Data Regeneration using SPJ-Cardinality Constraints

A PROJECT REPORT
SUBMITTED IN PARTIAL FULFILMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
Master of Technology
IN
Faculty of Engineering

BY

Prashik Rawale

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

June, 2022

Declaration of Originality

I, **Prashik Rawale**, with SR No. **04-04-00-10-42-20-1-17809** hereby declare that the material presented in the thesis titled

Data Regeneration using SPJ-Cardinality Constraints

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2020-2022**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

[illegible]

Date: 30-June-2022 Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Prof. Jayant R. Haritsa Advisor Signature

DEDICATED TO

My Parents
Mr. Keshao Rawale and Mrs. Pushpa Rawale

For their constant love, support and encouragement throughout my pursuit of education. My accomplishments and success are because they believed in me.

Acknowledgements

I want to express my deepest gratitude to my advisor, Prof. Jayant Haritsa. He has been an ideal teacher, mentor, and thesis supervisor, offering advice and encouragement with a perfect blend of insight and humor. I'm proud of and extremely grateful that you took me on as a student and continued to have faith in me over the years.

This project would not have been possible without the support of Anupam Sanghi. Many thanks to him for reading my numerous revisions and helping make some sense of the confusion. I have benefited greatly from your wealth of knowledge and meticulous editing.

I am fortunate to have been a part of the Database Systems Lab. I am indebted to several of my seniors and current students here for making this a joyful learning experience. Special credits to graduate scholars Tarun Kumar Patel and Shadab Ahmed who helped me find my footing as I started this work. Thank you for all the brainstorming sessions and late-night discussions that led to the completion of this project.

Most importantly, I am grateful for my family's unconditional, unequivocal, and loving support. To my family, I give everything, including this.

Abstract

Adequately testing a database engine requires synthesizing data that resembles the client data processing environments. Contemporary data regenerators use declarative formalisms for constructing synthetic data. In particular, they specify operator output volumes through row cardinality constraints. However, thus far, adherence to these volumetric constraints has been limited in the scope of operators handled. For instance, none of the frameworks provide a solution that supports cardinality constraints with Select-Project-Join (SPJ) operators. This project aims to provide a comprehensive solution for such constraints involving SPJ operators.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

RDBMS vendors often require synthetic data to capture the data processing scenarios on the client-side effectively. This need arises for use cases such as testing DBMS and database applications, benchmarking, etc.

In the past decade, several frameworks [1, 2, 4, 3] have been proposed that focus on *workload-aware data regeneration* using constraints derived from the execution of client query workloads, as described next.

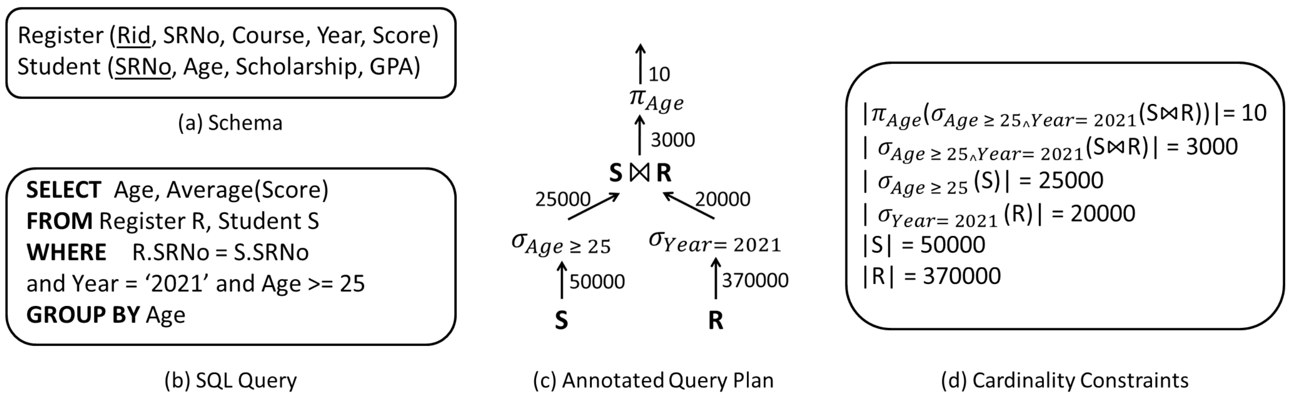
Workload-Aware Data Regeneration

Consider a sample client scenario where we have the database schema and an example query as shown in Fig. 1.1(a) and Fig. 1.1(b), respectively. Suppose that we get the execution plan for this query, by running the query at the client deployment, as shown in Fig. 1.1(c). Note that the edges in the plan tree are annotated with the number of rows flowing from one operator to the other. We refer to this plan as an Annotated Query Plan (AQP). The set of row-cardinality constraints (CCs) derived from this AQP is listed in Fig. 1.1(d).

The focus of the workload-aware data regeneration is to ensure *volumetric similarity*. That is, on running the client query workload on the synthetic database produced at the vendor site, the AQPs obtained are very similar to the ones fetched from the client site. In other words, the synthetic data should adhere to the CCs obtained with respect to the input client AQPs.

Cardinality Constraint. A CC dictates that the output of a given relational expression over the generated database should feature a specified number of rows. For Select-Project-Join (SPJ) query formulations, the canonical constraint representation is:

$$|\pi_{\mathbb{P}}(\sigma_f(T_1 \bowtie T_2 \bowtie ...T_N))| = k \tag{1.1}$$



Register (Rid, SRNo, Course, Year, Score)
Student (SRNo, Age, Scholarship, GPA)

(a) Schema

```
SELECT  Age, Average(Score)
FROM Register R, Student S
WHERE    R.SRNo = S.SRNo
and Year = '2021' and Age >= 25
GROUP BY Age
```

(b) SQL Query

(c) Annotated Query Plan

$|\pi_{Age}(\sigma_{Age \geq 25 \wedge Year = 2021}(S \bowtie R))| = 10$
$|\sigma_{Age \geq 25 \wedge Year = 2021}(S \bowtie R)| = 3000$
$|\sigma_{Age \geq 25}(S)| = 25000$
$|\sigma_{Year = 2021}(R)| = 20000$
$|S| = 50000$
$|R| = 370000$

(d) Cardinality Constraints

Figure 1.1: Example AQP and CCs

where k is the number of rows that are output after applying the complete relational expression, i.e., the output cardinality, $\mathbb{P}$ represents the set of attributes on which projection is applied (PAS), and f represents the filter conditions on the inner join of relations $T_1, T_2, ..., T_N$.

Background

The workload-aware data regeneration frameworks in the literature do not provide a comprehensive solution that handles CCs with SPJ operators. For instance, [2, 1, 3] model filter constraints using a linear programming (LP) based approach at its core. However, they lack the support for the projection operator. Likewise, [4] models filter and projection constraints in the LP; but due to being limited to a single relation, it does not support the join operator.

A critical shortcoming of the prior work is the lack of modeling the join constraints accurately. Unlike filter predicates that specify the constants/value-ranges that are permissible for the constrained columns, modeling join predicates require constructing dependence with respect to the join columns such that the generated tables obey the required join output cardinality. With large number of input constraints, this problem gets even more challenging.

A way to handle joins was used in [1, 3], where they constructed the denormalized tables first and then extracting the original tables from it. Specifically, for each table T to be constructed, a corresponding view V_T is synthesized first. This view captures the denormalized equivalent of T (excluding the key columns). These views allow rewriting the join expression on a single view. Therefore, processing on views help in generating correlations that are compatible with the various join cardinality in the input. For example, the views constructed with respect to the two tables in Figure 1.1(a) are as follows:

$$V_R(Course, Year, Score, Age, Scholarship, GPA),$$

$$V_S(Age, Scholarship, GPA)$$

Further, the first two CCs from Figure 1.1(d) can now be rewritten as:

$$|(\sigma_{Age \geq 25 \wedge Year=21}(V_R))| = 3000$$

$$|\pi_{Age}(\sigma_{Age \geq 25 \wedge Year=21}(V_R))| = 10$$

Using these views, filters on each view can be handled independently using the single table algorithm. However, the challenge then lies in extracting the original tables back from their denormalized versions. This is because these views need to obey *referential integrity* [1]. For example, the value-combinations for $Age, Scholarship, GPA$ in V_R should be a subset of V_S in order to replace the borrowed columns with the appropriate foreign-key value. In [1, 3], due to the lack of consideration of projection operation, adding only a few spurious tuples in the referenced table was sufficient to ensure referential integrity. This resulted in minor errors in satisfying the CCs. However, this approach cannot be used in presence of projection because each value-combination with respect to the borrowed columns need to be represented in the referenced table. Therefore, the LP formulation and the subsequent data generation from the solution need to explicitly model constraints to ensure referential integrity.

Our Contribution

In this work, we provide a comprehensive solution to handle the SPJ-cardinality constraints. Specifically, we use techniques proposed in PiGen [4] to model filter and projection conditions. These conditions were used in [4] to form individual LPs for each of the participating relations. Further, we also exploit the aforementioned denormalization strategy. A marked contrast is the way we model join conditions into the solution pipeline. We construct a *unified* LP for the *linked* (through referential constraints) tables. This LP models the referential constraints to ensure that the number of distinct value-combinations generated, with respect to the borrowed columns, in various intervals of the Foreign-Key table (referencing) view is upper bounded by the corresponding interval in the (referenced) Primary-Key table. Further, our *Key Curation* module ensure that the key values picked are such that the corresponding tuples in the dimension table have the prescribed number of distinct value-combinations for borrowed columns.

[1]The joins considered are restricted to PK-FK joins.

Additionally, our solution leverages the concept of *dynamic regeneration* [3], and constructs *Database Summary*, that ensures data can be generated on-demand during query processing while satisfying the input CCs. Therefore, no materialized table is required in the entire testing pipeline. Further, the time and space overheads incurred in constructing the summary is independent of the size of the table to be constructed and, in our evaluations, requires only a few 100 KBs of storage.

A detailed evaluation on a workload derived from the standard TPC-DS decision support benchmark has been conducted. The results demonstrates that the proposed solution accurately and efficiently models the SPJ CCs. As a case in point, for a workload of over 20 queries, leading to $\sim$130 CCs, the generated data satisfied all the CCs with perfect accuracy. Moreover, the entire summary production pipeline completed within viable time and space overheads.

Organization. In Section 2, we formalize the problem statement and present assumptions for our framework. Section 3 gives an overview of the proposed solution and some background concepts used in our framework. Section 4 explains about the overlapping projection constraints and workload decomposition to handle them. In Section 5, we model additional constraints for ensuring referential integrity in the synthesized database. LP formulation module is explained in Section 6 and Section 7 discusses how we materialize databases that satisfy the original workload characteristics. The experimental evaluation of our solution is shown in Section 8 and Section 9 concludes the report with a discussion of future work.

Chapter 2

Framework

In this section, we summarize the basic problem statement, and the underlying assumptions of our proposed solution.

Problem Statement. Given a workload $\mathcal{W}$ of SPJ queries with their corresponding AQPs, derived from an original database with schema S, the objective is to generate synthetic database $\mathcal{D}$ such that it conforms to S and the AQPs wrt $\mathcal{W}$.

Assumptions. We assume that the input queries consist of only PK-FK joins, and the filters and projections are on non-key columns. Further, for ease of presentation, we assume only star joins are present. The ideas can be extended to more general join queries too.

Output. Given S and $\mathcal{W}$, the proposed solution produces a collection of database summary. Each summary $s(\mathcal{D})$ can be used to deterministically produce the associated database $\mathcal{D}$. The databases produced are such that (a) all of them conform to S, and (b) for each query in $\mathcal{W}$, its AQP obtained from the original database matches the AQP obtained on at least one database instance.

Chapter 3

Solution Overview

We have extended the solution pipeline of PiGen [4] to include join operator in the CCs. The complete pipeline is illustrated in Figure 3.1. The green boxes represent the additional modules added/updated to the pipeline of PiGen. We now briefly explain these modules.

3.1 Denormalization

Inspired from [1, 3], we also have a denormalization step (described in Section ??) where the views are constructed first. After expressing the CCs in terms of the views, we can express the filter, projection and join output cardinality constraints as filter and projections on individual views.

3.2 Workload Decomposition

The solution for handling CCs with filter and projection operation, as proposed in [4], assumed non-overlapping constraints input prior to LP formulation stage. A pair of CCs were defined to overlapping if their PASs partially intersect and their filters overlap. For example, in the schema of 1.1, suppose one CC has PAS $\mathbb{P}_1$ of $(Course, Year)$ and other CC has PAS $\mathbb{P}_2$ of $(Year, Score)$ and let both CCs have same filter conditions. Then, these CCs are said to be overlapping. To handle this, [4] had an additional workload decomposition module that splits the input workload into sub-workloads such that each of them is free from these overlapping projection conflicts.

We have extended this case of overlapping projections to include the projection conflicts that surface in the presence of joins. For example, a pair of queries Q_1, Q_2 including a dimension table D, with PASs $\mathbb{P}_1$ and $\mathbb{P}_2$ induce a conflict if $\mathbb{P}_1, \mathbb{P}_2 \subseteq D$, and partially overlap with each other. We discuss the details of all the conflicts in Section ??. These conflicts are additionally

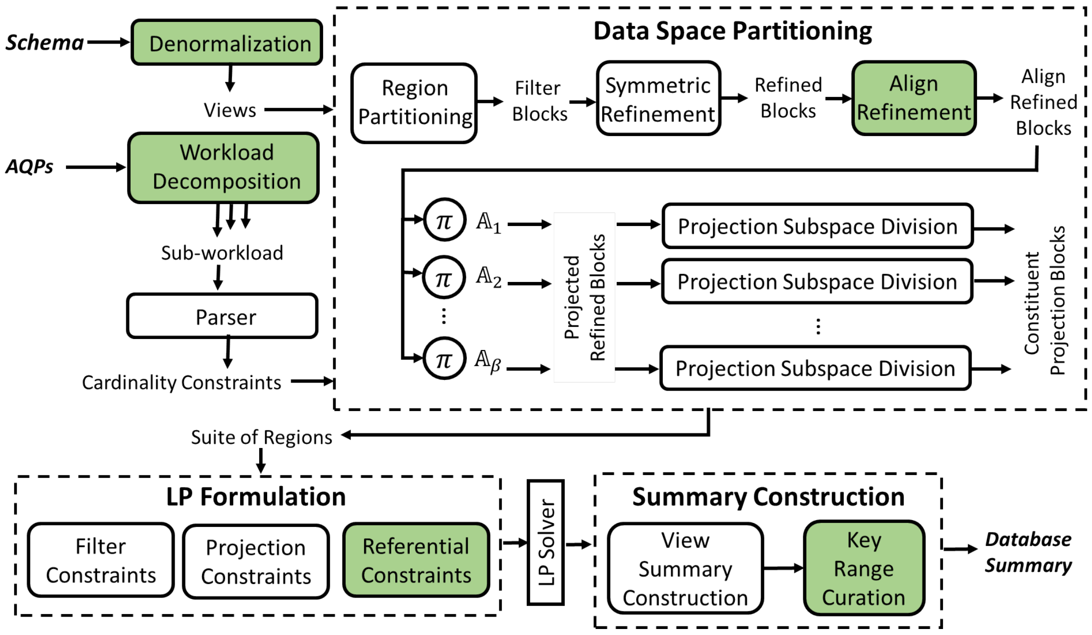


Figure 3.1: Solution Pipeline

used by the workload decomposition module to do the workload splitting.

3.3 Data Space Partitioning

Region Partitioning [3]. To model the filter predicates associated with $\mathbb{W}$, the data space of each view is logically partitioned into a set of blocks. Each block satisfies the condition that every data point in it satisfies the same subset of filter predicates. To do this partitioning, we leverage the *region partitioning* technique from [3, 4], which partitions the data space into the minimum number of blocks. Each resultant block is referred to as a filter-block (FB).

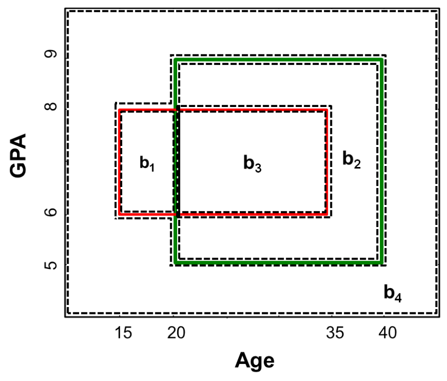


Figure 3.2: Region Partitioning

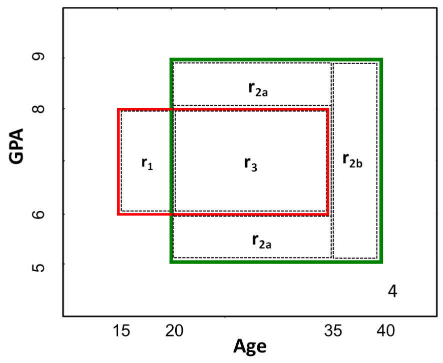


Figure 3.3: Symmetric Refinement

To make the above concrete, consider the following two filter CCs on Student table.

$$CC_1 : |(\sigma_{15 \leq Age < 35 \wedge 6 \leq GPA < 8}(V_S))| = 40000$$
$$CC_2 : |(\sigma_{20 \leq Age < 40 \wedge 5 \leq GPA < 9}(V_S))| = 45000$$

For simplicity, Figure 3.2 shows only the 2D data space comprising the Age and GPA attributes since no conditions exist on the other attributes. In this figure, the filter predicates are represented using regions delineated with colored solid-line boundaries. When region partitioning is applied on this scenario, it produces the four disjoint FBs: b_1, b_2, b_3, b_4, whose domains are depicted with dashed-line boundaries.

Symmetric Refinement [4]. To handle various projection subspaces (corresponding to the different PASs in the input queries) independently, a *Symmetric Refinement* strategy is adopted. Specifically, it refines an FB into a set of disjoint *refined blocks* (RBs) such that each resultant RB exhibits translation symmetry along each applicable projection subspace. That is, for each domain point of an RB along a particular PAS, the projection of the RB along the remaining attributes is identical.

For instance, consider b_2 in Figure 3.2. Clearly, it is asymmetric along Age – specifically, compare the spatial layout in the range $20 \leq Age < 35$ with that in $35 \leq Age < 40$. After refinement, this block breaks into r_{2a} and r_{2b} as shown in Figure 3.3 – it is easy to see that r_{2a} and r_{2b} are symmetric. This refinement allows for the values along different projection subspaces to be generated independently.

Align Refinement. To be able to obtain original tables back from their denormalized equivalents, the views need to obey referential integrity. We know that referential integrity constraint between a fact table F containing foreign key $F.fk$ referencing dimension table D with primary key $D.pk$ is expressed as $\pi_{F.fk}(F) \subseteq \pi_{D.pk}(D)$. The equivalent expression of this constraint in terms of the views is the following:

$$\pi_{\mathbb{B}}(V_F) \subseteq \pi_{\mathbb{B}}(V_D)$$

where $\mathbb{B}$ is the set of columns in V_D, and hence is borrowed in V_F. To be able to add referential constraints, the RBs of V_F need to be aligned with each other. Therefore, as a precursor, an *Align Refinement* stage is required, to ensure that each RB in V_F is either identical or is disjoint with another RB in V_F along the subspace spanned by $\mathbb{B}$. This is achieved by splitting the RBs into a set of *Aligned Refined Blocks* (ARBs). We discuss this further in Section 5.1.

Projection Subspace Division [4]. This technique divides each projection subspace into regions that allow modeling the unions into a summation of the cardinality of a subset of the regions obtained. For instance, by using projection subspace division strategy to divide the subspace with respect to *Age* attribute from our running example, we can express $|\pi_{Age}(b_{2a} \cup b_{2b} \cup b_3)|$ (projection of CC_2 along *Age*) as the following summation of cardinality of four projection regions: $|(\pi_{Age}(b_{2a}) \setminus \pi_{Age}(b_3))| + |(\pi_{Age}(b_{2a}) \cap \pi_{Age}(b_3))| + |(\pi_{Age}(b_3) \setminus \pi_{Age}(b_{2a}))| + |\pi_{Age}(b_{2b})|$.

PiGen's strategy for projection subspace division gives the minimum number of such projection regions. These are called *constituent projection blocks* (CPBs).

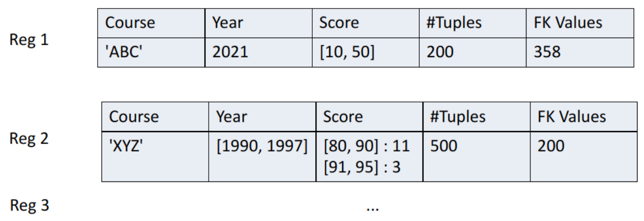
3.4 LP Formulation

After the above processing is completed for each view, we formulate a single linear program (LP) for modeling CCs. The LP is constructed using variables representing the cardinalities of ARBs and CPBs. Specifically, Filter Constraints and Projection Constraints are modelled for each view in the same way as proposed in PiGen.

Referential Constraints. We include additional constraints that ensure these referential dependencies. Specifically, these *referential constraints* ensure that for each interval of the borrowed columns, the number of distinct values present in V_F is at most equal to the number of distinct values present in V_D. Once this is ensured, the exact subset property is ensured in the final Key Curation stage.

3.5 Summary Construction.

The formulated LP is solved using an SMT solver. Once we obtain the solution, we build a summary data structure for each view that contains all the relevant information of the regions in that view required for regenerating the base relation of that view. Specifically, from the view



Reg 1

Course	Year	Score	#Tuples	FK Values
'ABC'	2021	[10, 50]	200	358

Reg 2

Course	Year	Score	#Tuples	FK Values
'XYZ'	[1990, 1997]	[80, 90] : 11 [91, 95] : 3	500	200

Reg 3

...

Figure 3.4: Sample Summary

summary the base relation summary is obtained by replacing the borrowed columns with the appropriate FK column. The process to do this is described next.

Key Range Curation. This final stage is responsible for the curation of tuples in F. Specifically, for each ARB r in V_F, to construct its equivalent in F, a range of fk values is assigned to it. This assignment is done using a range of pk values associated to a set of regions R in V_D, such that:

1. The regions in R are contained within the boundaries of r after projecting it along $\mathbb{B}$.

2. The tuples associated with the selected pk values have the desired number of distinct values along the PASs prescribed by the projections applied on the r.

In this way, we get the summary for each table, which is used for dynamic data regeneration. A sample summary is shown in Figure 3.4. Note that this summary does not reflect the exact solution of the query mentioned in Fig. 1.1

Chapter 4

Workload Decomposition

As discussed in Section **??**, the case of overlapping projection constraints is handled by splitting the workload into sub-workloads such that each sub-workload is free from such conflicts. In addition to the characterization of such overlapping projections in case of single table queries (mentioned in [4]), we have extended the class of overlapping projections to include the cases that appear in presence of joins. These additional cases of projection conflicts can be categorized based on the nature of referential dependencies as follows:

$1F : 1D$. Assume a pair of queries Q_1 and Q_2 comprising of a same pair of fact table F and dimension table D and the PASs applicable are $\mathbb{P}_1$ and $\mathbb{P}_2$ respectively, where $\mathbb{P}_1 \neq \mathbb{P}_2$. Further, the filter conditions in the queries intersect. In this case, Q_1 and Q_2 are conflicting if and $\mathbb{P}_1, \mathbb{P}_2 \subseteq D$. This is because there is an implied projection dependency between F and D with respect to $\mathbb{P}_1 \cup \mathbb{P}_2$ as well. Therefore, this adds projection constraints on F along $\mathbb{P}_1 \cup \mathbb{P}_2, \mathbb{P}_1$ and $\mathbb{P}_2$, which are overlapping.

$nF : 1D$. Assume a pair of queries Q_1, Q_2 with PASs $\mathbb{P}_1$ and $\mathbb{P}_2$. Further, both the queries involve a dimension table D such that the filters along D in the queries are overlapping. Now, if $\mathbb{P}_1, \mathbb{P}_2 \subseteq D$ and partially overlaps with each other, then this is a straightforward case of overlapping constraints on D. Therefore, Q_1 and Q_2 form a conflicting pair of queries.

$1F : nD$. Assume a query Q involving fact table F and dimension tables D_1 and D_2. Further, the PAS $\mathbb{P}$ applied on Q is such that $\mathbb{P} \subseteq D_1 \cup D_2$ and $\mathbb{P} \subsetneq D_1, \mathbb{P} \subsetneq D_1$. To ensure referential integrity, projection constraint on F along $\mathbb{P} \cap D_1$ and $\mathbb{P} \cap D_2$ is required. Both these constraints conflict with the preexisting projection constraint along $\mathbb{P}$.

The conflicts in the $1F : 1D$ and $nF : 1D$ category can be handled by splitting the workload into sub-workloads. Specifically, we construct a graph with each query being a vertex and adding an edge between two queries if there is conflict between them. Now, if we run vertex coloring algorithm on this, the subset of queries having the same color assigned form a sub-workload.

Unlike the previous two conflicts which were inter-query, the third case of $1F : nD$ type conflict is intra-query. A workaround to handle these queries is to generate all distinct tuples along $\mathbb{P} \cap D_1$ for filter compliant region of the dataspace in D_1 and along $\mathbb{P} \cap D_2$ in D_2. Subsequently, for F, the requisite number of distinct rows along $\mathbb{P}$ are generated by curating FKs from D_1 D_2. This is always possible since the distinct row cardinality along $\mathbb{P}$ in F can at most be the product of the distinct cardinality along $\mathbb{P} \cap D_1$ in D_1 and the distinct cardinality along $\mathbb{P} \cap D_2$ in D_2. Due to this explicit distinct row cardinalities being generated in D_1 and D_2, any other query with overlapping filters on D_1 (or D_2) will lead to a conflict. Again, we use the workload decomposition to also take care of these conflicts.

Chapter 5

Modeling RI Constraint

As discussed earlier, referential integrity has to be ensured in the data that is generated. We describe the modeling of this using a single pair of fact table F and dimension table D, i.e. the 1F:1D case. Let us first describe referential integrity in the view semantics.

Theorem 5.1 *Two tables F and D such that F holds FK referencing D satisfy a referential integrity dependency, iff the corresponding views V_F and V_D obey the following condition:*

$$\pi_{\mathbb{B}}(V_F) \subseteq \pi_{\mathbb{B}}(V_D)$$

where $\mathbb{B}$ is the set of columns in V_D that are borrowed by V_F.

In other words, for any region r in the data space of V_F, the following should hold:

$$\pi_{\mathbb{B}}(\sigma_{\mathbb{B}\in\overline{r}}(V_F)) \subseteq \pi_{\mathbb{B}}(\sigma_{\mathbb{B}\in\overline{r}}(V_D)) \tag{5.1}$$

where $\overline{r} = \pi_{\mathbb{B}}(r)$.

5.1 Align Refinement

The LHS in Equation 5.1 requires computing $\sigma_{\mathbb{B}\in\overline{r}}(V_F)$. That is, the interval in V_F along $\mathbb{B}$ which is aligned with a region r in V_F. To be able to determine this, the stage of Align Refinement is needed. The RBs for V_D, obtained from Symmetric Refinement, need to be aligned with each other along $\mathbb{B}$. That is, they are either identical or disjoint with each other along the subspace spanned by $\mathbb{B}$. Let us understand this using an example.

Example: Consider the RBs shown in Fig. 5.1 for a FK view where 'b' is a borrowed attribute. RB2 and RB3 have overlapping boundaries across column 'b' and are not aligned

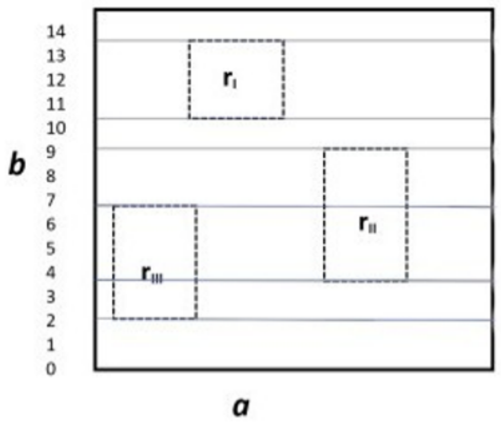


Figure 5.1: RBs before splitting

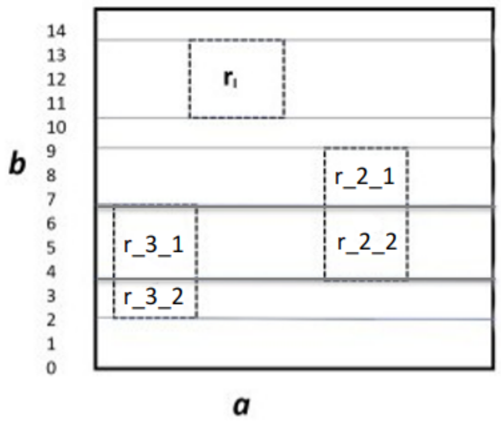


Figure 5.2: RBs after splitting (ARBs)

with each other along 'b' column. These RBs would be broken into smaller ARBs such that each ARB formed would be aligned with every other ARB as shown in Fig. 5.2.
Now, once we get blocks that are aligned, we can express the data subspace spanned by $\mathbb{B}$ using a collection of intervals such that a group of ARBs are associated with an interval. Let the group of ARBs associated with an interval I be represented as R_I. Now, we can rewrite the RI constraint as ensuring the following condition for each such interval I:

$$\pi_{\mathbb{B}}(\sigma_{\mathbb{B}\in I}(V_F)) = \pi_{\mathbb{B}}(\cup_{r\in R_I} r) \subseteq \pi_{\mathbb{B}}(\sigma_{\mathbb{B}\in I}(V_D)) \tag{5.2}$$

5.2 Region Mapping

The RHS in Equation 5.1 requires computing $\sigma_{\mathbb{B}\in\overline{r}}(V_D)$. That is, the fraction of V_D that is aligned with a region r in V_D along $\mathbb{B}$. Since, we have assumed star join queries, any filter on $F \bowtie D$ along $\mathbb{B}$ would have been also applied on D at one of the nodes in the AQP. Therefore, the domain split points in V_F along $\mathbb{B}$ would also be present in the domain split points of V_D.

As a result, we only need a mapping from the ARBs in V_F to ARBs in V_D to be able to express the RHS in the RI constraint using a union of blocks. We would like to clarify that we are only splitting fact table blocks into ARBs, but we use the term ARBs for dimension table also to be consistent in the naming. Therefore, ARBs in dimension table are same as RBs.

Now, say $M(r_f)$ be the set of ARBs in V_D that are aligned with a ARB $r_f \in V_F$ along $\mathbb{B}$. Then the Equation 5.2 can be rewritten as:

$$\pi_{\mathbb{B}}(\cup_{r_f \in R_I} r_f) \subseteq \pi_{\mathbb{B}}(\cup_{r_d \in M(r_f)} r_d) \tag{5.3}$$

Chapter 6

LP Formulation

In order to ensure that Equation 5.3 is satisfied for each ARB $r_f \in V_F$, as part of the LP formulation stage, we ensure that the LHS cardinality is upper bounded by the RHS cardinality. The explicit subset condition is ensured during summary construction. Therefore, the LP formulation ensures the following for each $r_f \in V_F$:

$$|\pi_{\mathbb{B}}(\cup_{r_f \in R_I} r_f)| \leq |\pi_{\mathbb{B}}(\cup_{r_d \in M(r_f)} r_d)| \tag{6.1}$$

6.1 Optimizations

The above condition can be simplified depending on the projection conditions applied on V_F. Specifically, there are the following three cases applicable for a region $r_f \in V_F$:

Case 1: No Projection along any attribute of $\mathbb{B}$ is applied on r_f.

Case 2: Projection along at least one attribute of F is applied on r_f.

Case 3: Projection only along a subset of $\mathbb{B}$ is applied on r_f.

We discuss each of these cases separately next.

Case 1

In this case, we can generate only one distinct value along $\mathbb{B}$ such that this value is also present in V_F. Therefore, the LP formulation stage needs to ensure that if $|r_f| > 0$ then $|r_d| > 0$, where $r_d \in M(r_f)$. This condition is expressed as the following referential constraint in the LP:

$$x_{r_f} \leq |F| \sum_{r_d \in M(r_f)} x_{r_d}$$

Here, x_r denotes a variable in the LP that represents the cardinality of the ARB r.

Case 2

Say the PAS applicable on r_f is $\mathbb{P}$. Here we replace $\mathbb{P}$ with $\mathbb{P} \setminus \mathbb{B}$. Further, again we generate only one distinct value along $\mathbb{B}$ in r_f. Note that since distinctness along a set of attributes also imply distinctness along the any of its superset, this condition ensures correctness. Hence, we use the same referential constraint in the LP as Case 1.

Case 3

In this case, we ensure the condition in Equation 6.1 as it is. This is achieved by using the CPBs. Let the CPBs associated with an ARB r along $\mathbb{B}$ be represented as $P(r)$. Then, the condition can be rewritten as:

$$\sum_{p_f \in P(r_f)} |p_f| \leq \sum_{p_d \in P(r_d), r_d \in M(r_f)} |p_d|$$

Therefore the following constraint in added in the LP in this case:

$$\sum_{p_f \in P(r_f)} y_{p_f} \leq \sum_{p_d \in P(r_d), r_d \in M(r_f)} y_{p_d}$$

Here, y_p denotes a variable in the LP that represents the cardinality of the CPB p.

Chapter 7

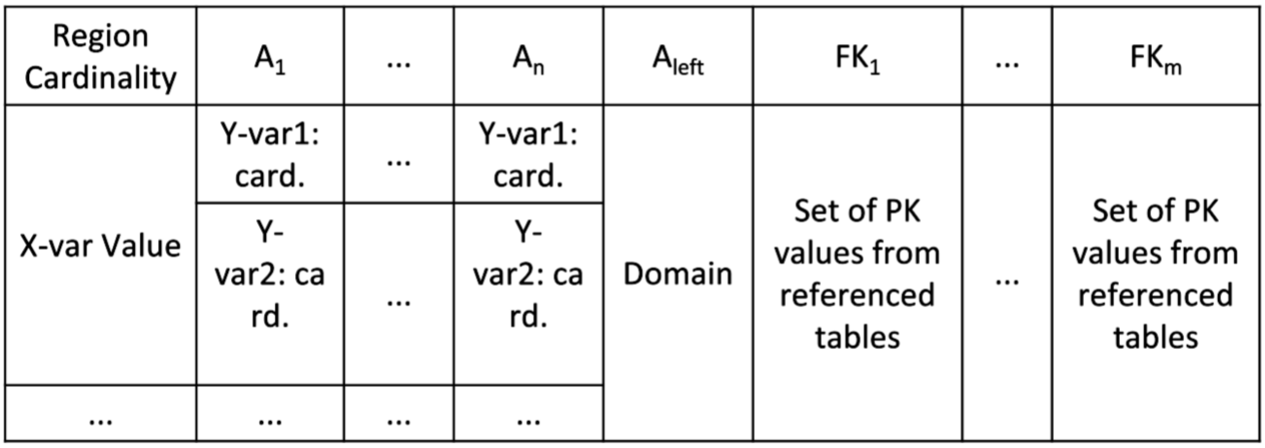
Data Generation

After adding all the constraints to LP, Z3, a popular SMT solver, is called to get the LP solution. A variable in the LP represents a region of a particular view, and its assigned value is the total number of rows in that region. We denote such a variable as 'X-var'. In addition, there are special variables associated with a region that represent smaller regions within that region, and their values indicate the number of distinct tuples that are to be instantiated from that region. We denote this special variable as 'Y-var'. A region may have zero or more such special variables associated with it. We summarize all information from the LP solution in a compact table summary (one summary for each table), which will subsequently be used for tuple generation. The summary data structure was used in [3, 4] and we build upon those concepts.

For a particular table, the summary is maintained region-wise. A sample template is shown in Fig. 8. In addition to the total cardinality (value of X-var), for each region, for every PAS acting on that region, all Y-vars associated with that PAS and their distinct cardinalities are maintained. Also, for the attributes not involved in any projections (A_{left}), only the domain is stored without any distinct cardinality. Besides this, to ensure referential integrity, we also store the Foreign Key values for every PK table to which this table references. Instantiating these FK values will be explained in the following subsection.

7.1 FK Curation

As discussed in section 4, for each region in the Fact table view (V_F), we obtain a map of corresponding regions in the Dimension table view (V_D). We will now use the same mapping to populate the FK values in the summary data structure. For every region r in V_F, we have the following 4 cases :



Region Cardinality	A_1	...	A_n	A_{left}	FK_1	...	FK_m
X-var Value	Y-var1: card.	...	Y-var1: card.	Domain	Set of PK values from referenced tables	...	Set of PK values from referenced tables
	Y-var2: card.	...	Y-var2: card.				
...	...	...	...				

Figure 7.1: Sample Region Summary

(1) No Projection constraints i.e. PAS = Φ. In this case, we use the FK region to PK region mapping to get the corresponding PK regions for region r. Since no projections are acting on r, a single FK value will suffice. Now, to ensure referential integrity, we pick any of the PK values from the set of corresponding regions to instantiate as the FK value in the summary of region r.

(2) Projection constraints are only on FK attributes i.e. PAS $\subseteq$ F. Since the projection constraints are only on the Fact table and [4] already handles such constraints, we need to only work on ensuring referential integrity. This is done by again using the corresponding regions of r, choosing one PK value from those regions, and using it as the FK value for region r.

(3) Projection constraints are on both FK and PK attributes i.e PAS $\subseteq$ F $\cup$ D, PAS $\not\subseteq$ F and PAS $\not\subseteq$ D. In this case, to simplify the generation process, we drop the borrowed attributes of D from the PAS acting on r. The idea is to generate the distinct cardinality from F only, thereby eliminating additional constraints for ensuring referential integrity. Now the problem is like case (2), and we proceed similarly. Suppose that the PAS consists of both A and B in our running example. Further, suppose that a particular constraint on this PAS requires 1000 rows in its output. To solve this, we generate 1000 unique values for the A column and couple it with any value from B's domain. Thus, the projection condition on A union B is satisfied, and referential integrity is maintained.

(4) Projection constraints are only on PK attributes i.e. PAS $\subseteq$ D. This case requires a distinct cardinality of tuples to be generated from the D table. So, for region r of the F table, we get the set of corresponding regions from the D table. We select as many unique PK values from those regions as the distinct cardinality. In the summary data structure, for region r, we store these values as the FK values to be used during tuple generation.

7.2 Tuple Generation

Using the information in summary, the tuples of the table are instantiated. Specifically, we iterate over each region and generate the number of rows specified in the associated total cardinality value. Each Y-var is picked for any region r and an associated PAS $\mathbb{A}$, and the corresponding partial tuples are generated. This gives a collection of partial tuples for $\mathbb{A}$, which may be less than the total cardinality. To make up the shortfall without altering the number of distinct values, we repeat the generated partial tuples until the total cardinality is reached. Any partial tuple within its boundaries can be picked for repetition for the Aleft component, which only has a single interval. Finally, partial tuples across all projection spaces of the region are concatenated to construct its output tuples.

If region r is of the Fact(F) table, then the summary also contains FK values for tuple generation. Based on the cases mentioned in the previous subsection, the FK values are instantiated as follows:

For any region r matching cases (1),(2), and (3) a single FK value would be stored in summary. Each tuple generated from such regions is given the same FK value.

For any region r matching case (4) a set of FK values would be stored in summary. Each Y-var value represents the distinct tuple cardinality for any such region r and an associated PAS $\mathbb{A}$. We select the same number of FK values stored in summary, and each partial tuple generated is assigned an FK value from these selected values in a round-robin manner.

Chapter 8

Experimental Evaluation

In this section, we evaluate the empirical performance of a Java based implementation of our proposed solution. The popular Z3 solver [6] is invoked to compute the solutions for the LP formulations. Our experiments cover the accuracy, time and space overheads aspects of our work. The experiments were conducted using the PostgreSQL v9.6 engine [5] on a vanilla HP Z440 workstation.

Workload Construction. We designed a workload of 28 SQL queries derived from the TPC-DS decision support benchmark such that they satisfy the aforementioned assumptions. These queries covered four fact tables and their corresponding dimension tables. These fact tables were – STORE_SALES (SS), CATALOG_SALES (CS), WEB_SALES (WS), INVENTORY (INV). The distribution of queries among these four tables were: SS (12 Queries), CS (6 Queries), WS (6 Queries), INV (4 Queries). We have shown the snapshot of a fraction of the schema graph in Figure 8.1. We can see the four fact tables with their dimension tables in the figure.

We show a sample SQL query from our input workload along with its corresponding AQP in Figure 8.2.

Sample SQL Query.

```
Select Distinct i_item_id
From store_sales, date_dim, item,
   customer_demographics, promotion
Where ss_sold_date_sk = d_date_sk
   and ss_item_sk = i_item_sk
   and cd_demo_sk = ss_cdemo_sk
   and p_promo_sk = ss_promo_sk
   and d_year = 2001 and cd_gender = 'M'
```

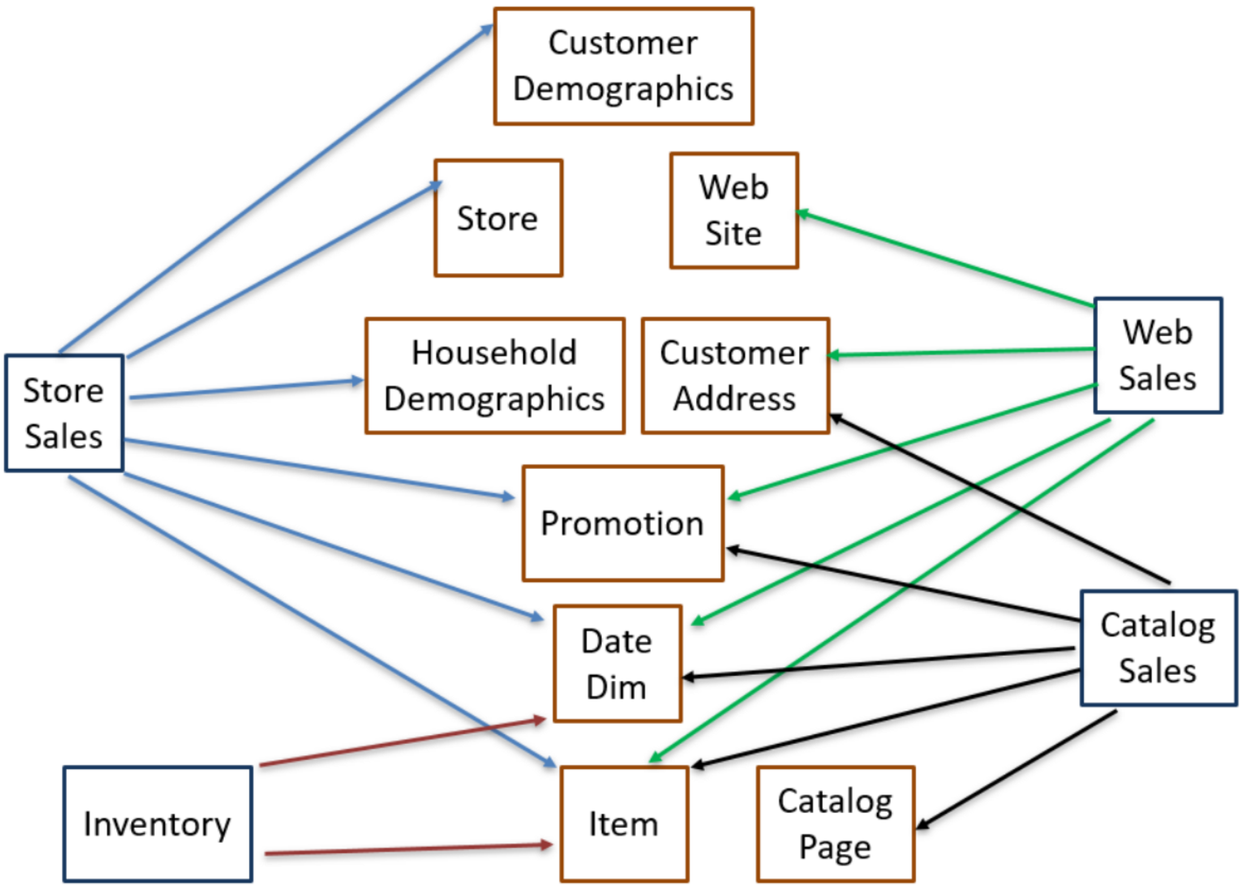


Figure 8.1: Schema Graph

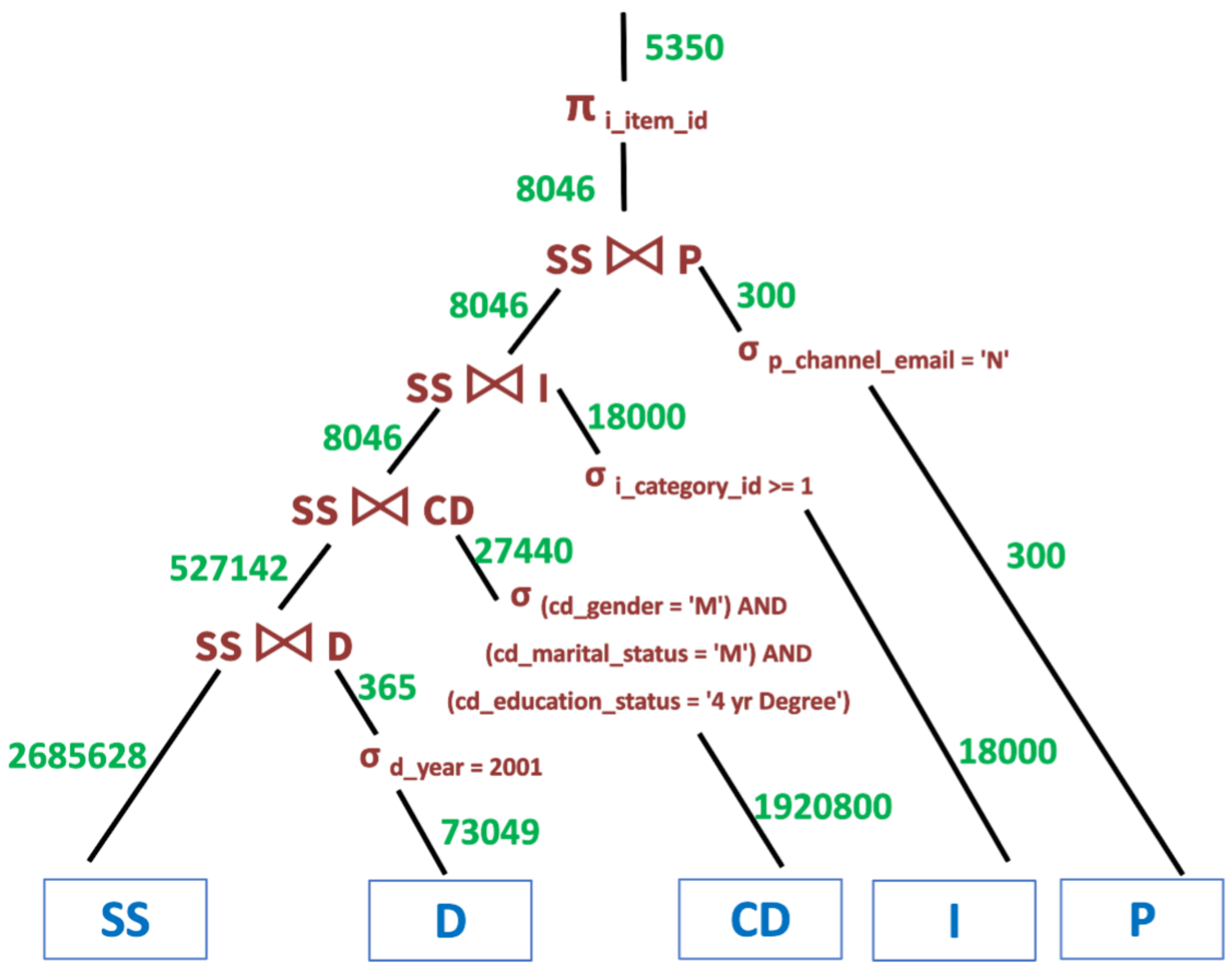


Figure 8.2: AQP of Sample Query

```
and cd_marital_status = 'M'
and cd_education_status = '4 yr Degree'
and p_channel_email = 'N' ;
```

These 28 queries led to a tally of 134 CCs. The constraints had PAS of upto 4 length. The join distribution in these CCs is shown in Figure 8.3. As we can see from the figure, the number of joins range from 1 to 4.

8.1 Workload Decomposition

We decompose this suite into three sub-workloads such that all the conflicts discussed are resolved. The complexity of these sub-workloads is quantitatively characterized in Table 8.1.

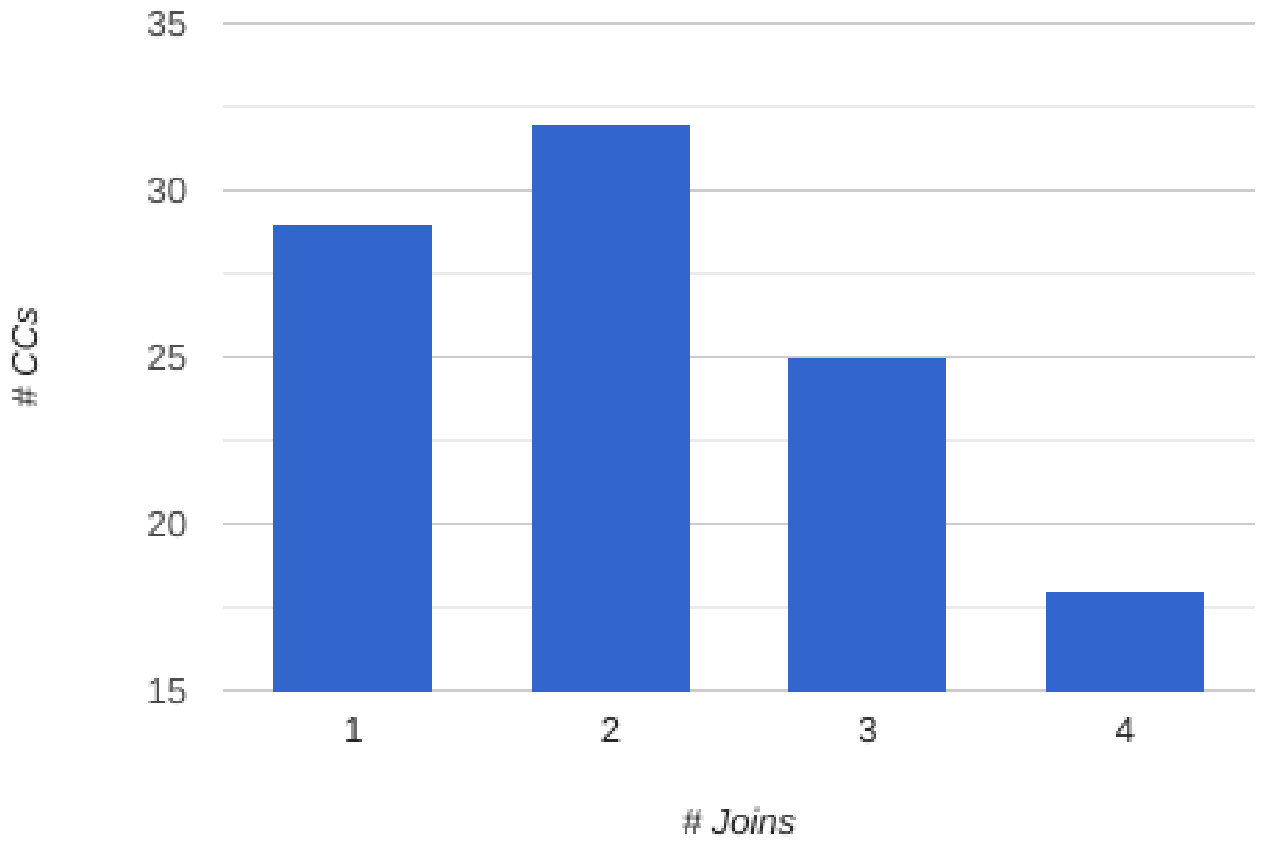


Figure 8.3: Distribution of Joins over All CCs

Table 8.1: Workload Characteristics

Workloads	#Queries	#CCs	Max. PAS Length	Max. Joins
W1	9	47	3	4
W2	8	39	3	4
W3	11	48	3	4

8.2 Constraint Accuracy

We ran our framework on the workloads mentioned before and the generated data satisfied all the constraints with **100% accuracy**. This is because – (a) additional constraints were included in the LP to ensure distinct cardinality relationship between each pair of fact table and its corresponding dimension table (s); (b) key curation ensured the explicit subset requirement, hence ensuring referential integrity.

8.3 Time and Space Overheads

We now turn our attention to the computational and resource overheads. The summary production times and sizes corresponding to the three sub-workloads is shown in Table 8.2. We see here that the summary construction times range from a few seconds to a few minutes. From a deployment perspective, these times appear acceptable since database generation is usually an offline activity. Moreover, the summary sizes are minuscule, within a few 100 KBs.

Drilling down into the summary production time, we find that all of it is consumed in LP solving stage. The collective time spent in other stages is usually less than ten seconds. The LPs constructed contains severely underdetermined system of constraints. Therefore, the number of variables in these LPs largely characterize the complexity of the LPs. To obtain a quantitative understanding of the LPs produced, we also report the number of variables constructed for the three LPs corresponding to the sub-workloads. As a case in point, the largest LP was formulated for workload W2 with about 32 thousand variables. This LP took 2 minutes to solve.

Table 8.2: Time Overheads

Workload	# of LP Variables	Total Time
W1	1218	5 s
W2	33157	2 min 13 s
W3	1196	6 s

Chapter 9

Conclusion and Future Work

Synthetic data generation from a set of cardinality constraints has been strongly advocated in the contemporary database testing literature. Our work expands the scope of the supported constraints to collectively include Select, Project and Join operators. The main challenge was to ensure referential integrity among tables so that the join constraints are handled effectively. Further, a unified LP was constructed comprising of cardinality constraints over all tables. The experimental evaluation on workloads derived from TPC-DS benchmark indicated that our solution accurately models the SPJ CCs and produces generation summaries with viable time and space overheads.

In this work we focused on Star schema queries and in our future work, we would like to extend the scope to more general schemas. Also, here we used the workload decomposition technique to work with the overlapping projection constraints. We would like to investigate if there exists other solution for the same.

Bibliography

[1] R. Kaushik A. Arasu and J. Li. Data Generation using Declarative Constraints. 2011. 1, 2, 3, 6

[2] S. Patwa A. Gilad and A. Machanavajjhala. Synthesizing Linked Data Under Cardinality and Integrity Constraints. 2021. 1, 2

[3] J. R. Haritsa A. Sanghi, R. Sood and S. Tirthapura. Scalable and Dynamic Regeneration of Big Data Volumes. 2018. 1, 2, 3, 4, 6, 7, 18

[4] S. Ahmed A. Sanghi and J. R. Haritsa. Projection-Compliant Database Generation. 2022. 1, 2, 3, 6, 7, 8, 9, 11, 18, 19

[5] PostgreSQL. https://www.postgresql.org/docs/9.6/. 21

[6] Z3. https://github.com/Z3Prover/z3. 21