' queries, or other information. There is an excellent survey on the topic by [MaNe11]; thus, here he restrict ourselves to a handful of characteristic approaches and refer the interested reader to [MaNe11] for a broader discussion.

The query recommendations that are related to our work can be classified in two orthogonal taxonomies, already found in [MaNe11]. In terms of the *data management environment* within which query recommendation takes place, we can distinguish between works in the general field of databases and works in the specific field of

OLAP. In terms of the *means* employed for the recommendation of queries, we can discern methods exploiting profiles, methods exploiting query logs and hybrid methods.

4.2. Database-related efforts

In [SDP09], the authors propose the enrichment of the results of a query with extra tuples that maybe have potential interest to the user. The method is entitled YMAL ("You May Also Like"), and tries to find tuples in the underlying relational database on the grounds of a principled tuple-recommendation approach. One of the contribution of [SPD09] is that the authors suggest a classification of methods for recommendation: (a) current state based, (b) history based, and (c) based on external sources.

The current-state approach makes use of the current query result and schema in conjunction to data of database to produce the YMAL result. To implement this approach the authors suggest three kinds of analysis: (i) local, (ii) global and (iii) hybrid analysis. Local analysis involves finding patterns in the results of a query and searching the rest of the database in order to add to the original result extra tuples that abide by the discovered patterns. The Global approach searches the database to find values that are correlated to the values involved in the selection condition of the submitted query; the k most correlated of these values are selected and tuples that contain them are recommended to the user. To calculate relevant tuples, the history-based approach uses (i) the previously submitted queries of the user, and, (ii) similar sessions of other users that have similar behavior of the current user. The last of these approaches, involves external sources and does not search the local database for relevant tuples, but the web or another schema.

In [Cha+11], the authors propose a recommender system called QueRIE (Query Recommendations for Interactive data Exploration). The main goal of this recommender system is to help the common user, who is not familiar with SQL and database schemata, to find parts of database with useful or interesting information. To this end, the authors have implemented a system with the ability of tracking the querying behavior of user and generating personalized query recommendation. Their

system is built on a simple premise inspired by Web recommender systems: if a user A has similar querying behavior to user B, then they are likely interested in the same data. Hence, the queries of user B can serve as a guide for user A.

4.3. OLAP-related methods

In [Car+08], the authors describe a method to help user to explore OLAP data. The proposed method combines OLAP and data mining techniques to facilitate the process of the exploration of a data cube by identifying the most relevant dimensions to expand. The implementation of this task is performed in a step by step approach. In each step the most relevant dimensions from the current session of the user are identified and then, the system suggests to the user which one to explore first. The dimensions are of relatively simple structure with two levels only (ALL and detailed). The main idea behind the method is that each dimension takes a degree of interest. Each time the degree of interest is calculated by the amount of information revealed when including the details of this dimension in the grouping of the detailed data (remember that each dimension has only two levels; thus including it in the group by practically means that the dimension's detailed values split the grouping space with a factor equal to their number).

A different approach for suggesting an OLAP query to user is introduced in [GMNS11]. Unlike [Car+08], the authors of [GMNS11] use the query log of previous users to find similar queries which can give information to user that he may not know it is available. The main idea is to recommend to the user the discoveries detected in former sessions of other users that investigated the same unexpected data as the current session. To this end, the proposed method analyzes the query log to discover pairs of cells at various levels of detail for which the measure values differ significantly. In addition, the method analyzes the current query such that to detect if a particular pair of cells for which the measure values differ significantly can be related to what is discovered in the log.

4.4. Advanced OLAP operators

Apart from recommending queries to the users, related research has explored the possibility of providing users with explanations for the results they observe in an OLAP report. We distinguish the work of Sarawagi in a series of papers in VLDB and briefly summarize the results.

In [Sar99], the *DIFF* operator is described with the aim to help the analyst get a concise set of tuples explaining the reasons for drops or increases observed at an aggregated level. As input the operator receives two cells of a report that are different. As output the operator returns a set of tuples that best describe this difference. To achieve this result, the paper proposes a greedy and a dynamic-programming algorithm. The idea is that the operator keeps as fixed the common selections that characterize the originally selected cells (so, it is important that they do have some common selection conditions for the computation to make sense) and drills-down the levels of aggregation for the involved hierarchy that is produced by the combination of these common dimensions. The crux of the approach is that it computed the respective difference when the data are aggregated for any of the tuples in this multidimensional space. Every tuple in this multi-level space is compared to its "parent" tuple (in one level of aggregation higher) and, if selected, it is placed in the top-N results that will ultimately be displayed to the user. For a tuple to make it in the top-N it has to contribute a significant percentage of the difference of the original cells compared to the contribution of its father.

In [Sar00] a tool that helps users explore the multidimensional OLAP data using their prior knowledge of the data is described. This tool uses a profile that tracks down the areas of the cube that the user has visited in the past, and thus, it is aware of what the user already knows about the data. Then, the tool guides the user to unexplored data that he will find most informative. The author in [Sar00] describes a method that uses the classical Maximum Entropy principle and a profile per user to recommend to the user the parts of the cube which contain the most surprising values compared to what the user has already seen

In [SaSa01], the authors introduce the operator *RELAX* which helps the user of OLAP data to go from a detailed level of information to a more general one, in order to verify whether a pattern observed at the detailed level is also present at a more

summarized level. The operator reports in a single step a summary of all possible maximal generalizations along various roll-up paths of the observed sub-cube. Their goal is to report all possible consistent and maximal generalizations. The term consistent is meaning that all subset of dimensions that are examined also abide by the pattern. On the other hand, the term maximal means that there is no superset of dimensions that investigated can yield consistent generalizations. For the implementation of this operator the authors develop a two stage algorithm. In the first stage, their algorithm finds all possible maximal generalizations using aggregation queries. In the second stage, the algorithm uses the results of the first stage and finds summarized exceptions of the generalizations.

4.5. Text synthesis from query results

In [SKAI08], the authors propose a method to synthesize a textual answer in response to a query over a relational database. The authors employ a graph model with nodes being attributes and relations, edges being part-of relationships and join relationships and labels for relations, attributes and edges (labels are used to produce a text for a query's result). The method takes a query as input, computes its result and tries to produce a sentence for each of the tuples that appear in the result. This is derived by following specific graph navigation patterns, each of which produces a different type of text.

4.6. Relationship of our work with the state of the art

Concerning all the above works, our method comes with an extensible architecture that is especially constructed with a mindset of plugging more and more of them, both at the part where new queries can be added and in the part where new analyses can be performed over their results. Our Act II resembles the DIFF operator to a certain extent, in the sense that it tries to explain the reasons of the originally observed result. DIFF goes one step further, in providing maximal explanations by picking the most profitable rows. Although DIFF can be integrated in our tool, the emphasis so far has been in coming up with a prototype that can provide a reasonable CineCube movie; research results like DIFF can be integrated in the tool in subsequent tool extensions

and revisions. The same applies for all the other advanced OLAP operators. Concerning text synthesis, we avoid describing the result of a query row-by-row, as [SKAI08] does. On the contrary, we provide an extensible architecture where each highlight extraction method comes with a generic text to describe the detected highlights. Of course, improvements on the produced text are clearly part of future work.

CHAPTER 5. CONCLUSIONS

5.1 Summary

5.2 Open Issues

5.1. Summary

In this paper we introduced a method that allows the generation of a CineCube movie, over an OLAP database, with a simple user query as starting point. We have shown how to complement the original query with additional queries and we search for interesting findings in their results. We have also discussed how to automate the generation of text describing these findings and how to convert this text to audio. Moreover, we have shown that all the above can be packaged in a PowerPoint presentation, practically presenting a small movie to the user. Our experiments have shown that the audio generation is several orders of magnitude over the other tasks; within these tasks, query execution takes again the lion's share of the execution time.

5.2. Open Issues

Extensibility. Extensibility comes in two flavors in our method: (a) extensibility of generated results and (b) extensibility of highlight detection within these results. There are plenty of works in query recommendation (see discussion in Ch. "Related Work"), pattern verification [SaSa01], trend analysis, future prediction, to name only a few, that can be added to the tasks included in a tool. Of course, a journey starts with a first step, and we believe this first step is the main contribution for this article.

Be compendious; if not, at least be concise! The single most important challenge that the research problem of answer-with-a-movie faces is the *identification of what to exclude*. The problem is not to add more and more recommendations or findings (at the price of time expenses): this can be done both effectively (too many algorithms to consider) and efficiently (or, at least, tolerably in terms of user time). The main problem is that it is very hard to keep the story both interesting and informative and, at the same time, automate the discovery of highlights and findings. To address this task, a clearly important topic of research involves the automatic ranking and pruning of highlights.

Can *I* be the director? Interactively maybe? Personalization and interactivity are two clear paths for extending the approach mentioned here. The enrichment of the architecture with extra knowledge –e.g., user profiles or crowd-wisdom (via user logs)- and the possibility of intervening and semi-automatically guiding the query generation are topics with clear potential.

Efficiency. Scaling with data size and complexity, let along with user needs, *in user time,* is also necessary for an effort like this to succeed. Techniques like multi-query optimization have a good chance to succeed, esp., since we operate with a known workload of queries as well as under the divine simplicity of OLAP.

REFERENCES

[Ali+12]	**J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, E. Turricchia. Similarity Measures for OLAP Sessions. Accepted in Knowledge And Information Systems (KAIS), available at http://www.julien.aligon.fr/wp-content/uploads/2012/09/kais.pdf**
[APOI]	The Apache POI Project. See https://poi.apache.org/
[Car+08]	V. Cariou, J. Cubillé, C. Derquenne, S. Goutier, F.Guisnel, H. Klajnmic, 2008. Built-In Indicators to Discover Interesting Drill Paths in a Cube. DaWaK (Turin, Italy, 2008), pp. 33-44, DOI=http://dx.doi.org/10.1007/978-3-540-85836-2_4
[Cha+11]	G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, N. Polyzotis, J. Varman, 2011. The QueRIE system for Personalized Query Recommendations. IEEE Data Eng. Bull. 34,2 (2011), pp. 55-60
[GMNS11]	A. Giacometti, P. Marcel, E. Negre, A. Soulet, 2011. Query Recommendations for OLAP Discovery-Driven Analysis. IJDWM 7,2 (2011), 1-25 DOI= http://dx.doi.org/10.4018/jdwm.2011040101
[JePT10]	C. S. Jensen, T. B. Pedersen, C. Thomsen, 2010. Multidimensional Databases and Data Warehousing. Synthesis Lectures on Data Management, Morgan & Claypool Publishers
[Man+05]	A. Maniatis, P. Vassiliadis, S. Skiadopoulos, Y. Vassiliou, G. Mavrogonatos, I. Michalarias, 2005. A presentation model and non-traditional visualization for OLAP. IJDWM, 1,1 (2005), 1-36. DOI= http://dx.doi.org/10.4018/jdwm.2005010101
[MaNe11]	P. Marcel, E. Negre, 2011. A survey of query recommendation techniques for data warehouse exploration. EDA (Clermont-Ferrand, France, 2011), pp.

119-134

[MARY]   DFKI. The MARY Text-to-Speech System. See http://mary.dfki.de/

[McKe97]   R. McKee, Story: substance, structure, style and the principles of screenwriting. HarperKollins pubs. 1997.

[Sap99]   Carsten Sapia: On Modeling and Predicting Query Behavior in OLAP Systems. DMDW 1999:2

[Sar00]   Sunita Sarawagi: User-Adaptive Exploration of Multidimensional Data. VLDB 2000:307-316

[Sar99]   S. Sarawagi, 1999. Explaining Differences in Multidimensional Aggregates. VLDB (Edinburgh, Scotland, 1999), pp. 42-53

[SaSa01]   G. Sathe, S. Sarawagi, 2001. Intelligent Rollups in Multidimensional OLAP Data. VLDB (Roma, Italy 2001), pp.531-540

[SDP09]   K. Stefanidis, M. Drosou, E. Pitoura, 2009. "You May Also Like" Results in Relational Databases. PersDB (Lyon, France, 2009).

[SKAI08]   A. Simitsis, G. Koutrika, Y. Alexandrakis, Y.E. Ioannidis, 2008. Synthesizing structured text from logical database subsets. EDBT (Nantes, France, 2008) pp. 428-439, DOI=http://doi.acm.org/10.1145/ 1353343.1353396

[Tuft97]   E.R. Tufte, 1997. Visual Explanations. Graphics Press

[VaSk00]   P. Vassiliadis, S. Skiadopoulos, 2000. Modelling and Optimization Issues for Multidimensional Databases. CAiSE (Stokholm, Sweden, 2000), pp. 482-497, DOI=http://dx.doi.org/10.1007/3-540-45140-4_32

SHORT CV

Dimitrios Gkesoulis was born in 1987 and finished high school in 2004. He obtained his B.Sc. in Computer Science in 2009 from the computer Science Department of the University of Ioannina. He entered the Graduate Program of the same institution at 2010 under the supervisor of Panos Vassiliadis. His research interests lie in the area of database systems, with particular emphasis on query recommendation.