# Data Generation using Join Constraints

Anupam Sanghi  Shadab Ahmed  Prashik Rawale  Jayant R. Haritsa

**Technical Report**
**TR-2022-01**
**(August 2022)**

Database Systems Lab
Dept. of Computational and Data Sciences
Indian Institute of Science
Bangalore 560012, India

https://dsl.cds.iisc.ac.in

**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.

# 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, 3, 4] 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(a) and Fig. 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(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(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:

$$|\pi_{\mathbb{P}}(\sigma_f(T_1 \bowtie T_2 \bowtie ...T_N))| = k \tag{1}$$

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, 4] model filter constraints using a linear programming (LP) based approach at its core. However, they lack the support for the projection

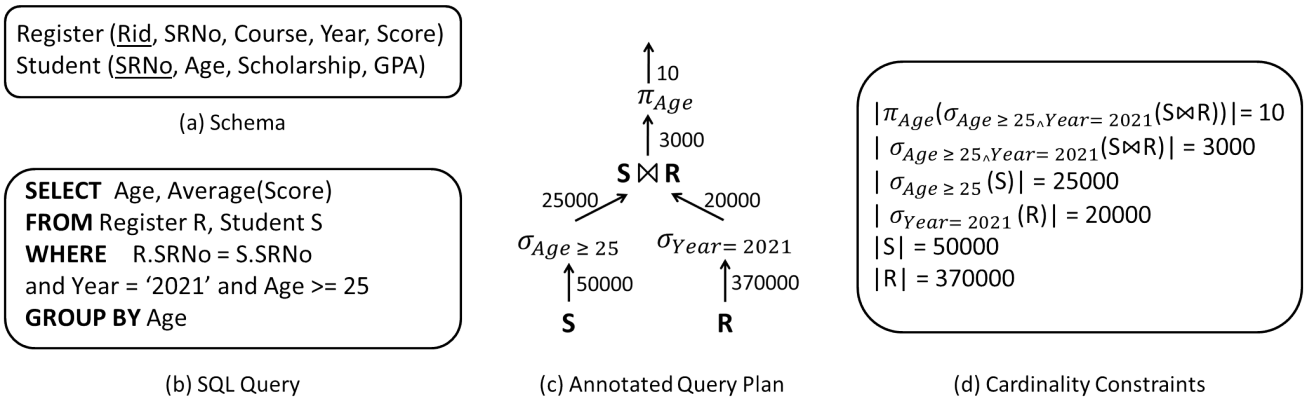


Figure 1: Example AQP and CCs

operator. Likewise, [3] 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, 4], 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(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(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, 4], 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.

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

## Our Contribution

In this work, we provide a comprehensive solution to handle the SPJ-cardinality constraints. Specifically, we use techniques proposed in PiGen [3] to model filter and projection conditions. These conditions were used in [3] 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.

Additionally, our solution leverages the concept of *dynamic regeneration* [4], 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.

# 2 Framework

In this section, we summarize the problem statement, the underlying assumptions, the output delivered, and a tabulation of the notations used in this chapter.

## 2.1 Problem Statement

Given an SPJ query-workload $\mathcal{W}$, with its corresponding set of AQPs $\mathcal{Q}$, derived from an original database with schema $\mathcal{S}$ and statistical metadata $\mathcal{M}$, the objective is to generate a synthetic database $\mathcal{D}$ such that it conforms to $\mathcal{S}$ and $\mathcal{Q}$. That is, the AQPs obtained from the original database match, wrt the cardinality annotations, the AQPs obtained on $\mathcal{D}$.

## 2.2 Assumptions

We assume that $\mathcal{W}$ comprises of only PK-FK joins. Further, we assume that the filters and projections are applied only on non-key columns. Again, we assume that $\mathcal{Q}$ is collectively feasible. Finally, for brevity, we present the ideas using tables with columns having float data type; the extension to other data types is straightforward.

## 2.3 Output

Given $\mathcal{S}$, $\mathcal{M}$, $\mathcal{W}$ and $\mathcal{Q}$, Hydra outputs a collection of database summaries $\mathbb{S}$. Each summary $s^{\mathcal{D}} \in \mathbb{S}$ can be used to deterministically produce the associated database $\mathcal{D}$. The databases produced are such

that: (a) all of them conform to $\mathcal{S}$, and (b) for each query in $\mathcal{W}$, its corresponding AQP in $\mathcal{Q}$ matches with the AQP obtained on at least one output database instance.

## 2.4 Notations

The main acronyms and key notations used in this chapter are summarized in Tables 1 and Table 2, respectively.

Table 1: Acronyms

| Acronym | Meaning |
|---|---|
| AQP | Annotated Query Plan |
| CC | Cardinality Constraint |
| SPJ | Select Project Join |
| PAS | Projection Attribute Set |
| FB | Filter Block |
| RB | Refined Block |
| ARB | Aligned Refined Block |
| CPB | Constituent Projection Block |
| PSD | Projection Subspace Division |
| NoPB | No Projection on a Subset of Borrowed Columns |
| PB | Projection on a Subset of Borrowed Columns |

Table 2: Notations

(a) Database Related

| Symbol | Meaning |
|---|---|
| $\mathcal{S}$ | Database Schema |
| $G_{\mathcal{S}}$ | Schema Graph |
| $\mathcal{D}$ | Output Database |
| $T$ | Output Table |
| $s^{\mathcal{D}}$ | Summary of $\mathcal{D}$ |
| $F$ | Fact Table |
| $D$ | Dimension Table |
| $V_T$ | View wrt $T$ |
| $\mathbb{B}$ | Borrowed Attribute-Set |

(b) Workload Related

| Symbol | Meaning |
|---|---|
| $q$ | Query |
| $\mathcal{W}$ | Query Workload |
| $\mathcal{Q}$ | Set of AQPs |
| $c$ | A c |
| $f$ | Filter Predicate |
| $\mathbb{A}$ | PAS |
| $l$ | Output row card. after filter |
| $k$ | Output row card. after projection |

(c) Block Related

| Symbol | Meaning |
|---|---|
| $r^T$ | RB wrt $V_T$ |
| $a^T$ | ARB wrt $V_T$ |
| $p^T$ | CPB wrt $V_T$ |
| $x(a^T)$ | variable for $\|a^T\|$ |
| $y(p^T)$ | variable for $\|p^T\|$ |

(d) Relation/Function Related

| Symbol | Meaning |
|---|---|
| $U(T)$ | Set of attributes in $T$ |
| $dom(.)$ | Domain of the input parameter |
| $M$ | A relation btw CCs and ARBs |
| $L$ | A relation btw CPBs and ARBs |
| $H$ | A relation btw ARBs wrt $V_F$ and $V_D$ |
| $J$ | A relation btw CPBs wrt $V_F$ and $V_D$ |

# 3 Design Principles

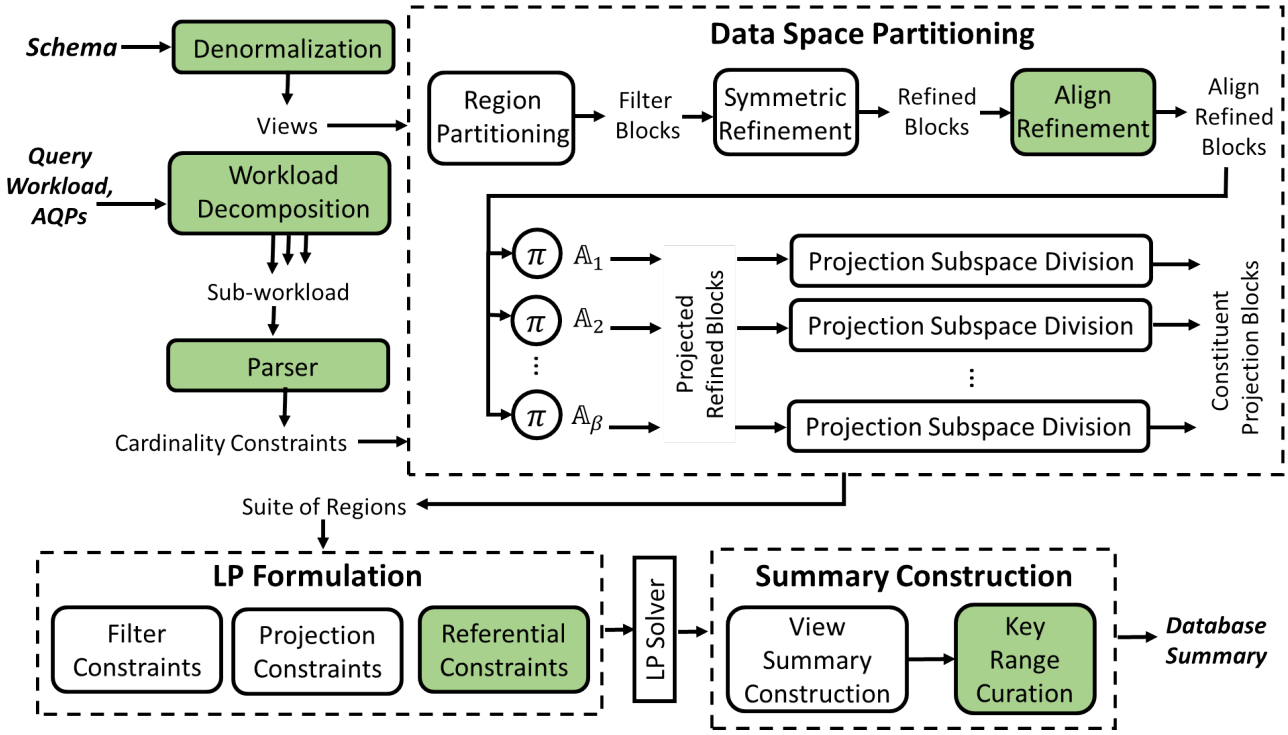


Figure 2: Join Solution Pipeline

The solution pipeline is illustrated in Figure 2. The green boxes illustrate the modules added/updated to handle joins. We briefly discuss each of the core modules in this section. For ease of exposition, we use the *fact and dimension table* terminology from data warehousing to refer to the tables having FK and the corresponding PK, respectively. Further, the notations $F$ and $D$ are used to denote a fact table and dimension table, respectively.

## 3.1 Denormalization

Inspired from DataSynth, we also construct views, where the view $V_T$ for a table $T$ is its denormalized equivalent (excluding key columns). Each column in a view is stored as a structure comprising of two fields: {Column Name, Column Path}. The Column Path is stored as an array of the foreign key columns involved in the query. Each view is associated with a view name and an array of the column structures. To populate this array, first a schema graph $G_S$ is constructed. Here, a vertex is made corresponding to each table in the database. For each PK-FK dependency, a directed edge is added from the FK table to PK table. The edge is annotated with the participating columns of both the relations. Note that, since two relations can have multiple PK-FK dependencies, there can be multiple directed edges between tables. We assume $G_S$ to be a DAG, which is common in real-world databases and is reflected in the benchmarks too.

Once $G_S$ is constructed, we traverse it in reverse topological order. For each edge $(F.fk, D.pk)$ in the order, the columns in view $V_D$ are added to $V_F$. For each such column $A$, the Column Path of $V_D.A$, prepended with $F.fk$, is inserted as the Column Path for $V_F.A$.

The join expression in a CC $c$, consists of series of PK-FK columns that participate in the joins in the subtree rooted at the AQP node corresponding to $c$. It is easy to see that this PK-FK column series can

be uniquely mapped to a path in $G_S$. The view $V_T$ corresponding to the source vertex in this path will have columns from all the tables included in the path. Also, it is easy to see that the filter and projection operation on a join expression can be equivalently written on $V_T$. Hence, in this way, each CC can be written as a filter and projection operation on a single view.

For our running example, the Views that are constructed are as follows:

$$V_{Reg}(Age, Scholarship, GPA, Course, Year, Score), \quad V_{Std}(Age, Scholarship, GPA)$$

Further, the aforementioned CCs from our running example, can be rewritten on the views $V_{Reg}$ and $V_{Std}$ as follows:

$$c_1 : |\pi_{GPA}(\sigma_{Age<20 \wedge GPA<6}(V_{Reg}))| = 2 \qquad c_2 : |\pi_{GPA}(\sigma_{Age\geq 20 \vee GPA<6}(V_{Reg}))| = 4$$
$$c_3 : |\sigma_{Age<20 \wedge GPA<6}(V_{Reg})| = 15000 \qquad c_4 : |\sigma_{Age\geq 20 \vee GPA<6}(V_{Reg})| = 250000$$
$$c_5 : |\sigma_{Age<20 \wedge GPA<6}(V_{Std}))| = 2000 \qquad c_6 : |\sigma_{Age\geq 20 \vee GPA<6}(V_{Std}))| = 42000$$
$$c_7 : |V_{Reg}| = 370000 \qquad c_8 : |V_{Std}| = 50000$$

As described earlier, referential integrity has to be ensured in the data that is generated. Let us first describe referential integrity in the view semantics.

**Theorem 1.** *Two tables $F$ and $D$, with $F$ having an FK column referencing table $D$, 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 $V_D$ borrowed by $V_F$.*

## 3.2 Workload Decomposition

In the previous chapter, a pair of constraints were defined to be overlapping if their PASs partially intersect and their filters overlap. To handle this, we 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 $q_1, q_2$ on a dimension table $D$, with PASs $\mathbb{A}_1$ and $\mathbb{A}_2$ respectively, induce a conflict if (a) the PASs $\mathbb{A}_1, \mathbb{A}_2 \subseteq D$, and partially overlap with each other, and (b) the filters in $q_1$ and $q_2$ intersect. We discuss the details of all the conflicts in Section 4. These conflicts are additionally used by the workload decomposition module to do the workload split.

## 3.3 Data Space Partitioning

**Region Partitioning and Symmetric Refinement.** To model the filter predicates associated with the input workload, 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 discussed in [4], which partitions the data space into the minimum number of filter-blocks (FBs).

To handle various projection subspaces (corresponding to the different PASs in the input queries) independently, a *Symmetric Refinement* strategy is adopted (discussed in PiGen [3]). 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.

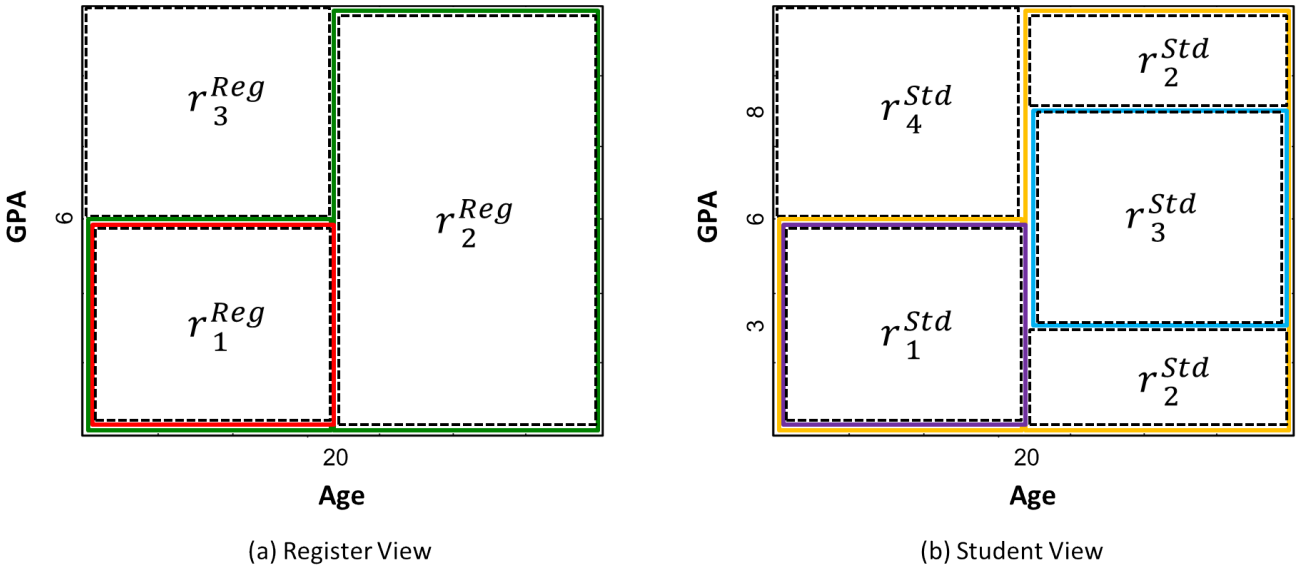


Figure 3: Partitioning of $Reg$ and $Std$ Views

To make the above concrete, the partitioning of data space of $V_{Reg}$ is shown in Figure 3(a). For ease of presentation, we show only the two dimensions that participate in the example queries. 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 three disjoint FBs: $r_1^{Reg}, r_2^{Reg}, r_3^{Reg}$, whose domains are depicted with dashed-line boundaries. For our running example, the resultant blocks are already symmetric. The partitioning with respect to $V_{Std}$ is also shown in Figure 3(b). Here, we have additionally added a constraint shown by blue solid-line to add complexity in the example.

**Align Refinement.** To obtain the original tables from their denormalized equivalents, the views need to obey referential integrity. As discussed in Theorem 1, the referential integrity constraint between fact table view $V_F$ and dimension table view $V_D$ is expressed as follows:

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

To add referential constraints, for a CC $c$ with PAS $\mathbb{A}$ (where $\mathbb{A} \subseteq \mathbb{B}$) applicable on $V_{Reg}$, its constituent RBs need to be aligned with each other along $\mathbb{A}$. By align we mean that the domain of RBs along $\mathbb{A}$ are either identical or disjoint. The blocks obtained after refinement are called *Aligned Refined Blocks* (ARBs). For example, $r_2^{Reg}$ in Figure 3(a) is split into $a_{2a}^{Reg}$ and $a_{2b}^{Reg}$ to ensure alignment with $a_1^{Reg}$, as shown in Figure 4(a). (The other RBs happen to be already aligned, so their equivalent ARBs are unaltered).

Further, we also need to ensure that each RB in $V_D$ is either completely contained or disjoint from the domain of each ARB in $V_F$. Therefore, we also do a refinement of RBs in $V_D$ as part of this module and produce ARBs that ensure alignment with $V_F$. For example, $r_2^{Std}$ and $r_3^{Std}$ in Figure 3(b) are split into $a_{2a}^{Std}, a_{2b}^{Std}$ and $a_{3a}^{Std}, a_{3b}^{Std}$ respectively, as shown in Figure 4(b). We discuss the details of this module in Section 5.

**Projection Subspace Division.** This technique divides each projection subspace into a set of *constituent projection blocks* (CPBs), as per PiGen. For our running example, the CPBs obtained for $V_{Reg}$

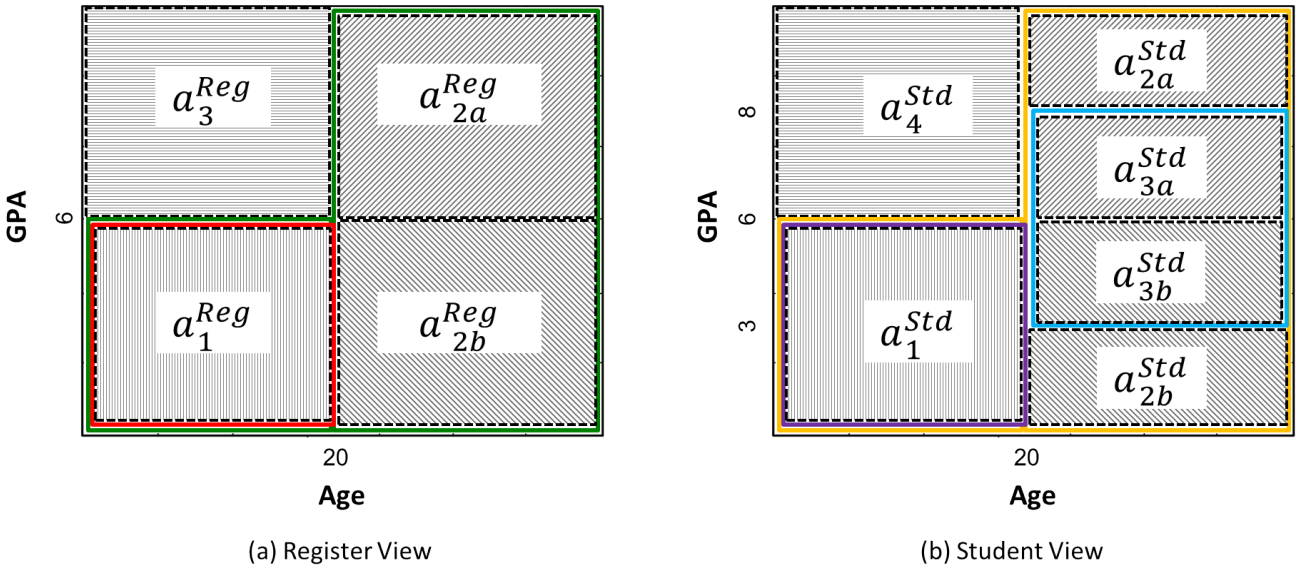


Figure 4: Align Refinement

are as follows:

$$p_1^{Reg} = \pi_{GPA}(a_1^{Reg}) \cap \pi_{GPA}(a_{2b}^{Reg}), \qquad p_2^{Reg} = \pi_{GPA}(a_1^{Reg}) \setminus \pi_{GPA}(a_{2b}^{Reg})$$
$$p_3^{Reg} = \pi_{GPA}(a_{2b}^{Reg}) \setminus \pi_{GPA}(a_1^{Reg}), \qquad p_4^{Reg} = \pi_{GPA}(a_{2a}^{Reg})$$

Further, the CPBs obtained for $V_{Std}$ are as follows:

$$p_1^{Std} = \pi_{GPA}(a_1^{Std}) \cap \pi_{GPA}(a_{3b}^{Std}), \qquad p_2^{Std} = \pi_{GPA}(a_1^{Std}) \setminus \pi_{GPA}(a_{3b}^{Std})$$
$$p_3^{Std} = \pi_{GPA}(a_1^{Std}) \cap \pi_{GPA}(a_{2b}^{Std}), \qquad p_4^{Std} = \pi_{GPA}(a_1^{Std}) \setminus \pi_{GPA}(a_{2b}^{Std})$$
$$p_5^{Std} = \pi_{GPA}(a_{3b}^{Std}) \setminus \pi_{GPA}(a_1^{Std}), \qquad p_6^{Std} = \pi_{GPA}(a_{2b}^{Std}) \setminus \pi_{GPA}(a_1^{Std})$$
$$p_7^{Std} = \pi_{GPA}(a_{2a}^{Std}), \qquad p_8^{Std} = \pi_{GPA}(a_{3a}^{Std})$$

## 3.4 LP Formulation

After the above processing is completed for each view, we formulate a *unified* LP for the *linked* (through referential constraints) tables. The LP is constructed using variables representing the cardinalities of ARBs and CPBs. Specifically, Filter Constraints and Projection Constraints are modeled for each view in the same way as proposed in the previous chapter. For modeling Referential Constraints we also do a Block Mapping where the ARBs and CPBs of $V_F$ are mapped to those of $V_D$. The referential constraints ensure that for each block in $V_F$ the number of distinct values along borrowed columns is upper bounded by the number of distinct values in the corresponding blocks in $V_D$. Once this is ensured, the exact subset property is ensured in the final Key Curation stage.

## 3.5 Summary Construction

Using the LP solution, we first build a summary data structure for each view that contains all the relevant information for extracting the corresponding base relation. Specifically, from the view 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.

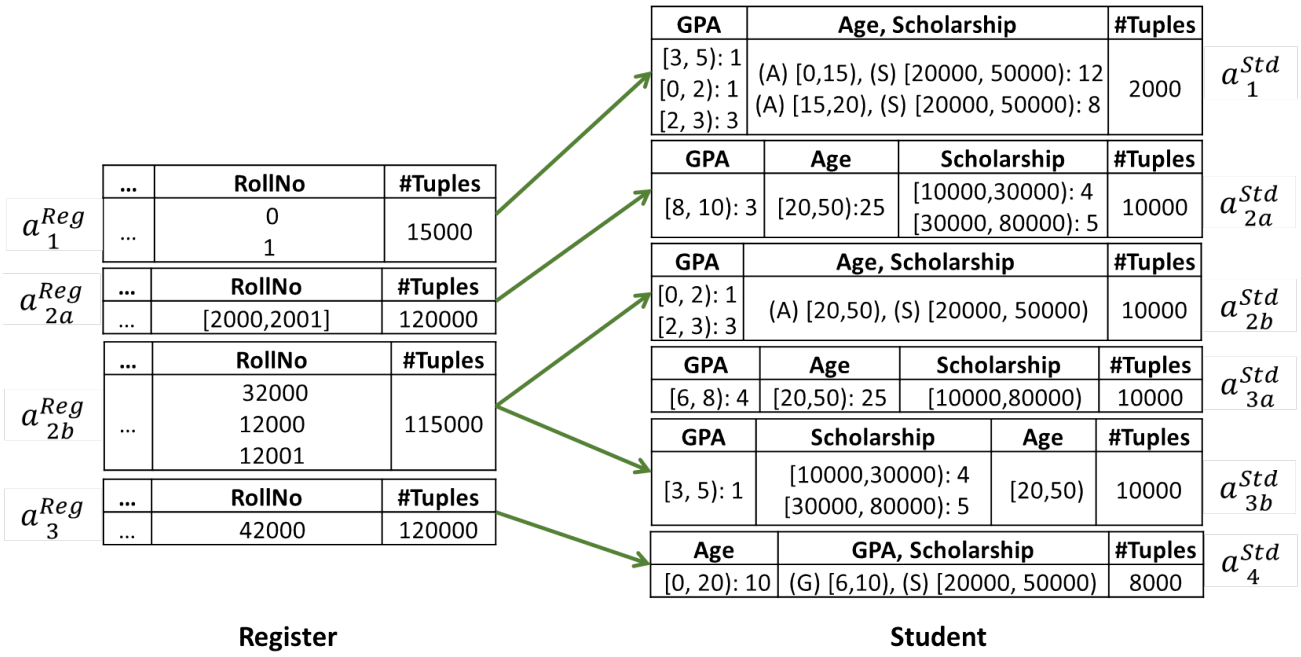


Figure 5: Sample Summary

**Key Range Curation.** This final stage is responsible for the curation of FK values in $F$. Specifically, for each ARB $a^F$ 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 blocks in $V_D$, such that:

1. The chosen $V_D$ blocks are contained within the boundaries of $a^F$ after projecting along $\mathbb{B}$.
2. The tuples associated with the selected PK values have the desired number of distinct values along the PAS prescribed by the projection applied on $a^F$.

In this way, we get the summary for each table, which is used for dynamic data regeneration. A sample summary is shown in Figure 5. To appreciate the summary in entirety, we also show the other columns in the $Std$ table schema. Therefore, we can see that the distinct row counts along different projection subspaces are represented similar to the discussion in the previous chapter. Additionally, we show the FK column in $Reg$ table summary having the range of $RollNo$ values to be included.

# 4 Workload Decomposition

As discussed in the previous section, the case of conflicting 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 (as per PiGen), 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 $q_2$ having common fact ($F$) and dimension ($D$) tables, and the PASs applicable are $\mathbb{A}_1$ and $\mathbb{A}_2$ respectively, where $\mathbb{A}_1 \neq \mathbb{A}_2$. Further, the filter conditions in the queries intersect. In this case, $q_1$ and $q_2$ are conflicting if $\mathbb{A}_1, \mathbb{A}_2 \subseteq D$. The conflict arises because there

is an implied projection dependency between $F$ and $D$ with respect to $\mathbb{A}_1 \cup \mathbb{A}_2$ as well. Therefore, $F$ is subjected to projection constraints along $\mathbb{A}_1 \cup \mathbb{A}_2$, $\mathbb{A}_1$ and $\mathbb{A}_2$, which are overlapping.

**multi-$F$ : 1$D$.** Assume a pair of queries $q_1, q_2$ with PASs $\mathbb{A}_1$ and $\mathbb{A}_2$. Further, both the queries involve a dimension table $D$ such that the filters along $D$ in the queries are overlapping. Now, if $\mathbb{A}_1, \mathbb{A}_2 \subseteq D$ and partially overlap with each other, then it is a straightforward case of overlapping constraints on $D$. Therefore, $q_1$ and $q_2$ form a conflicting pair of queries.

**1$F$ : multi-$D$.** Assume a query $q$ involving fact table $F$ and dimension tables $D_1$ and $D_2$. Further, the PAS $\mathbb{A}$ applied on $q$ is such that $\mathbb{A} \subseteq D_1 \cup D_2$ and $\mathbb{A} \nsubseteq D_1, \mathbb{A} \nsubseteq D_1$. To ensure referential integrity, projection constraints on $F$ along $\mathbb{A} \cap D_1$ and $\mathbb{A} \cap D_2$ are required. Both these constraints conflict with the preexisting projection constraint along $\mathbb{A}$.

The conflicts in the $1F : 1D$ and multi-$F : 1D$ 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 the graph, 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 $1F : 1D$ type conflict is intra-query. A workaround to handle these queries is to generate all distinct tuples along $\mathbb{A} \cap D_1$ for filter compliant region of the dataspace in $D_1$, and along $\mathbb{A} \cap D_2$ in $D_2$. Subsequently, for $F$, the requisite number of distinct rows along $\mathbb{A}$ are generated by curating FKs from $D_1$, $D_2$. This is always possible since the distinct row cardinality along $\mathbb{A}$ in $F$ can at most be the product of the distinct cardinality along $\mathbb{A} \cap D_1$ in $D_1$ and the distinct cardinality along $\mathbb{A} \cap D_2$ in $D_2$. Due to the distinct rows generation 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 workload decomposition to take care of these conflicts.

# 5 Align Refinement

To apply referential constraints on the various blocks of the fact table and dimension table views, we need to ensure that the blocks are well aligned. We next discuss the alignment process separately for the fact and dimension tables.

## 5.1 Fact Table Refinement

We know that the CPBs related to the same CC $c$ need to be assigned disjointed set of values, even if their domains overlap. Therefore, for solution tractability, as a first step, we ensure that for the CPBs that are related to the same constraint $c$ featuring PAS $\mathbb{A}$, have either identical or disjointed domains. This helps to divide the domain of $\mathbb{A}$, for a constraint $c$, into a set of intervals such that each constituent CPB is associated with an interval.

This CPB to interval mapping is done by ensuring that any two RBs related to the same CC with PAS $\mathbb{A}$ are aligned with each other along $\mathbb{A}$. That is, they are either identical or disjoint with each other along the subspace spanned by $\mathbb{A}$. Specifically, two blocks $r_1^F, r_2^F$ are considered aligned with each other if either $dom(\pi_{\mathbb{A}}(r_1^F)) = dom(\pi_{\mathbb{A}}(r_2^F))$ or $dom(\pi_{\mathbb{A}}(r_1^F)) \cap dom(\pi_{\mathbb{A}}(r_2^F)) = \emptyset$.

The Align Refinement module for a fact table view $V_F$ takes the RBs for the view as input and refines them such that the resultant blocks are mutually aligned. A block in $V_F$ obtained after refinement is called an Aligned Refined Block (ARB) and is denoted as $a^F$. In Figure 3(a), we saw $r_1^{Reg}$ and $r_2^{Reg}$ participate in a common constraint comprising of projection along $GPA$, and are not aligned along

$GPA$. As a result, $r_2^{Reg}$ is split in $a_{2a}^{Reg}$ and $a_{2b}^{Reg}$, as shown in Figure 4(a) after which alignment is preserved. Further we can see that the red constraint is associated with a single interval $I_1 : GPA \leq 6$ and the green constraint is associated with two intervals $I_1 : GPA \leq 6$ and $I_2 : GPA \geq 6$.

The algorithm for performing this refinement iterates over each pair of RBs $(r_1^F, r_2^F)$ related to the same CC with PAS $\mathbb{A}$, and outputs four (if non-empty domains) blocks, the domains of which are represented as follows:

$$dom(\pi_{U(F)\setminus\mathbb{A}}(r_1^F)) \times (dom(\pi_{\mathbb{A}}(r_1^F)) \setminus dom(\pi_{\mathbb{A}}(r_2^F))),$$
$$dom(\pi_{U(F)\setminus\mathbb{A}}(r_2^F)) \times (dom(\pi_{\mathbb{A}}(r_2^F)) \setminus dom(\pi_{\mathbb{A}}(r_1^F))),$$
$$dom(\pi_{U(F)\setminus\mathbb{A}}(r_1^F)) \times (dom(\pi_{\mathbb{A}}(r_1^F)) \cap dom(\pi_{\mathbb{A}}(r_2^F))),$$
$$dom(\pi_{U(F)\setminus\mathbb{A}}(r_2^F)) \times (dom(\pi_{\mathbb{A}}(r_1^F)) \cap dom(\pi_{\mathbb{A}}(r_2^F)))$$

### 5.2 Dimension Table Refinement

Each ARB $a^F$ along fact table view has to be given values for the borrowed columns that appear in the dimension table and are within its domain boundary. For this, we do a refinement of the RBs in dimension table as well so that each resultant block is either completely contained in the domain of a block $a^F$ (along $\mathbb{B}$) or is disjoint from it.

The Align Refinement module for a dimension table view $V_D$ takes the RBs for the view as input and refines them such that the resultant blocks are mutually aligned. A block in $V_D$ obtained after refinement is also called an aligned refined block (ARB) and is denoted as $a^D$. In Figure 3(b), the RBs $r_2^{Std}$ and $r_3^{Std}$ partially overlap with the blocks $a_{2a}^{Reg}$ and $a_{2b}^{Reg}$ (from Figure 4(a)) along the 2-D space $(GPA, Age)$. Therefore, we split these blocks into $a_{2a}^{Std}$, $a_{2b}^{Std}$, $a_{3a}^{Std}$, and $a_{3b}^{Std}$ ARBs, as shown in Figure 4(b).

The algorithm for performing this refinement iterates over each RB $r^D$ one by one. For an $r^D$, each ARB $a^F$ in $V_F$ is compared along $\mathbb{B}$, and if $dom(\pi_{\mathbb{B}}(a^F)), dom(r^D)$ partially overlap, then $r^D$ is broken into two blocks, the domains of which are represented as follows:

$$dom(r^D) \setminus dom(\pi_{\mathbb{B}}(a^F)), \quad dom(r^D) \cap dom(\pi_{\mathbb{B}}(a^F))$$

## 6 Block Mappings

To ensure referential integrity, we need to establish a mapping between the ARBs and CPBs in $V_F$ and $V_D$. We define these mappings first.

### 6.1 Aligned Refined Blocks Mapping

An ARB $a^F$ is related to an ARB $a^D$ by a relation $H$ iff the domain of $a^D$ is contained into domain of $a^F$ along the borrowed columns $\mathbb{B}$. That is:

$$(a^F, a^D) \in H \iff dom(a^D) \subseteq dom(\pi_{\mathbb{B}}(a^F))$$

For our running example, the ARB mapping is given as follows:

$$H = \{(a_1^{Reg}, a_1^{Std}), (a_{2b}^{Reg}, a_{2b}^{Std}), (a_{2b}^{Reg}, a_{3b}^{Std}), (a_{2a}^{Reg}, a_{2a}^{Std}), (a_{2a}^{Reg}, a_{3a}^{Std}), (a_3^{Reg}, a_4^{Std})\}$$

### 6.2 Constituent Projection Blocks Mapping

A CPB $p^F$ is related to a CPB $p^D$ by a relation $J$ iff for each ARB $a^F$ associated with $p^F$, there is an ARB $a^D$ associated with $p^D$, such that, the domain of $a^D$ is contained in the domain of $a^F$. That is:

$$(p^F, p^D) \in J \iff \forall a^F s.t.\ p^F L a^F, \exists a^D s.t.\ (p^D L a^D) \wedge (a^F H a^D)$$

Essentially all the CPBs in $D$ that are related to a CPB $p^F$ through $J$ together form its domain. For our running example, the CPB mapping is given as follows:

$$\begin{aligned} J = \{&(p_1^{Reg}, p_1^{Std}), (p_1^{Reg}, p_3^{Std}), (p_2^{Reg}, p_1^{Std}), (p_2^{Reg}, p_2^{Std}), (p_2^{Reg}, p_3^{Std}), (p_2^{Reg}, p_4^{Std}), \\ &(p_3^{Reg}, p_1^{Std}), (p_3^{Reg}, p_3^{Std}), (p_3^{Reg}, p_5^{Std}), (p_3^{Reg}, p_6^{Std}), (p_4^{Reg}, p_7^{Std}), (p_4^{Reg}, p_8^{Std})\} \end{aligned}$$

## 7 Referential Constraints

The referential constraints are imposed on the ARBs and CPBs depending on the nature of projections applied. Therefore, we first classify ARBs into two main categories:

**No Projection on a Subset of Borrowed Columns (NoPB)** If an ARB $a^F$ in $V_F$ is either not subjected to a projection constraint, or the projection is along PAS $\mathbb{A}$ such that $\mathbb{A} \nsubseteq \mathbb{B}$, then $a^F$ is included in the *NoPB* category. For example, $a_3^{Reg}$ is not subjected to any projection along the borrowed columns. Therefore, it belongs to the NoPB category.

**Projection on a Subset of Borrowed Columns (PB)** If an ARB $a^F$ in $V_F$ is subjected to a projection constraint with PAS $\mathbb{A}$ such that $\mathbb{A} \subseteq \mathbb{B}$, then we call $a^F$ to be in the *PB* category. For example, $a_1^{Reg}, a_{2a}^{Reg}, a_{2b}^{Reg}$ are all subjected to projection constraint along $GPA$ borrowed column. Therefore, these ARBs belong to the PB category.

### 7.1 NoPB Blocks

If there is a projection constraint applied on a block $a^F$ along a PAS $\mathbb{A}$, which is not a proper subset of $\mathbb{B}$, then this means that there is at least one attribute in $\mathbb{A}$ that was present in the original schema of the fact table $F$ itself. In this case, we replace the PAS from $\mathbb{A}$ to $\mathbb{A} \cap U(F)$, where $U(F)$ are the set of attributes in $F$. By ensuring distinctness for a subset of $\mathbb{A}$, we automatically get distinctness for $\mathbb{A}$ as well.

After the above pre-processing, each block $a^F$ in the NoPB category has no projection constraint along any attribute of $\mathbb{B}$. Therefore, we can generate any single value along $\mathbb{B}$ that lies in the domain boundary of the block. In other words, we can pick any value from the ARBs in $V_D$ that are related to $a^F$ by relation $H$. To do this, we need to ensure that if $a^F$ is populated then at least one related ARB $a^D$ is also populated. This is ensured by the following constraint:

$$|a^F| \leq |F| \sum_{a^D:(a^F,a^D)\in H} |a^D| \tag{2}$$

Here $|F|$ is a trivial upper bound on $a^F$.

## 7.2 PB Blocks

Consider a block $a^F$ on which a constraint with PAS $\mathbb{A}$, such that $\mathbb{A} \subseteq \mathbb{B}$, is applied. The distinct cardinality relationship has to be ensured at two levels for $a^F$ – (a) Constituent CPBs level, and (b) Constraint level, stemming from the interdependence of the ARBs. Specifically, the following constraints are applied:

**CPB Bound.** For a CPB $p^F$, its cardinality is upper bounded the summation of the CPBs in $D$ that form its domain. That is:

$$|p^F| \leq \sum_{p^D:(p^F,p^D)\in J} |p^D| \tag{3}$$

**Constraint Interval Bound.** For each interval $I$ obtained wrt a CC $c$ (after Align Refinement), the sum of the cardinalities of the CPBs wrt $V_F$ that are associated with $c$, and have domain as $I$, is upper bounded by the sum of the cardinalities of the CPBs wrt $V_D$, which are related to any of the aforementioned CPB wrt $V_F$, using $J$. That is,

$$\sum_{p^F:dom(p^F)=I\wedge(p^F,c)\in MoL} |p^F| \leq \sum_{p^D:(p^F,p^D)\in J\wedge dom(p^F)=I\wedge(p^F,c)\in MoL} |p^D| \tag{4}$$

## 7.3 LP Constraints

The LP has variables with respect to each ARB and CPB in both the fact table and dimension table views. Specifically, for a CPB $p$, we have a corresponding variable $y(p)$ denoting $|p|$, and for each ARB $a$, we have a corresponding variable $x(a)$ denoting $|a|$. Therefore, replacing these cardinality expressions (in Equations 2, 3, and 4) with the corresponding variables, we get the referential constraints. The filter and projection constraints are modeled as discussed in PiGen.

For our running example, following are the LP constraints for the two example AQPs:

**Filter Constraints**

$$\begin{aligned}
x(a_1^{Reg}) &= 15000 \\
x(a_1^{Reg}) + x(a_{2a}^{Reg}) + x(a_{2b}^{Reg}) &= 250000 \\
x(a_1^{Reg}) + x(a_{2a}^{Reg}) + x(a_{2b}^{Reg}) + x(a_3^{Reg}) &= 370000 \\
x(a_1^{Std}) &= 2000 \\
x(a_1^{Std}) + x(a_{2a}^{Std}) + x(a_{2b}^{Std}) + x(a_{3a}^{Std}) + x(a_{3b}^{Std}) &= 42000 \\
x(a_1^{Std}) + x(a_{2a}^{Std}) + x(a_{2b}^{Std}) + x(a_{3a}^{Std}) + x(a_{3b}^{Std}) + x(a_4^{Std}) &= 50000
\end{aligned}$$

**Projection Constraints**

$$\begin{aligned}
y(p_1^{Reg}) + y(p_2^{Reg}) &= 2 \\
y(p_1^{Reg}) + y(p_2^{Reg}) + y(p_3^{Reg}) + y(p_4^{Reg}) &= 5
\end{aligned}$$

**Referential Constraints**

[NoPB]

$$x(a_3^{Reg}) \leq 370000x(a_4^{Std})$$

[PB - CPB Bound]

$$\begin{aligned}
y(p_1^{Reg}) &\leq y(p_1^{Std}) + y(p_3^{Std}) \\
y(p_2^{Reg}) &\leq y(p_1^{Std}) + y(p_2^{Std}) + y(p_3^{Std}) + y(p_4^{Std}) \\
y(p_3^{Reg}) &\leq y(p_1^{Std}) + y(p_3^{Std}) + y(p_5^{Std}) + y(p_6^{Std}) \\
y(p_4^{Reg}) &\leq y(p_7^{Std}) + y(p_8^{Std})
\end{aligned}$$

[PB - Constraint Interval Bound]

$$\begin{aligned}
y(p_1^{Reg}) + y(p_2^{Reg}) + y(p_3^{Reg}) &\leq y(p_1^{Std}) + y(p_2^{Std}) + y(p_3^{Std}) + y(p_4^{Std}) + y(p_5^{Std}) + y(p_6^{Std}) \\
y(p_4^{Reg}) &\leq y(p_7^{Std}) + y(p_8^{Std})
\end{aligned}$$

In addition to these constraints, we also add the sanity constraints as discussed in PiGen. One possible solution to the LP is as follows:
[$Reg$ ARBs]:

$$x(a_1^{Reg}) = 15000, x(a_{2a}^{Reg}) = 120000, x(a_{2b}^{Reg}) = 115000, x(a_3^{Reg}) = 120000$$

[$Std$ ARBs]:

$$\begin{aligned}
x(a_1^{Std}) = 2000, x(a_{2a}^{Std}) = 10000, x(a_{2b}^{Std}) = 10000, \\
x(a_{3a}^{Std}) = 10000, x(a_{3b}^{Std}) = 10000, x(a_4^{Std}) = 8000
\end{aligned}$$

[$Reg$ CPBs]:

$$y(p_1^{Reg}) = 2, y(p_2^{Reg}) = 0, y(p_3^{Reg}) = 1, y(p_4^{Reg}) = 2$$

[$Std$ CPBs]:

$$\begin{aligned}
y(p_1^{Std}) = 2, y(p_2^{Std}) = 0, y(p_3^{Std}) = 4, y(p_4^{Std}) = 0, \\
y(p_5^{Std}) = 0, y(p_6^{Std}) = 0, y(p_7^{Std}) = 3, y(p_8^{Std}) = 4
\end{aligned}$$

# 8 Data Generation

From the LP solution, we get the following information for each view:

1. The cardinality for each ARB in the view.
2. The cardinality for each CPB in the view.

In this section we discuss how the database summary is constructed using this information and subsequently tuples are generated from it.

## 8.1 View Summary Construction

To begin with we first construct the view summaries independently using the same technique as in PiGen. To summarize, we first assign domain intervals to each CPB. Recall that two CPBs that are associated with a same constraint need to be disjoint even if there domains are overlapping. Therefore, during the domain interval assignment, we break the domain into disjoint intervals and assigned them to these CPBs. After this assignment, we have view summaries ready. In these summaries, we omit the domain assignment to the CPBs of the ARBs in $V_F$ that belong to PB category. As a next step, we need to extract relation summaries back from the views. Specifically, we need to replace the borrowed columns in the views with the appropriate FK column.

## 8.2 Key Curation

A number of CPBs in $V_F$ map to a CPB in $V_D$ using the relation $J$. Since these CPBs may require disjoint domain intervals assigned to them, we further split the domain interval assigned to a CPB in the $V_D$ based on the CPBs in $V_F$ related to it. Specifically, we do the following:

1. We construct a graph $G_p^D$ for each CPB $p^D$ in $V_D$. Construct a vertex for each CPB $p^F$ such that $(p^F, p^D) \in J$. An edge is added between two vertices if the corresponding CPBs are not related to a common constraint through $MoL$. In other words, these CPBs need not be disjoint and can therefore can be assigned the same values.

2. Extract maximal cliques from the graph. Number of maximal cliques correspond to the number of sub-domains in which the domain of $p^D$ is to be broken. Let the cardinality share of $p^D$ from the sub-domain $s$ be expressed using the variable $y_{p^D}(s)$.

3. Each vertex in the graph can be a part of multiple maximal cliques. This means, the corresponding CPB $p^F$ can be assigned values from multiple sub-domains. Let the share of $p^F$ from a sub-domain $s$ in the graph $G_p^D$ be expressed using variable $y_{p^F}(p^D, s)$.

4. Now, we compute the values for all the variables $y_{p^D}(s)$, $y_{p^F}(p^D, s)$ across all the graphs and maximal cliques such that:

   - Summation of the share of a dimension table CPB $p^D$ across all its sub-domains (maximal cliques) is equal to its total cardinality. That is:

   $$\sum_{s} y_{p^D}(s) = y_{p^D}$$

   .

   - Summation of the share of a fact table CPB $p^F$ across all the sub-domains (maximal cliques) in various graphs is equal to its total cardinality. That is:

   $$\sum_{(p^D, s)} y_{p^F}(p^D, s) = y_{p^F}$$

   - The cardinality share of a vertex is upper bounded by the cardinality share of the graph itself. That is:

   $$y_{p^F}(p^D, s) \leq y_{p^D}(s)$$

We use the above information to break the domain of $p^D$ into sub-domains and its associated cardinality divided wrt each sub-domain.

For our running example, considering only populated blocks, the domains of the CPBs of $V_{Std}$ are as follows:

$$p_1^{Std} : [3, 6), \quad p_3^{Std} : [0, 3), \quad p_7^{Std} : [8, 10), \quad p_8^{Std} : [6, 8)$$

Among these, the graphs corresponding to both $p_1^{Std}$ and $p_3^{Std}$ have two cliques corresponding to the singleton vertices $p_1^{Reg}$ and $p_3^{Reg}$. Therefore the domains of $p_1^{Std}$ and $p_3^{Std}$ are split into two sub-domains and the cardinality is also split. We show one possible split as follows:

$p_1^{Std}$(Split 1) : $[3, 5)$, Card.: 2 $\qquad$ $p_1^{Std}$(Split 2) : $[5, 6)$, Card.: 0

$p_3^{Std}$(Split 1) : $[0, 2)$, Card.: 1 $\qquad$ $p_3^{Std}$(Split 2) : $[2, 3)$, Card.: 3

Using these domain splits, we now show the domain assignments done to each of the populated CPBs of $V_{Reg}$:

$p_1^{Reg}$(Split 1) : $[3, 5)$, Card.: 1 $\qquad$ $p_1^{Reg}$(Split 2) : $[0, 2)$, Card.: 1

$p_3^{Reg}$(Split 1) : $[5, 6)$, Card.: 0 $\qquad$ $p_3^{Reg}$(Split 2) : $[2, 3)$, Card.: 1

$p_4^{Reg}$(Split 1) : $[8, 10)$, Card.: 2 $\qquad$ $p_4^{Reg}$(Split 2) : $[6, 8)$, Card.: 0

Next, we need to populate the FK column for each ARB in $V_F$. We do this depending on the nature of ARB.

**NoPB**

If an ARB $a^F$ is in the NoPB category, then we first pick any ARB $a^D$ such that $(a^F, a^D) \in H$. In the summary of $D$, we compute the cumulative ARB cardinality from the beginning till $a^D$. This gives the number of tuples generated before $a^D$ is generated. Let this value be $\gamma$. Since PK values are generated from 0, the tuple with PK value $\gamma$ will be the first tuple in $a^D$. Since any PK value in $a^D$ is a valid value for the FK column wrt $a^F$, we assign the FK value $\gamma$ to the entire $a^F$ block.

**PB**

Now we consider the case where an ARB $a^F$ is in the PB category. Each of its constituent CPBs are to be assigned FK value ranges. For a CPB $p^F$, we take its share with respect to each $(p^D, s)$ combination. We then iterate on the ARBs in $V_F$ that are related to $a^F$ using $H$ and get a target ARB that constitutes the CPB $p^D$. There we identify the fraction of $p^D$ that features $s$. Now, we identify the number of tuples generated before this point. This is by adding the total ARB cardinality before the target ARB and then adding the total distinct cardinality for the CPBs before the specific $p^D$ CPB. Further, the subdomain cardinalities before the specific $s$ are also added. This gives the first FK value that we need. Say this is fk-val. Then the range assigned is [fk-val, fk-val+ $y_{p^F}(p^D, s)$.

In this way, for a particular fact table, a sample template for an ARB summary is shown in Figure 6. There is total cardinality of the ARB, and for every PAS acting on it, all the CPBs associated with different projection subspaces and their distinct cardinalities are maintained. Further, for the attributes not involved in any projections ($\mathbb{A}_{left}$), only the domain is stored without any distinct cardinality. What is different here from the no-join case, is that we now also store the value ranges for each Foreign Key column. Each range is wrt to a CPB and its corresponding CPB in dimension table.

A sample database summary for our running example was already shown in Figure 5.

| $\mathbb{A}_1$ | $\mathbb{A}_2$ | ... | $\mathbb{A}_\alpha$ | $\mathbb{A}_{left}$ | $fk$ | |
|---|---|---|---|---|---|---|
| CPB-1: card., | CPB-1: card., | ... | CPB-1: card., | PB | Key Ranges | ARB Card. |
| CPB-2: card., | CPB-2: card., | ... | CPB-2: card., | | | |
| ... | ... | ... | ... | | | |

Figure 6: Sample ARB Summary

### 8.3 Tuple Generation

The tuples are generated similar to how we discussed in PiGen. Additionally, for FK columns, we use the value ranges present in the summary and do a round robin on them until the entire ARB is instantiated.

## 9 Experimental Evaluation

In this section, we evaluate the empirical performance of a Java based implementation of our proposed solution. The Z3 solver [5] 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 [] on a vanilla HP Z440 workstation.

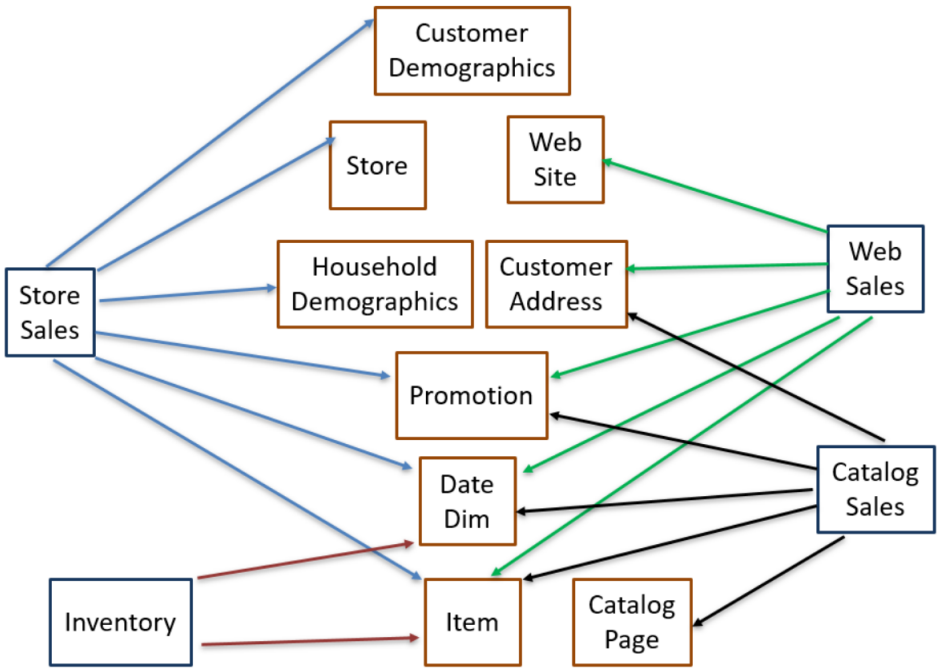


Figure 7: Schema Graph

**Workload Construction.** We designed a workload of 145 SQL queries derived from the TPC-DS decision support benchmark such that they satisfy the underlying 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 (73 Queries), CS (33 Queries), WS (32 Queries), INV (7 Queries). We have

shown the snapshot of a fraction of the schema graph in Figure 7. We can see the four fact tables with their dimension tables in the figure.

These 145 queries led to a tally of 540 CCs. The constraints had PAS of upto length ten. The join distribution in these CCs is shown in Figure 8. As we can see from the figure, the number of joins ranges from 1 to 4.

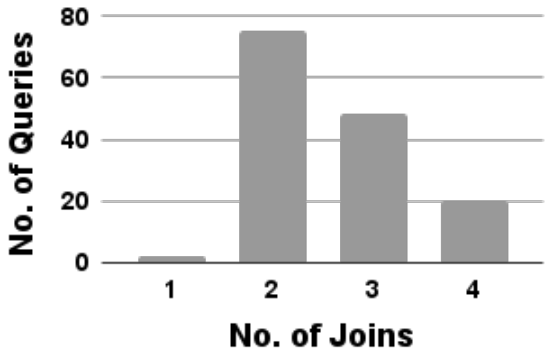


Figure 8: Distribution of Joins

We show a sample SQL query from our input workload below. The corresponding AQP is also shown in Figure 9.

**Sample SQL Query.**

```
Select Distinct i_item_id
From store_sales SS, date_dim D, item I, customer_demographics CD,
promotion P
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' and cd_marital_status = 'M'
  and cd_education_status = '4 yr Degree' and p_channel_email = 'N' ;
```

## 9.1 Workload Decomposition

We decompose the input workload into fifteen sub-workloads such that all the conflicts discussed are resolved. The complexity of these sub-workloads is quantitatively characterized in Table 3.

## 9.2 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 relationships between fact tables and their corresponding dimension tables; (b) key curation ensured the explicit subset requirement, hence ensuring referential integrity.

## 9.3 Time and Space Overheads

We now turn our attention to the computational and resource overheads. The summary production times and sizes corresponding to all the sub-workloads are shown in Table 4. We see here that the end-to-end

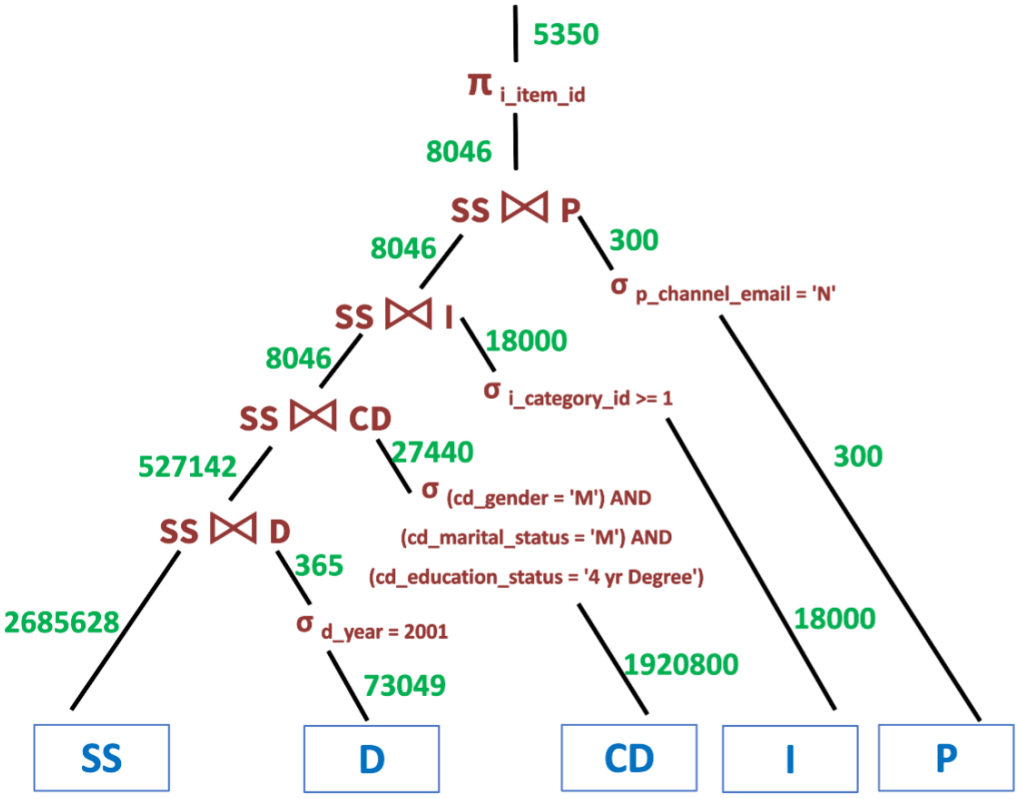


Figure 9: AQP of Sample Query

time for summary production for each sub-workload is less than a minute except for sub-workload 9 which is a couple of minutes. From a deployment perspective, these times appear acceptable since database generation is usually an offline activity. Moreover, the summary sizes, as shown in the table, are minuscule, within a few MBs. As mentioned earlier, these are independent of the data-scale of the original database.

To obtain a quantitative understanding, we report the sizes of the intermediate results at various pipeline stages also in Table 4 – specifically, the number of ARBs and CPBs created by Hydra are presented. The number of variables are simply the summation of these two quantities.

## 9.4 Performance of JOB Benchmark

We also evaluated the proposed solution on a workload of 35 SQL queries derived from the JOB benchmark based on IMDB dataset. This workload covered the three prominent fact tables – namely, MOVIE_KEYWORD (11 Queries), CAST_INFO (14 Queries), and MOVIE_COMPANIES (10 Queries), and their associated dimension tables. The number of joins in these queries range from 2 to 4. The workload led to a tally of 161 CCs.

The workload was split into 6 sub-workloads. The characteristics of these sub-workloads are shown in Table 5.

In this case as well, Hydra ensured 100% volumetric similarity. The computational and resource overheads are tabulated in Table 6. We see that the time is at most a couple of minutes and the summary sizes are also typically within a few MBs.

Table 3: Workload Characteristics (TPC-DS)

| Workload No. | #Queries | #CCs | Max PAS Length | Max joins |
|---|---|---|---|---|
| 1 | 13 | 56 | 2 | 5 |
| 2 | 10 | 45 | 4 | 5 |
| 3 | 8 | 33 | 10 | 4 |
| 4 | 8 | 26 | 3 | 4 |
| 5 | 11 | 44 | 5 | 4 |
| 6 | 10 | 38 | 2 | 5 |
| 7 | 9 | 22 | 3 | 3 |
| 8 | 10 | 32 | 4 | 3 |
| 9 | 10 | 47 | 4 | 5 |
| 10 | 9 | 41 | 6 | 5 |
| 11 | 8 | 30 | 2 | 5 |
| 12 | 7 | 20 | 2 | 4 |
| 13 | 8 | 27 | 6 | 5 |
| 14 | 14 | 39 | 2 | 4 |
| 15 | 10 | 40 | 5 | 5 |

Table 4: Overheads and Block Profile (TPC-DS)

| Workload | #ARBs | #CPBs | #Variables | Summary | |
|---|---|---|---|---|---|
| | | | | Time (s) | Space (MBs) |
| 1 | 89 | 583 | 672 | 4 | 1.2 |
| 2 | 81 | 162 | 243 | 2 | 0.6 |
| 3 | 39 | 27 | 66 | 24 | 5.4 |
| 4 | 40 | 76 | 116 | 4 | 3.4 |
| 5 | 120 | 6485 | 6605 | 14 | 5.8 |
| 6 | 53 | 69 | 122 | 2 | 0.2 |
| 7 | 38 | 137 | 175 | 2 | 0.8 |
| 8 | 48 | 66 | 114 | 3 | 7.8 |
| 9 | 98 | 8356 | 8634 | 121 | 2.2 |
| 10 | 65 | 122 | 187 | 3 | 1 |
| 11 | 46 | 29 | 75 | 2 | 0.06 |
| 12 | 24 | 70 | 94 | 42 | 4.8 |
| 13 | 36 | 78 | 114 | 2 | 0.14 |
| 14 | 75 | 1473 | 1548 | 12 | 2 |
| 15 | 58 | 243 | 301 | 6 | 7 |

# 10 Conclusion

Our work expands the scope of supported constraints to collectively include Select, Project and Join operators. The main challenge in handling joins in the CCs was to ensure referential integrity among tables. Our solution constructed a unified LP for the linked tables and added referential constraints that address the aforementioned challenge. Techniques of Align Refinement and Block Mapping aid in the process of adding these constraints. The experimental evaluation on workloads derived from TPC-DS and JOB benchmarks indicated that our solution accurately models the SPJ CCs and produces

Table 5: Workload Characteristics (IMDB)

| Workload No. | #Queries | #CCs | Max PAS Length | Max joins |
|---|---|---|---|---|
| 1 | 5 | 27 | 3 | 3 |
| 2 | 6 | 33 | 3 | 4 |
| 3 | 7 | 30 | 2 | 3 |
| 4 | 6 | 24 | 3 | 3 |
| 5 | 6 | 28 | 2 | 3 |
| 6 | 5 | 19 | 2 | 3 |

Table 6: Time and Space Overheads (IMDB)

| Workload | #Variables | Summary | |
|---|---|---|---|
| | | Time (s) | Space (MBs) |
| 1 | 94 | 1 | 0.4 |
| 2 | 66 | 1 | 0.3 |
| 3 | 1321 | 7 | 2.5 |
| 4 | 198 | 84 | 0.2 |
| 5 | 142 | 10 | 20 |
| 6 | 30 | 1 | 9.1 |

generation summaries within viable time and space overheads.

# References


[1] A. Arasu, R. Kaushik, and J. Li. Data Generation using Declarative Constraints. In *Proc. of ACM SIGMOD Conf.*, 2011.

[2] A. Gilad, S. Patwa, and A. Machanavajjhala. Synthesizing Linked Data Under Cardinality and Integrity Constraints. In *Proc. of ACM SIGMOD Conf.*, 2021.

[3] A. Sanghi, S. Ahmed, and J. R. Haritsa. Projection-Compliant Database Generation. In *PVLDB*, 15(5), 2022.

[4] A. Sanghi, R. Sood, J. R. Haritsa, and S. Tirthapura. Scalable and Dynamic Regeneration of Big Data Volumes. In *Proc. of 21st EDBT Conf.*, 2018, pgs. 301-312.

[5] Z3. https://github.com/Z3Prover/z3

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