Annotation-Enabled Interpretation and Analysis of Time-Series Data

Niveditha Venugopal  
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the Computer Sciences Commons

Recommended Citation
https://doi.org/10.15760/etd.6592

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.
Annotation-Enabled Interpretation and Analysis of Time-Series Data

by

Niveditha Venugopal

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

Thesis Committee:
David Maier, Chair
Kristin Tufte
Elliot Gall

Portland State University
2018
ABSTRACT

As we continue to produce large amounts of time-series data, the need for data analysis is growing rapidly to help gain insights from this data. These insights form the foundation of data-driven decisions in various aspects of life. Data annotations are information about the data such as comments, errors and provenance, which provide context to the underlying data and aid in meaningful data analysis in domains such as scientific research, genomics and ECG analysis. Storing such annotations in the database along with the data makes them available to help with analysis of the data. In this thesis, I propose a user-friendly technique for Annotation-Enabled Analysis through which a user can employ annotations to help query and analyze data without having prior knowledge of the details of the database schema or any kind of database programming language. The proposed technique receives the request for analysis as a high-level specification, hiding the details of the schema, joins, etc., and parses it, validates the input and converts it into SQL. This SQL query can then be executed in a relational database and the result of the query returned to the user. I evaluate this technique by providing real-world data from a building-data platform containing data about Portland State University buildings such as room temperature, air volume and CO₂ level. This data is annotated with information such as class schedules, power outages and control modes (for example, day or night mode). I test my technique with three increasingly sophisticated levels of use cases drawn from this building science domain: (1) Retrieve data with include or exclude annotation selection (2) Correlate data with include or exclude annotation selection (3) Align
data based on include annotation selection to support aggregation over multiple periods. I evaluate the technique by performing two kinds of tests: (1) To validate correctness, I generate synthetic datasets for which I know the expected result of these annotation-enabled analyses and compare the expected results with the results generated from my technique (2) I evaluate the performance of the queries generated by this service with respect to execution time in the database by comparing them with alternative SQL translations that I developed.
DEDICATION

To my mother, father, husband, brother, in-laws,

grandmother, family, friends and Sherry.
ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my thesis advisor, Dr. David Maier, for his continual guidance and support from the ideation to the completion of this thesis. My meetings with Professor Maier have always been enlightening and have transformed the way I think about things. I am deeply honored for having had the opportunity to work with him on this research.

I would like to thank Dr. Elliot Gall for being an active collaborator in the BuDS research group and for his valuable feedback on my work. I would also like to thank Dr. Kristin Tufte for reviewing and providing feedback on my thesis.

I feel privileged for having been a part of the BuDS research group who believed in me and gave me a platform to acquire a wide range of database skills. I am very thankful that I got a lot of opportunities in the research group meetings to brainstorm ideas for my thesis.

Finally, I would like to thank my husband, family and friends who constantly encouraged me and urged me to do better.
# CONTENTS

ABSTRACT .................................................................................................................................................. i  
DEDICATION........................................................................................................................................... iii  
ACKNOWLEDGEMENTS...................................................................................................................... iv  
LIST OF TABLES .................................................................................................................................... vii  
LIST OF FIGURES ................................................................................................................................ viii  

1. INTRODUCTION .............................................................................................................................. 1  
   1.1 ANNOTATION-ENABLED QUERIES ........................................................................... 3  
   1.2 THESIS CONTRIBUTIONS .............................................................................................. 6  

2. BuDS PROJECT OVERVIEW ..................................................................................................... 11  
   2.1 BuDS: TECHNICAL OVERVIEW ................................................................................. 12  
   2.2 SCHEMA OVERVIEW ..................................................................................................... 16  
   2.3 ANNOTATION-ENABLED INTERPRETATION ..................................................... 21  

3. ANNOTATION-ENABLED ANALYSIS ................................................................................... 26  
   3.1 SQLConverter: A TECHNICAL OVERVIEW ........................................................... 26  
   3.2 CHECKER ........................................................................................................................... 31  
   3.3 ROUTER ............................................................................................................................. 32  
   3.4 SQL GENERATORS ......................................................................................................... 32  
      3.4.1 RETRIEVE OPERATION ...................................................................................... 32  
      3.4.1.1 SCOPE .................................................................................................................... 33  
      3.4.1.2 GENERATION ..................................................................................................... 34  
      3.4.1.3 SAMPLE RESULTS ............................................................................................. 37  
      3.4.2 CORRELATE OPERATION .................................................................................. 41  
      3.4.2.1 SCOPE .................................................................................................................... 41  
      3.4.2.2 GENERATION ..................................................................................................... 42  
      3.4.2.3 SAMPLE RESULTS ............................................................................................. 45  
      3.4.3 ALIGNED-AVERAGE OPERATION ................................................................... 49  
      3.4.3.1 SCOPE .................................................................................................................... 50  
      3.4.3.2 GENERATION ..................................................................................................... 52  
      3.4.3.3 SAMPLE RESULTS ............................................................................................. 56  
   3.5 ROLE OF AEA IN BuDS ................................................................................................. 60
4. EVALUATION ................................................................................................................................. 61
  4.1 EVALUATION OF ANNOTATION-ENABLED INTERPRETATION ........................................... 62
  4.2 VALIDATION OF THE RETRIEVE OPERATION ................................................................. 66
  4.3 VALIDATION OF THE CORRELATE OPERATION ............................................................ 72
  4.4 VALIDATION OF THE ALIGNED-AVERAGE OPERATION ............................................... 77
  4.5 PERFORMANCE EVALUATION ......................................................................................... 83
     4.5.1 PERFORMANCE OF RETRIEVE OPERATION ........................................................... 84
     4.5.2 PERFORMANCE OF THE CORRELATE OPERATION ............................................... 89
     4.5.3 PERFORMANCE OF ALIGNED-AVERAGE OPERATION .......................................... 93
5. RELATED WORK .................................................................................................................... 97
  5.1 ANNOTATIONS IN VISUALIZATION ............................................................................... 98
  5.2 ANNOTATIONS IN TIME-SERIES DATA ......................................................................... 101
  5.3 ANNOTATION PROPAGATION ....................................................................................... 105
  5.4 SUPPORT FOR ANNOTATIONS IN TIME-SERIES DATABASES .................................. 108
  5.5 ANNOTATIONS IN NON-TIME-SERIES DATA .............................................................. 110
  5.6 ANNOTATIONS IN SCIENTIFIC DATA ........................................................................... 111
  5.7 ENERGY USAGE ANALYSIS IN BUILDINGS ................................................................ 112
6. CONCLUSION AND FUTURE WORK ............................................................................. 115
  6.1 FUTURE WORK ............................................................................................................... 115
  6.2 ROLE in BuDS ............................................................................................................... 117
  6.3 CONCLUDING REMARKS .......................................................................................... 117
7. REFERENCES ..................................................................................................................... 120
LIST OF TABLES

Table 4.1: Query Alternatives for Retrieve operation with exclude annotation selection and their average runtimes .......................................................................................................................... 88
Table 4.2: Query Alternatives for Correlate operation and their corresponding average runtimes .................................................................................................................................. 92
Table 4.3: Query Alternatives for Aligned-Average operation and their corresponding average runtimes .......................................................................................................................... 96
LIST OF FIGURES

Figure 1.1: Plot of classroom EB365 room temperature, 1st day of Fall 2017............ 2
Figure 1.2: Plot of classroom EB365 room temperature with Class Schedule Annotation, 1st week Fall 2017 ................................................................. 4
Figure 1.3: SQL Query for average room temperature profile for room EB 356 when there were classes in the room, Fall 2017 ................................................................. 5
Figure 1.4: Plot for average room temperature profile for room EB 356 when there were classes in the room, Fall 2017 ................................................................. 5
Figure 1.5: Input Specification File for Aligned-Average of Room Temperature ....... 9
Figure 2.1: BuDS Architecture Diagram................................................................. 13
Figure 2.2: ETL Flowchart ................................................................................... 15
Figure 2.3: Schema Diagram for TEC Data.......................................................... 18
Figure 2.4: Schema Diagram for Annotation....................................................... 19
Figure 2.5(a): Power Outage Annotation for Engineering Building...................... 20
Figure 2.5(b): Class Schedule Annotation for room EB 365 ............................. 20
Figure 2.6: Weekly Average Air Volume Profile for EB 365 with Class Schedule Annotation, Fall 2017 ..................................................................................... 22
Figure 2.7: Jupyter Notebook for generating the weekly average profile of air volume for EB 365, Fall 2017 ................................................................. 23
Figure 2.8: Room Temperature for EB 365 .......................................................... 24
Figure 3.1: AEA Technical Overview ................................................................. 28
Figure 3.2(a): Example Annotation Mapping Configuration file........................ 29
Figure 3.2(b): Example Schema Configuration file........................................... 30
Figure 3.3: Input Specification for Retrieve Operation with Include Class Schedule Annotation Selection ................................................................. 35
Figure 3.4: Output SQL for Retrieve Operation with Include Class Schedule Annotation Selection ................................................................. 36
Figure 3.5: Output of Retrieve Operation of EB 365 Room Temperature with Include Class Schedule Annotation Selection, Fall 2017 ................................. 37
Figure 3.6(a): Input Specification File for Retrieve Operation with Exclude Class Schedule Annotation Selection ........................................................................................................................................... 38
Figure 3.6(b): Output SQL for Retrieve Operation with Exclude Class Schedule Annotation Selection ........................................................................................................................................... 39
Figure 3.7: Results for Retrieve Operation of EB 365 Room Temperature with Exclude Class Schedule Annotation Selection, Fall 2017 .............................................................................................................. 39
Figure 3.8: Plot for Include versus Exclude Retrieve Operation of EB 365 Room Temperature, Fall 2017 .............................................................................................................................................................................................................................................. 39
Figure 3.9: Input Specification File for Correlate Operator between OAT and EB 365 Room Temperature with Include Class Schedule Annotation Selection, Fall 2017 .................................................. 44
Figure 3.10: Output SQL for Correlate Operation between OAT and EB 365 Room Temperature with Include Class Schedule Annotation Selection .............................................................................................................. 45
Figure 3.11: Correlation between OAT and EB 365 Room Temperature with Include Class Schedule Annotation Selection, Fall 2017 .............................................................................................................................................................................................................................................. 46
Figure 3.12(a): Input Specification File for Correlate Operator between OAT and EB 365 Room Temperature with Exclude Class Schedule Annotation Selection, Fall 2017 .............................................................................................................. 47
Figure 3.12(b): Output SQL for Correlate Operator between OAT and EB 365 Room Temperature with Exclude Class Schedule Annotation Selection, Fall 2017 .............................................................................................................. 48
Figure 3.13: Correlation between OAT and EB 365 Room Temperature with Exclude Class Schedule Annotation Selection, Fall 2017 .............................................................................................................................................................................................................................................. 48
Figure 3.14(a): EB 365 Room Temperature for 25th September 2017, Monday with Class Schedule Annotation .............................................................................................................................................................................................................................................. 50
Figure 3.14(b): EB 365 Room Temperature for 26th September 2017, Tuesday with Class Schedule Annotation .............................................................................................................................................................................................................................................. 50
Figure 3.15: Input Specification File for Aligned-Average Operator for morning classes with Include Annotation Selection .............................................................................................................................................................................................................................................. 54
Figure 3.16: Output SQL for Aligned-Average operation of morning classes with Include Annotation Selection .............................................................................................................................................................................................................................................. 56
Figure 3.17: Output of Aligned-Average of morning classes with Include Annotation Selection, Fall 2017 ................................................................. 58
Figure 3.18: Plot of Aligned-Average of morning classes in EB 365 with Include Annotation Selection, Fall 2017 ................................................................. 59
Figure 3.19: Plot of Aligned-Average of evening classes in EB 365 with Include Annotation Selection, Fall 2017 ................................................................. 59
Figure 3.20: How AEA fits in the BuDS Architecture ................................................................. 60
Figure 4.1(a): Stack plot showing Outside Air Temperature, Room Temperature, Air Volume, CO2 Level, Return Air CO2 Level, Return Air Humidity, Return Air Temperature for room EB 375 and Air Handling Unit corresponding to this room annotated with night control mode ......................................................................................... 63
Figure 4.1(b): Stack plot showing Outside Air Temperature, Room Temperature, Air Volume, CO2 Level, Return Air CO2 Level, Return Air Humidity, Return Air Temperature for room EB 375 and Air Handling Unit corresponding to this room annotated with Power Outages ......................................................................................... 64
Figure 4.1(c): Plot of EB 375 Air Volume with Power Outage Annotation, 26th August 2017 – 26th September 2017 ................................................................................................. 65
Figure 4.2(a): Input Specification File for Retrieve Operation with Include Annotation Selection ........................................................................................................ 67
Figure 4.2(b): Input Specification File for Retrieve Operation with Exclude Annotation Selection ........................................................................................................ 68
Figure 4.3(a): Results for Retrieve Operation with Include Annotation Selection ........................................................................................................ 68
Figure 4.3(b): Results for Retrieve Operation with Exclude Annotation Selection ........................................................................................................ 68
Figure 4.4(a): Query to return the intersection of the datasets returned by include and exclude filter of Retrieve Operation ........................................................................................................ 70
Figure 4.4(b): Result of the intersection of the datasets returned by include and exclude filter of Retrieve Operation ........................................................................................................ 70
Figure 4.5(a): Query to return the union of the datasets returned by include and exclude filter of Retrieve Operation ........................................................................................................ 71
Figure 4.5(b): Results of the union of the datasets returned by include and exclude filter of Retrieve ................................................................. 71
Figure 4.5(c): Number of rows in the result of the union of the datasets returned by Retrieve operation with include filter and Retrieve operation with exclude filter ... 71
Figure 4.6: Include and Exclude Correlation Coefficients between EB 365 room temperature and outside air temperature computed in Microsoft Excel .................. 73
Figure 4.7(a): Input Specification file for Correlation between EB 365 room temperature and outside air temperature when there were classes in the room .... 74
Figure 4.7(b): Query and Result for Correlation between EB 365 room temperature and outside air temperature when there were classes in the room ................... 75
Figure 4.8(a): Input Specification file for Correlation between EB 365 room temperature and outside air temperature when there were NO classes in the room ........................................................................................................................................ 76
Figure 4.8(b): Query and Result for Correlation between EB 365 room temperature and outside air temperature when there were NO classes in the room ........... 77
Figure 4.9(a): Synthetic dataset for EB 365 Room Temperature ................................. 79
Figure 4.9(b): Synthetic dataset for Class Schedule Annotation for room EB 365 ... 79
Figure 4.9(c): Plot of room temperature with Class Schedule Annotation for the synthetic dataset .................................................................................................................................. 80
Figure 4.9(d): Room Temperature values for the third session in synthetic dataset80
Figure 4.10: Input Specification File for Aligned-Average operation for EB 365 room temperature synthetic dataset ........................................................................................................................................ 81
Figure 4.11(a): Results for Aligned-Average of room temperature for EB 365 synthetic dataset ........................................................................................................................................ 82
Figure 4.11(b): Plot for Aligned-Average of room temperature for EB 365 synthetic dataset ........................................................................................................................................ 83
Figure 4.12(a): Query constructed by the AEA service, SQLConverter, for Retrieve operation with exclude Class Schedule Annotation, Fall 2017 .......................... 85
Figure 4.12(b): Runtime statistics for query constructed by the AEA service, SQLConverter, for Retrieve operation with exclude Class Schedule Annotation, Fall 2017 .......................................................................................................................................................... 85
Figure 4.13(a): Alternative query using “NOT EXISTS” clause for Retrieve operation with exclude Class Schedule Annotation, Fall 2017 ............................................................... 85
Figure 4.13(b): Runtime statistics for alternative query using “NOT EXISTS” clause for Retrieve operation with exclude Class Schedule Annotation, Fall 2017 ...................... 86
Figure 4.14(a): Alternative query using “EXCEPT” operator for Retrieve operation with exclude Class Schedule Annotation, Fall 2017 ............................................................... 86
Figure 4.14(b): Runtime statistics for alternative query using “EXCEPT” operator for Retrieve operation with exclude Class Schedule Annotation, Fall 2017 ...................... 87
Figure 4.15(a): Query constructed by the AEA service, SQLConverter, for Correlate operation with include Class Schedule Annotation, Fall 2017 ........................................... 90
Figure 4.15(b): Runtime statistics for query constructed by the AEA service, SQLConverter, for Correlate operation with include Class Schedule Annotation, Fall 2017 .......................................................................................................................................................... 91
Figure 4.16(a): Alternative query using direct join for Correlate operation with include Class Schedule Annotation, Fall 2017 ................................................................................................................................. 91
Figure 4.16(b): Runtime statistics for alternative query using direct join for Correlate operation with include Class Schedule Annotation, Fall 2017 ................................................................................................................................. 92
Figure 4.17(a): Query constructed by the AEA service, SQLConverter, for Aligned-Average operation using OFFSET technique with Class Schedule Annotation, Fall 2017 .......................................................................................................................................................... 94
Figure 4.17(b): Runtime statistics for alternate query for Aligned-Average operation using OFFSET technique with Class Schedule Annotation, Fall 2017 ......................... 94
Figure 4.18(a): Alternate query for Aligned-Average operation with Class Schedule Annotation, Fall 2017 ................................................................................................................................. 95
Figure 4.18(b): Runtime statistics for query constructed by the AEA service, SQLConverter, for Aligned-Average operation with Class Schedule Annotation, Fall 2017 .......................................................................................................................................................... 95
Figure 5.1: Kinds of data, annotation and uses of annotations .............................................. 98
Figure 5.2: Annotations in Google Charts [12] ................................................................. 99
Figure 5.3: System for annotating ECG data, image obtained from the patent [19] 103
Figure 5.4: Difference between AEA and Annotation Propagation systems ............ 106
1. INTRODUCTION

This thesis describes a technique for Annotation-Enabled Interpretation and Analysis of time-series data. I use building data to illustrate the functionality and usefulness of my technique. A building observation is a value that provides information about the state of a building or its environment. A building observation could be a measured quantity such as room temperature or it could be a control parameter such as cooling set point. In this text, I use the term observation also to mean a time-series of such values. To understand and analyze the various building observations such as water consumption, power consumption and CO₂ level, the Building Data and Sensors (BuDS) project was developed at Portland State University. This project aims to design and develop a data infrastructure and analysis framework for data captured from buildings. I provide a detailed overview of this project in Chapter 2. An initial form of analysis in the BuDS prototype was visualizing an observation over time to check for interesting patterns in its data. An important time-series observation that we analyzed is the room temperature in buildings. The Figure 1.1 shows the room temperature over time for a given classroom.
Looking at Figure 1.1, one might guess that the use of the classroom could be the cause for the spikes in the temperature. When we reviewed this data, our collaborator from Department of Mechanical Engineering was interested in distinguishing the rise in temperature associated with room occupancy during classes from the normal rise in temperature over the day. One way to analyze the difference in the cause of temperature rise would be to add annotations denoting class schedules during the different time periods of the day. This motivated the need for adding support for data annotations in the database.
1.1 ANNOTATION-ENABLED QUERIES

Data annotations are information about the data such as comments, errors and provenance\(^1\) that provide context to the data and aid in meaningful data analysis. These annotations can be captured in the database, but using them for analysis can be a challenge as it involves understanding the database schema and may require writing complex joins using inequality conditions. However, by adding annotation about class schedule information in the database, we had the ability to visualize the same room temperature data along with annotations as shown in Figure 1.2. This mechanism of employing annotations as a tool for visualization enabled us to provide a context to the data and understand it better. For example, in Figure 1.1, we can see that the room temperature plot has spikes, but we are not able to recognize the exact connections to room occupancy. But in Figure 1.2, we can clearly observe that the room temperature drops right at the end of the session. We can also clearly distinguish rise in room temperature owing to factors other than room occupancy during classes.

\(^1\) Data Provenance: Data provenance refers to records of the inputs, entities, systems, and processes that influence data of interest, providing a historical record of the data and its origins. [34]
This analysis guided us to ask another question which is, given that we store these annotations in the database, is there more we can do with them? For instance, our collaborator was interested in finding the average room temperature profile when there were sessions in the classroom. This question is an example of what I call annotation-enabled query, where the annotation influences the analysis on the underlying data. Annotation as a visualization tool has utility in understanding the basic data but also gives rise to other intriguing questions, which suggests that annotations can help data analysis in more ways.

I wrote an SQL query to address the question of finding the average room temperature profile when there were sessions in the classroom over a given time period. Figure 1.3 shows the code snippet for the same and Figure 1.4 shows a graphical plot of the result of this query.
Figure 1.3: SQL Query for average room temperature profile for room EB 356 when there were classes in the room, Fall 2017

```
WITH annotation AS
(SELECT
  rann.annotation_start_timestamp,
  rann.annotation_end_timestamp,
  rann.annotation_type_code,
  rann.annotation_type_value,
  rann.annotation_start_timestamp - interval '3 hour' as data_range_start,
  rann.annotation_end_timestamp - interval '2 hour' as data_range_end,
  TO_CHAR(rann.annotation_start_timestamp,'day') as day_of_week,
  EXTRACT(DOW FROM rann.annotation_start_timestamp) as daynumofweek
FROM smart_buildings."ROOM_ANNOTATION_DATE" rann
WHERE DATE_TRUNC('day',rann.annotation_start_timestamp) = date_trunc('day',timestamp '2017-09-24 00:00:00')
AND DATE_TRUNC('day',rann.annotation_end_timestamp) = date_trunc('day',timestamp '2017-12-06 00:00:00')
AND TO_CHAR(rann.annotation_start_timestamp,'HH24MI') = '0800'
AND TO_CHAR(rann.annotation_end_timestamp,'HH24MI') < '1200'
)

SELECT
  tec.tec_system_point_start_timestamp,
  tec.tec_system_point_end_timestamp,
  tec.float_value,
  CASE WHEN tec.tec_system_point_start_timestamp BETWEEN annot.annotation_start_timestamp AND annot.annotation_end_timestamp
  AND tec.tec_system_point_end_timestamp <= annot.annotation_end_timestamp
  THEN END AS indicator,
  tec.tec_system_point_start_timestamp - annot.annotation_start_timestamp as offset_time
FROM smart_buildings."TEC_SYSTEM_DETAIL_DATA" tec
INNER JOIN
  annotation annot
ON
  tec.tec_system_point_start_timestamp = annot.annotation_start_timestamp
AND tec.tec_system_point_end_timestamp < annot.annotation_end_timestamp
WHERE tec.tec_system_point = 'EWC.0031:ROOM TEMPS'

SELECT
  annot_data.offset_time,
  avg(annot_data.float_value) AS aligned_avg,
  max(annot_data.annot_indicator) AS annot_indicator
FROM annot_data
GROUP BY annot_data.offset_time
ORDER BY annot_data.offset_time
```

Figure 1.4: Plot for average room temperature profile for room EB 356 when there were classes in the room, Fall 2017
But this query would be very hard to write for a person who is not an expert in database programming or who is not informed about the database schema, which led to a follow up question, thereby laying the foundation for my thesis: Can we let such people take benefit of annotation data without having to write such complex queries? Hence, my objective for this thesis is to enable people to specify annotation-enabled queries for time-series data by just describing the parameters of the analysis such as the time range, observation and the kind of operation as simple key-value pairs without having to familiarize themselves with database programming or the database schema.

1.2 THESIS CONTRIBUTIONS

I created support for Annotation-Enabled Interpretation by creating annotation visualizations that includes adding annotations in plots along with data to aid in understanding data. I accomplished this by adding annotations as bars behind the data which is plotted as a line graph. I have created plots of this kind for various kinds of data such as room temperature, air volume and CO₂ level annotated with information such as class schedule, power outage and day or night control modes.

I propose a technique for supporting annotation-enabled queries in which the users can specify their analysis criteria through a high-level specification that is less complicated compared to writing the actual SQL query. For instance, the users can provide their analysis criteria through a user interface by selecting parameters
through drop-down menus and radio buttons. This high-level analysis specification hides details like how the data and annotations are stored in the database, which in turn relieves the users from knowing about SQL concepts such as SELECT, FROM, WHERE clauses and also the complex JOIN conditions. The service that I developed takes this specification in the form of a JavaScript Object Notation (JSON) file [1] and returns an SQL string statement that can be executed in the database to fetch the results of the query. Figure 1.5 shows a snapshot of an input specification file for finding the average room temperature profile when there were sessions in the room for which the output is the SQL shown above.

In this thesis, as part of the service, I provide support for three types of annotation-enabled queries of increasing levels of sophistication based on the questions from collaborators:

(1) Retrieve data with include or exclude annotation selection

Example: “Retrieve room temperature information from room “EB 365” in the time range of 24\textsuperscript{th} September 2017 to 6\textsuperscript{th} December 2017 when there were sessions in the room.”

(2) Correlate data with include or exclude annotation selection

Example: “Retrieve room temperature information from room “EB 365” in the time range of 24\textsuperscript{th} September 2017 to 6\textsuperscript{th} December 2017 when there were sessions in the room. Retrieve outside air temperature information in the time range of 24\textsuperscript{th} September 2017 to 6\textsuperscript{th} December 2017. Compute the correlation coefficient of these two observations.”
(3) Aligned-Average profile of an observation relative to annotated periods.

I define Aligned-Average as the time-by-time average profile of the observation by aligning them based on the start and end times of annotations. Example: “Compute the aligned-average temperature profile of room “EB 365” when there were classes between 8AM and 12 PM in the time range of 24th September 2017 and 6th December 2017. Also, include 3 hours before and 2 hours after the class in the average.”

There are other possibilities in Annotation-Enabled queries and above-mentioned use cases are a result of an initial investigation of the feasibility of the approach.
{  
"compute":  
{  
"operation":"aligned-average",  
"aligned_avg_grain":"day",  
"from_date":"2017-09-24 00:00:00",  
"to_date":"2017-12-06 00:00:00",  
"aligned_avg_start_time":"08:00",  
"aligned_avg_end_time":"12:00",  
"lead_time":"3",  
"trail_time":"2"
  },  
"select":"*",  
"source":[
  
{  
"compute":  
{  
"operation":"retrieve",  
"from_date":"2017-09-24 00:00:00",  
"to_date":"2017-12-06 00:00:00"
  },  
"select":["NWC.VV331:ROOM TEMP"],  
"source":"tec_system_detail_data_1",  
"annotation":[
  
{  
"annotate":"include",  
"annotate_type":"CS",  
"annotate_granularity":"room"
  }

  ]

  }

  ]

}

Figure 1.5: Input Specification File for Aligned-Average of Room Temperature
In Chapter 2, I provide an overview of the context of our own particular work – the Building Data and Sensors (BuDS) prototype. In Chapter 3, I explain the service that I developed for Annotation-Enabled Analysis called SQLConverter in detail. I discuss the evaluation strategies I used to test its utility in Chapter 4. Chapter 5 lists some of the relevant work in this field. Finally, in Chapter 6, I discuss possible directions of future work in this topic and provide my concluding remarks.
2. BuDS PROJECT OVERVIEW

People spend 90% of their time in buildings and hence it is important to study the effect of buildings on the health, safety and productivity of people in them. This was one of the main objectives of the Building Data and Sensors (BuDS) research group. Another objective is to analyze the energy efficiency in buildings. Since these two objectives have the potential to contradict each other, it makes the data analysis interesting and encourages the need to analyze the same data in different dimensions. Also, according to U.S. Energy Information Administration, in the year 2017, buildings used about 39% of the total U.S. energy consumption and studying building data can help in identifying faulty equipment that use a lot of energy in buildings. Hence, building data analysis can enable economic energy usage in buildings in a way that minimizes the effect on the comfort or health of the people in them. BuDS is a project developed at Portland State University that focusses on designing and developing a data infrastructure and analysis framework for data captured from buildings. As the name suggests, this project receives data about buildings from heterogenous sources including Building Management Systems (BMS)\(^2\) and our own sensors. Examples of data received are water and power consumption of buildings, room temperature, and air flow observations of Air Handling Units (AHUs).

\(^2\) BMS: A Building Management System is a computer-based control system installed in buildings that controls and monitors the building’s mechanical and electrical equipment such as ventilation, lighting, power systems, fire systems, and security systems. [33]
This chapter describes the technical features of the BuDS data infrastructure and data analysis capabilities. Section 2.1 gives a technical overview of the BuDS system. Section 2.2 describes in detail how data and annotations are stored in the database for the BuDS project. Section 2.3 provides an example of data analysis and visualization in the BuDS project. Chapter 3 focusses on the technical details of the service for Annotation-Enabled Analysis called SQLConverter that converts a high-level analysis specification into SQL query.

2.1 BuDS: TECHNICAL OVERVIEW

Figure 2.1 shows a high-level architecture diagram of the BuDS system. The BMS and the sensors send data in the form of CSV files, which is loaded into the Building Data PostgreSQL database through an Extract, Transform and Load (ETL) workflow. The BMS files arrive once a day at a particular time and contain data for the previous day. The data from our sensors is not configured to send data at a particular time at the time of writing this thesis.

There are three kind of data files:

1. Consumption Data: This file contains observations at the building level such as water consumption, gas consumption and electricity demand. Some of the observations in this file are cumulative in nature and are captured at 15-minute intervals.

2. TEC Data: This file contains observations at the room level, area level and building level such as room temperature, CO₂ levels, air flow and outside air
temperature. These observations are point-in-time observations that represent the values at an instant. These observations are captured at 5-minute intervals.

3. Sensor Data: While the BMS data (Consumption and TEC data) is sourced from the wired sensors in the buildings, these files contain observations about a room or area from wireless sensors used specifically for the BuDS project. Since these sensors are wireless, they have the flexibility to be moved from one location to another. They provide observations such as temperature, relative humidity and CO₂ levels about the location in which they are deployed. These observations are point-in-time observations that represent the values at that instant. We want to keep the data from these sensors flowing wirelessly to the database, but currently we have logging sensors that have to be downloaded.

Figure 2.1: BuDS Architecture Diagram
The ETL workflow through which the data is loaded into the database is depicted in Figure 2.2. Each CSV file goes through the steps in the flowchart to upload its data into the database. The first step in the workflow is to update the database with the header information contained in the file. This header information lists all the observations that are present in the file. Once the header information is captured, the data is loaded into a raw data (or staging) table in the format it is received without any reformatting of values. The database contains Metadata tables that contain information about the observations such as the description of the observation, which building the observation is about and the datatype of the observation. The next step in the workflow is to update the metadata information. Inventory information is a complete description of the available observations for a building or a room by time. After the inventory information is updated, the next step is to process the data and load it in the Processed data table. The main step in processing is to identify the data type of an observation and load the formatted value in a column corresponding to that data type. Currently we have three data types of observations: (1) integer, (2) floating point, (3) string. For example, the data type of room temperature is floating point and hence the value for this observation is populated in the column of floating point data type. In this step, if there are errors encountered, they are handled by the ETL error handler and the step is rerun until the file is successfully loaded in the database. Some of the errors encountered are data type mismatch, junk values and missing values. When these errors are encountered, the erroneous values are inserted in a table called “NULL_STRINGS” and the ETL job is rerun and it populates NULLS in place of these values in the Processed table. The occurrence of errors is one of the primary reasons
for having staging tables where the information is stored in the form it is received so that the BuDS system does not have loss of information due to these errors. The ETL workflow is written using Python and the database used is PostgreSQL.

Figure 2.2: ETL Flowchart
2.2 SCHEMA OVERVIEW

The bulk of the data in BuDS is time-series data representing point-in-time or cumulative measures of observations in buildings. The rest of the data, which are not time-series in nature, are metadata about these data. Annotations in the BuDS system are meant to go with the time-series data. For instance, CS (Class Schedule) annotation indicates the time in which there was a class in the room, PO (Power Outage) annotation indicates the power outage that occurred at a particular time in a particular location. Hence, treating time as a first-class citizen while designing the database was crucial. I provide a structural overview of the BuDS database describing how the data and annotations are stored in the database.

There are different kinds of information stored in the BuDS database. One of the main kinds of information captured is the location information. There is a location hierarchy that exists in BuDS which is: (1) Building, (2) Area, referring to a collection of rooms or other open spaces in a building that is serviced by an Air Handling Unit, and (3) Room. For each kind of location, there are tables in the database that store information pertaining to it. For instance, the Building table stores information such as building name, building address and its description. The Area table stores the building that it is in. The Room table stores information such as the building or area it belongs to and the floor number.

The database design for the three kinds of data – Consumption data, TEC data and Sensor data is very similar. Following are the information captured for each kind of data:
(1) Header: The Header information lists all the observations present in each building data file.

(2) Raw data: Raw datasets store the data as they are retrieved from the files without any reformatting of values.

(3) Metadata: Metadata are data about building observations such as the description of the observation, which building the observation is about and the datatype of the observation.

(4) Inventory: Inventory information tells which observations are available during which time period.

(5) Processed data: The data reformatted according to the data type of the observation and then stored as Processed data.

For these kinds of information, there are Header tables, Raw data tables, Metadata tables, Inventory tables and Processed tables, respectively.

As an example, the schema diagram for TEC data is shown in Figure 2.3.
The observations are stored as key-value pairs in the tables in order to accommodate new observations in the future without having to alter the database design. Each key also has a timestamp along with it which identifies the value for that observation at that particular time.

The annotation schema is shown in Figure 2.4. Annotations are stored separately from data. Similar to the Location entity, the annotations in the BuDS system are also hierarchical in nature. Each annotation item is attached to a time range and an annotation granularity. The BuDS system has a provision to annotate information at different granularities:

(1) Building-Level Annotation: This annotation granularity represents the annotation information about buildings. For example, a Power Outage
Annotation indicates a power outage in a certain building during a certain time range.

(2) Room-Level Annotation: This annotation granularity represents the annotation information about rooms in buildings. For example, a Class Schedule Annotation indicates a class session in a certain room during a certain time range.

(3) Point-Level Annotation: Observations are referred to as points in the Building Management System and hence, this annotation granularity represents the annotation information about a particular observation. For example: a Data Quality Annotation indicates a data quality issue with a certain observation during a certain time range.

![Figure 2.4: Schema Diagram for Annotation](image)

Data for each annotation granularity is stored in a table of its own. Currently there is no explicit Area-level annotation, however due to the presence of location hierarchy, an Area can inherit an annotation from the Building it belongs to. In case we require
a new annotation granularity, then we have to add a new table for this granularity in
the BuDS database. Apart from these tables, there are also metadata tables describing
each annotation. At the time of writing this thesis, the annotation storage is not fully
automated and insert scripts are written manually to load some of them and some are
loaded automatically. Figure 2.5(a) shows examples of Building-Level annotation
instances where an annotation item indicates a Power Outage in a certain building for
a certain time period. Figure 2.5(b) shows examples of Room-Level annotation
instances where each annotation item indicates a Class Schedule in a certain room for
a certain time period.

As stated previously, if we want to record a new observation series in the BuDS
database, no changes need to be made to the schema and annotations can be created
on this new observation as with any other observation. In case we want to add a new
annotation granularity, then this requires an addition of a new table in the schema
and a few additions and updates to the configuration files required by the Annotation-
Enabled Analysis service described in Chapter 3.
2.3 ANNOTATION-ENABLED INTERPRETATION

The data analysis and visualization framework currently used for BuDS is primarily through visualization libraries in Python using Jupyter notebook [2]. I also designed and developed a web interface that serves as a query interface and visualization platform for end users. I describe the details of Annotation-Enabled Interpretation through both the frameworks and provide examples of annotation visualizations that guided us towards Annotation-Enabled Analysis.

Jupyter notebook [2] is an open-source web application that allows us to create and share documents that contain live code, equations, visualizations and narrative text. Since both annotations and data are stored in the database, visualizing both of them together was not difficult. A lot of thought and discussion went into how these annotations can be visualized along with data so that it makes the analysis more meaningful. We decided to display annotations as bars behind the underlying data with x-axis representing the time domain and the y-axis representing the observation that we are visualizing. An example of such a visualization is presented in Figure 2.6.
This plot represents the weekly average air-volume profile for the room EB 365 during Fall of 2017 annotated with the class schedule information. As seen in the figure, we can observe interesting patterns in the air volume data when there are classes in session. For example, it can be observed that the air volume in the room EB 365 can be elevated for more than an hour after the class has ended.

Figure 2.7 shows a snapshot of the Jupyter notebook developed to produce the plot of weekly average air volume. The first step in creating the visualization is to import the necessary Python Libraries such as psycopg2 [3] for connecting to PostgreSQL database and matplotlib [4] for visualization. I was able to connect to the BuDS database from the Jupyter Notebook and get access to data for creating visualizations for Annotation-Enabled Interpretation. Once the connection to the database is established, I run the SQL query shown in the figure to fetch the dataset to visualize. Then I create the annotation column “Class Schedule” on-the-fly based on presence and absence of a value in “annotation_type_code” column. Then I use the visualization
library to plot the annotation information as bars and the air volume average profile as a line plot.

Figure 2.7: Jupyter Notebook for generating the weekly average profile of air volume for EB 365, Fall 2017
The other data analysis framework is a web interface that I developed for the BuDS project. It consists of a data explorer, which serves as a query interface for the users. The users can submit their queries through a form and the results are rendered back to them in both tabular format and graphical plots. A snapshot of this is shown in Figure 2.8.

![Figure 2.8: Room Temperature for EB 365](image)

As seen in the figure, this interface provides a compact view of the query inputs and the results in the form of a table and visualization in the same page. It provides the capability of making a slight alteration to the query and viewing the change in results immediately. It also has a feature for hovering over a particular point in the visualization to reveal the value under focus. I used Bootstrap [5], JavaScript [6] and Python Flask [7] to develop the web interface for the project and used Highcharts [8] to develop the visualizations.

In general, we observed that using the web interface is more fluid and quicker compared to Jupyter notebook for data analysis, but has limited functionality.
compared to Jupyter Notebooks and hence we found both the frameworks useful for analysis. It can also be noted that both these frameworks have different plotting styles, the result of the visualization libraries used.

These two data-analysis frameworks gave us a good platform to design the service for Annotation-Enabled queries. These analyses corroborated the need for Annotation-Enabled Analysis and viewing these visualizations suggested questions for such analyses to our collaborators.
3. ANNOTATION-ENABLED ANALYSIS

I define Annotation-Enabled Analysis as an approach to examine data in which annotations help specify the nature of analyses. In this technique, since the annotation information drives the analysis specification, the result of the analysis is derived by taking into consideration both the annotation information and the underlying data. A simple example of such an analysis is that of include versus exclude annotation analysis. The Retrieve operation with “include” annotation selection returns the data that is annotated and the Retrieve operation with “exclude” annotation selection returns the data that is not annotated. Though this question is simple, it is useful to ask and it is not very straightforward to write such a query for a person who is not well versed in database programming: the query not only requires expertise on a database programming language but also information as to how the annotations and data are stored in the database. In this section, I describe a service that I developed called SQLConverter that takes a high-level description of an analysis in the form of simple key-value pairs represented in JavaScript Object Notation (JSON) file format and returns, as output, an equivalent SQL query that can then be executed in the database. I describe the technical features of SQLConverter in Section 3.1 and all the Annotation-Enabled Analysis operations currently supported in the service in the sections following that.

3.1 SQLConverter: A TECHNICAL OVERVIEW

I designed SQLConverter keeping in mind that it should be a user-friendly method to specify the requirements of Annotation-Enabled Analysis. The service takes a high-
level specification of the analysis in the form of a JSON file as input and returns an SQL statement that can then be executed in the database to fetch the results of the query.

The basis for choosing a JSON file format was that JSON objects are in the form of simple key-value pairs and hence the users can express their query in a format that is very similar to filling in a form. The JSON file is also light-weight and can be transferred from one system to another with ease and hence is one of the most widely used data-interchange formats across the web. This feature of JSON enables the service to be easily plugged into other systems. We can add an Annotation-Enabled Analysis (AEA) feature to a web interface without having to do a lot of rework. The web application can pass the user inputs to the AEA service in the JSON file format and the service can be used without any changes to generate the SQL statement, which can then be executed in the database to fetch the results.

I designed the input specification file format so that it is both human readable and machine readable. This service can be used by a web application for Annotation-Enabled Analysis by receiving inputs from the user through a form and converting it into a format that is acceptable by the service. The application can then pass this input to the service which returns an output query that can then be executed in the database to fetch the results. These results can be rendered back to the user through the web application in both tabular format and visualizations.
Figure 3.1: AEA Technical Overview

Figure 3.1 shows the high-level architecture diagram of SQLConverter, the service for Annotation-Enabled Analysis. The service takes a high-level analysis specification in the form of JSON file as input, processes it and returns an SQL statement. It is also possible to specify the Retrieve operation as a sub-specification in the Correlate and Aligned-Average specification file. In addition to the input file, there are two configuration files that the service is initialized with:

1. Annotation Mapping Configuration: The BuDS system supports different annotation granularities, which are stored in separate tables. The mapping between an annotation granularity and its corresponding table in the database is provided to the service through the Annotation Mapping Configuration file.
(2) Schema Configuration: The Schema Configuration file consists of information from the Building Data database schema that is essential for the service to generate the SQL string. This information includes details such as columns to be included in the result for data tables and annotation tables, aliases for tables and JOIN conditions between the data table and different annotation tables involving timestamp columns.

Snapshots of an Annotation Mapping Configuration file and a Schema Configuration file are shown in Figure 3.2(a) and 3.2(b) respectively.

```json
{
    "room": "ROOM_LEVEL_ANNOTATION_INSTANCE",
    "point": "POINT_LEVEL_ANNOTATION_INSTANCE",
    "building": "BUILDING_LEVEL_ANNOTATION_INSTANCE"
}
```

Figure 3.2(a): Example Annotation Mapping Configuration file
The service consists of three layers of components: a checker, a router and an SQL Generator layer consisting of operations such as Retrieve, Correlate and Aligned-Average.
3.2 CHECKER

The checker component consists of two parts: (1) General Checks, (2) Operation-specific checks. The checker component reads the input file and inspects to see if it is formatted correctly. Every specification file should have at least three fields: “compute”, “select” and “source”. The “compute” field represents the operation that the user wants to compute. The “select” field specifies the building observation to be fetched. The “source” field specifies the table that contains the building observation. If these fields are not present in the input file, the file cannot be further processed. Some optional fields are “annotation”, “order_by”, etc. Another check that is performed for all the operations is date-format validation. The checker performs these preliminary validations and throws meaningful error messages back to the user when the input file fails these validations.

The next step is to perform operation-specific checks. The checker performs the checks that are relevant to the specific operation that the user wants to perform. For instance, one of the checks is to verify that the Correlate operator receives exactly two sources. It returns an error message to the user if there are more than two sources or if there is only one source specified in the input file. In the Aligned-Average operation, the service checks if all the required operation specific fields are present in the input such as “aligned_avg_grain”, “lead_time” and “trail_time”. Another check is to verify whether the aligned-average grain specified in the file is within the scope of the service.
After this step, if the specification file has passed all the operation-specific checks, it is considered to be a valid specification and the service will generate an output SQL for it.

3.3 ROUTER

After the checker has performed all its checks, the specification file is ready to be sent to the corresponding SQL Generator. The router is a simple component of the service that is responsible for directing the input files to the correct SQL Generator. Based on the operation specified in the input file, the file is then directed to the corresponding SQL Generator for generating the SQL. The router is a separate component to make it easy to add operators to the service.

3.4 SQL GENERATORS

There are currently three operations supported in this service: (1) Retrieve, (2) Correlate, (3) Aligned-Average. I describe each Generator in terms of its scope, generation and results in this section. I explain the capability, implementation and outcomes of each operation in Scope, Generation and Result subsections, respectively.

3.4.1 RETRIEVE OPERATION

The retrieve operation, as the name suggests, selects and returns the data specified by the user input. One of the main functionalities of the retrieve operation is to support include and exclude annotation selection. The include or exclude feature
filters data based on the presence or absence of annotation instances of the specified annotation type in that time range. The include annotation selection picks data only in the time periods of the annotation instances of the specified annotation type. The exclude annotation selection picks the data that are outside the time periods of the annotation instances of the specified annotation type. The Retrieve operator is simple but is useful as a sub-operator in other operators such as Correlate.

The include or exclude annotation filter turns out to be a useful feature in the BuDS system to study how the building observations relate to the presence and absence of annotations such as the Class Schedule annotation (indicating the presence of people in a room) and Power Outage annotation (indicating the response of the building meters to power outages). With this feature, once we retrieve room temperature data by applying an include or exclude annotation selection, it is fairly simple to compute the average room temperature from this result, which helps us compare the average room temperature with and without sessions in class and measure the impact of the room occupancy on the room temperature.

3.4.1.1 SCOPE

The scope of the retrieve operation is to return the observation values in the specified time range and apply the include or the exclude annotation filter. The output of the operation is an SQL statement that when executed in the Building Data database returns the results. Each result row contains time-range information, the value of the observation and the corresponding annotation information, if present.
3.4.1.2 GENERATION

I describe the design and implementation of the Retrieve operation in the SQLConverter service in this subsection. Figure 3.3 below shows a sample input specification file for the retrieve operation with an include annotation selection. The user’s intended request with this input file is:

“Retrieve room temperature information from room “EB 365” in the time range of 24th September 2017 to 6th December 2017 when there were class sessions in the room”.

The router in the service recognizes that the operation specified is “Retrieve”, hence it directs the request to the Retrieve SQL Generator.
The retrieve operator constructs the SELECT, FROM, WHERE and JOIN clauses of the SQL query from the input specification file. The SELECT clause is constructed from the Schema Configuration described in the previous section. Based on the “source” field, the Schema Configuration is consulted for the list of columns to be selected for that table. As seen in the input file, the “select” field is a list object, hence the retrieve operation can support selecting more than one building observation. The FROM clause of the SQL statement is constructed from the “source” field of the input. The source field of the input specifies the table in the database that contains the building observation. The WHERE clause needs to incorporate both “select” field in the input file to apply the filter condition for the building observation being fetched as well as
the time range specified by the “from_date” and “to_date” fields. The annotation-selection criteria are specified in the “annotation” field of the input. The “annotate_granularity” field provides the key to fetch the corresponding annotation table to join with. The “annotate_type” and the annotation selection (include or exclude) provided by the “annotate” field are the information required to build the JOIN condition between the data table and the annotation table. If the annotation selection is “include”, then the join becomes an INNER JOIN and if the annotation selection is “exclude”, then the join becomes a LEFT OUTER JOIN with selection on NULLS in annotation columns.

The output generated by the service for the aforementioned input is shown below in Figure 3.4.

```
SELECT
    tec.tec_system_point_start_timestamp,
    tec.tec_system_point_end_timestamp,
    tec.float_value,
    rann.annotation_start_timestamp,
    rann.annotation_end_timestamp,
    rann.annotation_type_code,
    rann.annotation_type_value,
    rann.annotation_value
FROM smart_buildings."TEC_SYSTEM_DETAIL_DATA_1" tec
INNER JOIN smart_buildings."ROOM_LEVEL_ANNOTATION_INSTANCE" rann
ON tec.tec_system_point_start_timestamp >= rann.annotation_start_timestamp
AND tec.tec_system_point_end_timestamp <= rann.annotation_end_timestamp
AND rann.annotation_type_code = 'CS'
WHERE tec.tec_system_point = 'NWC.VV331:ROOM TEMP'
AND tec.tec_system_point_start_timestamp BETWEEN '2017-09-24 00:00:00' AND '2017-12-06 00:00:00'
ORDER BY tec.tec_system_point_start_timestamp;
```

Figure 3.4: Output SQL for Retrieve Operation with Include Class Schedule Annotation Selection
3.4.1.3 SAMPLE RESULTS

I present example results obtained for the Retrieve operation for both include annotation selection and exclude annotation selection in this subsection. Figure 3.5 shows the results of executing the query shown above in Figure 3.4.

![Figure 3.5: Output of Retrieve Operation of EB 365 Room Temperature with Include Class Schedule Annotation Selection, Fall 2017](image)

As specified in the previous section, this result answers the query:

“Retrieve room temperature information from room “EB 365” in the time range of 24th September 2017 to 6th December 2017 when there were class sessions in the room”.

The start time of this query, 24th September 2017, was a Sunday and there were no classes on that day, hence the first row of the result starts with 25th September 2017.

As mentioned in the scope of the operator, Retrieve also supports exclude annotation selection. The user must provide “exclude” in the “annotate_type” field in the input file. The Retrieve operation with exclude annotation filter returns all the values excluding the annotation time ranges. For instance, if the user’s intended request is:
“Retrieve room temperature information from room “EB 365” in the time range of 24th September 2017 to 6th December 2017 when there were NO class sessions in the room”

This query can be expressed in the form of an input file as shown in Figure 3.6(a). The output SQL generated by the SQLConverter service for this input file is also shown in Figure 3.6(b). The result of the query is shown in Figures 3.7.

```
{
  "compute": {
    "operation": "retrieve",
    "from_date": "2017-09-24 00:00:00",
    "to_date": "2017-12-06 00:00:00"
  },
  "select": ["NWC.VV331:ROOM TEMP"],
  "source": "tec_system_detail_data_1",
  "annotation": [
    {
      "annotate": "exclude",
      "annotate_type": "CS",
      "annotate_granularity": "room"
    }
  ],
  "order_by": "timestamp"
}
```

Figure 3.6(a): Input Specification File for Retrieve Operation with Exclude Class Schedule Annotation Selection
Figure 3.6(b): Output SQL for Retrieve Operation with Exclude Class Schedule Annotation Selection

```
SELECT 
 tec.tec_system_point_start_timestamp, 
 tec.tec_system_point_end_timestamp, 
 tec.float_value, 
 rann.annotation_start_timestamp, 
 rann.annotation_end_timestamp, 
 rann.annotation_type_code, 
 rann.annotation_type_value, 
 rann.annotation_value 
 FROM smart_buildings."TEC_SYSTEM_DETAIL_DATA_1" tec 
 LEFT OUTER JOIN smart_buildings."ROOM_LEVEL_ANNOTATION_INSTANCE" rann 
 ON tec.tec_system_point_start_timestamp = rann.annotation_start_timestamp 
 AND tec.tec_system_point_end_timestamp = rann.annotation_end_timestamp 
 AND rann.annotation_type_code = 'CS' 
 WHERE tec.tec_system_point = 'WNC.VY331:ROOM TEMP' 
 AND tec.tec_system_point_start_timestamp BETWEEN '2017-09-24 00:00:00' AND '2017-12-06 00:00:00' 
 AND rann.annotation_start_timestamp IS NULL 
 ORDER BY tec.tec_system_point_start_timestamp;
```

Figure 3.7: Results for Retrieve Operation of EB 365 Room Temperature with Exclude Class Schedule Annotation Selection, Fall 2017

Figure 3.8: Plot for Include versus Exclude Retrieve Operation of EB 365 Room Temperature, Fall 2017
The difference between include annotation selection and exclude annotation selection is an additional WHERE condition to check for rows where the annotation columns are NULL. As seen in the result in Figure 3.7, all the annotation related columns are NULL as the Retrieve operator with exclude annotation selection fetches only those values in the time period when there are no “Class Schedule” annotations. Figure 3.8 shows a visualization of the include versus exclude Retrieve operation analysis for EB 365 room temperature for Fall 2017 where the green markers indicate room temperature values during class periods and red markers indicate room temperature values outside the class periods.
3.4.2 CORRELATE OPERATION

As per the statistical definition, the Pearson correlation function computes a correlation coefficient, which is a numerical measure that determines the strength of the relationship between two sets of numbers. This coefficient is a number between -1 and 1. A coefficient greater than 0 indicates a positive correlation between the two sets and a coefficient less than 0 indicates a negative correlation. If the correlation function returns 0, it indicates that there is no correlation between the two sets.

This operator is useful in the BuDS system because we have observed numerous instances where there is a strong relationship between observations that are measured. For instance, there is a correlation between the outside air temperature and the temperature in the room. Hence, it might be interesting to study whether these observations correlate differently under different conditions, such as the presence or absence of class sessions.

3.4.2.1 SCOPE

The scope of the Correlate operator defined in the service is to return the Pearson correlation coefficient of two observations over the specified time period while applying the include or exclude annotation selection to the observations. The operator does not check if the two variables specified in the input contain data series of the same length. If they are of unequal length, then only those observations that match on timestamp are accounted for in the Correlation operation. The output of the operation is an SQL statement, which when executed in the Building Data database,
returns the result. The result is a single row containing the correlation coefficient measure between the two observations.

3.4.2.2 GENERATION

I describe the design and implementation of the Correlate operation in the SQLConverter service in this section. Figure 3.9 below shows an example input specification file for the Correlate operation with include annotation selection. The user's intended request with this input file is:

“Retrieve room temperature information from room “EB 365” in the time range of 24th September 2017 to 6th December 2017 when there were class sessions in the room. Retrieve Outside Air Temperature (OAT) information in the time range of 24th September 2017 to 6th December 2017. Compute the correlation coefficient of these two measures when both are present.”

The router in the service recognizes that the operation specified is “Correlate”, hence it directs the request to the Correlate SQL Generator.

The input file for the Correlate operator should contain exactly two sources and this validation is performed at the checker component. The operation accepts exactly two sources because the correlation coefficient can be computed only on two variables. It can be observed that the sources in the input file of Correlate operation shown in Figure 3.9 are Retrieve operations themselves. Hence, I designed the Correlate operator such that it calls the Retrieve operator on each of the sources and constructs SQL queries for generating temporary datasets for each. Then, I make use of the native
PostgreSQL correlation function, CORR(x,y) to compute the correlation between the two sources. I use an INNER JOIN to combine the two temporary datasets because the CORR function expects its input as a list of pairs.

The output generated by the service for this input file is shown below in Figure 3.10.
Figure 3.9: Input Specification File for Correlate Operator between OAT and EB 365 Room Temperature with Include Class Schedule Annotation Selection, Fall 2017
3.4.2.3 SAMPLE RESULTS

The expected result for the Correlate operator is the correlation coefficient of the two observations provided in the input specification file. In this section, I present the results obtained for the Correlate operation for both include annotation selection and exclude annotation selection. Figure 3.11 shows the result of the query specified in the execution section as shown in Figure 3.10.
With the correlate operator, as specified, it is also possible to perform the operation with exclude annotation selection. As with the Retrieve operator, this variation is specified by providing “exclude” in the “annotate_type” field in the input file. A Snapshot of the input file and the corresponding output SQL generated by SQLConverter is shown in Figures 3.12(a) and 3.12(b) respectively. The result of executing the SQL in the database is shown in Figure 3.13.

Both the results indicate that the room temperature in EB 365 correlates positively with Outside Air Temperature (OAT). The correlation is stronger when there are no classes in the room as the human activity in the room is not an influence.
Figure 3.12(a): Input Specification File for Correlate Operator between OAT and EB 365 Room Temperature with Exclude Class Schedule Annotation Selection, Fall 2017
WITH dataset1 AS
(SELECT
tec.tec_system_point_start_timestamp,
tec.tec_system_point_end_timestamp,
tec.float_value,
rann.annotation_start_timestamp,
rann.annotation_end_timestamp,
rann.annotation_type_code,
rann.annotation_type_value,
rann.annotation_value
FROM smart_buildings."TEC_SYSTEM_DETAIL_DATA_1" tec
LEFT OUTER JOIN smart_buildings."ROOM_LEVEL_ANNOTATION_INSTANCE" rann
ON tec.tec_system_point_start_timestamp >= rann.annotation_start_timestamp
AND tec.tec_system_point_end_timestamp <= rann.annotation_end_timestamp
AND rann.annotation_type_code = 'CS'
WHERE tec.tec_system_point = 'NWC.VV331:ROOM TEMP'
AND rann.annotation_start_timestamp >= '2017-09-24 00:00:00'
AND rann.annotation_end_timestamp <= '2017-12-06 00:00:00'
AND rann.annotation_start_timestamp IS NULL),

dataset2 AS
(SELECT
tec.tec_system_point_start_timestamp,
tec.tec_system_point_end_timestamp,
tec.float_value
FROM smart_buildings."TEC_SYSTEM_DETAIL_DATA_1" tec
WHERE tec.tec_system_point = 'NWC.BLDG.OAT'
AND tec.tec_system_point_start_timestamp BETWEEN '2017-09-24 00:00:00' AND '2017-12-06 00:00:00')

SELECT corr(dataset1.float_value,dataset2.float_value) AS correlation
FROM dataset1
INNER JOIN
dataset2
ON dataset1.tec_system_point_start_timestamp = dataset2.tec_system_point_start_timestamp;

Figure 3.12(b): Output SQL for Correlate Operator between OAT and EB 365 Room Temperature with Exclude Class Schedule Annotation Selection, Fall 2017

Figure 3.13: Correlation between OAT and EB 365 Room Temperature with Exclude Class Schedule Annotation Selection, Fall 2017

<table>
<thead>
<tr>
<th>correlation</th>
<th>double precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.377392255208014</td>
</tr>
</tbody>
</table>
3.4.3 ALIGNED-AVERAGE OPERATION

I define *time-by-time average profile* of an observation as a series of averages of the observation grouped by time at which the observation occurred. I define Aligned-Average as the time-by-time average profile of the observation by aligning them based on the start and end times of annotations. An example of this operation in the BuDS system is to compute the time-by-time average room temperature profile when there were morning sessions in a certain room. In this scenario, there could be classes at different times in the mornings and hence it does not work to simply compute the average at each time point. An instance of this scenario is shown in Figures 3.14(a) and 3.14(b), when the session begins at a slightly different time on Mondays compared to Tuesdays. This scenario gave rise to aligning data based on a certain type of annotation and performing operations on them. In this instance, we select subsets of room temperature data in the period coinciding with start and end time of Class Schedule Annotation, align them and compute the time-by-time average profile of room temperature when there were classes in the room. By aligning the observation series based on annotations, we construct a *relative* time-series.
3.4.3.1 SCOPE

The scope of the Aligned-Average operator defined in the service is to return the time-by-time average profile of an observation in the specified time period by shifting the observation data in time so that the annotations line up.
The time period for the operation is stated by the following fields in the specification file:

1. *from_date*: This field marks the beginning of the analysis period.
2. *to_date*: This field marks the end of the analysis period.

There could be multiple annotations of the same kind on the same day. For instance, there could be multiple classes in the same room on the same day. If we want to consider only a subset of these annotations in the analysis, then we need a way to specify the start and end time within a day to be considered for this analysis. Following are the fields in the specification file to specify this requirement:

1. *aligned_avg_start_time*: This field indicates the start time from which the annotation needs to be considered for analysis in a day.
2. *aligned_avg_end_time*: This field indicates the end time until which the annotation needs to be considered for analysis in a day.

The operator supports two Aligned-Average grains:

1. **day**: In this grain, the data is aligned based on the start and the end time of annotations. The result of the Aligned-Average operator with this grain is the time-by-time average profile of the observation over a day.
2. **week**: In this grain, the data is aligned based on the start and end time of annotations as well as day of the week. The result of the Aligned-Average operator with this grain is a collection of time-by-time average profiles of the observation aligned by the day of the week and offset time. This variation in
Aligned-Average operator is useful if we wanted to compute the time-by-time average profile of an observation during annotated time periods for a particular day of the week.

The operator has the capability to include non-annotated prefixes and suffixes in the average profile. This capability is provided by the following two fields in the input specification:

(3) lead_time: This field indicates the number of hours to be included in the average profile before the start of each annotation.

(4) trail_time: This field indicates the number of hours to be included in the average profile after the end of each annotation.

At the time of writing this thesis, the operator does not support annotations of unequal durations or multiple annotations at the same time range. The output of the operation is an SQL statement that when executed in the Building Data database returns the result. The result contains the series of averages of the specified observation in the room stated in the input after aligning the annotated regions including the leading and trailing times.

3.4.3.2 GENERATION

I describe the design and implementation of the Aligned-Average operation in the SQLConverter service in this section. Figure 3.15 below shows an example input specification file for the Aligned-Average operation with include annotation selection. The user's intended request with this input file is:
“Compute the time-by-time aligned-average profile of room temperature in EB 365 when there were classes between 8AM and 12 PM in the time range of 24th September 2017 and 6th December 2017 over a day. Also, include 3 hours before and 2 hours after the class in the average.”

In this analysis,

- 8 AM and 12 PM signify the aligned_avg_start_time and aligned_avg_end_time fields respectively.
- The dates 24th September 2017 and 6th December 2017 signify the from_date and to_date fields respectively.
- The numbers 3 and 2 signify the lead_time and trail_time fields respectively.

The router in the service recognizes that the operation specified is “Aligned-Average”, hence it directs the request to the Aligned-Average SQL Operator.
It can be observed from the input file that it has a set of new fields compared to the Retrieve and Correlate input files. There are five fields specific to the Aligned-Average
operation: (1) aligned_avg_start_time, (2) aligned_avg_end_time, (3) lead_time, (4) trail_time and (5) aligned_avg_grain. The details of these fields are explained in the scope of the operator.

Unlike the Correlate operator in which the Retrieve operator could be used directly to fetch the datasets, the Aligned-Average operator requires the datasets in a format unique to it. I describe the methodology adopted to align the datasets based on annotations. First, based on the “annotation” field in the input file, a temporary dataset is constructed which contains all the annotation records in the specified date range specified in the fields from_date and to_date and the specified time range in the fields aligned_avg_start_time and aligned_avg_end_time. Then, for each record in the annotation dataset, temperature data is obtained from the data table and a field called offset time is created for each row. Offset time is computed as the difference between its corresponding annotation start time and the time at which the observation occurred. Once the data is aligned according to the offset time, I compute the time-by-time average profile of room temperature grouping them by the offset time. This analysis treats the missing data as NULLS and they are not considered for the average profile. The output generated by the service for this input file is shown below in Figure 3.16. The result of this operation is a relative time-series with zero corresponding to annotation start time.
3.4.3.3 SAMPLE RESULTS

The expected results for the Aligned-Average operation is time-by-time average profile of the observation aligned using the start and end times of the annotations. The result has an offset-time column that represents the alignment of data according to annotations. In this section, I present the results obtained for the Aligned-Average operation for the “day” aligned-average granularity for different time periods. The results of executing the SQL query generated by the service is shown in Figure 3.17. As seen in the figure, the result contains three columns:

- offset_time: This column is computed as the difference between the start time of the annotation instance and the time at which the observation occurred.
aligned_avg: This column contains the average room temperature when the sessions took place in the room.

annot_indicator: Since the operator has the utility to specify time regions around the annotated periods, it is useful to have a Boolean column that indicates the rows that are during the annotated time periods.

Figure 3.18 shows the visualization of the results as a graphical plot. The region shaded in pink is the region in which there was a session in the room. This plot shows the average room temperature of room EB 365 when there were sessions in the class between 8 AM and 12 PM during Fall 2017. It also includes the average room temperature 3 hours prior to the class and 2 hours after the class.

The same analysis is carried out for evening classes in EB 365 between 12 PM and 6 PM for Fall 2017 quarter is shown in Figure 3.19. The aligned-average for 5 hours before the session and 16 hours after the session is also computed as part of the operation. We can also see a peak from the morning classes in this result but they are not guaranteed to be aligned.

From this analysis we can observe that the room temperature seems to peak a little higher during evening classes but also drops off at the end of the session sooner compared to the room temperature during morning classes.
Figure 3.17: Output of Aligned-Average of morning classes with Include Annotation Selection, Fall 2017

<table>
<thead>
<tr>
<th>offset_time</th>
<th>aligned_avg numeric</th>
<th>annot_indicator integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>72.9679487179487179</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>72.9743597435974945</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>72.9615384615384615</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>72.9743597435974945</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>72.9615384615384615</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>72.9807692307692308</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>72.987194871794872</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>73.0064102564102564</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>73.0256410256410256</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>73.0192307692307692</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>73.0256410256410256</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>73.0192307692307692</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>73.0256410256410256</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>73.0192307692307692</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>73.0320512820512821</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>73.3461538461538462</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>73.4423076923076923</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>73.5487194871948719</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>73.6589743589743589</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>73.8589743589743589</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>74.0320512820512821</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>74.1987194871948719</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>74.3333333333333333</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>74.4166666666666667</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>74.5192307692307692</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>74.6602564102564103</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>74.7500000000000000</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>74.8269230769230769</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>74.8846153846153846</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>74.9615384615384615</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>74.9743597435974945</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>75.0000000000000000</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>75.0384615384615385</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>75.0705128205128205</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>75.0705128205128205</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>75.1025641025641026</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>75.1153846153846154</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>75.1025641025641026</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>75.0641025641025641</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>74.9807692307692308</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>74.9038461538461538</td>
<td></td>
</tr>
</tbody>
</table>
Figure 3.18: Plot of Aligned-Average of morning classes in EB 365 with Include Annotation Selection, Fall 2017

Figure 3.19: Plot of Aligned-Average of evening classes in EB 365 with Include Annotation Selection, Fall 2017
3.5 ROLE OF AEA IN BuDS

Figure 3.20 shows how the SQLConverter (AEA) service fits in the BuDS system. The AEA service is a package that can be imported into Jupyter Notebooks or even Python Flask server (web application). When the input specification file is passed to the service, it generates an SQL string that can be executed in the Building Data Database and the results of the query can be visualized in either of the data analysis frameworks. It is possible to easily build a service which takes the SQL query generated by SQLConverter (AEA Service) and executes the query in the database and fetches the results. One of the primary reasons for making the AEA service return the query and not directly the results is that it allows the user to build upon the analysis by modifying the query and studying the results. Also, the query returned by the service can be used as a subquery in other queries.
4. EVALUATION

I describe the evaluation strategies I used to test the correctness of SQLConverter, my implementation of Annotation-Enabled Analysis. The main goals of the evaluation are:

(1) Validate the correctness of the service

(2) Assess the performance of the translations

I provide sample visualizations in this section that I developed as part of Annotation-Enabled Interpretation to indicate the usefulness of these annotation visualizations. Though these visualizations do not prove correctness, they show that annotations can call out interesting aspects of data. Section 4.1 provides sample annotation visualizations in this category.

The first evaluation method is to create a synthetic dataset for which I know the expected result of the Annotation-Enabled Analysis and verify whether or not my service produces the same result. This method will provide some validations on the correctness of the service.

The next evaluation method is to analyze the performance of the query produced by SQLConverter and study how it compares with some alternate ways of writing the same query. Though databases have optimizers that try to produce the best plan to execute the query, it is necessary to perform this evaluation because there might be some information about the database that the optimizer might not be aware of and
hence might not come up with the right estimates. For instance, in the BuDS database, each data point generally joins with at most one annotation record.

4.1 EVALUATION OF ANNOTATION-ENABLED INTERPRETATION

In this subsection, I provide examples of Annotation-Enabled Interpretation that helped the BuDS research group to identify interesting patterns in data, indicating the usefulness of these visualizations.

Figure 4.1(a) shows a stack plot of various observations such as Outside Air Temperature (OAT), Room Temperature, Air Volume, CO$_2$ Level from room EB 375 and other observations such as Return Air CO$_2$ (RAC), Return Air Humidity (RAH) and Return Air Temperature (RAT) of the Air Handling Unit (AHU) catering to room EB 375 annotated with EB 375 day-night control mode. It can be observed that the drop in air volume observation lines up with the night mode and the rise in air volume lines up with the day mode. This pattern might indicate that the air flow is controlled based on this control mode.
Figure 4.1(a): Stack plot showing Outside Air Temperature, Room Temperature, Air Volume, CO2 Level, Return Air CO2 Level, Return Air Humidity, Return Air Temperature for room EB 375 and Air Handling Unit corresponding to this room annotated with night control mode.
Figure 4.1(b): Stack plot showing Outside Air Temperature, Room Temperature, Air Volume, CO₂ Level, Return Air CO₂ Level, Return Air Humidity, Return Air Temperature for room EB 375 and Air Handling Unit corresponding to this room annotated with Power Outages.

Figure 4.1(b) shows the same observations in a different time period annotated with Power Outage annotation. Power Outage annotations are announced power outages in a building that are scheduled to occur during a particular time period but might
take place for less than the total time period. The interesting thing to notice about this plot is that there are some observations such as room temperature, air volume and RAC that drop all the way to zero during a power outage. These drops might indicate that the BMS systems are affected during the power outage. There is also evidence from the plot that shows that there is an anomaly in the air flow values after the power outage where the air volume continues to drop to zero every twelve hours even after the power outage. Our collaborators were interested to see how long this trend continued. Figure 4.1(c) shows Air Volume for EB 375 annotated with Power Outage information for a longer time period. We noticed that this trend continued even ten days after the outage.

Figure 4.1(c): Plot of EB 375 Air Volume with Power Outage Annotation, 26th August 2017 – 26th September 2017
4.2 VALIDATION OF THE RETRIEVE OPERATION

I created a synthetic dataset for evaluating the Retrieve Operation with include and exclude annotation selection. I accomplished this by creating two new tables in the database:

1. **TEST_TEC_SYSTEM_DATA**: This table is a replica of TEC_SYSTEM_DETAIL_DATA table and contains synthetic room temperature data for 21st July 2018.

2. **TEST_ROOM_LEVEL_ANNOTATION_INSTANCE**: This table contains the EB 365 Class Schedule annotation information for the same day.

The dataset consists of room temperature observation values for room EB 365 between 10.00 AM and 11.00 AM on 21st July 2018 (Saturday). I added the room temperature observation values in TEST_TEC_SYSTEM_DATA and two class schedule annotations of ten-minute duration from 10:15 AM to 10:25 AM and from 10:40 AM to 10:50 AM in TEST_ROOM_LEVEL_ANNOTATION_INSTANCE table. Since the room temperature data is added at 5-minute intervals, the dataset considered for this evaluation consists of a total of twelve rows. The Retrieve operation with include annotation selection should return a total of four rows with two rows from the first class session and two rows from the second class session. The retrieve operation with exclude annotation selection should return the other eight rows.

In order to compute the Retrieve operation for this data using the service, I added the aforementioned tables to the Schema Configuration file and added an annotation
mapping between “test_room” key and “TEST_ROOM_LEVEL_ANNOTATION_INSTANCE” value in the Annotation Mapping Configuration file.

Figure 4.2(a) shows the input specification file for evaluation of Retrieve with include annotation selection and Figure 4.2(b) shows the input specification file for Retrieve operation with exclude annotation selection.

I then executed the SQL queries generated by the AEA service, SQLConverter for these input specification files in the database. The results of the queries are shown in Figure 4.3(a) and Figure 4.3(b). As seen in the figures, the results are as expected.

```json
{
    "compute": {
        "operation": "retrieve",
        "from_date": "2018-07-21 00:00:00",
        "to_date": "2018-07-21 23:55:00"
    },
    "select": ["NWC.VV331:ROOM TEMP"],
    "source": "TEST_TEC_SYSTEM_DATA",
    "annotation": [{
        "annotate": "include",
        "annotate_type": "CS",
        "annotate_granularity": "test_room"
    }]
}
```

Figure 4.2(a): Input Specification File for Retrieve Operation with Include Annotation Selection
{  
    "compute":  
    {  
        "operation": "retrieve",  
        "from_date": "2018-07-21 00:00:00",  
        "to_date": "2018-07-21 23:55:00"  
    },  
    "select": "NWC.VV331:ROOM TEMP",  
    "source": "TEST_TEC_SYSTEM_DATA",  
    "annotation": [{  
        "annotate": "exclude",  
        "annotate_type": "CS",  
        "annotate_granularity": "test_room"  
    }]  
}

Figure 4.2(b): Input Specification File for Retrieve Operation with Exclude Annotation Selection

<table>
<thead>
<tr>
<th>Start Time</th>
<th>End Time</th>
<th>Temperature Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-07-21 10:00:00</td>
<td>2018-07-21 10:01:00</td>
<td>70.25</td>
</tr>
<tr>
<td>2018-07-21 10:05:00</td>
<td>2018-07-21 10:06:00</td>
<td>70.25</td>
</tr>
<tr>
<td>2018-07-21 10:10:00</td>
<td>2018-07-21 10:11:00</td>
<td>70.25</td>
</tr>
<tr>
<td>2018-07-21 10:15:00</td>
<td>2018-07-21 10:16:00</td>
<td>70.25</td>
</tr>
</tbody>
</table>

Figure 4.3(a): Results for Retrieve Operation with Include Annotation Selection

<table>
<thead>
<tr>
<th>Start Time</th>
<th>End Time</th>
<th>Temperature Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-07-21 10:00:00</td>
<td>2018-07-21 10:01:00</td>
<td>70.25</td>
</tr>
<tr>
<td>2018-07-21 10:05:00</td>
<td>2018-07-21 10:06:00</td>
<td>70.25</td>
</tr>
<tr>
<td>2018-07-21 10:10:00</td>
<td>2018-07-21 10:11:00</td>
<td>70.25</td>
</tr>
<tr>
<td>2018-07-21 10:15:00</td>
<td>2018-07-21 10:16:00</td>
<td>70.25</td>
</tr>
</tbody>
</table>

Figure 4.3(b): Results for Retrieve Operation with Exclude Annotation Selection

I selected a day’s worth of real data from room EB 365 from the BuDS database for performing two additional checks. The data chosen for this test is room temperature values on 25th September 2017, the first day of Fall 2017 quarter. There were two class sessions in room from 10:15 AM to 12:05 PM and from 2:00 PM to 3:50 PM. A total of 288 room temperature values have been recorded for the room throughout

68
the day in the BuDS database. I carried out two additional tests which serve as supplementary checks to the Retrieve Operation with the include-exclude feature, which are listed as follows:

(1) Check if the intersection of the datasets returned by the Retrieve operation with include filter and Retrieve operation with exclude filter is empty. The query and the results of these tests are shown in Figures 4.4(a) and Figure 4.4(b) respectively. As seen in the figure, the result contains no rows and hence the intersection of the two datasets is empty.

(2) Check if the union of the datasets returned by the Retrieve operation with include filter and Retrieve operation with exclude filter is the complete dataset. The query and the results of these tests are shown in Figures 4.5(a) and Figure 4.5(b) respectively. The total number of rows in the result is 288 and hence the results confirm that the union of these two datasets is the complete dataset.

These tests help validate the correctness of the implementation of the Retrieve operator in the service.
Figure 4.4(a): Query to return the intersection of the datasets returned by include and exclude filter of Retrieve Operation

```sql
WITH incl_dataset AS
(SELECT tec.tec_system_point_start_timestamp, 
tec.tec_system_point_end_timestamp, 
tec.float_value, 
rann.annotation_start_timestamp, 
rann.annotation_end_timestamp, 
rann.annotation_type_code, 
rann.annotation_type_value, 
rann.annotation_value
FROM smart_buildings."TEC_SYSTEM_DETAIL_DATA_1" tec
INNER JOIN smart_buildings."ROOM_LEVEL_ANNOTATION_INSTANCE" rann 
ON tec.tec_system_point_start_timestamp >= rann.annotation_start_timestamp 
AND tec.tec_system_point_end_timestamp <= rann.annotation_end_timestamp 
AND rann.annotation_type_code = 'CS'
WHERE tec.tec_system_point = 'UMC.VV331:ROOM TEMP'
AND tec.tec_system_point_start_timestamp BETWEEN '2017-09-25 00:00:00' AND '2017-09-25 23:55:00')

, excl_dataset AS
(SELECT tec.tec_system_point_start_timestamp, 
tec.tec_system_point_end_timestamp, 
tec.float_value, 
rann.annotation_start_timestamp, 
rann.annotation_end_timestamp, 
rann.annotation_type_code, 
rann.annotation_type_value, 
rann.annotation_value
FROM smart_buildings."TEC_SYSTEM_DETAIL_DATA_1" tec
LEFT OUTER JOIN smart_buildings."ROOM_LEVEL_ANNOTATION_INSTANCE" rann 
ON tec.tec_system_point_start_timestamp >= rann.annotation_start_timestamp 
AND tec.tec_system_point_end_timestamp <= rann.annotation_end_timestamp 
AND rann.annotation_type_code = 'CS'
WHERE tec.tec_system_point = 'UMC.VV331:ROOM TEMP'
AND tec.tec_system_point_start_timestamp BETWEEN '2017-09-25 00:00:00' AND '2017-09-25 23:55:00'
AND rann.annotation_start_timestamp IS NULL)

SELECT * FROM incl_dataset 
INTERSECT 
SELECT * FROM excl_dataset
```

Figure 4.4(b): Result of the intersection of the datasets returned by include and exclude filter of Retrieve Operation
WITH incl_dataset AS
(SELECT tec.tec_system_point_start_timestamp,
tec.tec_system_point_end_timestamp,
tec.float_value,
rann.annotation_start_timestamp,
rann.annotation_end_timestamp,
rann.annotation_type_code,
rann.annotation_type_value,
rann.annotation_value
FROM smart_buildings."TEC_SYSTEM_DETAIL_DATA_I" tec
INNER JOIN smart_buildings."ROOM_LEVEL_ANNOTATION_INSTANCE" rann
ON tec.tec_system_point_start_timestamp >= rann.annotation_start_timestamp
AND tec.tec_system_point_end_timestamp <= rann.annotation_end_timestamp
AND rann.annotation_type_code = 'CS'
WHERE tec.tec_system_point = 'SNC.YY331:ROOM TEMP'
AND tec.tec_system_point_start_timestamp BETWEEN '2017-09-25 00:00:00' AND '2017-09-25 23:55:00')
,excl_dataset AS
(SELECT tec.tec_system_point_start_timestamp,
tec.tec_system_point_end_timestamp,
tec.float_value,
rann.annotation_start_timestamp,
rann.annotation_end_timestamp,
rann.annotation_type_code,
rann.annotation_type_value,
rann.annotation_value
FROM smart_buildings."TEC_SYSTEM_DETAIL_DATA_I" tec
LEFT OUTER JOIN smart_buildings."ROOM_LEVEL_ANNOTATION_INSTANCE" rann
ON tec.tec_system_point_start_timestamp >= rann.annotation_start_timestamp
AND tec.tec_system_point_end_timestamp <= rann.annotation_end_timestamp
AND rann.annotation_type_code = 'CS'
WHERE tec.tec_system_point = 'SNC.YY331:ROOM TEMP'
AND tec.tec_system_point_start_timestamp BETWEEN '2017-09-25 00:00:00' AND '2017-09-25 23:55:00'
AND rann.annotation_end_timestamp IS NULL)
SELECT * FROM incl_dataset
UNION ALL
SELECT * FROM excl_dataset

Figure 4.5(a): Query to return the union of the datasets returned by include and exclude filter of Retrieve Operation

Figure 4.5(b): Results of the union of the datasets returned by include and exclude filter of Retrieve Operation

Figure 4.5(c): Number of rows in the result of the union of the datasets returned by Retrieve operation with include filter and Retrieve operation with exclude filter

71
4.3 VALIDATION OF THE CORRELATE OPERATION

I selected a day’s worth of real data from room EB 365 from the BuDS database for evaluation of the Correlate operator. Through research, I understood that both PostgreSQL and Microsoft Excel use Pearson Correlation to compute the correlation coefficient between two sets of numbers. Hence, I compute the correlation coefficient of the chosen dataset in Microsoft Excel and compare the results with the results fetched by executing the query generated by my AEA service, SQLConverter. The expected correlation coefficient between room temperature and Outside Air Temperature when there were sessions in the room is computed in Microsoft Excel and the result is as shown in Figure 4.6. The include correlation coefficient between the observations is 0.719 and the exclude correlation coefficient is 0.948.
Figure 4.6: Include and Exclude Correlation Coefficients between EB 365 room temperature and outside air temperature computed in Microsoft Excel

I then create an input specification file for the Correlation operation, which is shown in Figure 4.7(a), and run the service with this input. The results after executing the query in the BuDS database is 0.718579579304187 as shown in Figure 4.7(b). It can be observed that the correlation coefficient computed by the service is similar to the one computed in Microsoft Excel with exception that Postgres’s CORR () operator returns a correlation coefficient with more significant figures than Microsoft Excel.
Figure 4.7(a): Input Specification file for Correlation between EB 365 room temperature and outside air temperature when there were classes in the room
Figure 4.7(b): Query and Result for Correlation between EB 365 room temperature and outside air temperature when there were classes in the room.

The same evaluation method is used for Correlate operation with exclude annotation selection. Figure 4.8(a) shows the input specification file used for the evaluation and 4.8(b) shows the query generated by the service and the result of executing it in the BuDS database which is equal to 0.948374601734865.
Figure 4.8(a): Input Specification file for Correlation between EB 365 room temperature and outside air temperature when there were NO classes in the room
It can be observed that the results of the Correlate operator from my AEA service match the results computed in Microsoft Excel. This test helps validate the correctness of the implementation of the Correlate operator in the service.

4.4 VALIDATION OF THE ALIGNED-AVERAGE OPERATION

I created a synthetic temperature and annotation dataset for room EB 365 on 22\textsuperscript{nd} July 2018 (Sunday). Figure 4.9(a) shows the room temperature data and Figure 4.9(b) shows the class schedule information for the same day. Figure 4.9(c) shows a
visualization of the room temperature for EB 365 on 22\textsuperscript{nd} July 2018 annotated with Class Schedule information.

It can be observed from the figure that there is a steady rise in room temperature when there are sessions in the room and immediate steady drop in temperature after the end of the session. The room temperature values I used during class sessions are between 73 degrees Fahrenheit and 77.5 degrees Fahrenheit with a step size of 0.25-degree Fahrenheit. I created five sessions of thirty-five minutes duration each. I constructed the data in such a way that the Aligned-Average profile of this dataset would be the same as the room temperature profile of the middle (third) class session of the day. The idea of this evaluation technique is to compute the time-by-time aligned-average profile of room temperature when there were classes in the room for this dataset and check if the average profile has room temperature values equal to the room temperature values during the third session between 12:05 PM and 12:40 PM shown in Figure 4.9(d).
**Figure 4.9(a):** Synthetic dataset for EB 365 Room Temperature

<table>
<thead>
<tr>
<th>tec_system_point</th>
<th>tec_system_point_start_timestamp</th>
<th>tec_system_point_end_timestamp</th>
<th>float_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>character varying(45)</td>
<td>timestamp without time zone</td>
<td>timestamp without time zone</td>
<td>numeric(7,2)</td>
</tr>
<tr>
<td>1</td>
<td>2018-07-22 01:00:00</td>
<td>2018-07-22 01:05:00</td>
<td>73.25</td>
</tr>
<tr>
<td>2</td>
<td>2018-07-22 01:10:00</td>
<td>2018-07-22 01:15:00</td>
<td>73.25</td>
</tr>
<tr>
<td>3</td>
<td>2018-07-22 01:20:00</td>
<td>2018-07-22 01:25:00</td>
<td>73.25</td>
</tr>
<tr>
<td>4</td>
<td>2018-07-22 01:30:00</td>
<td>2018-07-22 01:35:00</td>
<td>73.25</td>
</tr>
<tr>
<td>5</td>
<td>2018-07-22 01:40:00</td>
<td>2018-07-22 01:45:00</td>
<td>73.25</td>
</tr>
<tr>
<td>6</td>
<td>2018-07-22 01:50:00</td>
<td>2018-07-22 01:55:00</td>
<td>73.25</td>
</tr>
<tr>
<td>7</td>
<td>2018-07-22 02:00:00</td>
<td>2018-07-22 02:05:00</td>
<td>73.25</td>
</tr>
<tr>
<td>8</td>
<td>2018-07-22 02:10:00</td>
<td>2018-07-22 02:15:00</td>
<td>73.25</td>
</tr>
<tr>
<td>9</td>
<td>2018-07-22 02:30:00</td>
<td>2018-07-22 02:35:00</td>
<td>73.25</td>
</tr>
</tbody>
</table>

**Figure 4.9(b):** Synthetic dataset for Class Schedule Annotation for room EB 365

<table>
<thead>
<tr>
<th>building_code</th>
<th>room_number</th>
<th>annotation_type_code</th>
<th>annotation_type_value</th>
<th>annotation_start_timestamp</th>
<th>annotation_end_timestamp</th>
<th>annotation_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>character varying(45)</td>
<td>integer</td>
<td>character varying(16)</td>
<td>character varying(8)</td>
<td>timestamp without time zone</td>
<td>timestamp without time zone</td>
<td>numeric(10,2)</td>
</tr>
<tr>
<td>1</td>
<td>EB</td>
<td>365.CS</td>
<td>R1-101-001</td>
<td>2018-07-22 10:15:00</td>
<td>2018-07-22 10:50:00</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>EB</td>
<td>365.CS</td>
<td>R1-101-001</td>
<td>2018-07-22 11:10:00</td>
<td>2018-07-22 11:45:00</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>EB</td>
<td>365.CS</td>
<td>R1-101-001</td>
<td>2018-07-22 12:00:00</td>
<td>2018-07-22 12:45:00</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>EB</td>
<td>365.CS</td>
<td>R1-101-001</td>
<td>2018-07-22 13:00:00</td>
<td>2018-07-22 13:35:00</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>EB</td>
<td>365.CS</td>
<td>R1-101-001</td>
<td>2018-07-22 13:55:00</td>
<td>2018-07-22 14:30:00</td>
<td>31</td>
</tr>
</tbody>
</table>
Figure 4.9(c): Plot of room temperature with Class Schedule Annotation for the synthetic dataset

Figure 4.9(d): Room Temperature values for the third session in synthetic dataset

Figure 4.10 shows the input file that represents the query:

“Compute the time-by-time aligned-average profile of room temperature in room EB 365 when there were class sessions in the room between 8 AM and 2 PM on 22nd July 2018. Also, include 3 hours before and 3 hours after the class in the average.”
Figure 4.10: Input Specification File for Aligned-Average operation for EB 365 room temperature synthetic dataset

I then executed the SQL query generated by the AEA service in the Building Data database and visualized the result in Jupyter notebook. The tabular results are shown in Figure 4.11(a) and the visualization of the results is shown in Figure 4.11(b).
Figure 4.11 (a): Results for Aligned-Average of room temperature for EB 365 synthetic dataset

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Temperature</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>-01:00:00</td>
<td>73.85</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>-00:55:00</td>
<td>73.95</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>-00:50:00</td>
<td>74.15</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>-00:45:00</td>
<td>74.35</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>-00:40:00</td>
<td>74.55</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>-00:35:00</td>
<td>74.75</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>-00:30:00</td>
<td>74.95</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>-00:25:00</td>
<td>75.15</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>-00:20:00</td>
<td>74.95</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>-00:15:00</td>
<td>74.75</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>-00:10:00</td>
<td>74.55</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>-00:05:00</td>
<td>74.35</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>00:00:00</td>
<td>74.50</td>
<td>1</td>
</tr>
<tr>
<td>38</td>
<td>00:05:00</td>
<td>74.75</td>
<td>1</td>
</tr>
<tr>
<td>39</td>
<td>00:10:00</td>
<td>75.00</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>00:15:00</td>
<td>75.25</td>
<td>1</td>
</tr>
<tr>
<td>41</td>
<td>00:20:00</td>
<td>75.50</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>00:25:00</td>
<td>75.75</td>
<td>1</td>
</tr>
<tr>
<td>43</td>
<td>00:30:00</td>
<td>76.00</td>
<td>1</td>
</tr>
<tr>
<td>44</td>
<td>00:35:00</td>
<td>75.75</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>00:40:00</td>
<td>75.50</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>00:45:00</td>
<td>75.25</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>00:50:00</td>
<td>75.00</td>
<td></td>
</tr>
</tbody>
</table>
It can be seen from the result of aligned-average that the same data pattern of increase in room temperature is maintained when there are sessions in class and the room temperature also drops steadily after the end of the session. The average room temperature values when there were classes equals the room temperature values of the third session. Hence, the results validate that the Aligned-Average operation implementation in the SQLConverter service is correct.

4.5 PERFORMANCE EVALUATION

The optimizer in a Database Management System (DBMS) considers different plans for an SQL query and selects one based on the lowest estimated cost, which is in turn based on the estimates of sizes of the intermediate results. But there is information that the optimizer might not know such as the fact that a data record usually joins with at most one annotation record in the BuDS database. In such instances, the
optimizer's estimates might not be accurate and hence it might miss a good plan. This possibility motivates the need for evaluating the performance of the queries.

For each of the operations in the AEA service, I evaluate the query generated by my service against some chosen alternatives and compare their performance with respect to execution time. I used PostgreSQL’s “EXPLAIN” command with “ANALYZE” option to gather the runtime statistics of the execution of each SQL query and to inspect the plans that the DBMS chooses. These tests were carried out on a Postgres 9.5.13 database running on Ubuntu 16.04 with 8-core 2.5GHz Intel(R) Xeon(R) CPU and 8.17 GB of RAM.

4.5.1 PERFORMANCE OF RETRIEVE OPERATION

The Retrieve operation with include annotation filter does not have obvious alternative translations, hence I focus on evaluating the performance of the Retrieve operation with the exclude annotation filter. The Retrieve operation with exclude annotation filter fetches all the data that are outside the annotated time periods. An equivalent query can be written for this operation using the “NOT EXISTS” clause or the “EXCEPT” SQL operator. The query that I use for this evaluation is:

“Retrieve room temperature information from room “EB 365” in the time range of 24th September 2017 to 6th December 2017 when there were NO class sessions in the room”

I present the results of running this query using different alternatives for the Retrieve operation with exclude annotation selection.
Figure 4.12(a): Query constructed by the AEA service, SQLConverter, for Retrieve operation with exclude Class Schedule Annotation, Fall 2017

```
EXPLAIN ANALYZE
SELECT
tec.tec_system_point_start_timestamp,
tec.tec_system_point_end_timestamp,
tec.float_value,
ran.annotation_start_timestamp,
ran.annotation_end_timestamp,
ran.annotation_type_code,
ran.annotation_type_value,
ran.annotation_value
FROM smart_buildings."TEC_SYSTEM_DETAIL_DATA_1" tec
LEFT OUTER JOIN smart_buildings."ROOM_LEVEL_ANNOTATION_INSTANCE" ran
ON tec.tec_system_point_start_timestamp > ran.annotation_start_timestamp
AND tec.tec_system_point_end_timestamp <= ran.annotation_end_timestamp
AND ran.annotation_type_code = 'CS'
WHERE tec.tec_system_point = 'NNC.VV331:ROOM TEMP'
AND ran.annotation_start_timestamp IS NULL
ORDER BY tec.tec_system_point_start_timestamp;
```

Figure 4.12(b): Runtime statistics for query constructed by the AEA service, SQLConverter, for Retrieve operation with exclude Class Schedule Annotation, Fall 2017

```
```

Figure 4.13(a): Alternative query using “NOT EXISTS” clause for Retrieve operation with exclude Class Schedule Annotation, Fall 2017

```
EXPLAIN ANALYZE
SELECT
tec.tec_system_point_start_timestamp,
tec.tec_system_point_end_timestamp,
tec.float_value
FROM smart_buildings."TEC_SYSTEM_DETAIL_DATA_1" tec
WHERE NOT EXISTS
(SELECT annotation_value
FROM smart_buildings."ROOM_LEVEL_ANNOTATION_INSTANCE" ran
WHERE tec.tec_system_point_start_timestamp >= ran.annotation_start_timestamp
AND tec.tec_system_point_end_timestamp <= ran.annotation_end_timestamp
AND ran.annotation_type_code = 'CS'
AND tec.tec_system_point = 'NNC.VV331:ROOM TEMP'
AND ran.annotation_start_timestamp BETWEEN '2017-09-24 00:00:00' AND '2017-12-06 00:00:00'
ORDER BY tec.tec_system_point_start_timestamp;
```

Figure 4.13(a): Alternative query using “NOT EXISTS” clause for Retrieve operation with exclude Class Schedule Annotation, Fall 2017
Figure 4.13(b): Runtime statistics for alternative query using “NOT EXISTS” clause for Retrieve operation with exclude Class Schedule Annotation, Fall 2017

Figure 4.14(a): Alternative query using “EXCEPT” operator for Retrieve operation with exclude Class Schedule Annotation, Fall 2017
Figure 4.14(b): Runtime statistics for alternative query using “EXCEPT” operator for Retrieve operation with exclude Class Schedule Annotation, Fall 2017

The query (OUTER JOIN variant) generated by the AEA service and the runtime statistics for the query are shown in Figure 4.12(a) and 4.12(b) respectively. As seen in Figure 4.12(a), the Retrieve operation with exclude annotation selection is written as LEFT OUTER JOIN and selection of NULLS in the annotation columns. An equivalent query using “NOT EXISTS” clause (NOT EXISTS variant) is shown in Figure 4.13(a) and the results of the runtime statistics of this query is shown in Figure 4.13(b). It can be seen that the optimizer is able to correctly identify that the kind of join is ANTI JOIN between the data and annotation table in both the variants. Another observation is that the optimizer generates a similar execution plan for both the OUTER JOIN variant and the NOT EXISTS variant. Figure 4.14(a) shows an equivalent query written using the “EXCEPT” operator (EXCEPT variant) and Figure 4.14(b) shows its results. The EXCEPT variant computes the difference of the dataset included in an annotation instance from the entire dataset of that observation. Though using the EXCEPT variant for writing this query is the more obvious solution, it can be seen that
the optimizer does not figure out that this query is an ANTI JOIN between the data
table and has a more sub-optimal query plan where the optimizer scans the same data
twice.

I ran each of the queries five times, discarded the most expensive and the least
expensive runtimes and averaged the other three runtimes. Table 4.1 summarizes
these results. The runtimes of the different alternatives show that the translation of
the query can play an important role in performance.

<table>
<thead>
<tr>
<th>Query Alternatives</th>
<th>Average Runtimes (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTER JOIN Variant constructed by AEA Service</td>
<td>1.086</td>
</tr>
<tr>
<td>NOT EXISTS Variant</td>
<td>1.094</td>
</tr>
<tr>
<td>EXCEPT Variant</td>
<td>2.297</td>
</tr>
</tbody>
</table>

Table 4.1: Query Alternatives for Retrieve operation with exclude annotation selection and their
average runtimes
I use temporary datasets for storing the two data sources for which the correlation coefficient has to be computed. I call this approach as the TEMP variant. I adopted this strategy in order to reuse the Retrieve operation as a sub-operation in the Correlate operation. An alternative way to write this query is to directly join the two datasets without using temporary datasets and computing the correlation coefficient. I call this approach as the DIRECT JOIN variant. The query that I use for this evaluation is:

“Retrieve room temperature information from room “EB 365” in the time range of 24\textsuperscript{th} September 2017 to 6\textsuperscript{th} December 2017 when there were class sessions in the room. Retrieve Outside Air Temperature (OAT) information in the time range of 24\textsuperscript{th} September 2017 to 6\textsuperscript{th} December 2017. Compute the correlation coefficient of these two measures when both are present.”

I evaluate the performance of this query with respect to execution time for both the alternatives in this subsection.

Figure 4.15(a) shows the TEMP variant query generated by my service for computing the correlation measure between EB365 room temperature and outside air temperature when there were classes in the room for the Fall 2017 quarter. Figure 4.15(b) shows the runtime statistics of this query. Figure 4.16(a) shows the DIRECT JOIN variant in which a direct join is used between the two data sources and the correlation coefficient between the observations is computed on the joined dataset. Figure 4.16(b) shows the runtime statistics of the alternate translation. It can be
observed that the average runtime of both the queries is almost similar as seen in Table 4.2. The only difference in the execution plan of the two queries is that the result of dataset2 in the TEMP variant generated by the AEA service is materialized and ends up being marginally faster than the DIRECT JOIN variant as materializing a subquery implies storing the result of this subquery in memory during the execution of the query for faster access.

Figure 4.15(a): Query constructed by the AEA service, SQLConverter, for Correlate operation with include Class Schedule Annotation, Fall 2017
Figure 4.15(b): Runtime statistics for query constructed by the AEA service, SQLConverter, for Correlate operation with include Class Schedule Annotation, Fall 2017

Figure 4.16(a): Alternative query using direct join for Correlate operation with include Class Schedule Annotation, Fall 2017
Figure 4.16(b): Runtime statistics for alternative query using direct join for Correlate operation with include Class Schedule Annotation, Fall 2017

<table>
<thead>
<tr>
<th>Query Alternatives</th>
<th>Average Runtimes (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMP variant constructed by the AEA service</td>
<td>1.195</td>
</tr>
<tr>
<td>DIRECT JOIN variant</td>
<td>1.219</td>
</tr>
</tbody>
</table>

Table 4.2: Query Alternatives for Correlate operation and their corresponding average runtimes
4.5.3 PERFORMANCE OF ALIGNED-AVERAGE OPERATION

I aligned the data according to the start and end times of annotation by offsetting the time of occurrence of an observation by the start time of its corresponding annotation instance for computing the time-by-time average profile of an observation in the Aligned-Average operation. I call this approach the OFFSET variant. Another way to accomplish the alignment of datasets is by using the RANK () function. An alignment id is created by using the RANK () function of PostgreSQL ordered by time for each data record within the annotation instance that it is mapped to. Once the data is aligned using this alignment id, I compute the time-by-time average profile of an observation, grouping them by the alignment id. I initially used the RANK variant for generating the SQL query in the AEA service, and the performance evaluation revealed that the OFFSET variant yielded better plans that led me to revise the implementation of the Aligned-Average operator in the service. The query used for this evaluation is:

“Compute the time-by-time aligned-average profile of room temperature in EB 365 when there were classes between 8AM and 12 PM in the time range of 24th September 2017 and 6th December 2017 over a day. Also, include 3 hours before and 2 hours after the class in the average.”

I evaluate the performance of this query with respect to execution time for both the alternatives in this subsection.
Figure 4.17(a): Query constructed by the AEA service, SQLConverter, for Aligned-Average operation using OFFSET technique with Class Schedule Annotation, Fall 2017

```sql
SELECT
    annotated.annot_data.start_timestamp,
    annotated.annot_data.end_timestamp,
    annotated.annot_data.annotation_type,
    annotated.annot_data.annotation_value,
    annotated.annot_data.annotation_start_timestamp - interval '2 hour' as daters arrangement,
    annotated.annot_data.annotation_end_timestamp + interval '2 hour' as daters arrangement,
    EXTRACT(DAY FROM annotated.annot_data.annotation_start_timestamp) AS datersweek,
FROM smart_buildings."ROOM_LEVEL_ANNOTATION_INSTANCE" annotated,
WHERE DATE_TRUNC('day', annotated.annot_data.annotation_start_timestamp) = date_trunc('day', '2017-09-24 00:00:00')
AND DATE_TRUNC('day', annotated.annot_data.annotation_end_timestamp) = date_trunc('day', '2017-12-04 00:00:00')
AND EXTRACT(HOUR FROM annotated.annot_data.annotation_start_timestamp) < '12:00'
AND EXTRACT(HOUR FROM annotated.annot_data.annotation_end_timestamp) < '12:00'
INNER JOIN
    smart_buildings."TEC_SYSTEM_DETAIL_DATA" tec
ON annotated.annot_data.annotation_point_start_timestamp = tec.tec_system_point_start_timestamp
AND annotated.annot_data.annotation_point_end_timestamp = tec.tec_system_point_end_timestamp
ORDER BY annotated.annot_data.start_timestamp
```

Figure 4.17(b): Runtime statistics for alternate query for Aligned-Average operation using OFFSET technique with Class Schedule Annotation, Fall 2017
Figure 4.18(a): Alternate query for Aligned-Average operation with Class Schedule Annotation, Fall 2017

```
EXPLAIN ANALYZE WITH annotation AS (SELECT
  annotation_start_timestamp,
  annotation_end_timestamp,
  annotation_type_value,
  annotation_value,
  annotation_start_timestamp - interval '1 hour' as start_timestamp,
  annotation_end_timestamp + interval '2 hour' as end_timestamp,
  to_char(annotation_start_timestamp, 'day') as day_of_week,
  EXTRACT (EPOCH FROM (annotation_end_timestamp - annotation_start_timestamp))/86400 as duration,
  row_number() OVER () as annot_id
FROM smart_buildings
WHERE annotation_start_timestamp BETWEEN '2017-09-25 00:00:00' AND '2017-09-25 23:59:59'
AND annotation_end_timestamp BETWEEN '2017-12-04 00:00:00' AND '2017-12-04 23:59:59'
AND TO_CHAR(annotation_start_timestamp, 'HH24:MI') < '12:00'
)
, annot_data AS (SELECT
  tec.tec_system_point_start_timestamp,
  tec.tec_system_point_end_timestamp,
  tec.float_value,
  annot.annot_id,
  annot.annotation_start_timestamp,
  annot.annotation_end_timestamp,
  annot.day_of_week,
  annot.annotation_duration,
  CASE WHEN tec.tec_system_point_start_timestamp BETWEEN annot.annotation_start_timestamp AND annot.annotation_end_timestamp THEN 1
  ELSE 0 END AS annot_indicator,
  rank() OVER (PARTITION BY annot_id ORDER BY tec.tec_system_point_start_timestamp) AS align_id
FROM smart_buildings
LEFT JOIN annot
ON tec.tec_system_point_start_timestamp BETWEEN annot.annotation_start_timestamp AND annot.annotation_end_timestamp
AND tec.tec_system_point_end_timestamp BETWEEN annot.annotation_start_timestamp AND annot.annotation_end_timestamp
WHERE tec.tec_system_point = 'DEC.0711:ROOM TEMP'
)
SELECT
  annot.annot_align_id,
  avg(annot.annot_data.float_value) AS aligned_avg,
  max(annot.annot_data.annot_indicator) AS annot_indicator
FROM annot_data
GROUP BY annot.annot_align_id
ORDER BY annot.annot_align_id
```

Figure 4.18(b): Runtime statistics for query constructed by the AEA service, SQLConverter, for Aligned-Average operation with Class Schedule Annotation, Fall 2017
<table>
<thead>
<tr>
<th>Query Alternatives</th>
<th>Average Runtimes (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFFSET variant constructed by the service</td>
<td>26.73</td>
</tr>
<tr>
<td>RANK variant</td>
<td>33.74</td>
</tr>
</tbody>
</table>

Table 4.3: Query Alternatives for Aligned-Average operation and their corresponding average runtimes

The query generated by the AEA service for computing the Aligned-Average profile of room temperature in EB 365 when there are sessions in the room between 8 AM and 12 PM for Fall 2017 is shown in Figure 4.17(a). The runtime statistics of this query are shown in Figure 4.17(b). The alternative approach to compute the Aligned-Average profile using RANK technique is shown in Figure 4.18(a) and the runtime statistics of the query is shown in Figure 4.18(b). The average runtimes for both the OFFSET approach and RANK approach is shown in Table 4.3. It can be noted that the OFFSET variant is faster compared to the RANK variant and this might be because the RANK variant involves sorting the dataset for computing the rank for each data row within an annotation instance, which is avoided in the OFFSET variant.

I conclude that the initial query plan generated by the optimizer does make a difference in the performance of the query. I also adjusted the implementation of the Aligned-Average operator based on the results revealed by performance evaluation.
5. RELATED WORK

The concept of data annotation has been around for a while and there have been many applications for it such as commenting, search and provenance. We note that data annotations existed even in non-digital time-series data. For instance, the Wow! Signal discovered by an astronomer, Jerry R. Ehman, has a handwritten annotation added by him on an unusual radio signal received on August 15\textsuperscript{th}, 1977 by a radio telescope at Ohio State University [9]. The radio signal was used for the search of extraterrestrial intelligence. While there have been many applications for annotations in the past, Annotation-Enabled Analysis described in this thesis employs annotations for performing calculations on data.

The work related to using annotations in data analysis can be broadly classified into the following annotation use categories: (1) annotations in visualization, (2) annotations in time-series data, (3) annotation propagation, (4) support for annotations in a new class of databases called Time-Series Databases, (5) annotations in non-time-series data, (6) annotations in scientific data and (7) energy usage analysis in buildings. Figure 5.1 depicts the kinds of data, annotations and the uses of annotations that have been explored in literature.
5.1 ANNOTATIONS IN VISUALIZATION

Kibana is a visual exploration tool used for real-time data analysis in Elasticsearch and this tool has the facility to annotate data [10]. Francoeur, in his tutorial, gives a procedure for representing annotations generated by Machine Learning Jobs on the visualization of NginX logs dataset in Kibana [11]. The annotations generated by the job are used to detect unusual response code rates in the NginX web logs. He selects event rates, splits them by response codes and also selects remote IP as one of the influencers for the Machine Learning job. The job discovers some anomalies on certain dates where there were a lot of pages that had a 404 error and these dates are marked as anomalies. He creates visualizations that use annotations from the ML job along with actual data (web logs). He uses bomb markers for depicting annotations that represent anomalies in the visualization. Finally, the visualization that has
response codes from the NginX web logs annotated with anomalies is saved and can be used with any other filters.

Google Charts are interactive charts developed by Google for browsers and mobile devices [12]. They have the option of specifying annotations on their visualizations. Each Y value on the chart has fields such as “Annotation Title” and “Annotation Text” to go with it. There are settings for the annotations to be displayed and used as filters in the charts. As seen in Figure 5.2, the markers A and B are annotations added to the Google chart, which shows the number of pens and pencils sold over time.

![Figure 5.2: Annotations in Google Charts [12]](image)

Kibana, Google Charts and Annotation-Enabled Analysis have the capability of annotating time periods and visualizing data and annotations together in a graphical plot. While annotations are predominantly used for visualization purposes in Kibana and Google Charts, in Annotation-Enabled analysis, they are used for selecting subsets of data and performing calculations on them.

Dennis, et al. designed and developed a visualization exploratory tool called DAVID: Database for Annotation, Visualization and Integrated Discovery for analysis of
genome data [13]. The tool consists of mainly four modules: (1) Annotation Tool, (2) GoCharts, (3) KeggCharts and (4) DomainCharts. The users can use the annotation tool for uploading annotations to the gene lists and the tool provides a consolidated view of both annotated and unannotated gene lists along with a list of available annotations that can be added to the gene lists. GoCharts provides a graphical view of the genes classified by categories defined by the Gene Ontology Consortium (GO) [14]. The users can choose among three classification types, such as biological process, molecular function and cellular component and five levels of annotations that represent coverage and specificity. KeggCharts provides a graphical view similar to GoCharts but have links to annotations available in the KEGG (Kyoto Encyclopedia of Genes and Genomes) database [15]. DomainCharts have the ability to depict the genes among PFAM protein domains [16].

Though DAVID is used for analyzing and visualizing genome sequences, there is a close resemblance between annotating genome sequences and annotating time-series data because both are sequential data. DAVID is a system that integrates information from multiple different sources and provides a way for the users to annotate gene data and share their annotations with other users. DAVID allows the user to drill down genes based on an annotation and inspect it for other analyses. This feature of DAVID is similar to the include and exclude annotation selection feature in Annotation-Enabled Analysis.
5.2 ANNOTATIONS IN TIME-SERIES DATA

Fouse, et al. built a system called ChronoViz that aids annotation, visualization and analysis of multimodal time-series data, particularly in data-intensive fields such as behavioral science [17]. The field poses the constant challenge of analysis of unique case studies combined with digital data collected from multiple sources necessitating the researchers to keep notes about values and time points from one data set while processing corresponding time points from another dataset. While analyzing digital data from heterogenous sources, researchers transcribe and code data in a variety of ways and it becomes even more difficult when the researchers have to combine their hand-written notes in their analyses. ChronoViz addresses these challenges by providing interactive visual representations of data from multiple sources along with their annotations. It helps the researchers to navigate this data in flexible ways.

The relation between ChronoViz and Annotation-Enabled Analysis described in this thesis is that both the approaches use annotations to analyze data from heterogenous sources and they both annotate time periods. What differentiates my technique from ChronoViz is that ChronoViz is mainly targeted towards analyzing media data such as audio files and video files, whereas my technique is primarily focused on analyzing numerical data.

Another notable work in this category is that of Pressly, who developed TSPad, a tablet-PC based software tool that is designed to allow researchers to effectively collaborate on multiple time-series datasets such as stock prices, historical climate data and EEGs during their analyses [18]. This tool provides a distributed system that
enables the researchers to browse, annotate and search time-series data and also allows them to view the annotations of other researchers. The researchers can annotate time-series data with the help of a mouse in PC or a stylus in a tablet by marking a region of the time-series data which is then stored in the database and is available for viewing in future for them as well as others. The search functionality in TSPad allows the researchers to perform a similarity search based on their annotations either by drawing a waveform with an ink stroke or by selecting a subset of data from the database.

While both TSPad and Annotation-Enabled Analysis employ annotations in analyzing time-series data, TSPad helps researchers find similar data in the annotated regions in other parts of the database whereas Annotation-Enabled Analysis uses annotations to select subsets of data for further analyses. This work is one of the few cases that used annotations for analysis. This kind of similarity search could be an extension to AEA capabilities.

Ramseth describes a system in his patent that is used to display annotations on electrocardiographic time-series data [19]. This system provides an interactive display that allows the operator to annotate events by marking the beginning and end of a lead wave and attaching a description to that wave. There are different levels of annotations such as session-level annotation, epoch-level annotation and channel-level annotation. Based on the annotated regions, median heart beats can be calculated and the annotated points are usually included in the patient's report.
Figure 5.3 shows a snapshot of this system illustrating the placement of annotation markers to identify waveform features.

![Figure 5: System for annotating ECG data, image obtained from the patent [19]](image)

A similarity between this system and Annotation-Enabled Analysis is that both approaches select subsets of data based on annotations and perform calculations on them. While Ramseth’s system has the ability to provide a visual interface using which the users can annotate time-series, this feature is currently not supported in the BuDS system, however, it can be seen as a future enhancement to annotation capabilities in BuDS.
Hjelsvold, et al. describe a data model that has the ability of capturing the contents of a video document and indexing video streams [20]. A query language for this data model is also discussed in this paper. The data model for a video document is defined as a hierarchy of structural components in the document. Each video document is represented as a video stream. A video stream is a collection of video segments where each segment is a collection of video frames. Any video document can be represented in the form of a set of frame sequences. The data model also has a provision to index the video documents. An annotation can be added to arbitrary frame sequence by specifying the frame sequence and the annotation to go with it. The data model that the authors designed for a Television News Archive supports three kinds of annotation: (1) person annotation, (2) location annotation and (3) event annotation. The query language that is designed for this data model had the capabilities such as browsing the structure of the video documents, retrieving all the annotations associated with a frame sequence and retrieving videos based on the contents of the videos added as annotations.

A similarity between the data model for video documents and the BuDS data model is that both store the data and annotations separately in different tables and both the query language supported for video documents and AEA provide ways to analyze data based on annotations. While the video query language predominantly is used for querying the video contents based on person, location and event annotations, Annotation-Enabled Analysis employs annotations for selecting subsets of data and performing calculations on them.
5.3 ANNOTATION PROPAGATION

Widom, et al. describe a database called Trio that stores and manages accuracy, lineage and coverage in addition to base data [21]. TriQL is a language developed to query the Trio database and it is equipped with techniques to answer queries based on these additional attributes of data.

In this database, accuracy is captured at the granularity of a tuple and individual values. The accuracy of an attribute value can fall in one of the following categories: (1) an exact value, (2) a set of possible values each having a probability in the range [0,1] such that the sum of probabilities of all the values is 1, (3) start and end of a range of values – Trio assumes uniform distribution across minimum and maximum of the range, (4) a Gaussian distribution over a range of possible values. The accuracy of a tuple is expressed as confidence value in Trio. Each tuple has a confidence value in the range [0,1] indicating the likelihood that it belongs in the relation. The accuracy of a relation is captured as coverage value in the range [0,1] indicating the fraction of missing records in that relation. Every instance in the Trio database is associated with a Lineage relation.

The Lineage relation captures the following information about each tuple: (1) tuple-id, (2) derivation-type, which describes the type of derivation of the tuple, for example: query-based, program-based or update-based (3) time, (4) how-derived, which contains the query text, program, update statement, etc. (5) lineage-data, which describes the history of tuples by a list of relations or lineage query, IDs of previous tuples, etc.
The approach specifies how the lineage, accuracy and coverage information can be computed for data resulting from SQL operations such as JOIN, UNION, INTERSECTION and AGGREGATION. For instance, if there are two tuples t1 and t2 with confidence values c1 and c2 in the Trio database, when these two tuples are combined as a result of a JOIN operation in a query, then the confidence of the resulting tuple t3 is computed based on the confidence values c1 and c2. The similarity between Annotation-Enabled Analysis technique, that I have described in this thesis, and Trio is that both approaches employ annotations for analysis. While Trio treats annotation and data separately and uses the annotation information such as accuracy and lineage to compute the annotation for new data, Annotation-Enabled Analysis employs annotations to analyze and compute data as depicted in Figure 5.4.

![Figure 5.4: Difference between AEA and Annotation Propagation systems](image)

Tan, et al. describe an annotation-management system for relational databases where each data point has zero or more annotations attached to it [22]. The paper provides a methodology for storing and propagating annotations in the database and also presents a new language called pSQL that is used for annotation propagation in the database. An implementation of this annotation management system is DBNotes [23]. Three different types of annotation propagation schemes are described, namely, (1)
the default scheme that propagates annotations based on the source from which the data was copied, (2) the default-all scheme, which is invariant to the syntax in of the SQL query and generates the same annotation for the result irrespective of the way the SQL query is written, i.e., it propagates annotations in such a way that it applies to all equivalent ways of writing an SQL query, and (3) the custom scheme, where the user can specify how they want to propagate the annotations to the result.

This system employs annotations for identifying the provenance of data by annotating each tuple with its address in the database. Another use of annotations is for capturing more information about data in the form of metadata. Annotations are also used to identify the quality or security of a piece of data in this system.

Karvounarakis, et al. describe a method for querying the provenance of data in Collaborative Data Sharing Systems (CDSS) through a new language that they developed called ProQL [24]. This query language is useful for acquiring information such as the derivation of a tuple, the relationship between two tuples and tuples with common provenance. This system represents provenance information of data in the form of a graph called a Provenance Graph. This graph data model tracks the relationship between tuples, thus indicating how a tuple was derived in the model. This system also tracks annotations such as scores, probabilities and counts of tuples from the provenance of a tuple.

Eltabakh, et al. bring up the challenges with respect to different annotation granularities such as at the table, tuple, column and cell levels and present efficient storage schemes for annotations based on their granularities [25]. Annotations are
stored in separate tables called annotation tables, which have columns such as Annotation Id, Annotation Value, Annotation CoveredCells (range of cells covered by an annotation item) and additional Boolean Flag columns, which indicate whether the annotations have to be propagated or not based on the operations performed on data, such as OnUpdate PropagateFlag, OnAggregate PropagateFlag and ViewAnnotation PropagateFlag. The annotations are added to the data using extensions to SQL such as the ADD ANNOTATION clause, which allows the user to specify the annotation value for a particular data and annotation propagation clauses such as ON UPDATE PROPAGATE, which specifies that the annotation in the data source needs to be retained in the result.

While these systems focus on methods for propagating annotations through query transformations, Annotation-Enabled Analysis focusses on analyzing data based on annotations attached to it. While one of the uses of annotations in Annotation-Enabled Analysis is to capture metadata about data to aid in data analysis, it uses these annotations for selecting subsets of data and performing computations on them rather than propagating annotations to its analysis results.

5.4 SUPPORT FORANNOTATIONS IN TIME-SERIES DATABASES

Time-series databases are an emerging class of databases that are optimized for managing time-series data. These systems have the practice of recording each and every change to the system as a new row along with the time at which the change occurred. This powerful feature of time-series data allows us to measure the change
and analyze how something changed in the past, check how it is changing in the present and how it will affect the future. The two main reasons to use time-series databases are: (1) Scale: Since we capture every small change in system as new data, it is clear that time-series data can become huge very quickly. Time-series databases are designed for this kind of data as they introduce efficiencies by treating time as a primary dimension and these databases can be either relational databases or NoSQL databases. (2) Usability: Time-Series databases usually have operations for time-series data analysis such as data retention policies, support for continuous queries and flexible time aggregations.

InfluxDB is one such open-source time-series database developed by InfluxData [26]. It is designed for fast, high-availability storage and retrieval of time-series data. Chronograf is a user interface of InfluxDB and it provides support for data annotations. Users can add annotations to their data plots to mark important events, providing context to the data being visualized. The annotations are added to all the graphs for that time range, so that it helps users analyze the effect of such important events. Both Chronograf and Annotation-Enabled Analysis utilize annotations in the context of adding more meaning to the analysis. While Chronograf just puts the annotations on the visualizations, Annotation-Enabled Analysis employs annotations to filter a subset of data for further analyses in addition to supporting annotations on visualizations.

TimescaleDB is another open-source time-series relational database engineered from PostgreSQL which is designed to make SQL scalable and usable for time-series data
While there are some operations that are supported for data analytics, there is no support in particular for annotations in TimescaleDB.

5.5 ANNOTATIONS IN NON-TIME-SERIES DATA

Agosti and Ferro describe a generic annotation model that can be used for annotating digital content [28]. This model provides a formal definition for annotations, distinguishes the use of annotations as metadata and content and aims at defining relationships between annotations and describing how annotations can be added to digital objects. A new concept called a sign of an annotation is defined in this model that specifies the way annotation is represented. The different kinds of signs specified in the model are textual, graphical, video and auditive.

Bays, et al. explain a method for storing annotations in the database and using them for querying [29]. This method has a provision to annotate data at different levels such as a relational table, a tuple or a cell. Multiple annotations can be added to a particular data item. This work also states the significance of storing annotations independently from data so that the base data is not affected when performing annotation-specific operations such as inserting or updating annotations. Each annotation is also attached to the person or the application it came from. The method also defines a user group as the context of the annotation author. Based on the context or the group the reader of the data item belongs to, the annotations are displayed to them and this is a way to ensure credibility of the annotations to the reader. The method describes different query modes that allow the user to query for a particular
data item and retrieve all the annotations that go with it, select an annotation type in
the context of underlying data, select an annotation type and all the data items that
fall under the category and selecting particular data item in the context of an
annotation type.

There are many similarities between Annotation-Enabled Analysis and this system
such as annotating data at different levels of data such as a table, a tuple or a cell.
Annotation-Enabled Analysis also has a provision to add multiple annotations to a
single data item and supports analyses on multiple annotation types. While this
system focusses on retrieving data and annotations based on conditions on both,
Annotation-Enabled Analysis performs operations on data such as correlation and
average in addition to a simple retrieval of data based on annotations.

5.6 ANNOTATIONS IN SCIENTIFIC DATA

Bose and Buneman state the importance of annotations in different scientific
databases and discuss how the annotations are handled in different ones [30].
Annotations are stored at different levels including at the data, tuple and schema
levels. The authors define a *co-ordinate system* as a mechanism for describing the item
to which an annotation is attached. For instance, in molecular biology, gene identifiers
in various databases were used as a co-ordinate system, but there has also been the
emergence of use of linear co-ordinates determined by genetic sequencing and the
authors point out that the mapping between the different co-ordinate systems is not
1-1. Use cases such as these call for a stable and durable co-ordinate systems in
scientific databases. The work gives examples of scientific annotation systems such as UniProt and BioDAS, which are databases that store information on bioinformatics, and AstroDAS, which stores information on celestial-object observations. An entry from the Uniprot database shows that annotations are stored for an entire entry and also for segments of data in an entry. AstroDAS has a web interface through which the users can store and query annotations and its main aim is to enable the scientists to store and share scientific findings with a larger community.

5.7 ENERGY USAGE ANALYSIS IN BUILDINGS

Doukas, et al. describe an intelligent decision support system that uses rule sets for effective energy management in buildings [31]. The decision support system architecture has five main components including (1) indoor sensors, which capture the state of the buildings such as room temperature, (2) outdoor sensors, which provide information about outdoor conditions such as temperature, relative humidity and luminance, (3) controllers such as switches and valves, (4) a decision unit, which can interact with the sensors and controllers, and (5) a database, which stores all information pertaining to building energy usage. The decision unit has a way to intervene and change the control parameters based on their deviation from the user requirements. The authors state that this ability of the decision unit to intervene when necessary can result in efficient use of energy in buildings in certain scenarios. They tested their decision unit in one of the buildings of ZENON S.A. company in Athens, Greece. Their analysis led to the following conclusions:
(1) There were almost no scenarios where there was discomfort due to room temperature and this was attributed towards effective relative humidity and air quality control.

(2) Luminance levels inside buildings were also considered to be within the normal ranges.

(3) Comparing the energy usage data from the year they used the decision unit with that of the previous year's revealed that there was significant energy savings.

Pottinger, et. al, describe an information quality assessment framework that is used for assessing the quality of Building Information Models (BIMs) utilized in facility management (FM) [32]. The authors state that high quality BIMs are essential for effective facility management. The framework that they developed for assessing the information quality of BIMs takes into consideration both the physical FM subjects (assets and spaces) as well as model characteristics (objects, attributes and spatial information). The information quality standards that they come up with also considers the needs of the owner. One of the primary purposes of introducing annotations in the BuDS database was to assess the quality of information received from the Building Management Systems. This quality assessment framework can be studied to come up with various dimensions in which the data quality can be assessed in the BuDS database.

The purpose of this chapter is to provide a summary of some of the notable work done in the area of annotations. Annotations have been used for various purposes including
data analysis, propagation, visualization and search. Prominent work in each of this
category along with its comparison to my work in annotations have been summarized
in this chapter. While the vast majority of the work in literature utilized annotations
for selection, propagation and visualization of data, Annotation-Enabled Interpretation
and Analysis described in this thesis not only uses annotations for these purposes but
also uses them for performing quantitative analysis of data. Some systems described
in this chapter also presented ideas for future work in Annotation-Enabled Analysis.
6. CONCLUSION AND FUTURE WORK

Annotation-Enabled Interpretation and Analysis described in this thesis has provided an approach to data analysis that employs annotations for selecting subsets of data and performing computations on them and visualizing them. In this chapter, I describe a possible roadmap for future work in Annotation-Enabled Interpretation and Analysis and provide a summary of my work in this thesis.

In Section 6.1, I describe possible ideas of future work in this area. I describe my role in the BuDS project in Section 6.2 and provide a summary of my work and concluding remarks in Section 6.3.

6.1 FUTURE WORK

The work in this thesis is an initial investigation into the utility of Annotation-Enabled Interpretation and Analysis. I believe that there is potential to increase the capabilities of AEA in terms of both extending the functionality of its current operators and introducing new operators for other functionalities.

A useful extension to current AEA operators is to have the Aligned-Average operator support data alignment based on annotation instances that are not of the same length. An instance illustrating the usefulness of this extended feature is if we wanted to compute the time-by-time average room temperature profile for a particular room when there were morning classes and the classes were not of the same duration. There are multiple ways in which one could define the behavior of the operator in this use case, such as padding the shorter dataset with NULL values or trimming the larger
dataset to the shorter one and then computing the average on top of the resultant dataset. We are still investigating which of the alternatives adds more meaning to the analysis. There is also a thought that the semantics of the operator should be independent of the domain in which it is used and hence providing these extensions as optional features, that a user could pick based on his or her use case, is more valuable than strictly opting for one kind of behavior.

A new operator that might be useful in Annotation-Enabled Analysis is that of similarity search analogous to the one described in TSPad [18]. Similarity search denotes selecting a subset of observation data based on a particular annotation type and looking for other similar instances of the observation in the database. This operator might be useful in identifying patterns in data that are not annotated. For instance, in Building Management System, as stated in Section 4.1, we observed an anomaly in air volume data during a Power Outage annotation, with similarity search, we might be able to identify similar trends in air volume data on other days when there are no annotations available in the database. While this kind of similarity search does not provide sufficient evidence to come to any conclusion about the data, it helps us narrow down subsets of data for further investigation.

Another area of interest is to test the adaptability of the Annotation-Enabled Analysis service on other kinds of sequence data such as genome data, since this service is mainly built for time-series data.

Analyzing the efficiency of the SQL translations generated by the service and comparing them with other alternative SQL translations is something that has proved
to be productive and has led to crucial design changes in the implementation of some SQL generators in the service. Hence, I believe that this is an area that we need to continue to focus on in the future with every feature extension and addition of new operators to the AEA service.

6.2 ROLE in BuDS

In this subsection, I provide a summary of my contributions in the BuDS project. I have been involved with BuDS project since the beginning of the implementation phase of the project. I have played a role in the following:

1. Design and Development of the database schema
2. Design and Development of the ETL workflow for data
3. Generation of data analysis plots in Jupyter Notebook
4. Design and Development of a web interface for the BuDS project
5. Generation of Annotation-Enabled Analysis (AEA) visualizations in Jupyter Notebook and web interface
6. Design and Development of AEA service with support for Operations: Retrieve, Correlate and Aligned-Average

6.3 CONCLUDING REMARKS

In this thesis, I created support for Annotation-Enabled Interpretation by creating annotation visualizations in data-analysis frameworks such as Jupyter Notebooks and the web interface used for the BuDS prototype. These visualizations that have annotations in addition to data help in identifying patterns in data that motivated the
use of annotations for more complex use cases and led to the creation of the Annotation-Enabled Analysis service.

I created a service called SQLConverter for Annotation-Enabled Analysis that takes a high-level specification in the form of key-value pairs, parses it and converts the specification into an SQL query that can then be executed in the database to fetch the results. This service was mainly created for users who want to issue annotation-enabled queries, but who are not very familiar with database programming, thus shielding them from low level schema details of how the annotations and data are stored in the database. I created support for three operators, namely, (1) Retrieve data with include or exclude annotation selection (2) Correlate data with include or exclude annotation selection (3) Align data based on include annotation selection to support aggregation over multiple periods. I also evaluated the correctness and execution time performance of the AEA operators against some alternative SQL translations that I developed.
When I presented some of the results of the Annotation-Enabled Analysis to our collaborators in the BuDS project, they found the AEA service to be beneficial. The dimension that annotations provide to data analysis has proven to be useful in identifying many trends in building data such as rise and fall in room temperature due to sessions in a room, an air volume anomaly due to power outage, the variation in correlation between room temperature and outside air temperature with and without sessions in a room and the relationship between night control mode and the drop in air volume. I believe and conclude that this service is a valuable addition to the BuDS project and can be adapted to other time-series databases that store annotations in the databases.
7. REFERENCES


[33] "BMS," [Online]. Available:

[34] "Data Provenance," [Online]. Available: