Active Loop Detection for Applications that Access Databases*

JIASI SHEN, MIT EECS & CSAIL, USA
MARTIN RINARD, MIT EECS & CSAIL, USA

We present Shear, a new system that observes and manipulates the interaction between an application and its surrounding environment to learn a model of the behavior of the application. Shear implements active loop detection to infer the loop structures in the application. This technique repeatedly presents the application with the same input, altering the program’s interaction with the environment at precisely chosen execution points to elicit different program behaviors depending on the loop structure in the application. The ability to alter interactions between the application and the environment enables Shear to infer a broader range of loop structures otherwise undetectable given only the ability to observe application behavior. Active loop detection therefore enables Shear to infer a broader range of loop structures than previous approaches.

1 INTRODUCTION

Inferring models from the observed program behavior is a long-standing research goal [Aarts et al. 2013; Aarts and Vaandrager 2010; Angluin 1987; Cassel et al. 2016; Chow 1978; De Ruiter and Poll 2015; Fiterau-Broștean et al. 2016; Grinchtein et al. 2010; Isberner et al. 2014; Moore 1956; Radhakrishna et al. 2018; Raffelt et al. 2005; Vaandrager 2017; Volpato and Tretmans 2015; Wu et al. 2016]. Use cases of such inference include improving program comprehension [Alimadadi et al. 2014; Canfora et al. 2011; Cornelissen et al. 2011, 2009; Noughi et al. 2014] and producing higher-level representations that enable optimizations [Cheung et al. 2013; Kamil et al. 2016; Mendis et al. 2015]. When the inferred models are used to generate new executable code, this regeneration process can enable the migration of legacy programs to new platforms [Khadka et al. 2014; Shen and Rinard 2019] and the elimination of software supply chain security vulnerabilities [Vasilakis et al. 2021].

A central problem that arises in this context is inferring models of programs that contain loops [Biermann et al. 1975; Burtscher et al. 2005; Caballero et al. 2009; Elnozahy 1999; Heule et al. 2015; Ketterlin and Clauss 2008; Kobayashi 1984; Nistor et al. 2013; Noughi et al. 2014; Qi et al. 2012; Rodriguez et al. 2016; Shen and Rinard 2019; Wu 2018]. A standard approach is to apply heuristics that attempt to recognize repetitive structures in execution traces, then use those structures to infer the underlying presence of loops. Because there are, in general, multiple program structures that can generate the observed repetitions, prior approaches all limit either the program structures or the observed sequences that they can successfully work with. For example, one approach is to deploy algorithms that work for specific cases such as linear sequences of accessed addresses [Burtscher et al. 2005; Elnozahy 1999; Ketterlin and Clauss 2008; Nistor et al. 2013; Rodriguez et al. 2016]. Another approach is to use heuristics that may work in common cases but come with limited loop detection guarantees [Caballero et al. 2009; Heule et al. 2015; Wu 2018]. Yet another approach is to impose a variety of limitations on the structure of the underlying program to make the loop inference problem feasible [Kobayashi 1984; Noughi et al. 2014; Shen and Rinard 2019].

Although the problem of constructing a program from a behavior specification belongs to program synthesis, most existing program synthesis techniques cannot explicitly synthesize the precise loop structures that produce a specified repetition behavior. These techniques work with loops in restricted ways. One approach used by these techniques is to use parameterized templates that contain known loop or recursive structures, followed by synthesizing loop-free expressions

---

*This technical report is an updated version for https://hdl.handle.net/1721.1/131244.
as parameters and synthesizing loop-free callers that invoke these templates as sealed building blocks [Albarghouthi et al. 2013; Jha et al. 2010; Krogmeier et al. 2020; So and Oh 2018; Solar-Lezama et al. 2006; Srivastava et al. 2010]. Another approach is to synthesize recursive or iterative functions—such as map, filter, fold, sum, and replace—that apply on data structures—such as lists and trees—with known repetition boundaries [Ahmad and Cheung 2018; Chen et al. 2021; Feser et al. 2015; Gulwani 2011; Lee et al. 2018; Osera and Zdancewic 2015; Polikarpova et al. 2016; Smith and Albarghouthi 2016; Wang et al. 2017a,d]. Yet another approach is to rank multiple nonequivalent loop candidates with heuristics [Chen et al. 2020; Heule et al. 2015; Kulal et al. 2019; Lee et al. 2016; Pailoor et al. 2021; So and Oh 2017; Wang et al. 2017c; Ye et al. 2020].

Our Goal:
We address the problem of synthesizing loop constructs based on the observed executions of database-backed programs. We focus on lifting the repetitions in database traffic into a program with loop constructs. Many database programs contain loops that iterate over the rows retrieved by database queries. Meanwhile, inferring loops from repetitive or similar queries is not straightforward, since database programs often produce redundant data accesses or spurious repetitions [Chen et al. 2014, 2016; Finkelstein 1982; Giannikis et al. 2012; Harizopoulos et al. 2005; Sellis 1988; Yan et al. 2017; Yang et al. 2018]. Our goal is to synthesize the unique correct loop structures identical to those in the original program. We scope the range of supported programs with a domain-specific language (DSL) that expresses the programs’ externally observable behavior.

We aim at developing platform-independent techniques that analyze only the programs’ externally observable behavior, including the inputs, outputs, and database traffic. Starting from the observed behavior of an existing program, we synthesize a new program that produces the same behavior. Platform independence is desirable here, as database programs are often implemented with multiple rapidly-changing languages that are difficult to analyze statically [An and Tilevich 2020; Cornelissen et al. 2009; Maras et al. 2012] — techniques that require analyzing or modifying the source code may not adapt easily to new usage scenarios.

Our Approach:
In contrast to previous approaches, our algorithm explicitly synthesizes the unique correct loops, without many of the ad-hoc limitations required in prior work. A key reason for this advancement is that our technique not only observes the program executions but also manipulates them. Our algorithm repeatedly executes the program on the same input but systematically alters the interactions of the program with the environment at precisely chosen execution points. These alterations elicit different behaviors that expose information about the loop structure in the original program, enabling the disambiguation of loop constructs otherwise indistinguishable given only unaltered program executions. We call this technique active loop detection.

We implement and evaluate active loop detection in the context of Shear, a system for automatically inferring and regenerating programs that access relational databases. Shear works with applications that conform to the Shear DSL, which has loops that iterate over collections of rows returned from database queries. We find that the Shear active loop detection algorithm eliminates program structure restrictions present in previous systems, enabling Shear’s loop detection algorithm to significantly outperform those in previous systems: Shear targets a larger range of loop constructs, supports a larger range of programs, and successfully infers more programs (Section 6).

Contributions:
This paper makes the following contributions:

• Active Loop Detection: It presents active loop detection, which alters the program’s interaction with the environment at precisely chosen execution points to elicit different behaviors that enable more general loop detection algorithms. To the best of our knowledge, Shear is the first program synthesis technique that observes and intervenes in the executions of an existing program. This novel approach enables Shear to efficiently resolve loop-related
ambiguities, which have been one of the major challenges in inductive program synthesis based only on the end-to-end input-output specifications.

- **Soundness:** It presents a theorem that states that if the behavior of a program conforms to the DSL, then Shear infers the correct program. Shear infers the program as a black box, without requiring any analysis of the source code or the binary of the program. Hence Shear works well with programs written in any language or implementation styles, as long as the externally observable behavior of the programs can be expressed in the Shear DSL.

- **Results:** It presents experimental results from our Shear implementation. We evaluated the scalability and efficiency of active loop detection. We evaluated Shear’s expressiveness by using it to infer and regenerate open source applications in Java and Ruby on Rails, many of which violate the ad-hoc limitations in previous loop detection techniques. These results highlight the effectiveness of our approach in synthesizing loop and repetitive structures.

- **Other Contexts:** While the paper instantiates the idea of active loop detection in Shear, the idea is also applicable to a wide range of other scenarios that involve reverse engineering and program comprehension. The paper discusses how to apply active loop detection in other contexts. These potential extensions highlight the generalizability of active loop detection.

2 **EXAMPLE**

We present an example that illustrates how Shear identifies loops in a database program by manipulating the program’s interactions with the environment. Figure 1 presents the pseudo code of a task management program, where brackets (“[ . . . ]”) denote database queries. The program takes an input argument, t.id, and retrieves data from three database tables: tasks, comments, and users. It first retrieves a task specified by the input. When the task exists, the program retrieves comments under this task. For each comment, the program retrieves the user that made this comment, along with all tasks created by this user. After iterating over comments, the program retrieves the user to which the task is assigned, along with all tasks created by this user.

An example execution of the program uses the tasks table in Figure 2a, the comments table in Figure 2b, the users table empty, and the input t.id = 2. Figure 2c presents the resulting trace.

**Loop-Related Ambiguities:** The trace may be produced by five plausible loop structures (Figure 3). Each loop structure performs certain queries, iterates over the rows retrieved by a query, and optionally performs more queries after the loop ends. We use comments (after the “#” symbol) to represent queries produced across different loop iterations. Among these candidate loop structures, the only one that is consistent with the program (Figure 1) is Plausible Loop L (Figure 3a). All other candidates are nonequivalent and incorrect, but also indistinguishable with the trace alone. A key reason for these ambiguities is that the execution trace is unstructured—there are no pre-defined ways to split the trace into loop iterations.

Instead of using heuristics, Shear uses active loop detection to infer the unique correct loop structure. Shear first identifies potential queries over which a loop may iterate. In this example, these queries are $q_1$ and $q_3$, each of which retrieved two rows. For each of these locations, Shear performs three altered executions of the program to determine whether the potential loop is valid.

**Manipulating Interactions with Environment:** Shear uses a proxy between the program and the database to relay and manipulate the SQL queries in the database traffic. Shear first reuses the original inputs and database contents to start executing the program. When the program issues query $q_0$, Shear faithfully relays the database traffic for this query. Next, when the program issues query $q_1$, Shear strategically alters the SQL query into $q'_1$ before forwarding it to the database. The altered query $q'_1$ retrieves only the first row among the rows that would have been retrieved by the original query $q_1$. The database performs query $q'_1$ and retrieves the row as requested, which is then relayed through the proxy back to the program. After this manipulation, Shear resumes
normal program execution until it terminates. The manipulated execution produces the first altered trace that consists of queries \( q_0, q_1', q_2, q_3, q_6, \) and \( q_7. \) Figure 4b illustrates this manipulation.

**Shear** next obtains the second altered trace. Shear faithfully relays the database traffic for query \( q_0. \) When the program issues query \( q_1, \) Shear alters it into \( q_1'', \) which retrieves only the second row among the rows that would have been retrieved by query \( q_1. \) The database performs \( q_1'', \) whose rows are relayed through the proxy back to the program. After this manipulation, Shear resumes normal program execution until it terminates. The resulting altered trace consists of queries \( q_0, q_1'', q_4, q_5, q_6, \) and \( q_7. \) Figure 4c illustrates this manipulation.

Finally, Shear obtains the third altered trace, where the query \( q_1 \) is altered to \( q_1''' \) that retrieves both the first and the second rows in \( q_1. \) In this example, \( q_1''' \) retrieves the same results as \( q_1. \) Hence the third altered trace consists of queries \( q_0, q_1'', q_2, q_3, q_4, q_5, q_6, \) and \( q_7. \) Figure 4d illustrates this manipulation.

**Detecting Loop At Query \( q_1: ** Shear compares these three altered traces to determine if a loop iterates over the two rows retrieved by query \( q_1. \) Note that queries \( q_0, q_1', q_1'', \) and \( q_1''' \) are produced before any iterations of the hypothetical loop.

Shear first compares the lengths of the three altered traces to calculate the number of queries that would be produced by the subprogram after the hypothetical loop ends. This number is calculated by adding up the lengths of the first two altered traces after the hypothetical loop location (\( q_1' \) or \( q_1'' \)), then subtracting with the length of the third altered trace after the hypothetical loop location (\( q_1''' \)). In this example, this number is 2. Figure 5a illustrates this comparison.

Shear uses this number to identify queries that would be produced by hypothetical loop iterations. Specifically, Shear removes from each altered trace the last 2 queries and the queries before the hypothetical loop. Remaining queries in the first altered trace are \( q_2 \) and \( q_3, \) which would be produced by the first hypothetical loop iteration. Remaining queries in the second altered trace are \( q_4 \) and \( q_5, \) which would be produced by the second hypothetical loop iteration. Remaining queries in the third altered trace are \( q_2, q_3, q_4, \) and \( q_5, \) which would be produced by both first two hypothetical loop iterations.

Shear then uses these results to check if the hypothetical loop is valid. In this example, the queries produced by the first hypothetical iteration (\( q_2 \) and \( q_3 \)) comprise a strict prefix of the queries produced by both of the first two hypothetical iterations (\( q_2, q_3, q_4, \) and \( q_5). \) Also, the last 2 queries in all three altered traces are identical (\( q_6 \) and \( q_7). \) Based on these observations, Shear determines

```
<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>creator_id</th>
<th>assignee_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>
```

(a) Table tasks

```
<table>
<thead>
<tr>
<th>id</th>
<th>task_id</th>
<th>commenter_id</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
```

(b) Table comments

```
<table>
<thead>
<tr>
<th>id</th>
<th>task_id</th>
<th>assignee_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>
```

(c) SQL database traffic of an execution. The queries retrieve 1, 2, 0, 2, 0, 0, 0, and 0 rows, respectively.

Fig. 2. Example execution trace

![Fig. 1. Example program pseudo code](image-url)

![Fig. 4. Example execution trace](image-url)
that the program behavior is consistent with the existence of a hypothetical loop. \textit{Shear} therefore determines that a loop indeed iterates over the two rows retrieved by query \textit{q}_1.

**Detecting Non-Loop At Query \textit{q}_3:** Because query \textit{q}_3 also retrieved two rows during execution, there can potentially be a loop that iterates over the two rows retrieved by query \textit{q}_3 (Plausible Loop Z). To determine whether this loop exists, \textit{Shear} alters the database traffic for query \textit{q}_3 to obtain three altered traces. The first altered execution alters query \textit{q}_3 into query \textit{q}''_3 which retrieves only the first row in query \textit{q}_3. The resulting altered trace consists of queries \textit{q}_0, \textit{q}_1, \textit{q}_2, \textit{q}_3', \textit{q}_4, \textit{q}_5, \textit{q}_6, and \textit{q}_7. The second altered execution alters query \textit{q}_3 into query \textit{q}''''_3 which retrieves only the second row in query \textit{q}_3. The resulting altered trace consists of queries \textit{q}_0, \textit{q}_1, \textit{q}_2, \textit{q}_3'', \textit{q}_4, \textit{q}_5, \textit{q}_6, and \textit{q}_7. The third altered execution alters query \textit{q}_3 into query \textit{q}''''''_3 which retrieves both the first and the second rows in query \textit{q}_3. The resulting altered trace consists of queries \textit{q}_0, \textit{q}_1, \textit{q}_2, \textit{q}_3'''', \textit{q}_4, \textit{q}_5, \textit{q}_6, and \textit{q}_7.

\textit{Shear} first compares the lengths of the three altered traces to calculate the number of queries that would be produced by the subprogram after the hypothetical loop ends. This number is calculated...
by adding up the lengths of the first two altered traces after the hypothetical loop location ($q'_3$ or $q''_3$), then subtracting with the length of the third altered trace after the hypothetical loop location ($q'''_3$). In this example, this number is 4. Figure 5b illustrates the expected outcome of this comparison.

**Shear** then uses this number to identify the queries that would be produced by the hypothetical loop iterations. Specifically, **Shear** removes the last 4 queries in each altered trace and removes the leading queries up to the hypothetical loop location. For all of the three altered traces, there are no remaining queries. Figure 5c illustrates the actual outcome of this comparison.

**Shear** then uses these results to check if the hypothetical loop is valid. In this example, the queries produced by the first hypothetical iteration (no queries) do not comprise a strict prefix of the queries produced by both the first two hypothetical iterations (no queries). Based on these observations, **Shear** determines that the program behavior is inconsistent with the existence of a hypothetical loop. **Shear** determines that there are no loops that iterate over the two rows retrieved by query $q_3$. Hence, **Shear** rules out the incorrect Plausible Loop Z.

**Identifying Iteration Boundaries**: After **Shear** infers that a loop iterates over query $q_1$, it manipulates the database traffic again to calculate the loop iteration boundaries. For each row retrieved by $q_1$, **Shear** obtains an altered trace where $q_1$ is altered to retrieve only that single row. In this example, because $q_1$ retrieves only two rows, these altered traces are already obtained earlier when **Shear** detects the existence of the loop. **Shear** compares these traces to first calculate the number of queries in the trace that are generated by the after-loop subprogram. In this case, the after-loop subprogram generates two queries ($q_6$ and $q_7$). This result is used to identify the number of queries generated by each loop iteration. In this case, the first iteration generates two queries ($q_2$ and $q_3$) and the second iteration generates two queries ($q_4$ and $q_5$). Hence, **Shear** rules out the incorrect Plausible Loop W,X,Y and determines that the Plausible Loop L is correct.

**Discussion**: Active loop detection is based on several manipulated executions of the program per hypothetical loop. The algorithm works precisely with programs that may contain a variety of loop and repetitive structures including nested loops, consecutive loops, loops with conditional statements, and non-loop repetitive queries. To the best of our knowledge, **Shear** is the first program synthesis technique that intervenes in the executions of an existing program.

### 3 PRELIMINARIES

**Shear** instantiates active loop detection on a prior technique called Konure that infers and regenerates database applications [Shen and Rinard 2019]. We reiterate several key concepts from Konure and generalize them to express the new loop and repetitive structures supported by **Shear**.

#### 3.1 Overview of Inference and Regeneration of Database Programs

Program inference and regeneration is the process of observing a program’s behavior, inferring aspects of the behavior as a model in a certain domain, and using this inferred model to synthesize a new program (Figure 6a). The shared program inference framework in both Konure and **Shear** consists of a DSL and a recursive inference algorithm.
The DSL characterizes the externally observable behavior of database applications that can be inferred. The externally observable behavior consists of the input-output behavior and the database query traffic during execution. Key characteristics of the DSL are: (1) each statement performs an SQL query to retrieve rows from the database, (2) the retrieved rows determine the control flow, and (3) the data flow is largely visible in the database traffic.

A key technical difference between SHEAR and Konure is the extent to which these two systems interact with the program execution. Konure only observes the database traffic. In contrast, SHEAR both observes and manipulates the database traffic (Figure 6b). This new capability enables SHEAR to infer and regenerate fundamentally more expressive loop and repetition structures.

The inference algorithm uses an SMT solver to generate useful inputs and database values with which to execute the original program. As the algorithm executes the program, it observes the program behavior in terms of the database traffic and the outputs. Based on this observation, the algorithm updates a hypothesis of the inferred program structure as a partially expanded abstract syntax tree (AST). If the updated hypothesis still contains uncertainty, the algorithm resolves this uncertainty by using the SMT solver again to generate new inputs and database values that distinguish different hypotheses. The algorithm recursively expands nonterminal symbols in the inferred program’s AST.

The DSL and the recursive inference algorithm are designed together. Each recursive step of the algorithm identifies a unique correct production for expanding a nonterminal symbol in the DSL. The algorithm is guaranteed sound and complete for programs expressible in the DSL.

3.2 DSL for Inferrable Database Programs

Figure 7 presents the DSL for data retrieval programs that can be inferred and regenerated by SHEAR. It generalizes from Konure by adding the support for more general loop structures and eliminating heuristic restrictions on repetitions.

A program consists of sequences (Seq), conditionals (If), or loops (For). Each Query statement performs an SQL select operation that retrieves data from the database. Our current DSL supports SQL where clauses that select rows in which one column has the same value as another column (Col = Col) or the same value as a value in the context (Col = Orig). Selecting from multiple tables corresponds to an SQL inner join operation. The query stores the retrieved data in a unique variable (y) for later use. All variables must be defined before they are used. An If statement tests if its Query retrieves empty or nonempty data. A For statement iterates over the rows in its Query. The variable in the Query of a For statement is accessible within the loop body and holds the data for one selected row in every loop iteration. This variable, however, is not accessible by the subprogram that follows after the loop. Each Print statement is associated with a query and only prints values retrieved by its query. For loops may be nested.
The SHEAR DSL is the set of programs $S \subset \text{Prog}$ where the two branches of any If statement start with queries with different skeletons. As in Konure, this restriction is designed to enhance performance distinguishing Seq and If statements and does not affect the correctness of the algorithms.

### 3.3 Formalizing Program Behavior

We generalize definitions in prior work for characterizing nested loops and after-loop subprograms.

**Definition 1.** A context $\sigma = \langle \sigma_I, \sigma_D, \sigma_R \rangle \in \text{Context}$ (Figure 8) contains value mappings for the input parameters ($\sigma_I \in \text{Input}$), database contents ($\sigma_D \in \text{Database}$), and results retrieved by database queries ($\sigma_R \in \text{Result}$).

**Definition 2.** $[P]$ denotes the black box executable of a program $P \in \text{Prog}$. Note that SHEAR does not require analyzing the source code or the binary of $P$ when it infers $[P]$.

**Definition 3.** A query-result pair $(Q, r)$ has a query $Q \in \text{Query}$ and an integer $r \in \mathbb{Z}_{\geq 0}$ that counts the number of rows retrieved by $Q$ during execution.

**Definition 4.** A loop layout tree for a program $P \in \text{Prog}$ is a tree that represents information about the execution of loops (Figure 9). Each node in the tree is a query-result pair that corresponds to a query in $P$. Each node represents whether a loop in $P$ iterates over the corresponding query multiple times. In particular, when a loop in $P$ iterates over a query $r$ times ($r \geq 1$), the query’s corresponding node in the loop layout tree has $(r + 1)$ subtrees. The first $r$ subtrees $(T_1, \ldots, T_r)$ each corresponds to an iteration of the loop. The last subtree ($T_0$) corresponds to the remaining subprogram in $P$ that follows the loop.

For example, a loop layout tree may represent two nested loops in $P$ by nesting the inner subtree in some of the first $r$ subtrees of the outer tree. The inner subtree may occur multiple times, depending on the number of iterations of the outer loop in which the inner loop is executed. As another example, a loop layout tree may represent two consecutive loops in $P$ by nesting the latter subtree in the last subtree of the former tree.

**Definition 5.** An annotated trace is an ordered list of annotated query tuples. Each tuple, denoted as $\langle Q, r, \lambda \rangle$, has three components. The first component is a query $Q \in \text{Query}$. The second component is the number of rows retrieved by $Q$ during an execution. The third component is the annotated information of whether a loop was found to iterate over data retrieved by $Q$. If such loop was found then either (1) $\lambda$ is a non-negative integer that indicates the iteration index or (2) $\lambda = \text{AfterLoop}$ which indicates execution of the subprogram that follows the loop. If no such loop was found then $\lambda = \text{NotLoop}$. Each path from the root of the loop layout tree to a leaf generates a corresponding annotated trace.

**Definition 6.** A path constraint $W = (\langle Q_1, r_1, a_1 \rangle, \ldots, \langle Q_n, r_n, a_n \rangle)$ consists of a sequence of queries $Q_1, \ldots, Q_n \in \text{Query}$, row count constraints $r_1, \ldots, r_n$, boolean flags $d_1, \ldots, d_n$, and boolean flags $a_1, \ldots, a_n$. Each $r_i$ specifies the range of the number of rows in a query result, denoted as one of $(= 0)$, $(\geq 1)$, $(\geq 2)$, or (Any). Each $d_i$ is true if a loop iterates over the corresponding retrieved rows and false otherwise. Each $a_i$ is true if a loop iterates over the corresponding retrieved rows and the path enters the subprogram after the the loop. If the path enters the loop body, $a_i$ is false.

**Definition 7.** An annotated trace $t$ is consistent with path constraint $W$, denoted as $t \sim W$, if the path specified in $W$ is not longer than $t$, each query in $t$ matches a query in $W$, each row count in $t$ matches a requirement in $W$, and each after-loop status in $t$ matches a flag in $W$.

---

**Fig. 9. Grammar for loop layout trees**

\[
T \coloneqq \text{Nil} \mid (Q, r); T \mid (Q, r)[T_1, \ldots, T_r]; T_0
\]

\[Q \in \text{Query}, \quad r \in \mathbb{Z}_{\geq 0}\]
We next reiterate helper procedures in Konure. The inference algorithm takes an executable database program (Definition 2) along with specifications of its input parameter format and database schema. This restriction is with empty loop bodies, downgrading For statements with loop bodies that execute at most once. 

Definition 8. For a program $P \in \text{Prog}$ and a context $\sigma \in \text{Context}$, $\sigma \vdash P \downarrow_{\text{exec}} e$ and $\sigma \vdash P \downarrow_{\text{loops}} l$ denote evaluating $P$ in $\sigma$ to obtain a list of query-result pairs $e$ and a loop layout tree $l$, respectively. Figure 10 and Figure 11 define the evaluation. The CONNECTTREES procedure takes two loop layout trees and attaches the second tree to the last leaf in the first tree, then returns the resulting tree.

Definition 9. The skeleton of a program $P \in \text{Prog}$ is a program that is syntactically identical to $P$ except for replacing syntactic components derived from the Orig nonterminal (Figure 7) with an empty placeholder $\circ$. For any program $P \in \text{Prog}$, $\overline{P}$ is the semantically equivalent program obtained from $P$ by discarding unreachable branches in If and For statements, discarding For statements with empty loop bodies, downgrading For statements with loop bodies that execute at most once to If statements, and downgrading If statements with an unreachable branch or two semantically equivalent branches to Seq statements. For any program $P \in \text{Prog}$, $\mathcal{D}(P)$ is a predicate that is true if and only if the two branches of all conditional statements in $P$ start with queries with different skeletons (or one of the branches is empty).

Definition 10 (The Shear DSL). We define the Shear DSL as the set of programs $S \subset \text{Prog}$ defined as:

$$S = \{ \overline{P} \mid P \in \text{Prog}, \mathcal{D}(\overline{P}) = \text{true} \}$$

The predicate $\mathcal{D}(\overline{P}) = \text{true}$ states that the two branches of any If statement in $\overline{P}$ must start with queries with different skeletons (or one of the branches must be empty). This restriction is designed to enhance performance when distinguishing Seq from If statements and does not affect the correctness of the algorithms.

3.4 Helper Procedures for Program Execution and Solver Invocation

We next reiterate helper procedures in Konure. The inference algorithm takes an executable database program (Definition 2) along with specifications of its input parameter format and database schema.
The algorithm automatically interacts with the program to infer its functionality as a black box. After inference completes, it generates a new program. This regenerated program is guaranteed to have the same functionality as the original program as long as the original program’s behavior is expressible in the DSL.

**Infer:** The entry point to the inference algorithm (Algorithm 1). It takes an executable $P$ and configures an initial context $\sigma$ where all database tables are empty and the input parameters are distinct. It invokes **GetTrace** on $P$ to obtain an initial annotated trace $t$. This trace $t$ is used to invoke **InferProg**, which infers the program recursively. We present Shear’s generalized **InferProg** procedure in Section 5.

**GetTrace:** This procedure (Algorithm 2) executes the program, collects a trace, detects loops by invoking **DetectLoops**, and returns an annotated trace that satisfies a path constraint. We present Shear’s new **DetectLoops** procedure in Section 4.

Recall that the inference algorithm generates contexts, which specify the inputs and database values (Figure 8), by invoking an SMT solver. We reiterate this interface from Konure but generalize it for Shear to work with more sophisticated loop and repetitive structures. This interface consists of two procedures, **MakePathConstraint** and **SolveAndGetTrace**.

**MakePathConstraint:** This procedure takes a trace prefix $s_1$, a query $Q$, an integer $i$, and two lists of boolean flags $d$ and $a$. The procedure constructs a path constraint, $W_i$, which specifies that the program should execute down the same path as $s_1$, then perform $Q$ and retrieve $i$ rows. In particular, if $i = 0$ then $Q$ is required to retrieve zero rows (= 0). If $i = 1$ or $i = 2$ then $Q$ is required to retrieve at least $i$ rows ($\geq i$). If $i$ is not provided (Nil), then the row count for $Q$ is unconstrained.
Algorithm 1 Infer and regenerate a program expressible in the Shear DSL.

**Input:** \( \mathcal{P} \) is the executable of \( P \in \mathcal{S} \).

**Output:** Program equivalent to \( P \).

1: procedure \text{Infer}(\mathcal{P})
2: \( \sigma \leftarrow \) Database empty, input parameters distinct; \( t \leftarrow \text{GetTrace}(\mathcal{P}, \text{Nil}, \sigma) \);
3: return \text{InferProg}(\mathcal{P}, \text{Nil}, t)
4: end procedure

Algorithm 2 Execute a program and deduplicate the trace according to a path constraint

**Input:** \( \mathcal{P} \) is the executable of \( P \in \mathcal{S} \). \( W \) is a path constraint. \( \sigma \) is a context that satisfies \( W \).

**Output:** Annotated trace \( t \), \( t \sim W \), from executing \( \mathcal{P} \).

1: procedure \text{GetTrace}(\mathcal{P}, W, \sigma)
2: \( e \leftarrow \text{Execute}(\mathcal{P}, \sigma) \); \( l \leftarrow \text{DetectLoops}(\mathcal{P}, \sigma, e) \); return \text{MatchPath}(l, W)
3: end procedure

![Fig. 12. Illustration of Algorithm 4 when applied to a true loop that iterates over the \( k \)-th query](a) Intact trace (b) First altered trace (c) Second altered trace (d) Third altered trace

(Any). The path constraint uses \( d \) and \( a \) to specify the known looping structures in the trace prefix \( s_1 \) followed by \( Q \).

**SOLVEANDGETTRACE:** This procedure takes an executable \( \mathcal{P} \) and a path constraint \( W \). The procedure solves for a context \( \sigma \) that enables \( \mathcal{P} \) to produce a trace \( t \) that satisfies \( W \). If satisfiable, the procedure invokes \text{GetTrace} to execute the program and obtain a satisfying trace.

4. **ACTIVE LOOP DETECTION**

Shear takes a database program, infers its functionality, then regenerates a new version of the program with the same inferred functionality (Figure 6b). A key contribution of Shear is the idea of altering the program’s execution on the fly to resolve ambiguities. Shear alters the program’s interactions with the environment at precisely chosen execution points. Specifically, Shear detects potential loops in a database program by directly manipulating the SQL traffic during program execution. To infer whether a loop iterates over a specific query, Shear removes certain data from the corresponding database traffic and observes how it affects program execution. With a small number of such altered executions, Shear infers whether the execution point contains a loop and, if so, the structure of the loop.

4.1 **Manipulating Database Traffic with Proxy**

Shear manipulates the database traffic through a proxy interposed between the program and the database. In a normal execution of the program, the proxy faithfully relays the database traffic between the program and the database. When the program sends an SQL query to the proxy, it forwards the query to the database. When the database sends the retrieved data to the proxy, it
Algorithm 3 Execute program and alter database traffic at the specified query so that it retrieves a sub-list of rows

**Input:** $P$ is the executable of a program $P \in S$.
**Input:** $\sigma$ is a context.
**Input:** $k$ is an integer.
**Input:** $\rho$ is a list of distinct integers.
**Output:** List of SQL queries obtained from executing $P$ with $\sigma$, altering the $k$-th query to retrieve only the rows specified in $\rho$.

```
1: procedure ExecAndPick($P$, $\sigma$, $k$, $\rho$)
2:    $s \leftarrow$ Empty list
3:    $\langle \sigma_I, \sigma_D, \sigma_R \rangle \leftarrow \sigma$
4:    Populate the database with contents $\sigma_D$
5:    Execute $P$ with input parameters $\sigma_I$
6:    Use proxy to relay traffic between $P$ and database
7:    for each SQL query $q$ received from $P$ do
8:        if the $k$-th query then
9:            $q \leftarrow$ SQL query retrieving only rows $\rho$ in $q$
10:       end if
11:    end for
12:    $d \leftarrow$ Data retrieved from the database
13:    Send $d$ to $P$
14:    Append $q$ to $s$
15: return $s$
16: end procedure
```

forwards the data back to the program. In an execution of the program where Shear alters the database traffic, the Shear proxy (conceptually) removes certain data from the database traffic while the program runs. There are two general options to implement this alteration: (1) altering the queries sent from the program to the database or (2) altering the data sent from the database to the program. Shear uses the first option.

When the program sends a certain SQL query to the proxy, it alters the query so that it selects a certain sub-list of the rows that would have been retrieved if the query were intact. The proxy then sends the altered query to the database. The database performs the (altered) query to retrieve data and sends the data back to the proxy. The proxy forwards the (altered) retrieved data faithfully to the program.

**ExecAndPick:** Algorithm 3 presents the procedure for executing the program, altering its database traffic during execution. This procedure takes a program, a context, an integer $k$, and a list of distinct row indices $\rho$. It executes the program with the context while intercepting the database traffic.

For the first $(k - 1)$ queries that the program sends to the database, the proxy relays the traffic faithfully. Up to this point, the collected queries and their retrieved data are identical to what would have been collected from a normal execution of the program.

For the $k$-th query, the proxy alters the query so that it retrieves only a sub-list of the rows that would have been retrieved if the query were intact (line 9 of Algorithm 3). The sub-list of rows are
Algorithm 4 Detect if a loop iterates over a query by manipulating the database traffic in three program executions

Input: $P$ is the executable of $P \in S$. $\sigma$ is a context. $k$ is an integer, denoting a hypothetical loop at $k$-th query.
Output: Boolean $f$ represents whether a loop is found to iterate over the $k$-th query in the trace from executing $P$ with $\sigma$.
Output: Integer $l_a$ represents the number of queries in the trace produced by the subprogram that follows the detected loop.

1: procedure DetectHypo($P, \sigma, k$)
2:   $s_1 \leftarrow$ ExecAndPick($P, \sigma, [1]$) ; $s_2 \leftarrow$ ExecAndPick($P, \sigma, [2]$) ; $s_{12} \leftarrow$ ExecAndPick($P, \sigma, [1, 2]$)
3:   $s'_1 \leftarrow s_1[k+1, \ldots]$ ; $s'_2 \leftarrow s_2[k+1, \ldots]$ ; $s'_{12} \leftarrow s_{12}[k+1, \ldots]$  \hspace{1cm} $\triangleright$ Len. after hypo. loop
4:   $l_a \leftarrow$ Len($s'_1$) + Len($s'_2$) - Len($s'_{12}$)  \hspace{1cm} $\triangleright$ Len. of the first hypo. iteration
5:   $l_1 \leftarrow$ Len($s'_1$) - $l_a$  \hspace{1cm} $\triangleright$ Len. of the second hypo. iteration
6:   $l_2 \leftarrow$ Len($s'_2$) - $l_a$  \hspace{1cm} $\triangleright$ Len. of both hypo. iterations
7:   $l_{12} \leftarrow$ Len($s'_{12}$) - $l_a$  \hspace{1cm} $\triangleright$ Queries in the first hypo. iteration
8:   $\beta_1 \leftarrow s'_1[1, \ldots, l_1]$  \hspace{1cm} $\triangleright$ Queries in both hypo. iterations
9:   $\beta_{12} \leftarrow s'_{12}[1, \ldots, l_{12}]$  \hspace{1cm} $\triangleright$ Queries after the hypo. loop
10:  $\alpha_1 \leftarrow s'_1[l_1, 1, \ldots]$  \hspace{1cm} $\triangleright$ Queries after the hypo. loop
11:  $\alpha_{12} \leftarrow s'_{12}[l_{12}, 1, \ldots]$  \hspace{1cm} $\triangleright$ Queries after the hypo. loop
12:  \hspace{1cm} if $\alpha_1 = \alpha_{12} \triangleright$ PerfectPrefix($\beta_1, \beta_{12}$) then
13:     return $(true, l_a)$  \hspace{1cm} $\triangleright$ Traces consistent with hypo. loop
14: \hspace{1cm} end if
15: \hspace{1cm} return $(false, Nil)$  \hspace{1cm} $\triangleright$ Traces inconsistent with hypo. loop
16: end procedure

specified by the row indices in $\rho$. Our SHEAR implementation alters the query using standard SQL clauses "LIMIT", "OFFSET", and "ORDER BY". The proxy forwards the altered query to the database, which retrieves (altered) data for the query. The proxy sends the (altered) data back to the program, which processes the data and continues execution accordingly.

After this alteration, the SHEAR proxy continues to relay the rest of the database traffic faithfully until the program terminates. The procedure returns the list of collected SQL queries.

From the program’s viewpoint, SHEAR transparently removes certain rows from the data that would have been retrieved. Because SHEAR only removes rows, it only mildly disturbs the execution flow. In our experiments, this manipulation works reliably with all of our benchmark applications, without triggering any errors, warnings, or crashes.

4.2 Detecting One Loop

For each potential loop location in an execution trace, SHEAR first detects whether a loop exists, and if so, it then detects the boundaries of each iteration of the loop.

DetectHypo: Algorithm 4 infers whether a loop iterates over a specific query during an execution of the program. This procedure takes the program and a context $\sigma$. Executing the program with the context $\sigma$ would produce a trace. The third parameter, $k$, is an integer query index. The procedure detects if the $k$-th query in the trace was iterated over by a loop during program execution.

To detect whether a loop iterates over the $k$-th query, the procedure invokes ExecAndPick three times. Each invocation executes the program with $\sigma$ and alters the $k$-th query. The first
execution alters the $k$-th query to retrieve only the first row among all of the rows that would have been retrieved in an intact execution. The second execution alters the query to retrieve only the second row. The third execution alters the query to retrieve only the first two rows. Each execution produces an altered list of SQL queries. Because all three executions initially use the same context $\sigma$ to run the program, the three resulting altered lists are identical up to the $(k - 1)$-th query. These three altered lists may differ after the $k$-th query, depending on the structure of the program.

The DETECTHYPO procedure uses these three altered lists to infer whether the program contains a loop that iterates over the $k$-th query. The procedure obtains the three list suffixes starting from the $(k + 1)$-th query (line 3). The procedure compares these three list suffixes regarding their lengths and contents. Conceptually, it first assumes there would be a loop that iterates over the specified query. The procedure calculates the number of queries that would have been produced by only the first hypothetical loop iteration (line 5), the number for only the second hypothetical iteration (line 6), and the number for both the first and second iterations (line 7). The procedure then locates the queries that would have been produced by these hypothetical loop iterations (lines 8,9) and by any remaining queries in the program following the hypothetical loop (lines 10,11,12). Figure 12 illustrates these calculations. Finally, the DETECTHYPO procedure checks if the lengths and contents for these hypothetical iterations are consistent (line 13). The procedure invokes PerfectPrefix with variables $\beta_1$ and $\beta_{12}$, which represent the queries that would have been produced by the first hypothetical loop iteration and by the first two hypothetical loop iterations, respectively. The PerfectPrefix procedure takes two lists of SQL queries. It returns true if and only if (1) the first list is strictly shorter than the second list and (2) the queries in the first list are exactly the same as the corresponding queries at the beginning of the second list.

When a loop is found, the value $l_\alpha$ represents the number of queries in the execution trace that occur after all loop iterations end. Because this value indicates exactly where the loop ends, it enables Shear to distinguish the loop body and the after-loop subprogram without ambiguity.

This procedure detects a loop if and only if the hypothetical loop is consistent with the three (altered) executions. We show that it detects the potential loop accurately (Section 4.4).

**DetectIters**: When a loop has been detected, DETECTLOOPS invokes the DetectIters procedure (Algorithm 5) to identify the boundaries for each loop iteration. The DetectIters procedure takes a program, a context $\sigma$, and three integers $k, r, l_\alpha$. Shear invokes this procedure only when it

**Algorithm 5 Identify iteration lengths**

**Input:** $P$ is the executable of $P \in S$. $\sigma$ is a context. $k, r, l_\alpha$ are integers denoting a loop at $k$-th query, the number of iterations, and the number of queries after loop, respectively.

**Output:** List $l_\beta$ of $r$ integers, where the $i$-th ($i = 1, \ldots, r$) integer represents the number of queries produced by the $i$-th loop iteration.

```plaintext
1: procedure DetectIters($P$, $\sigma$, $k$, $r$, $l_\alpha$)
2: $l_\beta \leftarrow$ Empty list
3: for $i \leftarrow 1, \ldots, r$ do
4: $s_i \leftarrow$ ExecAndPick($P$, $\sigma$, $k$, $[i]$)
5: $l_i \leftarrow$ Len($s_i$) - $k - l_\alpha$
6: Append $l_i$ to $l_\beta$
7: end for
8: return $l_\beta$
9: end procedure
```

**Algorithm 6 Detect all loops**

**Input:** $P$ is the executable of $P \in S$. $\sigma$ is a context. $e$ is the list of query-result pairs obtained from executing $P$ with $\sigma$.

**Output:** Loop layout tree constructed from $e$.

```plaintext
1: procedure DetectLoops($P$, $\sigma$, $e$)
2: $L \leftarrow$ Empty list
3: $(Q_1, r_1), \ldots, (Q_n, r_n) \leftarrow e$
4: for $k = 1, \ldots, n$ do
5: if $r_k < 2$ then continue end if
6: $(f, l_\alpha) \leftarrow$ DetectIters($P$, $\sigma$, $k$)
7: if not $f$ then continue end if
8: $l_\beta \leftarrow$ DetectIters($P$, $\sigma$, $k$, $r_k$, $l_\alpha$)
9: Append $(k, l_\beta)$ to $L$
10: end for
11: return BuildLTTTree($e$, $L$)
12: end procedure
```
detects a loop that iterates over the $r$ rows retrieved by the $k$-th query in the trace from executing the program with $\sigma$. Because loops in the Shear DSL iterate over each row retrieved by the query (Section 3.2), the loop has $r$ iterations each accessing one row from the $k$-th query. The integer $l_\alpha$ is a result from DetectHypo (Algorithm 4) and equals the number of queries in the trace that are produced by the subprogram that follows the detected loop.

The procedure DetectIters identifies the length of each loop iteration in the trace. Specifically, the procedure invokes ExecAndPick for $r$ times. Each invocation executes the program with context $\sigma$, but altered so that the $k$-th query retrieves only one row each time. Each such execution produces a list of (altered) SQL queries. In each list, the first $(k-1)$ queries are intact, while the queries after the $k$-th query are altered. These suffix queries correspond to the queries that would have been produced by one iteration of the loop, followed by $l_\alpha$ queries that are produced by the subprogram in $P$ after the loop. Figure 13 illustrates these calculations.

The procedure calculates the lengths of each loop iteration, then returns all of these lengths as a list $l_\beta$. Using this list of lengths, it is straightforward to divide the intact execution trace (line 3 of Algorithm 6) into segments that correspond to the individual loop iterations.

### 4.3 Detecting and Representing All Loop Structures

**DetectLoops:** Algorithm 6 first executes the program to collect an intact trace. For each query in the trace, it invokes DetectHypo (Algorithm 4) to infer if a loop iterates over the query. If a loop is found, DetectLoops invokes DetectIters (Algorithm 5) to identify loop iteration boundaries in the trace. Both of these procedures leverage Shear’s proxy that alters database traffic during program execution. The DetectLoops procedure finally invokes BuildLLTree to construct a tree that represents the structure of all loops detected.

**BuildLLTree:** The BuildLLTree procedure takes a list of query-result pairs $e$, along with a list $L$ of all loops detected in $e$. When there are $m$ detected loops, the $j$-th loop iterates over the $k_j$-th query with iteration lengths $l_{\beta j}$. These lengths indicate the location of the queries in $e$ that are generated by each loop. When there are no nested loops, the queries generated by different loops do not overlap. On the other hand, when there are nested loops, the queries generated by an inner loop is a subset of the queries generated by the outer loop. The procedure constructs a loop layout tree that represents the structure of all loops detected in $e$. The procedure builds subtrees bottom-up, first building the subtrees for the last and innermost loops.

Programs in the Shear DSL have the following property: When a loop iterates over the $k$-th query and when Shear deletes the database traffic from the $k$-th query, the altered execution traces are the same with the original intact trace in all of the queries that do not belong to this loop. Any such altered trace differs from the intact trace only by lacking the queries that belong to certain iterations of this loop. As a result, the DetectLoops procedure is able to precisely identify the structures of all loops that iterated at least twice (see Section 5.2) in any intact trace. These algorithms work well for programs that may contain multiple loops, nested or otherwise.

### 4.4 Soundness Proof Outline

We outline a soundness proof for Shear’s active loop detection algorithm.

**Definition 11.** For program $P \in \text{Prog}$ and context $\sigma \in \text{Context}$, $\sigma[\rightarrow_{\leq k} P]$ denotes the updated context after evaluating $P$ in $\sigma$ for the first $k$ queries. Figure 14 defines this concept. $\text{Len}_{\text{exec}}(P, \sigma)$

---

1 A straightforward optimization is to avoid repeatedly executing the program with the same context and alterations.
\[
\sigma \vdash_0 P = \sigma \\
\sigma[Q.y \mapsto \sigma(Q)[|\sigma]\downarrow_0 P] = \sigma' \\
\sigma[|\sigma]\downarrow_{k+1} Q P] = \sigma' \\
|\sigma(Q)| = 0 \quad \sigma[|\sigma]\downarrow_{k+1} P] = \sigma' \\
\sigma[|\sigma]\downarrow_{k+1} \downarrow_0 P \text{ if } Q \text{ then } P_1 \text{ else } P_2 = \sigma' \\
|\sigma(Q)| > 0 \quad \sigma[Q.y \mapsto \sigma(Q)[|\sigma]\downarrow_{k+1} P_1] = \sigma' \\
\sigma[|\sigma]\downarrow_{k+1} \downarrow_0 P \text{ if } Q \text{ then } P_1 \text{ else } P_2 = \sigma' \\
|\sigma(Q)| = 0 \quad \sigma[|\sigma]\downarrow_{k+1} P] = \sigma' \\
\sigma[|\sigma]\downarrow_{k+1} \text{ for } Q \text{ do } P_1; P_2 = \sigma' \\
\sigma(Q) = (x_1, \ldots, x_r) \quad r > 0 \quad \sigma[|\sigma]\downarrow_{k+1} P_3] = \sigma' \\
\text{Len}_{\text{exec}}(P_1, \sigma[Q.y \mapsto x_j]) = k_j \quad \text{for all } i = 1, \ldots, r \\
\sigma[|\sigma]\downarrow_{k_1+\ldots+k_r+1} \text{ for } Q \text{ do } P_1; P_2 = \sigma' \\
\sigma(Q) = (x_1, \ldots, x_r) \quad 0 < j < r \\
\text{Len}_{\text{exec}}(P_1, \sigma[Q.y \mapsto x_j]) = k_j \quad \text{for all } i = 1, \ldots, j-1 \\
\text{Len}_{\text{exec}}(P_1, \sigma[Q.y \mapsto x_j]) > k_j \\
\sigma[|\sigma]\downarrow_{k_1+\ldots+k_j+1} \text{ for } Q \text{ do } P_1; P_2 = \sigma' \\
P, P_1, P_2 \in \text{Prog}, \quad Q \in \text{Query}, \quad \sigma, \sigma' \in \text{Context}, \quad y \in \text{Variable}, \\
r, i, j, k, k_1, \ldots, k_r \in \mathbb{Z}_{\geq 0}, \quad x_1, \ldots, x_r \in \text{CRow} \\
\]

<table>
<thead>
<tr>
<th>\text{(zero)}</th>
<th>\text{(seq)}</th>
<th>\text{(if-0)}</th>
<th>\text{(if-1)}</th>
<th>\text{(for-0)}</th>
<th>\text{(for-r)}</th>
<th>\text{(for-j)}</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma[\vdash_0 P] = \sigma$</td>
<td>$\sigma[Q.y \mapsto \sigma(Q)[</td>
<td>\sigma]\downarrow_0 P] = \sigma'$</td>
<td>$\sigma[</td>
<td>\sigma]\downarrow_{k+1} Q P] = \sigma'$</td>
<td>$</td>
<td>\sigma(Q)</td>
</tr>
</tbody>
</table>

Fig. 14. Updating a context after evaluating a program prefix

denotes the length of the trace from evaluating $P$ in $\sigma$, that is, \text{Len}_{\text{exec}}(P, \sigma) = \text{Len}(e)$ where $\sigma \vdash P \downarrow_0 e$.

**Definition 12.** For program $P \in \text{Prog}$ and list of query-result pairs $e$, $P - e = P'$ denotes the remaining subprogram after consuming $P$ with $e$. Figure 15 defines this concept. The CONNECTPROGS procedure takes a list of programs in Prog, connects them in the provided order using Seq, then restructures it so that the resulting program has no nested If statements and belongs to Prog.

**Definition 13.** For program $P \in \text{Prog}$ and context $\sigma \in \text{Context}$, a loop iterates over the $k$-th query for $r_k$ times if the following hold for some $P_1, P_2 \in \text{Prog}$: $\sigma \vdash P \downarrow_0 (Q_1, r_1), \ldots, (Q_n, r_n)$, $1 \leq k \leq n$, and $P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = \text{for } Q_k \text{ do } P_1; P_2$. In this case, $\sigma \vdash_k P \downarrow_0 e$ denotes the list of query-result pairs $e$ produced by the subprogram before the loop. $\sigma \vdash_k P \downarrow_0 e_i$ denotes the list of query-result pairs $e_i$ obtained from evaluating the $i$-th iteration of the loop ($i = 1, \ldots, r_k$). $\sigma \vdash_k P \downarrow_0 e$ denotes the list of query-result pairs $e$ produced by the subprogram after the loop. $\Lambda(P, \sigma, k) = \{e_1, \ldots, e_{r_k}\}$ denotes the list of lists of query-result pairs produced by each of the $r_k$ loop iterations. When no loops iterate over the $k$-th query, $\Lambda(P, \sigma, k) = \text{NotLoop}$. Figure 16 and Figure 17 define these concepts. $\text{Len}_{\text{iter}}(P, \sigma, k, i)$ denotes the number of queries produced by the $i$-th iteration of the loop, that is, $\text{Len}_{\text{iter}}(P, \sigma, k, i) = \text{Len}(e_i)$ where $\sigma \vdash_k P \downarrow_0 e_i$ ($i = 1, \ldots, r_k$).

**Definition 14.** For program $P \in \text{Prog}$, context $\sigma \in \text{Context}$, and integer $k$, $P[k]_{\sigma}$ denotes the $k$-th query evaluated when executing $P$ in $\sigma$. Figure 18 defines this concept.
Theorem 1. For any program $P$ and context $\sigma \in \text{Context}$, integer $k$ where a loop iterates over the $k$-th query, and list of integers $\rho_1, \ldots, \rho_m, \sigma \vdash_k: P \parallel_{\text{alter}}^{\rho_1, \ldots, \rho_m} e$ denotes the list of query-result pairs $e$ produced by an altered evaluation of $P$ in $\sigma$ where the loop that iterates over the $k$-th query performs only the iterations $\rho_1, \ldots, \rho_m$. Figure 19 defines this concept.

Recall that a key idea of SHEAR is to manipulate the program’s database traffic during executions to observe if the altered behavior matches hypothetical loop structures.

Definition 15. For program $P \in \text{Prog}$, context $\sigma \in \text{Context}$, integer $k$ where a loop iterates over the $k$-th query, and list of distinct integers $\rho_1, \ldots, \rho_m$, $\sigma \vdash_k: P \parallel_{\text{alter}}^{\rho_1, \ldots, \rho_m} e$ denotes the list of query-result pairs $e$ produced by an altered evaluation of $P$ in $\sigma$ where the loop that iterates over the $k$-th query performs only the iterations $\rho_1, \ldots, \rho_m$. Figure 19 defines this concept.

Proposition 1. For program $P \in \mathcal{S}$, context $\sigma \in \text{Context}$, and integer $k$, if $\Lambda(P, \sigma, k) = [e_1, \ldots, e_r]$, $\sigma \vdash_k: P \parallel_{\text{alter}}^{\rho_1, \ldots, \rho_m} (Q_1, r_1), \ldots, (Q_n, r_n)$, and $\text{EXECANDPick}(P, \sigma, k, []) = [q_1, \ldots, q_n]$, then we have $n = n'$ and $q_i$ corresponds to $Q_i$ for all $i = 1, \ldots, n$.

Proposition 2. For program $P \in \mathcal{S}$, context $\sigma \in \text{Context}$, integer $k$, and list of distinct integers $\rho_1, \ldots, \rho_m$ ($m > 0$), if $\Lambda(P, \sigma, k) = [e_1, \ldots, e_r]$, $1 \leq \rho_1 < \ldots < \rho_m \leq r$, $\sigma \vdash_k: P \parallel_{\text{alter}}^{\rho_1, \ldots, \rho_m} (Q_1, r_1), \ldots, (Q_n, r_n)$, and $\text{EXECANDPick}(P, \sigma, k, [\rho_1, \ldots, \rho_m]) = [q_1, \ldots, q_n]$, then we have $n = n'$ and $q_i$ corresponds to $Q_i$ for all $i = 1, \ldots, n$.

Theorem 1. For any program $P \in \mathcal{S}$, context $\sigma \in \text{Context}$, and integer $k$, if we have $\sigma \vdash_P \parallel_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n)$, $1 \leq k \leq n$, $r_k \geq 2$, and $\Lambda(P, \sigma, k) = [e_1, \ldots, e_{r_k}]$, then we have $\text{DETECTHYPO}(P, \sigma, k) = \langle \text{true}, \text{LEN}(e') \rangle$ where $\sigma \vdash_k: P \parallel_{\text{alter}} e'$.

Proof Sketch. By induction on $k$ and the derivation of loop body.
Fig. 16. Calculate query-result pairs that correspond to evaluating the subprograms before a loop, in each loop iteration, and after the loop

\[
\begin{align*}
\sigma \vdash P \downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) &\quad 1 \leq k \leq n \\
P - (Q_1, r_1), \ldots, (Q_k-1, r_{k-1}) &= \text{for } Q_k \text{ do } P_1; P_2 \\
\sigma \vdash k, P \downarrow_{\text{before}} (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) \\
\sigma \vdash P \downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) &\quad 1 \leq k \leq n \\
P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) &= \text{for } Q_k \text{ do } P_1; P_2 \\
\sigma[\rightarrow k, P] \vdash P_1 \downarrow_{\text{exec}} e \\
\sigma \vdash k, P \downarrow_{\text{iter}} e \\
\sigma \vdash P \downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) &\quad 1 \leq k \leq n \\
P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) &= \text{for } Q_k \text{ do } P_1; P_2 \\
l_j = \text{Len}_{\text{iter}}(P, \sigma, k, i) \quad \text{for all } i = 1, \ldots, j - 1 \\
2 \leq j \leq r_k \quad \sigma[\rightarrow k + l_1 + \cdots + l_j - 1] \vdash P_1 \downarrow_{\text{exec}} e \\
\sigma \vdash k, P \downarrow_{\text{iter}} e \\
\sigma \vdash P \downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) &\quad 1 \leq k \leq n \\
P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) &= \text{for } Q_k \text{ do } P_1; P_2 \\
\sigma[\rightarrow k-1] P \vdash P_2 \downarrow_{\text{exec}} e \\
\sigma \vdash k, P \downarrow_{\text{after}} e \\
P, P_1, P_2, \in \text{Prog}, \quad Q, Q_1, \ldots, Q_n \in \text{Query}, \quad \sigma \in \text{Context}, \quad n, i, j, k, r_1, \ldots, r_n, l_1, \ldots, l_{j-1} \in \mathbb{Z}_{\geq 0}
\end{align*}
\]

Fig. 17. Check if a loop iterates over a specific query and, if so, calculate each iteration’s corresponding query-result pairs

\[
\begin{align*}
\sigma \vdash P \downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) &\quad 1 \leq k \leq n \\
P - (Q_1, r_1), \ldots, (Q_k-1, r_{k-1}) &= Q_k P_1 \\
\Lambda(P, \sigma, k) = \text{NotLoop} \\
\sigma \vdash P \downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) &\quad 1 \leq k \leq n \\
P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) &= \text{if } Q_k \text{ then } P_1 \text{ else } P_2 \\
\Lambda(P, \sigma, k) = \text{NotLoop} \\
\sigma \vdash P \downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) &\quad 1 \leq k \leq n \\
P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) &= \text{for } Q_k \text{ do } P_1; P_2 \\
\sigma[\rightarrow k, P] \vdash e_j \quad \text{for all } j = 1, \ldots, r_k \\
\Lambda(P, \sigma, k) = [e_1, \ldots, e_r] \\
P, P_1, P_2, \in \text{Prog}, \quad Q, Q_1, \ldots, Q_n \in \text{Query}, \quad \sigma \in \text{Context}, \quad n, j, k, r_1, \ldots, r_n \in \mathbb{Z}_{\geq 0}
\end{align*}
\]

**Theorem 2.** For any program \( P \in S \), context \( \sigma \in \text{Context} \), and integer \( k \), if \( \sigma \vdash P \downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n), \quad 1 \leq k \leq n, \quad r_k \geq 2 \), and \( \Lambda(P, \sigma, k) = \text{NotLoop} \), then \( \text{DetectHypo}(P, \sigma, k) = (\text{false}, \text{Nil}) \).
Active loop detection enables Shear. Theorem 4. For any program \( P \), the loop layout trees for any program in the DSL. Theorem 3. For any program \( P \), context \( \sigma \), and integer \( k \), if \( \Lambda(P, \sigma, k) = [e_1, \ldots, e_r] \) and \( \sigma \vdash k \cdot P \mathbin\|_{\text{before}} e' \), then \( \text{DetectLoopIters}(P, \sigma, k, \text{Len}(e')) = [\text{Len}(e_1), \ldots, \text{Len}(e_r)] \).

Theorem 4. For any program \( P \), context \( \sigma \), and integer \( k \), if \( \sigma \vdash P \mathbin\|_{\text{exec}} e \) and \( \sigma \vdash P \mathbin\|_{\text{loops}} l \), then \( \text{DetectLoops}(P, \sigma, e) = l \).

These results indicate that Shear’s active loop detection technique is guaranteed to detect loops precisely and return the correct loop layout trees for any program in the Shear DSL.

5 USING ACTIVE LOOP DETECTION TO INFERENCE PROGRAMS

Shear instantiates active loop detection on a prior framework for program inference (Section 3.1). Active loop detection enables Shear to work with a wider range of loop and repetitive structures in database programs. We present two enhanced aspects of the program inference algorithm: (1)
traversing loop structures in the Shear DSL during the recursive inference algorithm and (2) using the partially inferred program AST to identify loops that iterated only once.

5.1 Recursive Inference Algorithm with Active Loop Detection

We present an enhanced version of the recursive inference algorithm from Konure. This enhanced algorithm is capable of reasoning about the new and more expressive loop structures in the Shear DSL. We follow the notation of the helper procedures in Section 3.4.

**InferProg:** Algorithm 7 infers programs in the Shear DSL that may contain sophisticated loop and repetitive structures, including nested loops and after-loop subprograms. This procedure recursively explores all relevant paths through a hypothetical DSL program and resolves Prog nonterminals as they are encountered. The procedure takes as parameters the executable $P$ and an annotated trace. The annotated trace consists of a prefix $s_1$ that corresponds to an explored path through the hypothetical DSL program and a suffix $s_2$ from the remaining unexplored part of the hypothetical DSL program. The first Query $Q$ in $s_2$ is generated by a Prog nonterminal to resolve. To resolve a Prog nonterminal, Shear examines three annotated traces $t_0$, $t_1$, and $t_2$. All of these traces are from executions that follow the same path to $Q$ as $s_1$. In the executions that generated $t_0$, $t_1$, and $t_2$, $Q$ retrieves zero rows, at least one row, and at least two rows, respectively. Shear encodes these requirements into path constraints by invoking MakePathConstraint. For these path constraints, the after-loop flags for $Q$ are all set to false. The @ operator performs list concatenation. Shear then obtains the satisfying traces (if they exist) by invoking SolveAndGetTrace.

If Shear detects a loop in $t_2$ that iterates over the rows retrieved by $Q$, Shear infers that $Q$ was generated by a For statement. To infer the subprograms of the For statement, InferProg obtains two additional annotated traces. The trace $t_{iter}$ is an annotated trace whose query $Q$ retrieves at least two rows and whose suffix is generated by the loop body. The trace $t_{after}$ is an annotated trace whose suffix is generated not by the loop body, but by the after-loop subprogram. The InferProg procedure then uses $t_{iter}$ and $t_{after}$ to recursively infer the loop-body subprogram and the after-loop subprogram, respectively. We discuss complications in Section 5.2.

If Shear does not detect a loop in $t_2$ that iterates over the rows retrieved by $Q$, Shear infers whether $Q$ was generated by an If statement or a Seq statement. This decision is based on whether the queries in $t_0$ that follow $Q$ differ from the queries in $t_1$ that follow $Q$. When Shear infers that $Q$ was generated by an If statement, the InferProg recursively infers the then-branch subprogram and else-branch subprogram using $t_1$ and $t_0$, respectively. When Shear infers that $Q$ was generated by a Seq statement, the procedure recursively infers the subsequent subprogram using one of $t_0$ or $t_1$, whichever exists.

**GetKnownLoops:** The GetKnownLoops procedure takes a trace prefix $s_1$. It returns two lists of boolean flags $d$ and $a$ that indicate the looping structures along the path of $s_1$ in the current inferred partial program. In particular, the boolean flags $d$ indicate the queries in $s_1$ over which a loop iterates. The boolean flags $a$ indicate whether the path enters the loop body or the after-loop subprogram at each of the known loop locations in $s_1$.

5.2 Loops that Iterated Only Once

A complication is when a loop iterated only once during program execution. This complication arises from the ability of the Shear DSL to express subprograms after loops.

Recall that each recursive call to InferProg corresponds to one Prog nonterminal along a path in the program’s AST from root to a leaf. The list of conceptual Prog nonterminals on the recursive call stack always corresponds to the list of queries in the provided trace prefix. Consequently, an annotated trace may contain queries from either the then branch or the else branch of an If
DetectLoops is the executable of a program $P \in S$. $s_1, s_2$ are a prefix and a suffix of an annotated trace. The length of $s_1$ is problematic. Similarly, an annotated trace is designed to contain queries from either the loop body or the after-loop subprogram of a For statement, but not both.

Recall that DetectLoops detects loops in the program as long as the loop iterated at least twice. When a loop iterated only once, the DetectLoops procedure does not immediately determine where the (only) loop iteration ends in the provided list of query-result pairs. The resulting loop layout tree therefore does not characterize this loop. When Shear traverses this tree to generate annotated traces, at least one resulting trace contains both the queries generated by the loop body (which iterated only once) and the queries generated after the loop. These traces, if untreated, would cause the InferProg procedure’s recursive steps to diverge from the structure of the hypothetical program’s AST.

Shear addresses this problem by constructing two new traces based on the problematic trace where the loop iterated only once. One of these new traces contains the queries before the loop and the queries inside the loop body. The other new trace contains the queries before the loop and the queries after the loop ends.

To achieve this goal, Shear repurposes its loop detection algorithm to identify the boundary of the (only one) loop iteration in the trace. Specifically, Shear first matches a trace $t$ against the known Prog nonterminals that have already been inferred. If a query $Q$ is known to be generated by a For statement but retrieved only one row in $t$, then the corresponding loop iterated only once. Hence the trace $t$ is problematic. Shear invokes ExecAndPick with an empty list $p$. In the altered
execution, the query \( Q \) is altered to retrieve zero rows. The corresponding loop iterates for zero times and continues execution after the loop. The resulting trace \( t_{\text{after}} \) contains only the after-loop queries, without any loop-body queries. Next, \( \text{Shear} \) uses \( t_{\text{after}} \) to locate the boundary of the loop body in \( t \) and discards all of the subsequent queries. The resulting trace \( t_{\text{iter}} \) contains only the loop-body queries, without any after-loop queries. Figure 20 illustrates these calculations. This way, even though the loop iterated only once in the original trace \( t \), \( \text{Shear} \) is able to discard \( t \) and replace it with two new traces \( t_{\text{iter}} \) and \( t_{\text{after}} \) that correspond precisely to the loop body and the after-loop subprogram, respectively.

5.3 Soundness Proof Outline

The soundness of the recursive inference algorithm builds on the results in Section 4.4. The proof structure for this part of the algorithm is similar to that of prior work [Shen and Rinard 2019, 2021]. Like prior work, we prove the following theorem for programs whose externally observable behavior is in \( S \) with no Print statements. The main difference in this proof is a new discussion of the after-loop subprogram during the structural induction proof of the \( \text{InferProg} \) procedure.

**Theorem 5.** For any program \( P \in S \), \( \text{Infer}(P) \) and \( P \) are identical except for the use of different but equivalent origin locations.

5.4 Discussion

\( \text{Shear} \)'s active loop detection algorithm works well with any context for (and any initial execution of) the program, regardless of whether the program contains repetitive queries or not. Loop detection is based mainly on how the trace changes when the \( \text{Shear} \) proxy removes certain iterations of a hypothetical loop. Loop boundaries are determined by how the trace changes when the proxy removes all but one loop iteration. By using active loop detection, \( \text{Shear} \) supports a wide range of programs in the \( \text{Shear} \) DSL.

A fundamental enhancement of \( \text{Shear} \) over Konure [Shen and Rinard 2019] is the new capability to infer and regenerate expressive loop structures. Konure detects loops by detecting repetitions in execution traces of programs that access relational databases. Unlike \( \text{Shear} \), Konure does not manipulate the database traffic and instead uses a collection of heuristics. These heuristics impose a range of restrictions on the structure of the program — for example, Konure requires the (unchecked) property that any query that follows a query in the DSL program that may retrieve multiple rows must not have the same query skeleton as any following query. It also requires any loop to be the last statement of the program, that is, Konure does not allow any other statements to follow after a loop ends. And it does not support nested loops.

The active loop detection algorithm in \( \text{Shear} \) eliminates all of these restrictions, enabling \( \text{Shear} \) to correctly infer a larger range of programs (Section 6). In other words, despite supporting a more expressive DSL, \( \text{Shear} \) still achieves the same level of soundness guarantees. This enhanced capability highlights the effectiveness of active loop detection.

6 EVALUATION

We evaluate our active loop detection technique with the following research questions:

- **RQ1: Scalability.** How does active loop detection scale with more complex loop structures?
• RQ2: Efficiency. How does active loop detection compare with enumerative search?

• RQ3: Effects on Inferring Benchmark Programs. How does SHEAR compare with existing techniques in the ability to infer database applications?

We performed experiments on Ubuntu 16.04 virtual machines with 6 cores and 4 GB memory. The host machine uses a processor with 6 cores (2.9 GHz Intel Core i9) and has 32 GB 2400 MHz DDR4 memory. Our SHEAR implementation uses Python 3.7.9 (PyPy3 7.3.3) and Z3 4.6.0.

6.1 RQ1: Scalability

We empirically evaluate the scalability of SHEAR’s loop detection algorithm along seven dimensions of program complexity. These dimensions are: (a) the number of consecutive loops in the program, (b) the number of iterations executed for a loop, (c) the number of non-loop repetitive queries in the program, (d) the number of queries in the body of a loop, (e) the number of non-loop queries that retrieve multiple rows during execution, (f) the number of both consecutive loops and database rows, and (g) the number of layers of nested loops. The dimension f is designed to simulate the effects of applying SHEAR to infer full programs, because our database solver often inserts more rows for longer program paths. By definition, the dimension f reflects the combined effects of both dimensions a and b.

Independent Variables: For each dimension of program complexity, we developed a set of synthetic Python programs (along with the associated input and database values) with various levels of complexity. Each program’s externally observable behavior is expressible in the SHEAR DSL. Specifically, we generated the synthetic programs for dimension (a) from Figure 21a with \( r = 2 \), \( b = 1 \), and varying values for \( l \), (b) from Figure 21a with \( l = 1 \), \( b = 1 \), and varying values for \( r \), (c) from Figure 21b with \( r = 8 \) and varying values for \( q \), (d) from Figure 21a with \( r = 2 \), \( l = 1 \), and varying values for \( b \), (e) from Figure 21a with \( r = 2 \), \( b = 0 \), and varying values for \( l \), (f) from Figure 21a with \( b = 1 \) and varying values for both \( r \) and \( l \) (\( r = l \)), and (g) from Figure 21c with varying values for \( n \). Here \( r \) denotes the number of rows we insert into the table \( t1 \) in the database.

Note that for the complexity dimensions b and f, setting \( r = 1 \) means inserting only one row into the table \( t1 \), which causes the loop to iterate only once. Loops that iterated only once are not identified by loop detection alone (Section 5.2).

Dependent Variables: For each dimension of complexity and value of the independent variable, we executed the synthetic program with the associated input and database values. We measured the wall-clock time for executing the program. This execution produces an intact trace. We then used this intact trace (along with the executable program) to invoke SHEAR’s active loop detection algorithm. We measured the wall-clock time for detecting loops in this trace.

Experimental Results: Figure 22 presents the scalability results for SHEAR’s loop detection. Each subfigure corresponds to a dimension of complexity. The horizontal axes represent the independent variables. The vertical axes represent the dependent variables in numbers of seconds.

```python
for n_loops in range(l):
    s = do_sql("SELECT * FROM t1")
    for row in s:
        for n_body in range(b):
            do_sql("SELECT * FROM t0")
```

(a) with \( l \) consecutive loops, where each loop body has \( b \) queries

```python
do_sql("SELECT * FROM t1")
for n_query in range(q):
    do_sql("SELECT * FROM t2")
```

(b) with \( q \) non-loop repetitive queries

```python
def rec(n_layer):
    if n_layer <= 0:
        return
    s = do_sql("SELECT * FROM t0")
    for row in s:
        rec(n_layer - 1)
    rec(n)
```

(c) with \( n \) layers of nested loops

Fig. 21. Programs for testing scalability
The green circles represent program execution times. In the first five subfigures, the program execution times vary between 2.9–3.7 seconds (average 3.1 seconds). In Figure 22f, the program execution times vary between 3.0–8.1 seconds (average 4.8 seconds). In Figure 22g, the program execution times vary between 3.1–4.7 seconds for 0–6 layers of nested loops and between 6.1–11.4 seconds for 7–9 layers. The program execution time is positively correlated with the length of the intact execution trace. The traces in our experiments contain between 0–48 queries (Figure 22a), 2–17 queries (Figure 22b), 3–33 queries (Figure 22d), 0–16 queries (Figure 22e), 2–272 queries (Figure 22f), and 1–127 queries for 0–6 layers of nested loops and 255–1023 queries for 7–9 layers (Figure 22g).

The blue triangles represent loop detection times. The loop detection times vary between 0–456 seconds in the first five subfigures, between 0–1336 seconds in Figure 22f, and between 2–978 seconds in Figure 22g for up to 6 layers of nested loops. The algorithm identifies loop structures in the trace, specifically, whether any loops iterated over any queries and where the loop iteration boundaries are. Shear correctly identifies the unique correct loop structures for all of these traces.

**Discussion:** In these experiments, the growths of the loop detection times are correlated mainly with the number of altered program executions during loop detection. For the plotted values, the loop detection times scale linearly with the number of loops (Figure 22a), the number of loop iterations (Figure 22b), and the number of queries retrieving multiple rows (Figure 22e). For the plotted values, the loop detection times remain stable with various numbers of non-loop repetitions (Figure 22c) and numbers of queries in the loop body (Figure 22d) and scale quadratically with the number of both loops and database rows (Figure 22f). In theory, the asymptotic growths of the loop detection times are also correlated with the times for executing the program and the times for parsing the execution traces. These two time components both grow linearly along dimensions a–e and quadratically along the dimension f. The loop detection times grow exponentially with increasing layers of nested loops (Figure 22g). We attribute this phenomenon to the exponential growth in the lengths of execution traces for deeper nested loops. Overall, Shear’s active loop detection algorithm scales polynomially along the first six dimensions of program complexity.

### 6.2 RQ2: Efficiency

We evaluate the asymptotic efficiency of active loop detection by contrasting the loop detection time against the size of the search space of candidate program structures. We note that many program synthesis techniques are based on enumerative search, often with pruning strategies [Ahmad and Cheung 2018; Albarghouthi et al. 2013; Biemann et al. 1975; Chen et al. 2021; Feser et al. 2015; Gulwani 2010; Ji et al. 2020b; Lee et al. 2016, 2018; Osera and Zdancewic 2015; Pailoor et al. 2021; Polikarpova et al. 2016; So and Oh 2017, 2018; Wang et al. 2017b,c,d; Ye et al. 2020].

**Independent Variables:** We conducted two experiments that calculate the search space sizes. The first experiment uses the same set of independent variables as in Section 6.1. The second experiment uses the maximum depth of programs as the independent variable, where depth is defined as the maximum number of SQL queries along any path in the program’s AST.

**Dependent Variables:** The first experiment uses the intact traces collected from executing the synthetic programs for the various dimensions of complexity and values of the independent variables. For each trace, we calculated the number of candidate loop structures that may generate the trace. Specifically, we calculated the numbers of different ways to split the trace into loop iterations as follows. Potential loops may iterate over a query $q$ in the trace if (1) the query retrieves $r$ rows of data, where $r \geq 2$, and (2) the subsequent query $q'$ occurs $m$ times in the remaining trace, where $m \geq r$. For each such query $q$ with potential loops, we count all of the potential ways to allocate the $m$ occurrences of $q'$ into $r$ loop iterations. Each of the first $(r - 1)$ loop iterations must end right before the next iteration starts. The last loop iteration may potentially end at any
subsequent query in the trace. For each of these potential ways to split the trace into loop iterations, we recursively calculate the numbers of potential loops nested inside the loop iterations or located after the loop ends.

The second experiment is based on theoretical calculation. For each maximum program depth, we calculated the number of candidate AST program structures under this depth. Specifically, we calculated the number of different expansions of the Prog nonterminal symbol with the formula:

\[ N_{k+1} = N_k + (N_k - 1) + (N_k - 1)N_k + 1, \]

where \( N_k \) denotes the number of different AST “Prog” expansions for programs whose depths are at most \( k \) (\( k \geq 1 \)). The four addends on the right-hand side correspond to the cases where the topmost Prog nonterminal symbol expands into Seq, If, For, and \( \epsilon \), respectively. The base case is \( N_1 = 2 \), which accounts for an empty program and a program that consists of a single query.

**Experimental Results:** Figure 23 presents the results for the first experiment. Each subfigure corresponds to one dimension of program complexity. The horizontal axes represent the independent variables. The vertical axes represent the sizes of the search space of candidate loop structures given an intact trace. To evaluate the efficiency of Shear’s active loop detection algorithm, we compare the asymptotic growths of the search space sizes against the loop detection times. For the plotted values, the search space sizes grow at least exponentially with increasing numbers of consecutive loops (Figure 23a), numbers of queries retrieving multiple rows (Figure 23e), and numbers of both loops and database rows (Figure 23f). In contrast, the Shear loop detection times grow polynomially along these dimensions (Figure 22a, Figure 22e, and Figure 22f). Compared to the loop detection times, the search space sizes also grow asymptotically faster with increasing numbers of non-loop repetitions (Figure 23c), numbers of queries in the loop body (Figure 23d), and numbers of layers of nested loops (Figure 23g). We attribute these differences to Shear’s ability to detect loops efficiently. The only dimension along which the search space sizes grow slower than
loop detection times is the number of loop iterations (Figure 23b). We attribute this difference to the increased number of program executions required for SHEAR to detect the boundaries for an increased number of loop iterations.

Figure 24 presents the results for the second experiment. The horizontal axis represents the maximum depth of a program in the SHEAR DSL. The vertical axis represents the size of the search space of candidate AST structures. Because these numbers are too large for our plotting software, we plot them after taking logarithm with base 10. This search space grows double-exponentially with the maximum program depth.

Discussion: Overall, the asymptotic performance of SHEAR’s active loop detection algorithm (Figure 22) is very efficient compared to the search space of candidate loop structures (Figure 23). We attribute this efficiency to the fact that SHEAR’s algorithm does not require enumerative search—instead, it is based on manipulating the program executions and comparing the trace lengths. SHEAR’s loop detection is not only efficient but also precise. The detected loop structures are guaranteed to be the unique correct answers (Section 4.4).

We note that the number of candidate loop structures is only a small fraction of the number of candidate AST programs. A naïve search space of candidate AST programs may contain programs whose depths are less than or equal to the trace length, up to 272 in our experiments. Such a search space may contain up to $N^{272}$ (defined above) candidate AST structures in our experiments. This search space is way beyond the plotting capability of Figure 24 and infeasible to enumerate. In contrast, the SHEAR loop detection algorithm terminates within half an hour on a laptop computer for all of these experiments (Figure 22).

6.3 RQ3: Effects on Inferring Benchmark Programs

Benchmarks: We evaluate SHEAR on the following benchmark applications and a synthetic test suite. Each application has several commands; SHEAR infers one command at a time. Each command takes input parameters, performs SQL queries accordingly, and outputs some of the retrieved data.

- ** RailsCollab Project Manager:** RailsCollab [rai 2021] is an open source project management and collaboration tool, built with Ruby on Rails, with over 250 stars on GitHub. The source code contains 11944 lines of Ruby, HTML, CSS, and JavaScript. RailsCollab maintains multiple task lists, tasks, milestones, time records, and messages. RailsCollab retrieves data from 24 relevant tables with 270 columns. Its commands enable users to navigate these contents.

- ** Kanban Task Manager:** Kanban [kan 2021] is an open source task management system, built with Ruby on Rails, with over 600 stars and 200 forks on GitHub. The source code contains 1653 lines of JavaScript, SASS, Ruby, and HTML. Kanban maintains boards. Each board may contain multiple lists. Each list may contain multiple cards, each of which may have comments. Kanban retrieves data from 4 relevant tables with 42 columns. Its commands enable users to navigate boards, lists, cards, and comments.

- ** Todo Task Manager:** Todo [tod 2021] is an open source task-tracking tool, built with Ruby on Rails, with over 100 stars and 80 forks on GitHub. The source code contains 1340 lines of HTML, JavaScript, Ruby, CSS, and SASS. Todo maintains multiple lists. Each list may contain multiple tasks. Todo retrieves data from 2 relevant tables with 10 columns. Its commands enable users to navigate lists and tasks.

- ** Fulcrum Task Manager:** Fulcrum [ful 2018] is an open source project planning tool, built with Ruby on Rails, with over 1500 stars on GitHub. The source code contains 3642 lines of JavaScript, Ruby, SASS, and HTML. Fulcrum maintains multiple projects. Each project may
contain multiple stories. Each story may contain multiple notes. Fulcrum retrieves data from 5 relevant tables with 55 columns. Its commands enable users to navigate the contents of projects, stories, and notes, as well as the users who created these contents.

- **Kandan Chat Room**: Kandan [kan 2018] (distinct from Kanban) is an open source chat room application, built with Ruby on Rails, with over 2700 stars on GitHub. The source code contains 8438 lines of JavaScript, CoffeeScript, SASS, CSS, Ruby, and HTML. Kandan maintains multiple chat rooms (so-called channels) that users can access. Kandan retrieves data from 4 relevant tables with 41 columns. Its commands enable users to navigate chat rooms and messages (so-called activities) and display relevant user information.

- **Enki Blogging Application**: Enki [enk 2018] is an open source blogging application, built with Ruby on Rails, with over 2700 stars and 280 forks on GitHub. The source code contains 2589 lines of Ruby, HTML, JavaScript, CSS, and SASS. Enki maintains multiple pages and posts, each of which may have comments. Enki retrieves data from 5 relevant tables with 39 columns. Its commands enable the author of the blog to navigate pages, posts, and comments.

- **Blog**: The Blog application is an example obtained from the Ruby on Rails website [rai 2018]. The source code contains 232 lines of HTML, Ruby, and JavaScript. Blog maintains information about blog articles and blog comments. Blog retrieves data from 2 relevant tables with 11 columns. It implements a command that retrieves all articles and a command that retrieves a specific article and its associated comments.

- **Student Registration**: The student registration application is a student registration system adapted from an earlier version of a program developed by the MITRE Corporation, which was developed specifically for studying the detection and nullification of SQL injection attacks [nis 2020]. The application was written in Java and interacts with a MySQL database [Widenius and Axmark 2002] via JDBC [Reese 2000]. The source code contains 1264 lines of Java. It retrieves data from 5 relevant tables with 17 columns.

- **Synthetic**: A set of synthetic Python programs with repetitions, nested loops, and consecutive loops designed to challenge other loop detection techniques.

Five of these applications – RailsCollab, Kanban, Fulcrum, Kandan, and Enki – are studied in a recent survey [Yan et al. 2017]. We identified Todo from popular Ruby on Rails projects on GitHub. Five of the applications – Fulcrum, Kandan, Enki, Blog, and Student – were used in the evaluation of Konure [Shen and Rinard 2019]. The synthetic test suite highlights the capability of **Shear**.

**Characteristics of Loop and Repetition Structures**: Our benchmarks have structures as follows.

- **Non-Loop Repetitive Queries**: Codes “R” and “r” denote that a command contains non-loop repetitive queries that violate the heuristic assumptions in other loop detection systems. Specifically, the code “R” denotes that the command may generate an execution trace that contains non-adjacent repetitions. The code “r” denotes generating adjacent repetitions.

- **Control Structures in the Loop Body**: The code “C” denotes that a command has a loop whose loop body contains control structures such as conditional statements or loops.

- **Nested Loops**: The code “N_i” denotes that a command contains i layers of nested loops when the command’s externally observable behavior is expressed in the Shear DSL.

- **After-Loop Subprograms**: The code “A” denotes that a command contains after-loop subprograms. The code “A_i” denotes that the command contains i loops, one after the other.

Commands with code “R” are out of the scope of Konure and Kobayashi’s algorithm [Kobayashi 1984]. Commands with code “r” are out of the scope of Konure, DaViS [Noughi et al. 2014], and Kobayashi’s algorithm. Commands with code “C” are out of the scope of DaViS and Kobayashi’s algorithm. Commands with code “N_i” are out of the scope of Konure, DaViS, and Kobayashi’s algorithm. Commands with code “A” or “A_i” are out of the scope of Konure.
In contrast, only 13 of these commands are supported by at least one of the three other techniques. Overall, 25 benchmark commands (15 from open source applications) are out of the scope of Konure, 18 commands (14 open source) are out of the scope of DaViS, and 25 commands (15 open source) are out of the scope of Kobayashi’s algorithm. In contrast, all of these commands are supported by Shear. We attribute these differences to Shear’s

**Experimental Results for Loop Detection:** Table 1 summarizes commands that contain loops or non-loop repetitive queries as expressed in the Shear DSL. The first two columns (Command and App) present the command and application names. The next (Time) presents the wall-clock inference time. The next two (Q and For) present the numbers of SQL and For statements as expressed in the DSL. The next (Struct) presents the characteristics of loop and repetitive structures outlined above. The next two (S and Kn) represent whether the command is supported by Shear and Konure, respectively. The next two (D and Kb) represent whether the execution traces produced by the command are compatible with DaViS and Kobayashi’s algorithm, respectively.

Compared to these three other techniques, Shear supports a substantially more complex set of loop and repetitive structures. All 25 benchmark commands with loops (whose “For” column is nonzero) are supported by Shear. In contrast, only 13 of these commands are supported by at least one of the three other techniques. Only 4 of these commands are supported by all of the three other techniques. All 6 benchmark commands without loops but contain repetitive queries (whose “For” column is zero) are supported by Shear. In contrast, only one of these commands is supported by at least one of the three other techniques. Overall, 25 benchmark commands (15 from open source applications) are out of the scope of Konure, 18 commands (14 open source) are out of the scope of DaViS, and 25 commands (15 open source) are out of the scope of Kobayashi’s algorithm. In contrast, all of these commands are supported by Shear. We attribute these differences to Shear’s

<table>
<thead>
<tr>
<th>Command</th>
<th>App</th>
<th>Time</th>
<th>Q</th>
<th>For</th>
<th>Struct</th>
<th>S</th>
<th>Kn</th>
<th>D</th>
<th>Kb</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_projects_id_messages</td>
<td>RailsCollab</td>
<td>7203.3s</td>
<td>34</td>
<td>3</td>
<td>R, C, A2</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>get_projects_id_messages_id</td>
<td>RailsCollab</td>
<td>4326.4s</td>
<td>21</td>
<td>0</td>
<td>r</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>get_projects_id_messages_display_list</td>
<td>RailsCollab</td>
<td>7461.4s</td>
<td>35</td>
<td>3</td>
<td>R, C, A2</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>get_projects_id_times (fixed 500 error)</td>
<td>RailsCollab</td>
<td>8739.7s</td>
<td>38</td>
<td>1</td>
<td>R, C, A</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>get_projects_id_times_id</td>
<td>RailsCollab</td>
<td>3846.7s</td>
<td>55</td>
<td>0</td>
<td>R</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>get_projects_id_milestones_id</td>
<td>RailsCollab</td>
<td>5544.6s</td>
<td>25</td>
<td>2</td>
<td>R, r, A</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>get_users_id (fixed 500 error)</td>
<td>RailsCollab</td>
<td>960.3s</td>
<td>12</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>get_api_lists</td>
<td>Kanban</td>
<td>1557.2s</td>
<td>9</td>
<td>3</td>
<td>C, N3</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>get_api_lists_id</td>
<td>Kanban</td>
<td>765.5s</td>
<td>9</td>
<td>2</td>
<td>C, N2</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>get_api_cards</td>
<td>Kanban</td>
<td>780.3s</td>
<td>8</td>
<td>2</td>
<td>C, N2</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>get_api_cards_id</td>
<td>Kanban</td>
<td>398.4s</td>
<td>8</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>get_api_boards_id</td>
<td>Kanban</td>
<td>1329.2s</td>
<td>10</td>
<td>3</td>
<td>C, N3, A</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>get_home</td>
<td>Todo</td>
<td>529.3s</td>
<td>5</td>
<td>1</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>get_lists_id_tasks</td>
<td>Todo</td>
<td>565.9s</td>
<td>6</td>
<td>1</td>
<td>C, A</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>get_lists_id_tasks (fixed 404 error)</td>
<td>Todo</td>
<td>742.2s</td>
<td>6</td>
<td>1</td>
<td>C, A</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>get_channels</td>
<td>Kandan</td>
<td>3762.2s</td>
<td>16</td>
<td>2</td>
<td>C</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>get_home</td>
<td>Enki</td>
<td>757.2s</td>
<td>9</td>
<td>1</td>
<td>A</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>get_archives</td>
<td>Enki</td>
<td>659.4s</td>
<td>6</td>
<td>1</td>
<td>C, A</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>get_admin_posts</td>
<td>Enki</td>
<td>397.5s</td>
<td>3</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>get_admin (trimmed)</td>
<td>Enki</td>
<td>773.9s</td>
<td>7</td>
<td>2</td>
<td>A2</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>Student</td>
<td>171.1s</td>
<td>5</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>repeat_2</td>
<td>Synthetic</td>
<td>82.0s</td>
<td>3</td>
<td>0</td>
<td>r</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>repeat_3</td>
<td>Synthetic</td>
<td>102.7s</td>
<td>4</td>
<td>0</td>
<td>R, r</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>repeat_4</td>
<td>Synthetic</td>
<td>122.2s</td>
<td>5</td>
<td>0</td>
<td>R, r</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>repeat_5</td>
<td>Synthetic</td>
<td>143.0s</td>
<td>6</td>
<td>0</td>
<td>R, r</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>nest</td>
<td>Synthetic</td>
<td>393.8s</td>
<td>3</td>
<td>2</td>
<td>C, N2</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>after_2</td>
<td>Synthetic</td>
<td>220.8s</td>
<td>4</td>
<td>2</td>
<td>R, A2</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>after_3</td>
<td>Synthetic</td>
<td>328.7s</td>
<td>6</td>
<td>3</td>
<td>R, A2</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>after_4</td>
<td>Synthetic</td>
<td>441.1s</td>
<td>8</td>
<td>4</td>
<td>R, A2</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>after_5</td>
<td>Synthetic</td>
<td>549.4s</td>
<td>10</td>
<td>5</td>
<td>R, A2</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>example (Section 2)</td>
<td>Synthetic</td>
<td>293.3s</td>
<td>6</td>
<td>1</td>
<td>R, A</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
capability to precisely disambiguate hypothetical loops, instead of relying on heuristic assumptions as in other techniques.

Compared to Konure, SHEAR supports 19 additional benchmark commands that contain loops and 6 additional benchmark commands without loops. The capability to work with sophisticated loop and repetitive structures enables SHEAR to support commands in real-world applications that are larger and more complex.

**Experimental Results for Program Inference:** We applied SHEAR to infer and regenerate 53 commands in our benchmarks. These commands include 9 from RailsCollab, 7 from Kanban, 3 from Todo, 8 from Fulcrum, 6 from Kandan, 7 from Enki, 2 from Blog, 1 from Student, and 10 from Synthetic. Among these supported commands, 25 (47%) are out of the scope of Konure (Table 1). We present the SHEAR inference performance below and compare it with Konure.

Table 2 presents a summary of SHEAR’s inference effort. The first column (Command) presents the name of the command. The next column (Launch) presents the average time required to tear down, restart, and execute the application (or its web server) in the SHEAR environment. The next column (Runs) presents the number of executions that SHEAR used to infer the command. The next column (= Intact + Alt) presents the number of intact executions and number of altered executions, respectively, that SHEAR used to infer the program. The next column (Solves) presents the number of invocations of the Z3 SMT solver that SHEAR executed to infer the model for the program. The next column (Time) presents the wall-clock inference time. The last column (AlgTime) presents the time spent purely in the inference algorithm. These numbers are calculated by subtracting the total wall-clock time with the total time spent on tearing down, restarting, and executing each command (which is the product of the launch time and the number of runs).

To infer a model for the command as expressed in the SHEAR DSL, the number of total command executions range between 5–434 (average 84.6). Among these executions, the intact executions range between 5–88 (average 22.6). The number of intact executions is positively correlated with the size of the program and the need for database disambiguation, which is positively correlated with the size of the program, the number of outputs, and the number of columns in the database schema. The altered executions range between 0–361 (average 62.0). The number of altered executions is positively correlated with the number of queries that retrieve multiple rows during execution, as well as the number of rows that each such query retrieves (see Section 6.1). The number of altered executions range between 0–7.8 times (average 2.6 times) that of intact executions. The number of solver invocations (Section 5.1) range between 6–474 (average 86.9).

The wall-clock inference times range between 39–8740 seconds (average 1197 seconds). Among the 25 benchmark commands that are out of the scope of Konure (Table 1), SHEAR’s inference times range between 82–8740 seconds (average 1909 seconds). Among the 28 benchmark commands that are supported by both SHEAR and Konure, SHEAR’s inference times range between 39–3762 seconds (average 560 seconds). The former commands are often larger and more complex than the latter.

We next compare SHEAR and Konure on the 21 shared benchmark commands that are used in the evaluation of both systems.\(^2\) For these shared commands, the number of total command executions in SHEAR range between 5–233 (average 47.0). Among these executions, the intact executions range between 5–38 (average 16.4). The altered executions range between 0–201 (average 30.6). As comparison, the number of executions in Konure range between 2–23 (average 9.5). Note that Konure has only intact executions. The intact execution numbers in SHEAR are slightly higher than the corresponding numbers reported in Konure. We attribute this difference to the different uses of the solver when implementing these two systems. The number of altered executions in

---

\(^2\)Our benchmarks contain 7 commands that can be supported by Konure but were not used in Konure’s evaluation.
Shear range between 0–6.3 times (average 1.6 times) that of intact executions. We attribute this overhead to Shear’s algorithm that identifies a wider range of loop and repetitive structures.

For the shared commands, the number of solver invocations (Section 5.1) in Shear range between 9–307 (average 68.5). As comparison, the number of solver invocations in Konure range between 1–242 (average 70.7). The number of solver invocations is positively correlated with the size of the program, the number of command executions, and the need for database disambiguation. These numbers in both systems are roughly in the same range.

The total wall-clock inference time consists of command execution and pure inference (spent mainly around the database solver). The pure inference time is positively correlated with the number of solves and the size of the program. In theory, the pure inference time required by Shear and Konure are similar. In our experiments for the shared commands, the pure inference time in Shear range between 2.8–348 seconds (average 102.2 seconds). As comparison, the pure inference time in Konure range between 2.4–5985 seconds (average 710.3 seconds). In other words, the pure inference times in Shear are often shorter than that of Konure. We attribute this difference to the different uses of the solver when implementing these two systems. The percentage of wall-clock inference time spent by Shear on command executions range between 48.3%–95.9% (average 81.6%). As comparison, this percentage in Konure range between 4.3%–89.1% (average 37.4%). We attribute this difference to the fact that Shear often executes the commands more times to identify more sophisticated loop and repetitive structures.

For the shared commands, the total wall-clock inference time in Shear range between 52–3762 seconds (average 626.9 seconds). As comparison, the total wall-clock inference time in Konure range between 21–6300 seconds (average 809.7 seconds). Shear’s inference times are generally shorter than Konure for commands that required many solver invocations, such as get_projects_id, get_projects_id_users, get_channels, and get_channels_id_activities. We attribute this difference to the different uses of the solver when implementing these two systems. Shear’s inference times are generally longer than Konure for commands that required many executions, such as get_users and get_admin_posts. We attribute this difference to the increased number of command executions by Shear for identifying more expressive loop and repetitive structures.

Experimental Results for Program Regeneration: Table 3 presents a summary of the regenerated Python implementations. We present all of the regenerated programs in Appendix A. The first column (Command) presents the name of the command. The next column (App) presents the name of the application to which the command belongs. The next column (In) presents the number of input parameters for the command. The Q, If, For, Out, and LoC columns present the number of SQL statements, If statements, For statements, lines that generate output, and lines of code. The Comp column presents the number of lines of computation code, calculated by subtracting the number of output statements from the number of the total lines of code.

For each benchmark command, Shear regenerates a Python program. The regenerated programs have between 1–55 (average 9.8) SQL queries, between 0–15 (average 2.3) If statements, between 0–5 (average 0.9) For statements, between 0–50 (average 12.7) lines that generate output, and between 7–147 (average 36.0) total lines of code. The lines of computation code range between 5–97 (average 23.3).

We compared the regenerated programs for the shared commands. For each shared command in Enki, Fulcrum, Kandan, and Student, Shear and Konure infer and regenerate equivalent programs. For each command in Blog, Shear and Konure infer and regenerate equivalent programs except for a minor difference in the scripts for restarting and executing the application.

Environmental Setup and Scope: To enable our Shear implementation to work with these benchmark applications, we configured the environment as follows. We disabled SQL caching in the Ruby on Rails framework. We disabled integrity checks in the MySQL server. We set default
values for encrypted passwords so that our solver generates contexts that enable any valid user to log in. We set defaults for some columns that are compared against constant values, such as setting an admin flag to always true.

Some of these applications implement data retrieval commands that are out of the scope of Shear. Six such commands in RailsCollab and two in Kandan retrieve files or folders. One in Kanban and one in Fulcrum retrieves metadata such as session keys and history updates. Four in RailsCollab and one in Kanban iterate over the rows retrieved by an earlier query that does not immediately precede the first iteration of the loop body. One in RailsCollab contains conditional statements that do not depend on whether the preceding query retrieves empty or nonempty. The majority of the remaining data retrieval commands in Enki and RailsCollab involve application-specific calculations such as concatenating multiple input strings, checking whether a datetime is smaller than another, and enumerating a set of activity type strings. Shear infers all of the data retrieval commands in Todo, Blog, Student, and Synthetic.

**Discussion:** Compared to three other techniques, Shear supports a wider range of loop and repetitive structures that occur in real-world database applications. This new capability enables Shear to infer and regenerate a wider range of larger and more complex database applications than Konure. When applied on identical benchmarks that are supported by both Shear and Konure, the two systems infer and regenerate equivalent results, with the main difference being that Shear may execute the benchmarks more. This overhead enables Shear to infer and regenerate a wider range of programs with more sophisticated loop and repetitive structures. This overhead often becomes less noticeable for larger programs, as the inference algorithm spends a larger portion of the time interacting with the solver. We note that almost half of our benchmark commands contain loop and repetitive structures that are supported by only Shear but not Konure.

## 7 ADAPTATIONS FOR OTHER CONTEXTS

We identify the following properties of Shear’s domain as the key enablers of active loop detection.

- **P1:** The program has strong boundaries between components.
- **P2:** The component interactions can be altered transparently during program execution.
- **P3:** Loops in the program iterate over collections that move across component boundaries.
- **P4:** Each loop iteration operates independently on a single element of the collection.
- **P5:** The execution of each loop iteration generates a deterministic and nonempty trace.

For the Shear DSL, the database traffic (P1) can be altered transparently (P2), the loops iterate over rows retrieved from the database (P3 and P4), and the loop body is always nonempty (P5).

We identify other domains with similar properties and discuss how Shear’s instrumentation can be adapted to work with collections in these domains.

**Adaptations for Mimic:** Mimic [Heule et al. 2015] is designed to synthesize models for opaque JavaScript functions. Mimic uses JavaScript proxy objects [Jav 2019] to forward the interactions between an opaque function and the underlying objects and to record the traces of memory accesses (P1). JavaScript proxy objects can alter the return values for accesses on object properties [Jav 2019] (P2). Potential loops in the Mimic paper [Heule et al. 2015] that may benefit from active loop detection include the functions every, filter, forEach, map, reduce, max, min, and sum (P3–P5). A way to incorporate active loop detection into Mimic is to change its proxy objects to manipulate the results for certain memory accesses while executing the opaque function. For example, the updated Mimic proxy objects could respond the caller function with altered array lengths or elements. These

---

3For a Ruby on Rails application, these commands correspond to the routes that handle HTTP GET requests with an index action, a show action, or an action that displays the current user. We count such routes only if their corresponding actions are implemented and access the database.
extensions would allow Mimic to detect a wider range of loops from traces, such as multiple loops, nested loops, and loops with multiple conditionals in the loop bodies.

**Adaptations for DaViS:** DaViS [Noughi et al. 2014] is designed to visualize program execution to aid comprehension. DaViS extracts the SQL queries performed by a program during execution, using an aspect-based technique [Cleve and Hainaut 2008] (P1). This implementation collects SQL traces by using Java AspectJ advice, which can alter function arguments [Kiczales et al. 2001] (P2). DaViS detects potential loops whose loop body has only one SQL query that references the query preceding the loop (P3–P5). A way to incorporate active loop detection into DaViS is to change its aspect-based tracing technique to manipulate the SQL queries or results during program execution. For example, the updated DaViS tracing advice could alter certain SQL queries at program points before the queries are performed. These extensions would allow DaViS to detect a wider range of loops from traces, such as loops with multiple queries in the loop bodies.

**Adaptations for Nero:** Nero [Wu 2018] is designed to synthesize database programs from seed Python programs. Nero uses wrappers over certain Python data structures to record a trace of accesses to these data structures during program execution (P1). These wrappers can alter the function return values (P2). Nero works with loops that iterate over Python lists and dictionaries where different iterations are nonempty and independent from each other (P3–P5). A way to incorporate active loop detection into Nero is to change its wrappers to manipulate the data structure accesses during program execution. For example, the updated Nero wrappers could respond the caller with altered data structure elements. These extensions would allow Nero to detect loop boundaries accurately, including the end locations of the last iterations of any loops.

**Adaptations for Dispatcher:** Dispatcher [Caballero et al. 2009; Caballero and Song 2013] is designed to reverse engineer a protocol by observing the executions of an application that sends and receives messages following the protocol. Dispatcher collects execution traces of the application with a whole-system emulator [Song et al. 2008] (P1), which can alter the program state during program execution (P2). Dispatcher uses two loop detection methods, among which the “dynamic” method is based on detecting repetitions in a trace. Loops in the application often arise from processing sequences of data fields (P3), whose boundaries are often specified by length fields or delimiters in the message [Caballero et al. 2009; Caballero and Song 2013; Caballero et al. 2007] (P4 and P5). A way to incorporate active loop detection into Dispatcher is to change its emulator to manipulate the application’s memory buffers during execution. For example, the updated Dispatcher emulator could alter the length, the offset, or certain delimiter values in the buffer before allowing the application to enter a hypothetical loop. These extensions would allow Dispatcher to accurately determine the loop boundaries, as well as work with protocols with more sophisticated loop structures.

8 RELATED WORK

Detecting loops from repetitions has been a recurring problem in many areas, including program synthesis, program comprehension, performance profiling, and protocol reverse engineering.

**Detecting Loops from High-Level Execution Traces:** Prior systems that detect loops from program traces have used heuristics to partially address the loop detection problem. These systems either (1) do not attempt to fully identify the loop structure [Caballero et al. 2009; Heule et al. 2015; Wu 2018] or (2) impose restrictions to avoid dealing with ambiguous repetitions [Kobayashi 1984; Noughi et al. 2014; Shen and Rinard 2019]. Mimic [Heule et al. 2015] uses a probabilistic approach to rank candidate loops but does not guarantee identifying the correct loops. Nero [Wu 2018] uses instrumentation to identify where each loop iteration starts in the trace, but does not identify where the last loop iteration ends unless it already knows the loop body. Dispatcher’s [Caballero et al. 2009; Caballero and Song 2013] “dynamic” loop detection technique detects repetitions from traces using
heuristics and does not accurately determine where the loop ends in the trace. DaViS [Noughi et al. 2014] uses heuristics to identify possible nested queries from the SQL trace, where the loop body contains exactly one SQL query. Kobayashi’s algorithm [Kobayashi 1984] is based on repetitions and would not work with programs that contain conditionals in a loop body or ambiguous repetitive instructions. We compare with Konure [Shen and Rinard 2019] in Section 5.4.

**Synthesizing Loops from Observed Executions:** Program synthesis is an active research area. In contrast to systems that synthesize loops by observing executions of an existing program [Biermann et al. 1975; Heule et al. 2015; Qi et al. 2012], SHEAR (1) manipulates the program execution, (2) fully and precisely identifies loop structures in the program, and (3) directly calculates the loop structures based on several program executions, instead of enumerative search.

**Synthesizing Regular Expressions:** A fundamental difference between SHEAR and existing regular expression synthesis techniques is that SHEAR intervenes in program executions to obtain an efficient top-down inference algorithm. There are two kinds of regular expression synthesis algorithms. Techniques that work only with positive and negative examples [Chen et al. 2020; Lee et al. 2016; Ye et al. 2020] provide no guarantees that they will infer the exact regular expression and rely on heuristics to deliver an ordered list of synthesized expressions. SHEAR, in contrast, produces a single correct loop structure within the SHEAR DSL. Techniques that do deliver a single correct regular expression require the ability to ask an oracle whether a candidate is correct [Angluin 1987]. SHEAR does not need this oracle, but instead works with an existing program.

**Memory Address Trace Compression:** Many techniques identify potential loops in memory address traces, often to compress the traces for storage or to improve runtime performance of predicted loops [Burtscher et al. 2005; Elnozahy 1999; Ketterlin and Clauss 2008; Rodriguez et al. 2016]. These techniques are often based on detecting linear progressions from the memory addresses in the traces. SHEAR, in contrast, does not require knowledge of the internal memory layouts.

**Detecting Loops with Static Analysis or Program State:** There are many techniques that detect potential loops according to control flow graphs, instruction addresses, memory addresses, stack frames, register values, taints, or other forms of low-level runtime information [Ahmad and Cheung 2018; Caballero et al. 2009; Caballero and Song 2013; Caballero et al. 2007; Carbin et al. 2011; Cheung et al. 2013; Hayashizaki et al. 2011; Kamil et al. 2016; Kling et al. 2012; Mendis et al. 2015; Moseley et al. 2007; Sato et al. 2011; Tubella and Gonzalez 1998]. In contrast to these techniques, SHEAR treats the program as a black-box executable and observes only the SQL queries visible in the network traffic as the program communicates with an external database.

**Active Learning:** In contrast to most active learning approaches in machine learning [Settles 2009], our approach symbolically reasons about potential program structures and obtains unique correct answers. Closest related to SHEAR are active learning techniques that automatically generate inputs during program inference [Cambronero et al. 2019; Heule et al. 2015; Jha et al. 2010; Rinard et al. 2018; Shen et al. 2021; Shen and Rinard 2019; Vasilakis et al. 2021; Wu 2018]. SHEAR differs from all these techniques in that it manipulates the program execution in addition to observing it. Other active learning techniques related to programming languages include for pruning the search space during program synthesis [Ji et al. 2020a; Si et al. 2018], for learning data structure specifications [Gehr et al. 2015], for learning input grammars [Bastani et al. 2017], for learning points-to specifications [Bastani et al. 2018], for learning models of the design patterns [Jeon et al. 2016], and for learning event-transition classifiers [Bowring et al. 2004].

**Inductive Program Synthesis:** These techniques use end-to-end input-output behavior specifications provided by a user [Albarghouthi et al. 2013; Feser et al. 2015; Gulwani 2011; Ji et al. 2020a; Lee et al. 2016, 2018; Osera and Zdancewic 2015; Smith and Albarghouthi 2016; So and Oh 2017, 2018; Wang et al. 2017b,c,d; Ye et al. 2020]. In contrast, SHEAR automatically generates the
behavior specifications by executing an existing program. The observations are not only end-to-end input-output behavior but also contain the database traffic during execution.

**State-Machine Model Learning and Fuzzing:** These techniques infer program functionality as automata and state transitions [Aarts et al. 2013; Aarts and Vaandrager 2010; Angluin 1987; Cassel et al. 2016; Chow 1978; De Ruiter and Poll 2015; Fiterău-Broștean et al. 2016; Grinchtein et al. 2010; Isberner et al. 2014; Moore 1956; Radhakrishna et al. 2018; Raffelt et al. 2005; Vaandrager 2017; Volpato and Tretmans 2015; Wu et al. 2016]. In contrast, SHEAR infers and regenerates the full functionality of certain database programs.

9 CONCLUSION

The need to infer loop constructs from observations of program executions has repeatedly arisen in a range of fields. We present new active loop detection algorithms that automatically infer loop structures from execution traces. The algorithms strategically alter the program’s database traffic at precisely chosen execution points to elicit different behaviors, which differ depending on the loop structure in the underlying program. Results from our implementation highlight the effectiveness of active loop detection at eliminating many of the limitations present in other systems.
Table 2. SHEAR’s performance on inferring benchmark commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Launch</th>
<th>Runs</th>
<th>% Intact + Alt</th>
<th>Solves</th>
<th>Time</th>
<th>AlgTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_projects_id_messages</td>
<td>10.6s</td>
<td>425</td>
<td>68 + 357</td>
<td>358</td>
<td>7203.3s</td>
<td>2716.8s</td>
</tr>
<tr>
<td>get_projects_id_messages_id</td>
<td>10.4s</td>
<td>77</td>
<td>62 + 15</td>
<td>327</td>
<td>4326.4s</td>
<td>3526.7s</td>
</tr>
<tr>
<td>get_projects_id_messages_display_list</td>
<td>10.6s</td>
<td>415</td>
<td>71 + 344</td>
<td>391</td>
<td>7461.4s</td>
<td>3073.3s</td>
</tr>
<tr>
<td>get_projects_id_times (fixed 500 error)</td>
<td>10.6s</td>
<td>434</td>
<td>73 + 361</td>
<td>401</td>
<td>8739.7s</td>
<td>4125.1s</td>
</tr>
<tr>
<td>get_projects_id_times_id</td>
<td>10.6s</td>
<td>97</td>
<td>88 + 9</td>
<td>474</td>
<td>3846.7s</td>
<td>2821.9s</td>
</tr>
<tr>
<td>get_projects_id_milestones_id</td>
<td>10.4s</td>
<td>170</td>
<td>78 + 92</td>
<td>411</td>
<td>5544.6s</td>
<td>3774.5s</td>
</tr>
<tr>
<td>get_projects</td>
<td>10.1s</td>
<td>24</td>
<td>12 + 12</td>
<td>38</td>
<td>295.3s</td>
<td>53.1s</td>
</tr>
<tr>
<td>get_companies_id</td>
<td>10.1s</td>
<td>20</td>
<td>14 + 6</td>
<td>52</td>
<td>257.3s</td>
<td>55.6s</td>
</tr>
<tr>
<td>get_users_id (fixed 500 error)</td>
<td>10.2s</td>
<td>73</td>
<td>24 + 49</td>
<td>87</td>
<td>960.3s</td>
<td>218.3s</td>
</tr>
<tr>
<td>get_api_lists</td>
<td>7.4s</td>
<td>181</td>
<td>23 + 158</td>
<td>45</td>
<td>1557.2s</td>
<td>220.1s</td>
</tr>
<tr>
<td>get_api_lists_id</td>
<td>7.4s</td>
<td>91</td>
<td>19 + 72</td>
<td>44</td>
<td>765.5s</td>
<td>93.0s</td>
</tr>
<tr>
<td>get_api_cards</td>
<td>7.4s</td>
<td>89</td>
<td>17 + 72</td>
<td>35</td>
<td>780.3s</td>
<td>122.4s</td>
</tr>
<tr>
<td>get_api_cards_id</td>
<td>7.4s</td>
<td>43</td>
<td>19 + 24</td>
<td>50</td>
<td>398.4s</td>
<td>79.4s</td>
</tr>
<tr>
<td>get_api_boards_id</td>
<td>7.1s</td>
<td>167</td>
<td>25 + 142</td>
<td>48</td>
<td>1329.2s</td>
<td>149.8s</td>
</tr>
<tr>
<td>get_api_users_current</td>
<td>7.7s</td>
<td>5</td>
<td>5 + 0</td>
<td>6</td>
<td>41.3s</td>
<td>3.0s</td>
</tr>
<tr>
<td>get_api_users_id</td>
<td>7.2s</td>
<td>5</td>
<td>5 + 0</td>
<td>6</td>
<td>39.0s</td>
<td>3.1s</td>
</tr>
<tr>
<td>get_home</td>
<td>8.8s</td>
<td>57</td>
<td>12 + 45</td>
<td>26</td>
<td>529.3s</td>
<td>25.8s</td>
</tr>
<tr>
<td>get_lists_id_tasks</td>
<td>9.1s</td>
<td>59</td>
<td>14 + 45</td>
<td>35</td>
<td>565.9s</td>
<td>31.8s</td>
</tr>
<tr>
<td>get_lists_id_tasks (fixed 404 error)</td>
<td>8.8s</td>
<td>80</td>
<td>17 + 63</td>
<td>44</td>
<td>742.2s</td>
<td>41.8s</td>
</tr>
<tr>
<td>get_home</td>
<td>6.5s</td>
<td>19</td>
<td>7 + 12</td>
<td>42</td>
<td>165.2s</td>
<td>41.9s</td>
</tr>
<tr>
<td>get_projects</td>
<td>6.5s</td>
<td>19</td>
<td>7 + 12</td>
<td>42</td>
<td>165.4s</td>
<td>42.4s</td>
</tr>
<tr>
<td>get_projects_id</td>
<td>6.5s</td>
<td>47</td>
<td>14 + 33</td>
<td>83</td>
<td>584.1s</td>
<td>276.3s</td>
</tr>
<tr>
<td>get_projects_id_stories</td>
<td>6.4s</td>
<td>41</td>
<td>17 + 24</td>
<td>64</td>
<td>366.3s</td>
<td>103.4s</td>
</tr>
<tr>
<td>get_projects_id_stories_id</td>
<td>6.5s</td>
<td>41</td>
<td>17 + 24</td>
<td>67</td>
<td>363.7s</td>
<td>98.5s</td>
</tr>
<tr>
<td>get_projects_id_stories_id_notes</td>
<td>6.5s</td>
<td>41</td>
<td>17 + 24</td>
<td>65</td>
<td>364.7s</td>
<td>98.0s</td>
</tr>
<tr>
<td>get_projects_id_stories_id_notes_id</td>
<td>6.5s</td>
<td>43</td>
<td>19 + 24</td>
<td>81</td>
<td>391.6s</td>
<td>114.0s</td>
</tr>
<tr>
<td>get_projects_id_users</td>
<td>6.5s</td>
<td>39</td>
<td>12 + 27</td>
<td>77</td>
<td>525.0s</td>
<td>272.7s</td>
</tr>
<tr>
<td>get_channels</td>
<td>15.0s</td>
<td>233</td>
<td>32 + 201</td>
<td>100</td>
<td>3762.2s</td>
<td>263.7s</td>
</tr>
<tr>
<td>get_channels_id_activities</td>
<td>15.0s</td>
<td>80</td>
<td>38 + 42</td>
<td>157</td>
<td>1388.2s</td>
<td>188.5s</td>
</tr>
<tr>
<td>get_channels_id_activities_id</td>
<td>14.9s</td>
<td>52</td>
<td>28 + 24</td>
<td>60</td>
<td>844.3s</td>
<td>71.7s</td>
</tr>
<tr>
<td>get_me</td>
<td>15.1s</td>
<td>38</td>
<td>20 + 18</td>
<td>68</td>
<td>637.7s</td>
<td>62.0s</td>
</tr>
<tr>
<td>get_users</td>
<td>14.8s</td>
<td>103</td>
<td>28 + 75</td>
<td>307</td>
<td>1872.4s</td>
<td>345.0s</td>
</tr>
<tr>
<td>get_users_id</td>
<td>14.9s</td>
<td>40</td>
<td>22 + 18</td>
<td>78</td>
<td>678.7s</td>
<td>82.5s</td>
</tr>
<tr>
<td>get_home</td>
<td>9.4s</td>
<td>75</td>
<td>17 + 58</td>
<td>55</td>
<td>757.2s</td>
<td>52.7s</td>
</tr>
<tr>
<td>get_archives</td>
<td>9.5s</td>
<td>65</td>
<td>15 + 50</td>
<td>30</td>
<td>659.4s</td>
<td>44.3s</td>
</tr>
<tr>
<td>get_admin_comments_id</td>
<td>9.8s</td>
<td>6</td>
<td>6 + 0</td>
<td>13</td>
<td>62.4s</td>
<td>3.7s</td>
</tr>
<tr>
<td>get_admin_pages</td>
<td>9.2s</td>
<td>23</td>
<td>8 + 15</td>
<td>21</td>
<td>220.7s</td>
<td>9.2s</td>
</tr>
<tr>
<td>get_admin_pages_id</td>
<td>9.8s</td>
<td>5</td>
<td>5 + 0</td>
<td>11</td>
<td>51.8s</td>
<td>3.0s</td>
</tr>
<tr>
<td>get_admin_posts</td>
<td>9.7s</td>
<td>39</td>
<td>11 + 28</td>
<td>13</td>
<td>397.5s</td>
<td>20.9s</td>
</tr>
<tr>
<td>get_admin (trimmed)</td>
<td>9.6s</td>
<td>75</td>
<td>15 + 60</td>
<td>24</td>
<td>773.9s</td>
<td>55.8s</td>
</tr>
<tr>
<td>get_article_id</td>
<td>4.7s</td>
<td>17</td>
<td>11 + 6</td>
<td>25</td>
<td>92.1s</td>
<td>11.7s</td>
</tr>
<tr>
<td>get_articles</td>
<td>4.4s</td>
<td>13</td>
<td>7 + 6</td>
<td>9</td>
<td>60.8s</td>
<td>3.3s</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>3.0s</td>
<td>47</td>
<td>18 + 29</td>
<td>56</td>
<td>171.1s</td>
<td>31.6s</td>
</tr>
<tr>
<td>repeat_2</td>
<td>2.8s</td>
<td>25</td>
<td>7 + 18</td>
<td>6</td>
<td>82.0s</td>
<td>11.2s</td>
</tr>
<tr>
<td>repeat_3</td>
<td>2.8s</td>
<td>31</td>
<td>7 + 24</td>
<td>6</td>
<td>102.7s</td>
<td>14.8s</td>
</tr>
<tr>
<td>repeat_4</td>
<td>2.8s</td>
<td>37</td>
<td>7 + 30</td>
<td>6</td>
<td>122.2s</td>
<td>17.5s</td>
</tr>
<tr>
<td>repeat_5</td>
<td>2.8s</td>
<td>43</td>
<td>7 + 36</td>
<td>6</td>
<td>143.0s</td>
<td>20.9s</td>
</tr>
<tr>
<td>nest</td>
<td>2.9s</td>
<td>115</td>
<td>13 + 102</td>
<td>12</td>
<td>393.8s</td>
<td>56.3s</td>
</tr>
<tr>
<td>after_2</td>
<td>2.9s</td>
<td>63</td>
<td>15 + 48</td>
<td>14</td>
<td>220.8s</td>
<td>35.6s</td>
</tr>
<tr>
<td>after_3</td>
<td>2.9s</td>
<td>93</td>
<td>21 + 72</td>
<td>20</td>
<td>328.7s</td>
<td>56.2s</td>
</tr>
<tr>
<td>after_4</td>
<td>2.9s</td>
<td>123</td>
<td>27 + 96</td>
<td>26</td>
<td>441.1s</td>
<td>79.5s</td>
</tr>
<tr>
<td>after_5</td>
<td>2.9s</td>
<td>153</td>
<td>33 + 120</td>
<td>32</td>
<td>549.4s</td>
<td>102.2s</td>
</tr>
<tr>
<td>example (Section 2)</td>
<td>2.8s</td>
<td>89</td>
<td>23 + 66</td>
<td>52</td>
<td>293.3s</td>
<td>44.8s</td>
</tr>
<tr>
<td>Command</td>
<td>App</td>
<td>In</td>
<td>Q</td>
<td>If</td>
<td>For</td>
<td>Out</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>----</td>
<td>---</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>get_projects_id_messages</td>
<td>RailsCollab</td>
<td>2</td>
<td>34</td>
<td>9</td>
<td>3</td>
<td>39</td>
</tr>
<tr>
<td>get_projects_id_messages_id</td>
<td>RailsCollab</td>
<td>3</td>
<td>21</td>
<td>5</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>get_projects_id_messages_display_list</td>
<td>RailsCollab</td>
<td>2</td>
<td>35</td>
<td>9</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>get_projects_id_times</td>
<td>RailsCollab</td>
<td>2</td>
<td>38</td>
<td>11</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>get_projects_id_times_id</td>
<td>RailsCollab</td>
<td>3</td>
<td>55</td>
<td>15</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>get_projects_id_milestones_id</td>
<td>RailsCollab</td>
<td>3</td>
<td>25</td>
<td>6</td>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td>get_projects</td>
<td>RailsCollab</td>
<td>1</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>get_companies_id</td>
<td>RailsCollab</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>get_users_id</td>
<td>RailsCollab</td>
<td>2</td>
<td>12</td>
<td>5</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>get_api_lists</td>
<td>Kanban</td>
<td>1</td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>27</td>
</tr>
<tr>
<td>get_api_lists_id</td>
<td>Kanban</td>
<td>2</td>
<td>9</td>
<td>2</td>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td>get_api_cards</td>
<td>Kanban</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>get_api_cards_id</td>
<td>Kanban</td>
<td>2</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>get_api_boards_id</td>
<td>Kanban</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>3</td>
<td>33</td>
</tr>
<tr>
<td>get_api_users_current</td>
<td>Kanban</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>get_api_users_id</td>
<td>Kanban</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>get_home</td>
<td>Todo</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>get_lists_id_tasks</td>
<td>Todo</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>get_lists_id_tasks (fixed 404 error)</td>
<td>Todo</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>get_lists_id_tasks (fixed 500 error)</td>
<td>RailsCollab</td>
<td>2</td>
<td>12</td>
<td>5</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>get_channels</td>
<td>Fulcrum</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>get_channels_id</td>
<td>Fulcrum</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>get_channels_id</td>
<td>Fulcrum</td>
<td>2</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>get_channels_id</td>
<td>Fulcrum</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>get_channels_id</td>
<td>Fulcrum</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>get_channels_id</td>
<td>Fulcrum</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>get_channels_id</td>
<td>Fulcrum</td>
<td>4</td>
<td>10</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>get_channels_id</td>
<td>Fulcrum</td>
<td>2</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>get_channels_id</td>
<td>Fulcrum</td>
<td>1</td>
<td>16</td>
<td>4</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>get_channels_id</td>
<td>Fulcrum</td>
<td>2</td>
<td>16</td>
<td>6</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>get_channels_id</td>
<td>Fulcrum</td>
<td>3</td>
<td>11</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>get_channels_id_activities</td>
<td>Kandan</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>get_channels_id_activities_id</td>
<td>Kandan</td>
<td>1</td>
<td>11</td>
<td>3</td>
<td>0</td>
<td>45</td>
</tr>
<tr>
<td>get_channels_id_activities</td>
<td>Kandan</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>get_channels_id_activities_id</td>
<td>Kandan</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>get_channels_id_activities_id</td>
<td>Kandan</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>get_channels_id_activities_id</td>
<td>Kandan</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>get_channels_id_activities_id</td>
<td>Enki</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>get_channels_id_activities_id</td>
<td>Enki</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>get_channels_id_activities_id</td>
<td>Enki</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>get_channels_id_activities (trimmed)</td>
<td>Enki</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>get_channels_id_activities (trimmed)</td>
<td>Blog</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>get_channels_id_activities (trimmed)</td>
<td>Blog</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>Student</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>Synthetic</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>Synthetic</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>Synthetic</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>Synthetic</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>Synthetic</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>Synthetic</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>Synthetic</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>Synthetic</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>Synthetic</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>liststudentcourses</td>
<td>Synthetic</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>
REFERENCES


A.1 RailsCollab Project Manager Command get_projects_id_messages

```python
def get_projects_id_messages(conn, inputs):
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'."uid" = :x0 LIMIT 1", ('x0': inputs[0]))
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'."id" = :x0 LIMIT 1", ('x0': util.get_one_data(s0, 'users', 'id')))
        s2 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects' INNER JOIN 'people' ON 'projects'."id" = 'people'."project_id" WHERE 'people'."user_id" = :x0 AND 'projects'.completed_on IS NULL ORDER BY 'projects'.priority ASC, 'projects'.name ASC", ('x0': util.get_one_data(s0, 'users', 'id')))
        s3 = util.do_sql(conn, SELECT 'time_records'.* FROM 'time_records' WHERE 'time_records'."assigned_to_user_id" = :x0 AND (start_date IS NOT NULL AND done_date IS NULL), ('x0': util.get_one_data(s0, 'users', 'id')))
        s4 = util.do_sql(conn, SELECT 'projects'.* FROM 'projects' WHERE 'projects'."id" = :x0 LIMIT 1", ('x0': inputs[1]))
        outputs.extend(util.get_data(s4, 'projects', 'id'))
        outputs.extend(util.get_data(s4, 'projects', 'name'))
        if util.has_rows(s4):
            s5 = util.do_sql(conn, SELECT 'companies'.* FROM 'companies' WHERE 'companies'."id" = :x0 LIMIT 1", ('x0': util.get_one_data(s0, 'users', 'company_id')))
            outputs.extend(util.get_data(s5, 'companies', 'id'))
            outputs.extend(util.get_data(s5, 'companies', 'name'))
            outputs.extend(util.get_data(s5, 'companies', 'homepage'))
            if util.has_rows(s5):
                s6 = util.do_sql(conn, SELECT 'companies'.* FROM 'companies' WHERE 'companies'."id" IS NULL LIMIT 1", ())
                s7 = util.do_sql(conn, SELECT COUNT(*) FROM 'messages' WHERE 'messages'."project_id" = :x0", ('x0': inputs[1]))
                s8 = util.do_sql(conn, SELECT COUNT(*) FROM 'messages' WHERE 'messages'."project_id" = :x0", ('x0': inputs[1]))
                if util.has_rows(s8):
                    s18 = util.do_sql(conn, SELECT 'messages'.* FROM 'messages' WHERE 'messages'."project_id" = :x0 ORDER BY created_on DESC LIMIT 10 OFFSET 0", ('x0': inputs[1]))
                    outputs.extend(util.get_data(s18, 'messages', 'id'))
                    outputs.extend(util.get_data(s18, 'messages', 'project_id'))
    s19 = util.do_sql(conn, SELECT 'users'.* FROM 'users' WHERE 'users'."id" = :x0 LIMIT 1", ('x0': util.get_one_data(s18, 'users', 'id')))
    outputs.extend(util.get_data(s19, 'users', 'id'))
    s20 = util.do_sql(conn, SELECT 'projects'.* FROM 'projects' WHERE 'projects'."id" = :x0 LIMIT 1", ('x0': util.get_one_data(s18, 'messages', 'project_id')))
    outputs.extend(util.get_data(s20, 'projects', 'id'))
    s21 = util.do_sql(conn, SELECT 'people'.* FROM 'people' WHERE 'people'."user_id" = :x0 AND 'people'."project_id" = :x1 LIMIT 1", ('x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s20, 'messages', 'project_id')
    outputs.extend(util.get_data(s21, 'people', 'project_id'))
    s22 = util.do_sql(conn, SELECT 'people'.* FROM 'people' WHERE 'people'."user_id" = :x0 AND 'people'."project_id" = :x1", ('x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s18, 'messages', 'project_id'))
    outputs.extend(util.get_data(s22, 'people', 'project_id'))
    outputs.extend(util.get_data(s22, 'people', 'user_id'))
    if util.has_rows(s22):
        pass
```
else:
	s23 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0,'users','id')})

outputs.extend(util.get_data(s23, 'people', 'project_id'))
	s24 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1)", {'x0': util.get_one_data(s18, 'messages', 'project_id')})

s18 = s18_all
	s25 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1)", {'x0': inputs[1]})

outputs.extend(util.get_data(s25, 'people', 'project_id'))
	s26 = util.do_sql(conn, "SELECT COUNT(*) FROM 'categories' WHERE 'messages'.project_id = :x0 AND 'messages'.is_important = 1", {'x0': inputs[1]})

s27 = util.do_sql(conn, "SELECT COUNT(*) FROM 'categories' WHERE 'categories'.project_id = :x0", {'x0': inputs[1]})

if util.has_rows(s27):
	s28 = util.do_sql(conn, "SELECT 'categories'.* FROM 'categories' WHERE 'categories'.project_id = :x0", {'x0': inputs[1]})

outputs.extend(util.get_data(s28, 'categories', 'id'))

outputs.extend(util.get_data(s28, 'categories', 'project_id'))

outputs.extend(util.get_data(s28, 'categories', 'name'))

s28_all = s28

for s28 in s28_all:
	s29 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects' WHERE 'projects'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s28, 'categories', 'project_id')})

outputs.extend(util.get_data(s29, 'projects', 'id'))

outputs.extend(util.get_data(s29, 'projects', 'name'))
	s30 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})

x1: util.get_one_data(s18, 'messages', 'project_id'))

outputs.extend(util.get_data(s30, 'people', 'project_id'))

outputs.extend(util.get_data(s30, 'people', 'project_id'))

outputs.extend(util.get_data(s30, 'people', 'user_id'))

s31 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})

x1: util.get_one_data(s28, 'categories', 'project_id'))

outputs.extend(util.get_data(s31, 'people', 'project_id'))

outputs.extend(util.get_data(s31, 'people', 'user_id'))

if util.has_rows(s31):

    pass

else:
	s32 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s28, 'categories', 'project_id')})

s33 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s28, 'categories', 'project_id')})

s28 = s28_all

pass

else:

    pass

else:
	s9 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})

x1: inputs[1]})

outputs.extend(util.get_data(s9, 'people', 'project_id'))

outputs.extend(util.get_data(s9, 'people', 'user_id'))

s10 = util.do_sql(conn, "SELECT COUNT(*) FROM 'messages' WHERE 'messages'.project_id = :x0 AND 'messages'.is_important = 1", {'x0': inputs[1]})
	s11 = util.do_sql(conn, "SELECT COUNT(*) FROM 'categories' WHERE 'categories'.project_id = :x0", {'x0': inputs[1]})
if util.has_rows(s11):
    s12 = util.do_sql(conn, "SELECT 'categories'.* FROM 'categories'
    WHERE 'categories'.project_id = :x0 LIMIT 1", {'x0': inputs[11]})
outputs.extend([util.get_data(s12, 'categories', 'id')])
outputs.extend([util.get_data(s12, 'categories', 'project_id')])
s12_all = s12
    for s12 in s12_all:
        s13 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects'
        WHERE 'projects'.id = :x0 LIMIT 1", {'x0': util
        .get_one_data(s12, 'categories', 'id')})
outputs.extend([util.get_data(s13, 'projects', 'id')])
outputs.extend([util.get_data(s13, 'projects', 'name')])
s14 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
        WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util
        .get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(
        s12, 'categories', 'project_id')})
outputs.extend([util.get_data(s14, 'people', 'project_id')])
outputs.extend([util.get_data(s14, 'people', 'user_id')])
    if util.has_rows(s15):
        pass
else:
    s16 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
        WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util
        .get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(
        s12, 'categories', 'project_id')})
outputs.extend([util.get_data(s16, 'people', 'project_id')])
    s17 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
        WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util
        .get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(
        s12, 'categories', 'project_id')})
outputs.extend([util.get_data(s17, 'people', 'project_id')])
    pass
else:
    pass
else:
    pass
else:
    pass
else:
    pass
return util.add_warnings(outputs)

A.2 RailsCollab Project Manager Command get_projects_id_messages_id

def get_projects_id_messages_id(conn, inputs):
    util.clear_warnings()
    outputs = []
s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.username = :x0
        LIMIT 1", {'x0': inputs[0]})
if util.has_rows(s0):
    s1 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id = :x0
        LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
s2 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects' INNER JOIN 'people' ON 'projects'.id = 'people'.project_id WHERE 'people'.user_id = :x0 AND (projects.completed_on IS NULL) ORDER BY projects.priority ASC, projects.name ASC", {'x0': util.get_one_data(s0, 'users', 'id')})
s3 = util.do_sql(conn, "SELECT 'time_records'.* FROM 'time_records'
        WHERE 'time_records'.assigned_to_user_id = :x0 AND (start_date IS NOT NULL AND
done_date IS NULL)", {'x0': util.get_one_data(s0, 'users', 'id')})
s4 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects' WHERE 'projects'.id
        = :x0 LIMIT 1", {'x0': inputs[11]})
outputs.extend([util.get_data(s4, 'projects', 'id')])
if util.has_rows(s4):
s5 = util.do_sql(conn, "SELECT 'companies' FROM 'companies' WHERE 'companies'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'company_id')})

if util.has_rows(s5):
    s6 = util.do_sql(conn, "SELECT 'companies' FROM 'companies' WHERE 'companies'.'id' IS NULL LIMIT 1", {''})
    s7 = util.do_sql(conn, "SELECT 'messages' FROM 'messages' WHERE 'messages'.'project_id' = :x0 AND 'messages'.'id' = :x1 ORDER BY created_on DESC LIMIT 1", {'x0': inputs[1], 'x1': inputs[2]})
outputs.extend(util.get_data(s7, 'messages', 'project_id'))
outputs.extend(util.get_data(s7, 'messages', 'title'))
outputs.extend(util.get_data(s7, 'messages', 'text'))
if util.has_rows(s7):
    s8 = util.do_sql(conn, "SELECT 'projects' FROM 'projects' WHERE 'projects'.'id' = :x0 LIMIT 1", {'x0': inputs[1]})
outputs.extend(util.get_data(s8, 'projects', 'id'))
outputs.extend(util.get_data(s8, 'projects', 'name'))
s9 = util.do_sql(conn, "SELECT 'people' FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s9, 'people', 'project_id'))
outputs.extend(util.get_data(s9, 'people', 'user_id'))
if util.has_rows(s9):
    s10 = util.do_sql(conn, "SELECT 'projects' FROM 'projects' WHERE 'projects'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'companies', 'created_by_id')})
outputs.extend(util.get_data(s10, 'users', 'id'))
outputs.extend(util.get_data(s10, 'users', 'display_name'))
s12 = util.do_sql(conn, "SELECT 'milestones' FROM 'milestones' WHERE 'milestones'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'messages', 'milestone_id')})
outputs.extend(util.get_data(s12, 'milestones', 'id'))
outputs.extend(util.get_data(s12, 'milestones', 'project_id'))
outputs.extend(util.get_data(s12, 'milestones', 'name'))
s13 = util.do_sql(conn, "SELECT 'people' FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s13, 'people', 'project_id'))
outputs.extend(util.get_data(s13, 'people', 'user_id'))
s14 = util.do_sql(conn, "SELECT 'people' FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s14, 'people', 'project_id'))
outputs.extend(util.get_data(s14, 'people', 'user_id'))
s15 = util.do_sql(conn, "SELECT 'people' FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s15, 'people', 'project_id'))
outputs.extend(util.get_data(s15, 'people', 'user_id'))
s16 = util.do_sql(conn, "SELECT 'people' FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s16, 'people', 'project_id'))
outputs.extend(util.get_data(s16, 'people', 'user_id'))
s17 = util.do_sql(conn, "SELECT 'categories' FROM 'categories' WHERE 'categories'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'messages', 'category_id')})
outputs.extend(util.get_data(s17, 'categories', 'id'))
outputs.extend(util.get_data(s17, 'categories', 'name'))
s18 = util.do_sql(conn, "SELECT 'users' FROM 'users' INNER JOIN 'message_subscriptions' ON 'users'.'id' = 'message_subscriptions'.'user_id' WHERE 'message_subscriptions'.'message_id' = :x0", {'x0': inputs[2]})
outputs.extend(util.get_data(s18, 'users', 'id'))
outputs.extend(util.get_data(s18, 'users', 'display_name'))
s19 = util.do_sql(conn, "SELECT 'people' FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s19, 'people', 'project_id'))
outputs.extend(util.get_data(s19, 'people', 'user_id'))
Active Loop Detection for Applications that Access Databases

47

A.3 RailsCollab Project Manager Command get_projects_id_messages_display_list

def get_projects_id_messages_display_list (conn, inputs):
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'username' = :x0 LIMIT 1", ('x0': inputs[0]))
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'id' = :x0 LIMIT 1", ('x0': util.get_one_data(s0, 'users', 'id')))
        s2 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects' INNER JOIN 'people' ON 'projects'.'id' = 'people'.'project_id' WHERE 'people'.'user_id' = :x0 AND (projects.completed_on IS NULL ORDER BY projects.priority ASC, projects.name ASC", ('x0': util.get_one_data(s0, 'users', 'id')))
        s3 = util.do_sql(conn, "SELECT 'time_records'.* FROM 'time_records' WHERE 'time_records'.'assigned_to_user_id' = :x0 AND (start_date IS NOT NULL AND done_date IS NULL)", ('x0': util.get_one_data(s0, 'users', 'id')))
        s4 = util.do_sql(conn, "SELECT 'projects'.'id' FROM 'projects' WHERE 'projects'.'id' = :x0 LIMIT 1", ('x0': inputs[1]))
        outputs.extend(util.get_data(s4, 'projects', 'id'))
        if util.has_rows(s4):
            s5 = util.do_sql(conn, "SELECT 'companies'.'id' FROM 'companies' WHERE 'companies'.'id' = :x0 LIMIT 1", ('x0': util.get_one_data(s0, 'users', 'id')))
            outputs.extend(util.get_data(s5, 'companies', 'id'))
            if util.has_rows(s5):
                s6 = util.do_sql(conn, "SELECT 'companies'.'id' FROM 'companies' WHERE 'companies'.'id' IS NULL LIMIT 1", ({}))
                s7 = util.do_sql(conn, "SELECT COUNT(*) FROM 'messages' WHERE 'messages'.'project_id' = :x0", ('x0': inputs[1]))
                s8 = util.do_sql(conn, "SELECT COUNT(*) FROM 'messages' WHERE 'messages'.'project_id' = :x0", ('x0': inputs[1]))
                if util.has_rows(s8):
                    s18 = util.do_sql(conn, "SELECT 'messages'.'id' FROM 'messages' WHERE 'messages'.'project_id' = :x0 ORDER BY created_on DESC LIMIT 10 OFFSET 0", ('x0': inputs[1]))
                    outputs.extend(util.get_data(s18, 'messages', 'id'))
                    outputs.extend(util.get_data(s18, 'messages', 'project_id'))
                    outputs.extend(util.get_data(s18, 'messages', 'title'))
                    s18_all = s18
                    for s18 in s18_all:
                        s19 = util.do_sql(conn, "SELECT 'projects'.'id' FROM 'projects' WHERE 'projects'.'id' = :x0 LIMIT 1", ('x0': util.get_one_data(s18, 'messages', 'project_id')))
                        outputs.extend(util.get_data(s19, 'projects', 'id'))
                        outputs.extend(util.get_data(s19, 'projects', 'name'))
if outputs.extend(util.get_data(s32, 'messages', 'project_id'))

s21 = util.do_sql(conn, 'SELECT `people`.* FROM `people` WHERE `project_id` = :x1 LIMIT 1', {'x1': util.get_one_data(s21, 'project_id')})

outputs.extend(util.get_data(s21, 'project_id'))

if util.has_rows(s21):
    s25 = util.do_sql(conn, "SELECT ` projects`.id FROM `projects` WHERE `project_id` = :x2 LIMIT 1", {'x2': util.get_one_data(s25, 'project_id')})

outputs.extend(util.get_data(s25, 'project_id'))

s22 = util.do_sql(conn, "SELECT `people`.* FROM `people` WHERE `project_id` = :x2 AND `messages`.id = :x0 LIMIT 1", {'x0': util.get_one_data(s22, 'messages'), 'id': inputs[1]})

outputs.extend(util.get_data(s22, 'messages', 'project_id'))

s23 = util.do_sql(conn, "SELECT `people`.* FROM `people` WHERE `project_id` = :x2 AND `messages`.id = :x0 LIMIT 1", {'x0': util.get_one_data(s23, 'messages'), 'id': inputs[1]})

outputs.extend(util.get_data(s23, 'messages', 'project_id'))

s24 = util.do_sql(conn, "SELECT `users`.* FROM `users` WHERE `project_id` = :x2 AND `messages`.id = :x0 LIMIT 1", {'x0': util.get_one_data(s24, 'messages'), 'id': inputs[1]})

outputs.extend(util.get_data(s24, 'messages', 'project_id'))

s26 = util.do_sql(conn, "SELECT `messages`.id FROM `messages` WHERE `project_id` = :x2 AND `messages`.is_important = 1", {'x2': inputs[1]})

outputs.extend(util.get_data(s26, 'messages', 'project_id'))

s27 = util.do_sql(conn, 'SELECT COUNT(*) FROM `messages` WHERE `project_id` = :x2 AND `messages`.id = :x0', {'x0': inputs[1]})

if util.has_rows(s27):
    s29 = util.do_sql(conn, "SELECT ` categories`.id FROM `categories` WHERE `project_id` = :x2", {'x2': inputs[1]})

outputs.extend(util.get_data(s29, 'categories', 'id'))

s29_all = s29

for s29 in s29_all:
    s30 = util.do_sql(conn, "SELECT ` projects`.id FROM `projects` WHERE `project_id` = :x3", {'x3': inputs[2]})

outputs.extend(util.get_data(s30, 'projects', 'id'))

s31 = util.do_sql(conn, 'SELECT ` messages`.id FROM `messages` WHERE `project_id` = :x3', {'x3': inputs[2]})

outputs.extend(util.get_data(s31, 'messages', 'id'))

s32 = util.do_sql(conn, 'SELECT ` people`.id FROM ` people` WHERE `project_id` = :x3', {'x3': inputs[2]})

outputs.extend(util.get_data(s32, 'people', 'id'))

if util.has_rows(s32):
    pass
else:

s33 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
  WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'
x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.
  get_one_data(s1, 'categories', 'project_id')})
s34 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
  WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'
x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.
  get_one_data(s2, 'categories', 'project_id')})
s29 = s29_all
else:
  pass
else:
  s9 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
  WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'
x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s9, 'people', 'project_id'))
outputs.extend(util.get_data(s9, 'people', 'user_id'))
s10 = util.do_sql(conn, "SELECT COUNT(*) FROM 'messages' WHERE 'messages'.project_id = :x0 AND 'messages'.is_important = 1", {'
x0': inputs[1]})
s11 = util.do_sql(conn, "SELECT COUNT(*) FROM 'categories' WHERE 'categories'.project_id = :x0", {'
x0': inputs[1]})
if util.has_rows(s11):
s12 = util.do_sql(conn, "SELECT 'categories'.* FROM 'categories'
  WHERE 'categories'.project_id = :x0", {'x0': inputs[1]})
outputs.extend(util.get_data(s12, 'categories', 'id'))
outputs.extend(util.get_data(s12, 'categories', 'project_id'))
s12_all = s12
for s12 in s12_all:
s13 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects'
  WHERE 'projects'.id = :x0 LIMIT 1", {'x0': util.
    get_one_data(s12, 'categories', 'project_id')})
outputs.extend(util.get_data(s13, 'projects', 'id'))
outputs.extend(util.get_data(s13, 'projects', 'name'))
s14 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
  WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'
x0': util.
  get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(
  s12, 'categories', 'project_id')})
outputs.extend(util.get_data(s14, 'people', 'project_id'))
outputs.extend(util.get_data(s14, 'people', 'user_id'))
s15 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
  WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'
x0': util.
  get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(
  s12, 'categories', 'project_id')})
outputs.extend(util.get_data(s15, 'people', 'project_id'))
outputs.extend(util.get_data(s15, 'people', 'user_id'))
if util.has_rows(s15):
  pass
else:
  s16 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
  WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'
x0': util.
  get_one_data(s0, 'users', 'id'), 'x1': util.
  get_one_data(s12, 'categories', 'project_id')})
s17 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
  WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'
x0': util.
  get_one_data(s0, 'users', 'id'), 'x1': util.
  get_one_data(s12, 'categories', 'project_id')})
s12 = s12_all
else:
  pass
else:
  pass
else:
  pass
else:
  pass
return util.add_warnings(outputs)
A.4 RailsCollab Project Manager Command get_projects_id_times

For this command we use a modified version of RailsCollab, where we fixed a 500 error when a task's task list ID does not match any records in the database. Specifically, we changed the statement “url_for hash_for_task_path(:id => self.id, :active_project => self.project_id, :only_path => host.nil?, :host => host)” into “url_for hash_for_task_path(:id => self.id, :active_project => self.project_id, :only_path => host.nil?, :host => host) rescue nil” in the file app/models/task.rb.
if s13 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id'))} else:
outputs.extend(util.get_data(s13, 'people', 'project_id'))

s34 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE 'people'.user_id = :x0 AND 'people'.project_id = :x1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id'))} outputs.extend(util.get_data(s34, 'people', 'project_id'))

if util.has_rows(s34):
pass
else:

s35 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", ('x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id')))

s36 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE 'people'.user_id = :x0 AND 'people'.project_id = :x1", ('x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id')))

else:

s26 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects' WHERE 'projects'.id = :x0 LIMIT 1", ('x0': util.get_one_data(s10, 'time_records', 'project_id'))}

outputs.extend(util.get_data(s26, 'projects', 'id'))

s27 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id'))}

outputs.extend(util.get_data(s27, 'people', 'project_id'))

outputs.extend(util.get_data(s27, 'people', 'user_id'))

s28 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE 'people'.user_id = :x0 AND 'people'.project_id = :x1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id'))}

outputs.extend(util.get_data(s28, 'people', 'project_id'))

outputs.extend(util.get_data(s28, 'people', 'user_id'))

if util.has_rows(s28):
pass
else:

s29 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id'))}

s30 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE 'people'.user_id = :x0 AND 'people'.project_id = :x1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id'))}

else:

s12 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s10, 'time_records', 'assigned_to_user_id'))})

outputs.extend(util.get_data(s12, 'users', 'display_name'))

s13 = util.do_sql(conn, "SELECT 'tasks'.* FROM 'tasks' WHERE 'tasks'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s10, 'time_records', 'task_id'))})

if util.has_rows(s13):
s19 = util.do_sql(conn, "SELECT 'task_lists'.* FROM 'task_lists' WHERE 'task_lists'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s13, 'tasks', 'task_list_id'))})
outputs.extend(util.get_data(s19, 'task_lists', 'project_id'))
s20 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects'
WHERE 'projects'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s10, 'time_records', 'project_id'))}
outputs.extend(util.get_data(s20, 'projects', 'id'))
outputs.extend(util.get_data(s20, 'projects', 'name'))
s21 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s10, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id'))}
outputs.extend(util.get_data(s21, 'people', 'project_id'))
outputs.extend(util.get_data(s21, 'people', 'user_id'))
s22 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s10, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id'))}
outputs.extend(util.get_data(s22, 'people', 'project_id'))
outputs.extend(util.get_data(s22, 'people', 'user_id'))
if util.has_rows(s22):
    pass
else:
    s23 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id'))}
outputs.extend(util.get_data(s23, 'people', 'project_id'))
outputs.extend(util.get_data(s23, 'people', 'user_id'))
s14 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects'
WHERE 'projects'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s10, 'time_records', 'project_id'))}
outputs.extend(util.get_data(s14, 'projects', 'id'))
outputs.extend(util.get_data(s14, 'projects', 'name'))
s15 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id'))}
outputs.extend(util.get_data(s15, 'people', 'project_id'))
outputs.extend(util.get_data(s15, 'people', 'user_id'))
s16 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id'))}
outputs.extend(util.get_data(s16, 'people', 'project_id'))
outputs.extend(util.get_data(s16, 'people', 'user_id'))
if util.has_rows(s16):
    pass
else:
    s17 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': util.get_one_data(s10, 'time_records', 'project_id'))}
outputs.extend(util.get_data(s17, 'people', 'project_id'))
outputs.extend(util.get_data(s17, 'people', 'user_id'))
s10 = s10_all
s27 = util.do_sql(conn, "SELECT 'people'.* FROM 'people'
WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s8, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s27, 'people', 'project_id'))
outputs.extend(util.get_data(s27, 'people', 'user_id'))
else:
    s9 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x8 AND project_id = :x1) LIMIT 1", {'x8': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
    outputs.extend(util.get_data(s9, 'people', 'project_id'))
    outputs.extend(util.get_data(s9, 'people', 'user_id'))
else:
    pass
else:
    pass
else:
    pass
return util.add_warnings(outputs)

A.5 RailsCollab Project Manager Command get_projects_id_times_id

def get_projects_id_times_id (conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'."username" = :x0 LIMIT 1", {'x0': inputs[0]})
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'."id" = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
        s2 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects' INNER JOIN 'people' ON 'projects'."id" = 'people'."project_id" WHERE 'people'."user_id" = :x0 AND (projects.completed_on IS NULL) ORDER BY projects.priority ASC, projects.name ASC", {'x0': util.get_one_data(s0, 'users', 'id')})
        s3 = util.do_sql(conn, "SELECT 'time_records'.* FROM 'time_records' WHERE 'time_records'."assigned_to_user_id" = :x0 AND (start_date IS NOT NULL AND done_date IS NULL)", {'x0': util.get_one_data(s3, 'projects', 'id')})
        s4 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects' WHERE 'projects'."id" = :x0 LIMIT 1", {'x0': inputs[1]})
        outputs.extend(util.get_data(s4, 'projects', 'id'))
    if util.has_rows(s4):
        s5 = util.do_sql(conn, "SELECT 'companies'.* FROM 'companies' WHERE 'companies'."id" = :x0 LIMIT 1", {'x0': util.get_one_data(s5, 'users', 'company_id')})
    if util.has_rows(s5):
        s6 = util.do_sql(conn, "SELECT 'companies'.* FROM 'companies' WHERE 'companies'."id" IS NULL LIMIT 1", {})
        s7 = util.do_sql(conn, "SELECT 'time_records'.* FROM 'time_records' WHERE 'time_records'."project_id" = :x0 AND 'time_records'."id" = :x1 LIMIT 1", {'x0': inputs[1], 'x1': inputs[2]})
        outputs.extend(util.get_data(s7, 'time_records', 'id'))
        outputs.extend(util.get_data(s7, 'time_records', 'project_id'))
        outputs.extend(util.get_data(s7, 'time_records', 'name'))
        outputs.extend(util.get_data(s7, 'time_records', 'description'))
    if util.has_rows(s7):
        s8 = util.do_sql(conn, "SELECT 'projects'.* FROM 'projects' WHERE 'projects'."id" = :x0 LIMIT 1", {'x0': inputs[1]})
        outputs.extend(util.get_data(s8, 'projects', 'id'))
        outputs.extend(util.get_data(s8, 'projects', 'name'))
        s9 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
        outputs.extend(util.get_data(s9, 'people', 'project_id'))
        outputs.extend(util.get_data(s9, 'people', 'user_id'))
    if util.has_rows(s9):
        s37 = util.do_sql(conn, "SELECT 'companies'.* FROM 'companies' WHERE 'companies'."id" = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'time_records', 'assigned_to_company_id')})
        outputs.extend(util.get_data(s37, 'companies', 'name'))
    if util.has_rows(s37):
        s47 = util.do_sql(conn, "SELECT 'tasks'.* FROM 'tasks' WHERE 'tasks'."id" = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'time_records', 'task_id')})
        outputs.extend(util.get_data(s47, 'tasks', 'id'))
        outputs.extend(util.get_data(s47, 'tasks', 'text'))
if util.has_rows(s47):
    s51 = util.do_sql(conn, "SELECT `tasks`.* FROM `tasks` WHERE `tasks`. `id` = :x0 LIMIT 1", {'x0': util.get_one_data(s47, 'tasks', 'task_list_id')})
    outputs.extend(util.get_data(s51, 'tasks_list', 'project_id'))

if util.has_rows(s51):
    s52 = util.do_sql(conn, "SELECT `people`.* FROM `people` WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
    outputs.extend(util.get_data(s52, 'people', 'project_id'))

outputs.extend(util.get_data(s52, 'people', 'user_id'))
outputs = util.get_one_data(s7, 'display_name')

if util.has_rows(s39):
    s43 = util.do_sql(conn, "SELECT `tasks`.* FROM `tasks` WHERE `tasks`. `id` = :x0 LIMIT 1", {'x0': util.get_one_data(s39, 'tasks', 'task_list_id')})
    outputs.extend(util.get_data(s43, 'tasks_list', 'project_id'))

if util.has_rows(s43):
    s44 = util.do_sql(conn, "SELECT `people`.* FROM `people` WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

s48 = util.do_sql(conn, "SELECT `people`.* FROM `people` WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s48, 'people', 'project_id'))

s49 = util.do_sql(conn, "SELECT `people`.* FROM `people` WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s49, 'people', 'project_id'))

s50 = util.do_sql(conn, "SELECT `people`.* FROM `people` WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s50, 'people', 'project_id'))

else:
    pass

s47 = util.do_sql(conn, "SELECT `people`.* FROM `people` WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s47, 'people', 'project_id'))

s39 = util.do_sql(conn, "SELECT `tasks`.* FROM `tasks` WHERE `tasks`. `id` = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'time_records', 'assigned_to_user_id')})
outputs.extend(util.get_data(s39, 'tasks', 'display_name'))

s38 = util.do_sql(conn, "SELECT `users`.* FROM `users` WHERE `users`. `id` = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'time_records', 'assigned_to_user_id')})
outputs.extend(util.get_data(s38, 'users', 'id'))

s44 = util.do_sql(conn, "SELECT `people`.* FROM `people` WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s44, 'people', 'project_id'))
outputs.extend(util.get_data(s44, 'people', 'project_id'))
outputs.extend(util.get_data(s44, 'people', 'user_id'))

s45 = util.do_sql(conn, "SELECT 'people' FROM people WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s45, 'people', 'project_id'))
outputs.extend(util.get_data(s45, 'people', 'user_id'))

s46 = util.do_sql(conn, "SELECT 'people' FROM people WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s46, 'people', 'project_id'))
outputs.extend(util.get_data(s46, 'people', 'user_id'))
else:
    pass
else:
s40 = util.do_sql(conn, "SELECT 'people' FROM people WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s40, 'people', 'project_id'))
outputs.extend(util.get_data(s40, 'people', 'user_id'))
s41 = util.do_sql(conn, "SELECT 'people' FROM people WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s41, 'people', 'project_id'))
outputs.extend(util.get_data(s41, 'people', 'user_id'))
s42 = util.do_sql(conn, "SELECT 'people' FROM people WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s42, 'people', 'project_id'))
outputs.extend(util.get_data(s42, 'people', 'user_id'))
else:
s10 = util.do_sql(conn, "SELECT 'people' FROM people WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
s11 = util.do_sql(conn, "SELECT 'companies' FROM companies WHERE 'companies'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'time_records', 'assigned_to_company_id')})
outputs.extend(util.get_data(s11, 'companies', 'name'))
if util.has_rows(s11):
s25 = util.do_sql(conn, "SELECT 'tasks' FROM tasks WHERE 'tasks'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'time_records', 'task_id')})
outputs.extend(util.get_data(s25, 'tasks', 'id'))
outputs.extend(util.get_data(s25, 'tasks', 'text'))
if util.has_rows(s25):
s31 = util.do_sql(conn, "SELECT 'task_lists' FROM task_lists WHERE 'task_lists'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s25, 'tasks', 'task_list_id')})
outputs.extend(util.get_data(s31, 'task_lists', 'project_id'))
if util.has_rows(s31):
s32 = util.do_sql(conn, "SELECT 'people' FROM people WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
s33 = util.do_sql(conn, "SELECT 'people' FROM people WHERE 'people'.user_id = :x0 AND 'people'.project_id = :x1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
s34 = util.do_sql(conn, "SELECT 'people' FROM people WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
s35 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE 'people'."user_id" = :x0 AND 'people'."project_id" = :x1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

s36 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

else:
    pass
else:
    s26 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

s27 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE 'people'."user_id" = :x0 AND 'people'."project_id" = :x1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

s28 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

s29 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE 'people'."user_id" = :x0 AND 'people'."project_id" = :x1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

s30 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

else:
    s12 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'."id" = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'time_records', 'assigned_to_user_id')})

outputs.extend(util.get_data(s12, 'users', 'display_name'))

s13 = util.do_sql(conn, "SELECT 'tasks'.* FROM 'tasks' WHERE 'tasks'."id" = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'time_records', 'task_id')})

outputs.extend(util.get_data(s13, 'tasks', 'id'))

outputs.extend(util.get_data(s13, 'tasks', 'text'))

if util.has_rows(s13):

    s19 = util.do_sql(conn, "SELECT 'task_lists'.* FROM 'task_lists' WHERE 'task_lists'."id" = :x0 LIMIT 1", {'x0': util.get_one_data(s13, 'tasks', 'task_list_id')})

outputs.extend(util.get_data(s19, 'task_lists', 'project_id'))

if util.has_rows(s19):

    s20 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

s21 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE 'people'."user_id" = :x0 AND 'people'."project_id" = :x1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

s22 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

s23 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE 'people'."user_id" = :x0 AND 'people'."project_id" = :x1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

s24 = util.do_sql(conn, "SELECT 'people'.* FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})

else:
    pass
else:
A.6 RailsCollab Project Manager Command get_projects_id_milestones_id

```python
def get_projects_id_milestones_id(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'username' = :x0 LIMIT 1", {'x0': inputs[0]})
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT 'people'.'id' FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': inputs[0], 'x1': inputs[1]})
        s2 = util.do_sql(conn, "SELECT 'projects'.'id' FROM 'projects' INNER JOIN 'people' ON 'projects'.'id' = 'people'.'project_id' WHERE 'people'.'user_id' = :x0 AND (projects.completed_on IS NULL) ORDER BY projects.priority ASC, projects.name ASC", {'x0': inputs[0]})
        s3 = util.do_sql(conn, "SELECT 'time_records'.'id' FROM 'time_records' WHERE 'time_records'.'assigned_to_user_id' = :x0 AND (start_date IS NOT NULL AND done_date IS NULL)", {'x0': inputs[0]})
        s4 = util.do_sql(conn, "SELECT 'projects'.'id' FROM 'projects' WHERE 'projects'.'id' = :x0 LIMIT 1", {'x0': inputs[0]})
        outputs.extend(util.get_data(s4, 'projects', 'id'))
    if util.has_rows(s4):
        s5 = util.do_sql(conn, "SELECT 'companies'.'id' FROM 'companies' WHERE 'companies'.'id' = :x0 LIMIT 1", {'x0': inputs[0]})
        if util.has_rows(s5):
            s6 = util.do_sql(conn, "SELECT 'companies'.'id' FROM 'companies' WHERE 'companies'.'id' IS NULL LIMIT 1", {})  
        s7 = util.do_sql(conn, "SELECT 'milestones'.'id' FROM 'milestones' WHERE 'milestones'.'project_id' = :x0 AND 'milestones'.'id' = :x1 LIMIT 1", {'x0': inputs[0], 'x1': inputs[1]})
        outputs.extend(util.get_data(s7, 'milestones', 'id'))
        outputs.extend(util.get_data(s7, 'milestones', 'name'))
        outputs.extend(util.get_data(s7, 'milestones', 'description'))
    if util.has_rows(s7):
        s8 = util.do_sql(conn, "SELECT 'projects'.'id' FROM 'projects' WHERE 'projects'.'id' = :x0 LIMIT 1", {'x0': inputs[0]})
        outputs.extend(util.get_data(s8, 'projects', 'id'))
        outputs.extend(util.get_data(s8, 'projects', 'name'))
```

s15 = util.do_sql(conn, "SELECT 'people' FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s8, 'users', 'id'), 'x1': inputs[1]}))
outputs.extend(util.get_data(s9, 'people', 'project_id'))
s10 = util.do_sql(conn, "SELECT 'people' FROM 'people' WHERE 'people'.user_id = :x0 AND 'people'.project_id = :x1", {'x0': util.get_one_data(s8, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s10, 'people', 'project_id'))
if util.has_rows(s10):
    s11 = util.do_sql(conn, "SELECT 'companies' FROM 'companies' WHERE 'companies'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'milestones', 'assigned_to_company_id')})
outputs.extend(util.get_data(s11, 'companies', 'name'))
if util.has_rows(s11):
    s19 = util.do_sql(conn, "SELECT 'messages' FROM 'messages' WHERE 'messages'.milestone_id = :x0", {'x0': inputs[2]})
outputs.extend(util.get_data(s19, 'messages', 'id'))
outputs.extend(util.get_data(s19, 'messages', 'milestone_id'))
s19_all = s19
for s19 in s19_all:
    s20 = util.do_sql(conn, "SELECT 'users' FROM 'users' WHERE 'users'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s9, 'messages', 'created_by_id')})
outputs.extend(util.get_data(s20, 'users', 'id'))
outputs.extend(util.get_data(s20, 'users', 'display_name'))
s19 = s19_all
s21 = util.do_sql(conn, "SELECT 'task_lists' FROM 'task_lists' WHERE 'task_lists'.milestone_id = :x0 ORDER BY 'order' DESC", {'x0': inputs[2]})
outputs.extend(util.get_data(s21, 'task_lists', 'id'))
outputs.extend(util.get_data(s21, 'task_lists', 'milestone_id'))
outputs.extend(util.get_data(s21, 'task_lists', 'name'))
s22 = util.do_sql(conn, "SELECT 'people' FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s8, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s22, 'people', 'project_id'))
outputs.extend(util.get_data(s22, 'people', 'user_id'))
s23 = util.do_sql(conn, "SELECT 'people' FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s9, 'people', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s23, 'people', 'project_id'))
s24 = util.do_sql(conn, "SELECT 'people' FROM 'people' WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'x0': util.get_one_data(s10, 'people', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s24, 'people', 'project_id'))
outputs.extend(util.get_data(s24, 'people', 'user_id'))
else:
    s12 = util.do_sql(conn, "SELECT 'users' FROM 'users' WHERE 'users'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s7, 'milestones', 'assigned_to_user_id')})
outputs.extend(util.get_data(s12, 'users', 'display_name'))
s13 = util.do_sql(conn, "SELECT 'messages' FROM 'messages' WHERE 'messages'.milestone_id = :x0", {'x0': inputs[2]})
outputs.extend(util.get_data(s13, 'messages', 'id'))
outputs.extend(util.get_data(s13, 'messages', 'milestone_id'))
s13_all = s13
for s13 in s13_all:
    s14 = util.do_sql(conn, "SELECT 'users' FROM 'users' WHERE 'users'.id = :x0 LIMIT 1", {'x0': util.get_one_data(s13, 'messages', 'created_by_id')})
outputs.extend(util.get_data(s14, 'users', 'id'))
outputs.extend(util.get_data(s14, 'users', 'display_name'))
s13 = s13_all
s15 = util.do_sql(conn, "SELECT 'task_lists' FROM 'task_lists' WHERE 'task_lists'.milestone_id = :x0 ORDER BY 'order' DESC", {'x0': inputs[2]})
Active Loop Detection for Applications that Access Databases

A.7 RailsCollab Project Manager Command get_projects

def get_projects(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT "users".* FROM "users" WHERE "users"."username" = :x0 LIMIT 1", ('x0': util.get_one_data(s0, 'users', 'id')))
    outputs.extend((util.get_data(s0, 'users', 'id')))
    s1 = util.do_sql(conn, "SELECT "projects".* FROM "projects" INNER JOIN "people" ON "projects"."id" = "people"."project_id" WHERE "people"."username" = :x0 AND (projects.completed_on IS NULL ORDER BY projects.priority ASC, projects.name ASC), ('x0': util.get_one_data(s0, 'users', 'id')))
    s2 = util.do_sql(conn, "SELECT "companies".* FROM "companies" WHERE "companies"."id" = :x0 LIMIT 1", ('x0': util.get_one_data(s0, 'companies', 'id')))
    outputs.extend((util.get_data(s2, 'companies', 'id')))
    outputs.extend((util.get_data(s1, 'people', 'company_id')))
    s3 = util.do_sql(conn, "SELECT "time_records".* FROM "time_records" WHERE "time_records"."assigned_to_user_id" = :x0 AND (start_date IS NOT NULL AND done_date IS NULL)", ('x0': util.get_one_data(s0, 'users', 'id')))
    outputs.extend((util.get_data(s3, 'company_id')))}
    s4 = util.do_sql(conn, "SELECT "projects".* FROM "projects"", ()
    s5 = util.do_sql(conn, "SELECT "companies".* FROM "companies" WHERE "companies"."id" = :x0 LIMIT 1", ('x0': util.get_one_data(s0, 'companies', 'id')))
    outputs.extend((util.get_data(s5, 'companies', 'id')))
    outputs.extend((util.get_data(s5, 'companies', 'name')))
    if util.has_rows(s5):
        s6 = util.do_sql(conn, "SELECT "companies".* FROM "companies" WHERE "companies"."id" IS NULL LIMIT 1", ()
    else:
        pass
    return util.add_warnings(outputs)
A.8 RailsCollab Project Manager Command get_companies_id

```python
def get_companies_id(conn, inputs):
    outputs = []
s0 = util.do_sql(conn, "SELECT 'users' FROM 'users' WHERE 'username' = :x0 LIMIT 1", ('x0': inputs[0]))
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT 'companies' FROM 'companies' WHERE 'id' = :x0 LIMIT 1", ('x0': util.get_one_data(s0, ['users', 'id'])))
        s2 = util.do_sql(conn, "SELECT 'projects' FROM 'projects' INNER JOIN 'people' ON 'projects'.id = 'people'.project_id WHERE 'people'.user_id = :x0 AND (projects.completed_on IS NULL) ORDER BY projects.priority ASC, projects.name ASC", ('x0': util.get_one_data(s0, ['users', 'id'])))
        s3 = util.do_sql(conn, "SELECT 'time_records' FROM 'time_records' WHERE 'time_records'.assigned_to_user_id = :x0 AND (start_date IS NOT NULL AND done_date IS NULL)", ('x0': util.get_one_data(s0, ['users', 'id'])))
        s4 = util.do_sql(conn, "SELECT 'companies' FROM 'companies' WHERE 'companies'.id = :x0 LIMIT 1", ('x0': inputs[1]))
    outputs.extend(util.get_data(s0, 'companies'))
    outputs.extend(util.get_data(s1, 'companies'))
    outputs.extend(util.get_data(s2, 'companies'))
    outputs.extend(util.get_data(s3, 'companies'))
    outputs.extend(util.get_data(s4, 'companies'))
    if util.has_rows(s4):
        s5 = util.do_sql(conn, "SELECT 'companies' FROM 'companies' WHERE 'companies'.id IS NULL LIMIT 1", ())
    else:
        pass
    else:
        pass
    return util.add_warnings(outputs)
```

A.9 RailsCollab Project Manager Command get_users_id

For this command we use a modified version of RailsCollab, where we fixed a 500 error when a user's IM type ID does not match any records in the database. Specifically, we changed the statement "<td><img src="<%= im_value.im_type.icon_url %>"><alt="<%= im_value.im_type.name rescue nil %>"></td>" into "<td><img src="<%= im_value.im_type.icon_url rescue nil %>"><alt="<%= im_value.im_type.name rescue nil %>"></td>" in the file app/views/users/_card.html.erb.
Active Loop Detection for Applications that Access Databases

```python
outputs.extend(util.get_data(s4, 'users', 'display_name'))
outputs.extend(util.get_data(s4, 'users', 'title'))
outputs.extend(util.get_data(s4, 'users', 'office_number'))
outputs.extend(util.get_data(s4, 'users', 'fax_number'))
outputs.extend(util.get_data(s4, 'users', 'home_number'))
if util.has_rows(s4):
    s5 = util.do_sql(conn, "SELECT 'companies'.* FROM 'companies' WHERE 'companies'.'id' = x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'company_id')})
outputs.extend(util.get_data(s5, 'companies', 'id'))
outputs.extend(util.get_data(s5, 'companies', 'name'))
outputs.extend(util.get_data(s5, 'companies', 'homepage'))
if util.has_rows(s5):
    s6 = util.do_sql(conn, "SELECT 'companies'.* FROM 'companies' WHERE 'companies'.'id' IS NULL LIMIT 1", ())
    s7 = util.do_sql(conn, "SELECT 'companies'.* FROM 'companies' WHERE 'companies'.'id' = x0 LIMIT 1", (x0: util.get_one_data(s4, 'users', 'company_id')))
outputs.extend(util.get_data(s7, 'companies', 'id'))
outputs.extend(util.get_data(s7, 'companies', 'name'))
if util.has_rows(s7):
    s8 = util.do_sql(conn, "SELECT 'companies'.* FROM 'companies' WHERE 'companies'.'id' IS NULL LIMIT 1", ())
    s9 = util.do_sql(conn, "SELECT COUNT(*) FROM 'user_im_values' WHERE 'user_im_values'.'user_id' = x0 ORDER BY im_type_id DESC", {'x0': inputs[1]})
    outputs.extend(util.get_data(s8, 'user_im_values', 'user_id'))
    outputs.extend(util.get_data(s9, 'user_im_values', 'value'))
    s10_all = s10
    for s10 in s10_all:
        s11 = util.do_sql(conn, "SELECT 'im_types'.* FROM 'im_types' WHERE 'im_types'.'id' = x0 LIMIT 1", (x0: util.get_one_data(s10, 'user_im_values', 'im_type_id')))
        outputs.extend(util.get_data(s11, 'im_types', 'icon'))
s10 = s10_all
pass
else:
    pass
else:
    pass
else:
    pass
else:
    pass
return util.add_warnings(outputs)
```

A.10 Kanban Task Manager Command get_api_lists

```python
def get_api_lists(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'email' = x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': inputs[0]})
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT 'lists'.* FROM 'lists' INNER JOIN 'boards' ON 'lists'.'board_id' = 'boards'.'id' INNER JOIN 'board_members' ON 'boards'.'id' = 'board_members'.'board_id' WHERE 'board_members'.'member_id' = x0 ORDER BY 'lists'.'position' ASC", {'x0': util.get_one_data(s0, 'users', 'id')})
        outputs.extend(util.get_data(s1, 'lists', 'id'))
    outputs.extend(util.get_data(s1, 'lists', 'board_id'))
    outputs.extend(util.get_data(s1, 'lists', 'title'))
    outputs.extend(util.get_data(s1, 'lists', 'position'))
```
s1_all = s1
for s1 in s1_all:
    s2 = util.do_sql(conn, "SELECT `boards`.* FROM `boards` WHERE `boards`\`id` = :x0 LIMIT 1", (\'x0\': util.get_one_data(s1, \'lists\', \'board_id\')))
outputs.extend(util.get_data(s2, \'boards\', \'id\'))

s3 = util.do_sql(conn, "SELECT `cards`.* FROM `cards` WHERE `cards`\`list_id` = :x0 ORDER BY `cards`\`.position` ASC", (\'x0\': util.get_one_data(s1, \'lists\', \'id\')))
outputs.extend(util.get_data(s3, \'cards\', \'id\'))

s4 = util.do_sql(conn, "SELECT COUNT(*) FROM `card_comments` WHERE `card_comments`\`.card_id` = :x0 ORDER BY `users`\`.id` ASC LIMIT 1", (\'x0\': util.get_one_data(s1, \'cards\', \'id\')))

s5 = util.do_sql(conn, "SELECT `users`.* FROM `users` WHERE `users`\`id` = :x0 ORDER BY `users`\`.id` ASC LIMIT 1", (\'x0\': util.get_one_data(s1, \'cards\', \'id\')))

s6 = util.do_sql(conn, "SELECT `card_comments`.* FROM `card_comments` WHERE `card_comments`\`.card_id` = :x0 ORDER BY `card_comments`\`.created_at` DESC", (\'x0\': util.get_one_data(s1, \'cards\', \'id\')))

for s6 in s6_all:
    s7 = util.do_sql(conn, "SELECT `cards`.* FROM `cards` WHERE `cards`\`id` = :x0 ORDER BY `cards`\`.position` ASC LIMIT 1", (\'x0\': util.get_one_data(s6, \'card_comments\', \'card_id\')))
outputs.extend(util.get_data(s7, \'cards\', \'id\'))
outputs.extend(util.get_data(s7, \'cards\', \'list_id\'))
outputs.extend(util.get_data(s7, \'cards\', \'title\'))
outputs.extend(util.get_data(s7, \'cards\', \'description\'))
outputs.extend(util.get_data(s7, \'cards\', \'due_date\'))
outputs.extend(util.get_data(s7, \'cards\', \'position\'))

s8 = util.do_sql(conn, "SELECT `users`.* FROM `users` WHERE `users`\`id` = :x0 ORDER BY `users`\`.id` ASC LIMIT 1", (\'x0\': util.get_one_data(s6, \'card_comments\', \'commenter_id\')))
outputs.extend(util.get_data(s8, \'users\', \'id\'))
outputs.extend(util.get_data(s8, \'users\', \'email\'))
outputs.extend(util.get_data(s8, \'users\', \'bio\'))
outputs.extend(util.get_data(s8, \'users\', \'full_name\'))
s6 = s6_all

pass

s3 = s3_all
pass

s1 = s1_all
pass

else:
    pass

return util.add_warnings(outputs)

A.11 Kanban Task Manager Command get_api_lists_id

def get_api_lists_id(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT `users`.* FROM `users` WHERE `users`\`.email` = :x0 ORDER BY `users`\`.id` ASC LIMIT 1", (\'x0\': inputs[0]))
    if util.has_rows(s0):
s1 = util.do_sql(conn, "SELECT `lists`.* FROM `lists` INNER JOIN `boards` ON `lists`.
  `board_id` = `boards`.
  `id` INNER JOIN `board_members` ON `boards`.
  `id` = `board_members`.
  `board_id` WHERE `board_members`.
  `member_id` = x0 AND `lists`.
  `id` = x1 ORDER BY `lists`.
  `position` ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
outputs.extend(util.get_data(s1, 'lists', 'id'))
outputs.extend(util.get_data(s1, 'lists', 'board_id'))
outputs.extend(util.get_data(s1, 'lists', 'title'))
outputs.extend(util.get_data(s1, 'lists', 'position'))
if util.has_rows(s1):
  s2 = util.do_sql(conn, "SELECT `boards`.* FROM `boards` WHERE `boards`.
  `id` = x0 LIMIT 1", {'x0': util.get_one_data(s1, 'lists', 'board_id')})
outputs.extend(util.get_data(s2, 'boards', 'id'))
s3 = util.do_sql(conn, "SELECT `cards`.* FROM `cards` WHERE `cards`.
  `list_id` = x0 ORDER BY `cards`.
  `position` ASC", {'x0': inputs[1]})
outputs.extend(util.get_data(s3, 'cards', 'id'))
outputs.extend(util.get_data(s3, 'cards', 'title'))
outputs.extend(util.get_data(s3, 'cards', 'description'))
outputs.extend(util.get_data(s3, 'cards', 'due_date'))
outputs.extend(util.get_data(s3, 'cards', 'position'))
s3_all = s3
for s3 in s3_all:
  s4 = util.do_sql(conn, "SELECT COUNT(*) FROM `card_comments` WHERE `card_comments`.
  `card_id` = :x0", {'x0': util.get_one_data(s3, 'cards', 'id')})
s5 = util.do_sql(conn, "SELECT `users`.* FROM `users` WHERE `users`.
  `id` = :x0 ORDER BY `users`.
  `id` ASC LIMIT 1", {'x0': util.get_one_data(s3, 'cards', 'assignee_id')})
outputs.extend(util.get_data(s5, 'users', 'id'))
outputs.extend(util.get_data(s5, 'users', 'email'))
outputs.extend(util.get_data(s5, 'users', 'bio'))
s6 = util.do_sql(conn, "SELECT `card_comments`.* FROM `card_comments` WHERE `card_comments`.
  `card_id` = :x0 ORDER BY `card_comments`.
  `created_at` DESC", {'x0': util.get_one_data(s3, 'cards', 'id')})
outputs.extend(util.get_data(s6, 'card_comments', 'card_id'))
outputs.extend(util.get_data(s6, 'card_comments', 'content'))
s6_all = s6
for s6 in s6_all:
  s7 = util.do_sql(conn, "SELECT `cards`.* FROM `cards` WHERE `cards`.
  `id` = :x0 ORDER BY `cards`.
  `position` ASC LIMIT 1", {'x0': util.get_one_data(s6, 'card_comments', 'card_id')})
outputs.extend(util.get_data(s7, 'cards', 'id'))
outputs.extend(util.get_data(s7, 'cards', 'title'))
outputs.extend(util.get_data(s7, 'cards', 'description'))
outputs.extend(util.get_data(s7, 'cards', 'due_date'))
s8 = util.do_sql(conn, "SELECT `users`.* FROM `users` WHERE `users`.
  `id` = :x0 ORDER BY `users`.
  `id` ASC LIMIT 1", {'x0': util.get_one_data(s6, 'card_comments', 'commenter_id')})
outputs.extend(util.get_data(s8, 'users', 'id'))
outputs.extend(util.get_data(s8, 'users', 'email'))
outputs.extend(util.get_data(s8, 'users', 'bio'))
s6 = s6_all
pass
s3 = s3_all
pass
else:
  pass
else:
  pass
return util.add_warnings(outputs)

A.12 Kanban Task Manager Command get_api_cards

```python
def get_api_cards (conn, inputs):
```
return
46
45
44
else
41
40
39
outputs . extend ( util . get_data (s7 , 'users' , 'id'))
38
outputs . extend (util . get_data(s7 , 'users' , 'bio'))
37
outputs . extend (util . get_data(s7 , 'users' , 'email'))
36
outputs . extend (util . get_data(s7 , 'users' , 'full_name'))
35
s7 = util . do_sql (conn , "SELECT COUNT(*) FROM 'users' WHERE 'users'.'id' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': util . get_one_data(s5 , 'card_comments' , 'card_id')})
34
outputs . extend (util . get_data(s7 , 'users' , 'email'))
33
outputs . extend (util . get_data(s7 , 'users' , 'bio'))
32
outputs . extend (util . get_data(s7 , 'users' , 'full_name'))
31
s7_all = s7
30
for s7 in s7_all:
29
s6 = util . do_sql (conn , "SELECT 'cards'.'* FROM 'cards' WHERE 'cards'.'id' = :x0 ORDER BY 'cards'.'position' ASC LIMIT 1", {'x0': util . get_one_data(s5 , 'card_comments' , 'card_id')})
28
outputs . extend (util . get_data(s6 , 'cards' , 'id'))
27
outputs . extend (util . get_data(s6 , 'cards' , 'title'))
26
outputs . extend (util . get_data(s6 , 'cards' , 'description'))
25
outputs . extend (util . get_data(s6 , 'cards' , 'due_date'))
24
outputs . extend (util . get_data(s6 , 'cards' , 'position'))
23
s5 = util . do_sql (conn , "SELECT 'card_comments'.'* FROM 'card_comments' WHERE 'card_comments'.'card_id' = :x0 ORDER BY 'card_comments'.'created_at' DESC" , {'x0': util . get_one_data(s1 , 'cards' , 'id')})
22
outputs . extend (util . get_data(s5 , 'card_comments' , 'card_id'))
21
outputs . extend (util . get_data(s5 , 'card_comments' , 'content'))
20
s5_all = s5
19
for s5 in s5_all:
18
s4 = util . do_sql (conn , "SELECT 'users'.'* FROM 'users' WHERE 'users'.'id' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': util . get_one_data(s1 , 'cards' , 'id')})
17
outputs . extend (util . get_data(s4 , 'users' , 'id'))
16
outputs . extend (util . get_data(s4 , 'users' , 'email'))
15
outputs . extend (util . get_data(s4 , 'users' , 'bio'))
14
outputs . extend (util . get_data(s4 , 'cards' , 'full_name'))
13
s1 = util . do_sql (conn , "SELECT COUNT(*) FROM 'card_comments' WHERE 'card_comments'.'card_id' = :x0 ORDER BY 'card_comments'.'card_id' ASC LIMIT 1", {'x0': util . get_one_data(s1 , 'cards' , 'id')})
12
outputs . extend (util . get_data(s1 , 'cards' , 'title'))
11
outputs . extend (util . get_data(s1 , 'cards' , 'description'))
10
outputs . extend (util . get_data(s1 , 'cards' , 'due_date'))
9
s1_all = s1
8
for s1 in s1_all:
7
s3 = util . do_sql (conn , "SELECT 'lists'.'* FROM 'lists' WHERE 'lists'.'id' = :x0 ORDER BY 'lists'.'position' ASC LIMIT 1", {'x0': util . get_one_data(s1 , 'cards' , 'list_id')})
6
outputs . extend (util . get_data(s3 , 'lists' , 'id'))
5
s2 = util . do_sql (conn , "SELECT COUNT(*) FROM 'card_comments' WHERE 'card_comments'.'card_id' = :x0", {'x0': util . get_one_data(s2 , 'cards' , 'id')})
4
outputs . extend (util . get_data(s2 , 'cards' , 'list_id'))
3
s1 = util . do_sql (conn , "SELECT 'cards'.'* FROM 'cards' INNER JOIN 'lists' ON 'cards'.'list_id' = 'lists'.'id' INNER JOIN 'boards' ON 'lists'.'board_id' = 'boards'.'id' INNER JOIN 'board_members' ON 'boards'.'id' = 'board_members'.'board_id' WHERE 'board_members'.'member_id' = :x0 ORDER BY 'cards'.'position' ASC , 'lists'.'position' ASC LIMIT 1", {'x0': util . get_one_data(s0 , 'users' , 'id')})
2
outputs . extend (util . get_data(s1 , 'cards' , 'id'))
1
for s1 in s1_all:
0
s0 = util . do_sql (conn , "SELECT 'users'.'* FROM 'users' WHERE 'users'.'email' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': inputs[0]})

A.13 Kanban Task Manager Command get_api_cards_id

```python
def get_api_cards_id (conn , inputs):
    util . clear_warnings ()
    outputs = []
```
Active Loop Detection for Applications that Access Databases

A.14 Kanban Task Manager Command get_api_boards_id

```python
def get_api_boards_id(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.email = :x0 ORDER BY 'users'.id ASC LIMIT 1", {'x0': inputs[0]})
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT 'cards'.* FROM 'cards' INNER JOIN 'boards' ON 'cards'.board_id = :x0 INNER JOIN 'board_members' ON 'boards'.id = 'board_members'.board_id WHERE 'board_members'.member_id = :x0 AND 'cards'.id = :x1 ORDER BY 'cards'.position ASC, 'boards'.position ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
        outputs.extend(util.get_data(s1, 'cards', 'id'))
        s2 = util.do_sql(conn, "SELECT COUNT(*) FROM 'card_comments' WHERE 'card_comments'.card_id = :x0", {'x0': inputs[1]})
        s3 = util.do_sql(conn, "SELECT 'lists'.* FROM 'lists' WHERE 'lists'.id = :x0 ORDER BY 'lists'.position ASC LIMIT 1", {'x0': util.get_one_data(s1, 'cards', 'list_id')})
        outputs.extend(util.get_data(s3, 'lists', 'id'))
        s4 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id = :x0 ORDER BY 'users'.id ASC LIMIT 1", {'x0': util.get_one_data(s1, 'cards', 'assignee_id')})
        outputs.extend(util.get_data(s4, 'users', 'id'))
        s5 = util.do_sql(conn, "SELECT 'card_comments'.* FROM 'card_comments' WHERE 'card_comments'.card_id = :x0 ORDER BY 'card_comments'.created_at DESC", {'x0': inputs[1]})
        outputs.extend(util.get_data(s5, 'card_comments', 'card_id'))
        s5_all = s5
        for s5 in s5_all:
            s6 = util.do_sql(conn, "SELECT 'cards'.* FROM 'cards' WHERE 'cards'.id = :x0 ORDER BY 'cards'.position ASC LIMIT 1", {'x0': util.get_one_data(s5, 'card_comments', 'id')})
            outputs.extend(util.get_data(s6, 'cards', 'id'))
            outputs.extend(util.get_data(s6, 'cards', 'list_id'))
            outputs.extend(util.get_data(s6, 'cards', 'title'))
            outputs.extend(util.get_data(s6, 'cards', 'description'))
            outputs.extend(util.get_data(s6, 'cards', 'due_date'))
            s7 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id = :x0 ORDER BY 'users'.id ASC LIMIT 1", {'x0': util.get_one_data(s5, 'card_comments', 'commenter_id')})
            outputs.extend(util.get_data(s7, 'users', 'id'))
            outputs.extend(util.get_data(s7, 'users', 'email'))
            outputs.extend(util.get_data(s7, 'users', 'bio'))
            s5 = s5_all
    else:
        pass
    else:
        pass
    return util.add_warnings(outputs)
```
```python
s1 = util.do_sql(conn, "SELECT `boards`.* FROM `boards` INNER JOIN `board_members` ON `boards`.'id' = `board_members`.'board_id' WHERE `board_members`.'member_id' = :x0 AND `boards`.'id' = :x1 LIMIT 1", {'x0': inputs[0], 'x1': inputs[1]})
outputs.extend(util.get_data(s1, 'boards', 'id'))
outputs.extend(util.get_data(s1, 'boards', 'name'))
outputs.extend(util.get_data(s1, 'boards', 'description'))
if util.has_rows(s1):
s2 = util.do_sql(conn, "SELECT `lists`.* FROM `lists` WHERE `lists`.'board_id' = :x0 ORDER BY `lists`.'position' ASC", {'x0': inputs[1]})
outputs.extend(util.get_data(s2, 'lists', 'id'))
outputs.extend(util.get_data(s2, 'lists', 'name'))
outputs.extend(util.get_data(s2, 'lists', 'title'))
outputs.extend(util.get_data(s2, 'lists', 'position'))
for s2 in s2_all:
s3 = util.do_sql(conn, "SELECT `cards`.* FROM `cards` WHERE `cards`.'list_id' = :x0 ORDER BY `cards`.'position' ASC", {'x0': util.get_one_data(s2, 'cards', 'id')})
outputs.extend(util.get_data(s3, 'cards', 'id'))
outputs.extend(util.get_data(s3, 'cards', 'list_id'))
outputs.extend(util.get_data(s3, 'cards', 'title'))
outputs.extend(util.get_data(s3, 'cards', 'description'))
outputs.extend(util.get_data(s3, 'cards', 'due_date'))
outputs.extend(util.get_data(s3, 'cards', 'position'))
s3_all = s3
for s3 in s3_all:
s4 = util.do_sql(conn, "SELECT COUNT(*) FROM `card_comments` WHERE `card_comments`.'card_id' = :x0", {'x0': util.get_one_data(s3, 'cards', 'id')})
s5 = util.do_sql(conn, "SELECT `users`.* FROM `users` WHERE `users`.'id' = :x0 ORDER BY `users`.'id' ASC LIMIT 1", {'x0': util.get_one_data(s3, 'cards', 'id')})
outputs.extend(util.get_data(s5, 'users', 'id'))
outputs.extend(util.get_data(s5, 'users', 'email'))
outputs.extend(util.get_data(s5, 'users', 'bio'))
outputs.extend(util.get_data(s5, 'users', 'full_name'))
s6 = util.do_sql(conn, "SELECT `card_comments`.* FROM `card_comments` WHERE `card_comments`.'card_id' = :x0 ORDER BY `card_comments`.'created_at' DESC", {'x0': util.get_one_data(s3, 'cards', 'id')})
outputs.extend(util.get_data(s6, 'card_comments', 'card_id'))
outputs.extend(util.get_data(s6, 'card_comments', 'content'))
s6_all = s6
for s6 in s6_all:
s7 = util.do_sql(conn, "SELECT `cards`.* FROM `cards` WHERE `cards`.'id' = :x0 ORDER BY `cards`.'position' ASC LIMIT 1", {'x0': util.get_one_data(s6, 'card_comments', 'card_id')})
outputs.extend(util.get_data(s7, 'cards', 'id'))
outputs.extend(util.get_data(s7, 'cards', 'list_id'))
outputs.extend(util.get_data(s7, 'cards', 'title'))
outputs.extend(util.get_data(s7, 'cards', 'description'))
outputs.extend(util.get_data(s7, 'cards', 'due_date'))
outputs.extend(util.get_data(s7, 'cards', 'position'))
s8 = util.do_sql(conn, "SELECT `users`.* FROM `users` WHERE `users`.'id' = :x0 ORDER BY `users`.'id' ASC LIMIT 1", {'x0': util.get_one_data(s6, 'card_comments', 'commenter_id')})
outputs.extend(util.get_data(s8, 'users', 'id'))
outputs.extend(util.get_data(s8, 'users', 'email'))
outputs.extend(util.get_data(s8, 'users', 'bio'))
outputs.extend(util.get_data(s8, 'users', 'full_name'))
s6 = s6_all
```
```python
A.15 Kanban Task Manager Command get_api_users_current

```python
def get_api_users_current(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'email' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': inputs[0]})
    outputs.extend(util.get_data(s0, 'users', 'id'))
    outputs.extend(util.get_data(s0, 'users', 'email'))
    outputs.extend(util.get_data(s0, 'users', 'bio'))
    return util.add_warnings(outputs)

A.16 Kanban Task Manager Command get_api_users_id

```python
def get_api_users_id(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'email' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': inputs[0]})
    outputs.extend(util.get_data(s0, 'users', 'id'))
    outputs.extend(util.get_data(s0, 'users', 'email'))
    outputs.extend(util.get_data(s0, 'users', 'bio'))
    return util.add_warnings(outputs)

A.17 Todo Task Manager Command get_home

```python
def get_home(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0_all = s0
    for s0 in s0_all:
        s1 = util.do_sql(conn, "SELECT 1 AS one FROM 'tasks' WHERE 'tasks'.'list_id' = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'lists', 'id')})
        if util.has_rows(s1):
            s3 = util.do_sql(conn, "SELECT 'tasks'.'* FROM 'tasks' WHERE 'tasks'.'list_id' = :x0", {'x0': util.get_one_data(s0, 'lists', 'id')})
            outputs.extend(util.get_data(s3, 'tasks', 'id'))
            outputs.extend(util.get_data(s3, 'tasks', 'name'))
            s4 = util.do_sql(conn, "SELECT COUNT(*) FROM 'tasks' WHERE 'tasks'.'list_id' = :x0 AND 'tasks'.'done' = 1", {'x0': util.get_one_data(s0, 'lists', 'id')})
        else:
            s2 = util.do_sql(conn, "SELECT COUNT(*) FROM 'tasks' WHERE 'tasks'.'list_id' = :x0 AND 'tasks'.'done' = 1", {'x0': util.get_one_data(s0, 'lists', 'id')})
            s0 = s0_all
    return util.add_warnings(outputs)

A.18 Todo Task Manager Command get_lists_id_tasks

```python
```
A.19 Todo Task Manager Command get_lists_id_tasks

For this command we use a modified version of Todo, where we fixed a 404 error when the provided list ID does not match any records in the database. Specifically, we changed the statement "prevlist = List.find listid" into "prevlist = List.find listid rescue nil" in the file app/views/tasks/index.html.erb.

A.20 Fulcrum Task Manager Command get_home
A.21 Fulcrum Task Manager Command get_projects

```python
def get_projects (conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'email' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': inputs[0]})
    outputs.extend(util.get_data(s0, 'users', 'email'))
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'id' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
        outputs.extend(util.get_data(s1, 'users', 'email'))
        s2 = util.do_sql(conn, "SELECT DISTINCT 'projects'.'id' = 'projects_users'.'project_id' WHERE 'projects_users'.'user_id' = :x0", {'x0': util.get_one_data(s0, 'users', 'id')})
        outputs.extend(util.get_data(s2, 'projects', 'id'))
        s3 = util.do_sql(conn, "SELECT 'projects'.'id' AS 'id' FROM 'projects' WHERE 'projects'.'user_id' = :x0 ORDER BY 'projects'.'id' ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
        outputs.extend(util.get_data(s3, 'projects', 'id'))
        s4 = util.do_sql(conn, "SELECT 'projects'.'id' = 'projects_users'.'project_id' WHERE 'projects_users'.'user_id' = :x0", {'x0': util.get_one_data(s0, 'users', 'id')})
        outputs.extend(util.get_data(s4, 'projects', 'id'))
    else:
        pass
    return util.add_warnings(outputs)
```

A.22 Fulcrum Task Manager Command get_projects_id

```python
def get_projects_id (conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.'id' FROM 'users' WHERE 'users'.'email' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': inputs[0]})
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT 'users'.'id' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
```
s2 = util.do_sql(conn, "SELECT DISTINCT 'projects'.id FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.id = 'projects_users'.project_id WHERE 'projects_users'.user_id = :x0", ("x0": util.get_one_data(s0, 'users', 'id')))

s3 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id = :x0 ORDER BY 'users'.id ASC LIMIT 1", ("x0": util.get_one_data(s0, 'users', 'id')))

s4 = util.do_sql(conn, "SELECT DISTINCT 'projects'.id FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.id = 'projects_users'.project_id WHERE 'projects_users'.user_id = :x0 AND 'projects'.id = :x1 LIMIT 1", ("x0": util.get_one_data(s0, 'users', 'id'), "x1": inputs[1]))

outputs.extend(util.get_data(s4, 'projects', 'id'))
outputs.extend(util.get_data(s4, 'projects', 'name'))
if util.has_rows(s4):
    s6 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' INNER JOIN 'projects_users' ON 'users'.id = 'projects_users'.user_id WHERE 'projects_users'.project_id = :x0", ("x0": inputs[1]))
outputs.extend(util.get_data(s6, 'users', 'id'))
outputs.extend(util.get_data(s6, 'users', 'email'))
outputs.extend(util.get_data(s6, 'users', 'name'))
outputs.extend(util.get_data(s6, 'users', 'initials'))

s7 = util.do_sql(conn, "SELECT DISTINCT 'projects'.id FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.id = 'projects_users'.project_id WHERE 'projects_users'.user_id = :x0", ("x0": util.get_one_data(s0, 'users', 'id'))) outputs.extend(util.get_data(s7, 'projects', 'id'))
outputs.extend(util.get_data(s7, 'projects', 'name'))
else:
    s5 = util.do_sql(conn, "SELECT DISTINCT 'projects'.id FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.id = 'projects_users'.project_id WHERE 'projects_users'.user_id = :x0", ("x0": util.get_one_data(s0, 'users', 'id'))) outputs.extend(util.get_data(s5, 'projects', 'id'))

if util.has_rows(s5):
    s6 = util.do_sql(conn, "SELECT 'stories'.* FROM 'stories' WHERE 'stories'.project_id IN (:x0)", ("x0": inputs[1]))
outputs.extend(util.get_data(s6, 'stories', 'id'))
outputs.extend(util.get_data(s6, 'stories', 'title'))
outputs.extend(util.get_data(s6, 'stories', 'description'))
outputs.extend(util.get_data(s6, 'stories', 'estimate'))
outputs.extend(util.get_data(s6, 'stories', 'requested_by_id'))
outputs.extend(util.get_data(s6, 'stories', 'owned_by_id'))
outputs.extend(util.get_data(s6, 'stories', 'project_id'))
if util.has_rows(s6):
    s7 = util.do_sql(conn, "SELECT 'notes'.* FROM 'notes' WHERE 'notes'.story_id IN (:x0)", ("x0": util.get_data(s6, 'stories', 'id')))

A.23 Fulcrum Task Manager Command get_projects_id_stories

def get_projects_id_stories(conn, inputs):
    util.clear_warnings()
outputs = []
s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.email = :x0 ORDER BY 'users'.id ASC LIMIT 1", ("x0": inputs[0]))
if util.has_rows(s0):
    s1 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id = :x0 ORDER BY 'users'.id ASC LIMIT 1", ("x0": util.get_one_data(s0, 'users', 'id'))) s2 = util.do_sql(conn, "SELECT DISTINCT 'projects'.id FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.id = 'projects_users'.project_id WHERE 'projects_users'.user_id = :x0", ("x0": util.get_one_data(s0, 'users', 'id'))) s3 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id = :x0 ORDER BY 'users'.id ASC LIMIT 1", ("x0": util.get_one_data(s0, 'users', 'id'))) s4 = util.do_sql(conn, "SELECT DISTINCT 'projects'.id FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.id = 'projects_users'.project_id WHERE 'projects_users'.user_id = :x0 AND 'projects'.id = :x1 LIMIT 1", ("x0": util.get_one_data(s0, 'users', 'id')))

if util.has_rows(s4):
    s6 = util.do_sql(conn, "SELECT 'stories'.* FROM 'stories' WHERE 'stories'.project_id IN (:x0)", ("x0": inputs[1]))
outputs.extend(util.get_data(s6, 'stories', 'id'))
outputs.extend(util.get_data(s6, 'stories', 'title'))
outputs.extend(util.get_data(s6, 'stories', 'description'))
outputs.extend(util.get_data(s6, 'stories', 'estimate'))
outputs.extend(util.get_data(s6, 'stories', 'requested_by_id'))
outputs.extend(util.get_data(s6, 'stories', 'owned_by_id'))
outputs.extend(util.get_data(s6, 'stories', 'project_id'))
if util.has_rows(s6):
    s7 = util.do_sql(conn, "SELECT 'notes'.* FROM 'notes' WHERE 'notes'.story_id IN (:x0)", ("x0": util.get_data(s6, 'stories', 'id')))

Jiasi Shen and Martin Rinard
```python
A.24 Fulcrum Task Manager Command get_projects_id_stories_id

def get_projects_id_stories_id(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users' FROM 'users' WHERE 'users'.email = :x0 ORDER BY 'users'.id ASC LIMIT 1", {'x0': inputs[0]})
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT 'users' FROM 'users' WHERE 'users'.id = :x0 ORDER BY 'users'.id ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
    s2 = util.do_sql(conn, "SELECT DISTINCT 'projects' FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.id = 'projects_users'.project_id WHERE 'projects_users'.user_id = :x0", {'x0': util.get_one_data(s0, 'users', 'id')})
    s3 = util.do_sql(conn, "SELECT 'users' FROM 'users' WHERE 'users'.id = :x0 ORDER BY 'users'.id ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
    s4 = util.do_sql(conn, "SELECT DISTINCT 'projects' FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.id = 'projects_users'.project_id WHERE 'projects_users'.user_id = :x0 AND 'projects'.id = :x1 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
    if util.has_rows(s4):
        s5 = util.do_sql(conn, "SELECT DISTINCT 'projects' FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.id = 'projects_users'.project_id WHERE 'projects_users'.user_id = :x0", {'x0': util.get_one_data(s0, 'users', 'id')})
        else:
            pass
    return util.add_warnings(outputs)
```
A.25 Fulcrum Task Manager Command get_projects_id_stories_id_notes

```python
def get_projects_id_stories_id_notes(conn, inputs):
util.clear_warnings()
outputs = []
s0 = util.do_sql(conn, "SELECT 'users' AS FROM 'users' WHERE 'users'.'email' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': inputs[0]})
if util.has_rows(s0):
    s1 = util.do_sql(conn, "SELECT 'users' AS FROM 'users' WHERE 'users'.'id' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
s2 = util.do_sql(conn, "SELECT DISTINCT 'projects' AS FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.'id' = 'projects_users'.'project_id' WHERE 'projects_users'.'user_id' = :x0", {'x0': util.get_one_data(s0, 'users', 'id')})
s3 = util.do_sql(conn, "SELECT 'users' AS FROM 'users' WHERE 'users'.'id' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
s4 = util.do_sql(conn, "SELECT DISTINCT 'projects' AS FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.'id' = 'projects_users'.'project_id' WHERE 'projects_users'.'user_id' = :x0 AND 'projects'.'id' = :x1 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
if util.has_rows(s4):
    s5 = util.do_sql(conn, "SELECT 'stories' AS FROM 'stories' WHERE 'stories'.'project_id' = :x0 AND 'stories'.'id' = :x1 LIMIT 1", {'x0': inputs[2], 'x1': inputs[3]})
s6 = util.do_sql(conn, "SELECT 'notes' AS FROM 'notes' WHERE 'notes'.'story_id' = :x0", {'x0': inputs[3]})
s7 = util.do_sql(conn, "SELECT DISTINCT 'projects' AS FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.'id' = 'projects_users'.'project_id' WHERE 'projects_users'.'user_id' = :x0", {'x0': util.get_one_data(s0, 'users', 'id')})
s8 = util.do_sql(conn, "SELECT 'notes' AS FROM 'notes' WHERE 'notes'.'story_id' = :x0", {'x0': inputs[2]})
outputs.extend(util.get_data(s8, 'notes', 'id'))
outputs.extend(util.get_data(s6, 'notes', 'story_id'))
else:
    s5 = util.do_sql(conn, "SELECT DISTINCT 'projects' AS FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.'id' = 'projects_users'.'project_id' WHERE 'projects_users'.'user_id' = :x0", {'x0': util.get_one_data(s0, 'users', 'id')})
    s7 = util.do_sql(conn, "SELECT DISTINCT 'projects' AS FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.'id' = 'projects_users'.'project_id' WHERE 'projects_users'.'user_id' = :x0", {'x0': util.get_one_data(s0, 'users', 'id')})
    s8 = util.do_sql(conn, "SELECT 'notes' AS FROM 'notes' WHERE 'notes'.'story_id' = :x0", {'x0': inputs[2]})
    outputs.extend(util.get_data(s8, 'notes', 'id'))
    outputs.extend(util.get_data(s8, 'notes', 'story_id'))
else:
    pass
return util.add_warnings(outputs)
```

A.26 Fulcrum Task Manager Command get_projects_id_stories_id_notes_id

```python
def get_projects_id_stories_id_notes_id(conn, inputs):
util.clear_warnings()
outputs = []
s0 = util.do_sql(conn, "SELECT 'users' AS FROM 'users' WHERE 'users'.'email' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': inputs[0]})
if util.has_rows(s0):
    s1 = util.do_sql(conn, "SELECT 'users' AS FROM 'users' WHERE 'users'.'id' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
s2 = util.do_sql(conn, "SELECT DISTINCT 'projects' AS FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.'id' = 'projects_users'.'project_id' WHERE 'projects_users'.'user_id' = :x0", {'x0': util.get_one_data(s0, 'users', 'id')})
s3 = util.do_sql(conn, "SELECT 'users' AS FROM 'users' WHERE 'users'.'id' = :x0 ORDER BY 'users'.'id' ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
s4 = util.do_sql(conn, "SELECT DISTINCT 'projects' AS FROM 'projects' INNER JOIN 'projects_users' ON 'projects'.'id' = 'projects_users'.'project_id' WHERE 'projects_users'.'user_id' = :x0 AND 'projects'.'id' = :x1 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
if util.has_rows(s4):
```
s6 = util.do_sql(conn, "SELECT 'stories' . * FROM 'stories' WHERE 'stories' . 'project_id' = :x0 AND 'stories' . 'id' = :x1 LIMIT 1", {'x0': inputs[1], 'x1': inputs[2])
if util.has_rows(s6):
    s7 = util.do_sql(conn, "SELECT DISTINCT 'projects' . * FROM 'projects' WHERE 'projects' . 'user_id' = :x0", ('x0': util.get_one_data(s6, 'users', 'id')))

outputs.extend((util.get_data(s7, 'notes', 'id')))

if util.has_rows(s7):
    s8 = util.do_sql(conn, "SELECT 'notes' . * FROM 'notes' WHERE 'notes' . 'story_id' = :x0 AND 'notes' . 'id' = :x1 LIMIT 1", ('x0': inputs[2], 'x1': inputs[3]))

outputs.extend((util.get_data(s8, 'notes', 'id')))

if util.has_rows(s8):
    pass
else:
    s9 = util.do_sql(conn, "SELECT DISTINCT 'projects' . * FROM 'projects' INNER JOIN 'projects_users' ON 'projects' . 'id' = 'projects_users' . 'project_id' WHERE 'projects_users' . 'user_id' = :x0", ('x0': util.get_one_data(s8, 'users', 'id')))

else:
    s7 = util.do_sql(conn, "SELECT DISTINCT 'projects' . * FROM 'projects' INNER JOIN 'projects_users' ON 'projects' . 'id' = 'projects_users' . 'project_id' WHERE 'projects_users' . 'user_id' = :x0", ('x0': util.get_one_data(s7, 'users', 'id')))

else:
    pass

return util.add_warnings(outputs)

A.27 Fulcrum Tasker Task Command get_projects_id_users

def get_projects_id_users (conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users' . * FROM 'users' WHERE 'users' . 'name' = :x0 ORDER BY 'users' . 'id' ASC LIMIT 1", {'x0': inputs[0])

if util.has_rows(s0):
    s1 = util.do_sql(conn, "SELECT 'users' . * FROM 'users' WHERE 'users' . 'id' = :x0 ORDER BY 'users' . 'id' ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'))

    s2 = util.do_sql(conn, "SELECT DISTINCT 'projects' . * FROM 'projects' . 'id' = 'projects_users' . 'project_id' WHERE 'projects_users' . 'user_id' = :x0", ('x0': util.get_one_data(s0, 'users', 'id')))

    s3 = util.do_sql(conn, "SELECT 'users' . * FROM 'users' WHERE 'users' . 'id' = :x0 ORDER BY 'users' . 'id' ASC LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id'))

    s4 = util.do_sql(conn, "SELECT DISTINCT 'projects' . * FROM 'projects' INNER JOIN 'projects_users' ON 'projects' . 'id' = 'projects_users' . 'project_id' WHERE 'projects_users' . 'user_id' = :x0 AND 'projects' . 'id' = :x1 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')}, {'x1': inputs[1])

    outputs.extend((util.get_data(s4, 'projects', 'name')))

if util.has_rows(s4):
    s5 = util.do_sql(conn, "SELECT 'projects' . * FROM 'projects' INNER JOIN 'projects_users' ON 'projects' . 'id' = 'projects_users' . 'project_id' WHERE 'projects_users' . 'user_id' = :x0", ('x0': util.get_one_data(s0, 'users', 'id')))

    s6 = util.do_sql(conn, "SELECT DISTINCT 'users' . * FROM 'users' INNER JOIN 'projects_users' ON 'users' . 'id' = 'projects_users' . 'user_id' WHERE 'projects_users' . 'project_id' = :x0", ('x0': inputs[1])

    outputs.extend((util.get_data(s6, 'users', 'id')))

    outputs.extend((util.get_data(s6, 'users', 'email')))

    outputs.extend((util.get_data(s6, 'users', 'name')))

    outputs.extend((util.get_data(s6, 'users', 'initials')))

    s7 = util.do_sql(conn, "SELECT DISTINCT 'projects' . * FROM 'projects' INNER JOIN 'projects_users' . 'project_id' WHERE 'projects_users' . 'user_id' = :x0", ('x0': util.get_one_data(s0, 'users', 'id')))

    outputs.extend((util.get_data(s7, 'projects', 'id')))

    outputs.extend((util.get_data(s7, 'projects', 'name'))

else:
    s5 = util.do_sql(conn, "SELECT DISTINCT 'projects'.* FROM 'projects' INNER JOIN
    'projects_users' ON 'projects'..'id' = 'projects_users'..'project_id' WHERE
    'projects_users'..'user_id' = :x0", {'x0': util.get_one_data(s0, 'users', 'id')})
else:
    pass
return util.add_warnings(outputs)

A.28 Kandan Chat Room Command get_channels

def get_channels(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'..'* FROM 'users' WHERE 'users'..'username' = :x0
    LIMIT 1", {'x0': inputs[0]})
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT 'users'..'* FROM 'users' WHERE 'users'..'id' = :x0
        LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
        s2 = util.do_sql(conn, "SELECT 'channels'..'* FROM 'channels'" , ())
    if util.has_rows(s2):
        s5 = util.do_sql(conn, "SELECT 'activities'..'* FROM 'activities'
        WHERE 'channel_id' = :x0", {'x0': util.get_data(s2, 'channels', 'id')})
    if util.has_rows(s5):
        s10 = util.do_sql(conn, "SELECT 'users'..'* FROM 'users' WHERE 'users'..'id'
        IN :x0", {'x0': util.get_data(s5, 'activities', 'user_id')})
    if util.has_rows(s10):
        s11 = util.do_sql(conn, "SELECT 'users'..'* FROM 'users' WHERE 'users'..'id'
        = :x0 LIMIT 1", {'x0': util.get_one_data(s10, 'users', 'id')})
        s12 = util.do_sql(conn, "SELECT 'channels'..'* FROM 'channels'" , ())
        outputs.extend(util.get_data(s12, 'channels', 'id'))
        outputs.extend(util.get_data(s12, 'channels', 'name'))
        outputs.extend(util.get_data(s12, 'channels', 'user_id'))
        s12_all = s12
        for s12 in s12_all:
            s13 = util.do_sql(conn, "SELECT COUNT(*) FROM 'activities'
            WHERE 'activities'..'channel_id' = :x0", {'x0': util.get_one_data(s12, 'channels', 'id')})
            s14 = util.do_sql(conn, "SELECT 'activities'..'* FROM 'activities'
            WHERE 'activities'..'channel_id' = :x0 ORDER BY id DESC LIMIT 30 OFFSET 0
            ", {'x0': util.get_one_data(s12, 'channels', 'id')})
            outputs.extend(util.get_data(s14, 'activities', 'id'))
            outputs.extend(util.get_data(s14, 'activities', 'content'))
            outputs.extend(util.get_data(s14, 'activities', 'channel_id'))
            outputs.extend(util.get_data(s14, 'activities', 'user_id'))
        if util.has_rows(s14):
            s15 = util.do_sql(conn, "SELECT 'users'..'* FROM 'users' WHERE 'users'
           ..'id' IN :x0", {'x0': util.get_data(s14, 'activities', 'user_id')})
            outputs.extend(util.get_data(s15, 'users', 'id'))
            outputs.extend(util.get_data(s15, 'users', 'email'))
    elif util.has_rows(s14):
        s11 = util.do_sql(conn, "SELECT 'users'..'* FROM 'users' WHERE 'users'..'id'
        = :x0 LIMIT 1", {'x0': util.get_one_data(s11, 'users', 'id')})
        outputs.extend(util.get_data(s11, 'users', 'first_name'))
        outputs.extend(util.get_data(s11, 'users', 'last_name'))
    else:
        pass
    s12 = s12_all
    for s12 in s12_all:
        s1 = util.do_sql(conn, "SELECT 'users'..'* FROM 'users' WHERE 'users'..'id'
        = :x0 LIMIT 1", {'x0': util.get_one_data(s1, 'users', 'id')})
        s7 = util.do_sql(conn, "SELECT 'channels'..'* FROM 'channels'" , ())
        outputs.extend(util.get_data(s7, 'channels', 'id'))
        outputs.extend(util.get_data(s7, 'channels', 'name'))
        outputs.extend(util.get_data(s7, 'channels', 'user_id'))
        s7_all = s7
        for s7 in s7_all:
```python
s8 = util.do_sql(conn, "SELECT COUNT(*) FROM 'activities' WHERE 'activities'."channel_id" = :x0", ('x0': util.get_one_data(s7, 'channels', 'id')))
s9 = util.do_sql(conn, "SELECT 'activities' .* FROM 'activities' WHERE 'activities'."channel_id" = :x0 ORDER BY id DESC LIMIT 30 OFFSET 0" , ("x0": util.get_one_data(s7, 'channels', 'id')))
s7 = s7_all
pass
else:
s3 = util.do_sql(conn, "SELECT 'users' .* FROM 'users' WHERE 'users'.'id' = :x0
LIMIT 1", ('x0': util.get_one_data(s0, 'users', 'id')))
s4 = util.do_sql(conn, "SELECT 'channels' .* FROM 'channels'", ())
else:
pass
return util.add_warnings(outputs)

A.29 Kandan Chat Room Command get_channels_id_activities

def get_channels_id_activities(conn, inputs):
    util.clear_warnings()
    outputs = []
s0 = util.do_sql(conn, "SELECT 'users' .* FROM 'users' WHERE 'users'.'username' = :x0
LIMIT 1", ('x0': inputs[0]))
if util.has_rows(s0):
s1 = util.do_sql(conn, "SELECT 'users' .* FROM 'users' WHERE 'users'.'id' = :x0
LIMIT 1", ('x0': util.get_one_data(s0, 'users', 'id')))
s2 = util.do_sql(conn, "SELECT 'channels' .* FROM 'channels'", ())

if util.has_rows(s5):
s7 = util.do_sql(conn, "SELECT 'activities' .* FROM 'activities' WHERE 'activities'."channel_id" IN :x0", ('x0': util.get_data(s2, 'channels', 'id')))

if util.has_rows(s5):
s0 = util.do_sql(conn, "SELECT 'users' .* FROM 'users' WHERE 'users'.'id' = :x0
IN :x0", ('x0': util.get_data(s5, 'activities', 'user_id')))
s1 = util.do_sql(conn, "SELECT 'users' .* FROM 'users' WHERE 'users'.'id' = :x0
LIMIT 1", ('x0': util.get_one_data(s0, 'users', 'id')))
s2 = util.do_sql(conn, "SELECT 'channels' .* FROM 'channels' WHERE 'channels'.'id' = :x0
LIMIT 1", ('x0': inputs[1]))
if util.has_rows(s12):
s3 = util.do_sql(conn, "SELECT 'activities' .* FROM 'activities' WHERE 'activities'."channel_id" = :x0 ORDER BY id LIMIT 1", ('x0': inputs[1]))
outputs.extend(util.get_data(s13, 'activities', 'id'))
outputs.extend(util.get_data(s13, 'activities', 'content'))
outputs.extend(util.get_data(s13, 'activities', 'channel_id'))
s4 = util.do_sql(conn, "SELECT 'activities' .* FROM 'activities' WHERE 'activities'."channel_id" = :x0 ORDER BY id DESC LIMIT 30", ('x0': inputs[1]))
outputs.extend(util.get_data(s14, 'activities', 'id'))
outputs.extend(util.get_data(s14, 'activities', 'content'))
outputs.extend(util.get_data(s14, 'activities', 'channel_id'))
s5 = util.do_sql(conn, "SELECT 'users' .* FROM 'users' WHERE 'users'.'id' = :x0
IN :x0", ('x0': util.get_one_data(s14, 'activities', 'user_id')))
outputs.extend(util.get_data(s15, 'users', 'id'))
outputs.extend(util.get_data(s15, 'users', 'email'))
outputs.extend(util.get_data(s15, 'users', 'first_name'))
outputs.extend(util.get_data(s15, 'users', 'last_name'))
outputs.extend(util.get_data(s15, 'users', 'username'))
else:
pass
else:
pass
else:
s6 = util.do_sql(conn, "SELECT 'users' .* FROM 'users' WHERE 'users'.'id' = :x0
LIMIT 1", ('x0': util.get_one_data(s0, 'users', 'id')))
```

---

The code snippet defines a function `get_channels_id_activities` that retrieves data from a database for a Kandan Chat Room command. It uses SQL queries to fetch user and activity data, constructs output from these data, and handles cases where data is not available.
A.30  Kandan Chat Room Command get_channels_id_activities_id

```python
def get_channels_id_activities_id(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'username' = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
        s2 = util.do_sql(conn, "SELECT 'channels'.* FROM 'channels' WHERE 'channels'.'channel_id' IN :x0", {'x0': util.get_data(s2, 'channels', 'id')})
        if util.has_rows(s2):
            s5 = util.do_sql(conn, "SELECT 'activities'.* FROM 'activities' WHERE 'activities'.'channel_id' IN :x0", {'x0': util.get_data(s5, 'channels', 'id')})
            if util.has_rows(s5):
                s8 = util.do_sql(conn, "SELECT 'users'.'id' FROM 'users' WHERE 'users'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s8, 'users', 'id')})
                s9 = util.do_sql(conn, "SELECT * FROM 'activities' WHERE 'activities'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s9, 'activities', 'id')})
                s10 = util.do_sql(conn, "SELECT * FROM 'channels' WHERE 'channels'.'id' = :x0 LIMIT 1", {'x0': inputs[2]})
                outputs.extend([util.get_data(s10, 'channels', 'content')])
                else:
                    s6 = util.do_sql(conn, "SELECT 'users'.'id' FROM 'users' WHERE 'users'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s6, 'users', 'id')})
                    s7 = util.do_sql(conn, "SELECT * FROM 'activities' WHERE 'activities'.'id' = :x0 LIMIT 1", {'x0': inputs[2]})
                    outputs.extend([util.get_data(s7, 'activities', 'content')])
                    else:
                        s3 = util.do_sql(conn, "SELECT 'users'.'id' FROM 'users' WHERE 'users'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s3, 'users', 'id')})
                        s4 = util.do_sql(conn, "SELECT * FROM 'activities' WHERE 'activities'.'id' = :x0 LIMIT 1", {'x0': inputs[2]})
                        outputs.extend([util.get_data(s4, 'activities', 'content')])
            else:
                pass
        else:
            pass
    return util.add_warnings(outputs)
```

A.31  Kandan Chat Room Command get_me

```python
def get_me(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.'id' FROM 'users' WHERE 'users'.'username' = :x0 LIMIT 1", {'x0': inputs[0]})
    outputs.extend([util.get_data(s0, 'users', 'id')])
    outputs.extend([util.get_data(s0, 'users', 'email')])
```
outputs.extend(util.get_data(s0, 'users', 'first_name'))
outputs.extend(util.get_data(s0, 'users', 'last_name'))
outputs.extend(util.get_data(s0, 'users', 'username'))
if util.has_rows(s0):
    s1 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id = :x0 LIMIT 1", {':x0': util.get_one_data(s0, 'users', 'id')})
outputs.extend(util.get_data(s1, 'users', 'id'))
outputs.extend(util.get_data(s1, 'users', 'email'))
outputs.extend(util.get_data(s1, 'users', 'first_name'))
outputs.extend(util.get_data(s1, 'users', 'last_name'))
outputs.extend(util.get_data(s1, 'users', 'username'))
s2 = util.do_sql(conn, "SELECT 'channels'.* FROM 'channels'", {})
if util.has_rows(s2):
    s4 = util.do_sql(conn, "SELECT 'activities'.* FROM 'activities' WHERE 'activities'.channel_id IN :x0", {':x0': util.get_one_data(s4, 'channels', 'id')})
if util.has_rows(s4):
    s6 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id IN :x0 LIMIT 1", {':x0': util.get_one_data(s6, 'users', 'id')})
s7 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id = :x0 LIMIT 1", {':x0': util.get_one_data(s7, 'users', 'id')})
outputs.extend(util.get_data(s7, 'users', 'id'))
outputs.extend(util.get_data(s7, 'users', 'email'))
outputs.extend(util.get_data(s7, 'users', 'first_name'))
outputs.extend(util.get_data(s7, 'users', 'last_name'))
outputs.extend(util.get_data(s7, 'users', 'username'))
else:
    s5 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id = :x0 LIMIT 1", {':x0': util.get_one_data(s5, 'users', 'id')})
outputs.extend(util.get_data(s5, 'users', 'id'))
outputs.extend(util.get_data(s5, 'users', 'email'))
outputs.extend(util.get_data(s5, 'users', 'first_name'))
outputs.extend(util.get_data(s5, 'users', 'last_name'))
outputs.extend(util.get_data(s5, 'users', 'username'))
else:
    s3 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id = :x0 LIMIT 1", {':x0': util.get_one_data(s3, 'users', 'id')})
outputs.extend(util.get_data(s3, 'users', 'id'))
outputs.extend(util.get_data(s3, 'users', 'email'))
outputs.extend(util.get_data(s3, 'users', 'first_name'))
outputs.extend(util.get_data(s3, 'users', 'last_name'))
outputs.extend(util.get_data(s3, 'users', 'username'))
else:
    pass
return util.add_warnings(outputs)

def get_users(conn, inputs):
    util.clear_warnings()
    outputs = []
s0 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.username = :x0 LIMIT 1", {':x0': inputs[8]})
outputs.extend(util.get_data(s0, 'users', 'id'))
outputs.extend(util.get_data(s0, 'users', 'email'))
outputs.extend(util.get_data(s0, 'users', 'first_name'))
outputs.extend(util.get_data(s0, 'users', 'last_name'))
outputs.extend(util.get_data(s0, 'users', 'username'))
if util.has_rows(s0):
    s1 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.id = :x0 LIMIT 1", {':x0': util.get_one_data(s1, 'users', 'id')})
outputs.extend(util.get_data(s1, 'users', 'id'))
outputs.extend(util.get_data(s1, 'users', 'email'))
outputs.extend(util.get_data(s1, 'users', 'first_name'))
outputs.extend(util.get_data(s1, 'users', 'last_name'))
outputs.extend(util.get_data(s1, 'users', 'username'))
s2 = util.do_sql(conn, "SELECT 'channels'.* FROM 'channels'", {})
if util.has_rows(s2):
```python
s5 = util.do_sql(conn, "SELECT 'activities'.* FROM 'activities' WHERE 'activities'.'channel_id' IN :x0, ('x0': util.get_data(s2, 'channels', 'id'))
if util.has_rows(s5):
    s8 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'id' IN :x0, ('x0': util.get_data(s8, 'users', 'id'))
    outputs.extend(util.get_data(s8, 'users', 'id'))
    outputs.extend(util.get_data(s8, 'users', 'email'))
    outputs.extend(util.get_data(s8, 'users', 'first_name'))
    outputs.extend(util.get_data(s8, 'users', 'last_name'))
    s9 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'id' = :x0 LIMIT 1", ('x0': util.get_one_data(s8, 'users', 'id')))
    outputs.extend(util.get_data(s9, 'users', 'id'))
    outputs.extend(util.get_data(s9, 'users', 'email'))
    outputs.extend(util.get_data(s9, 'users', 'first_name'))
    outputs.extend(util.get_data(s9, 'users', 'last_name'))
    s10 = util.do_sql(conn, "SELECT 'users' FROM 'users'", ())
    outputs.extend(util.get_data(s10, 'users', 'id'))
    outputs.extend(util.get_data(s10, 'users', 'username'))
else:
    s6 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'id' = :x0 LIMIT 1", ('x0': util.get_one_data(s6, 'users', 'id')))
    outputs.extend(util.get_data(s6, 'users', 'id'))
    outputs.extend(util.get_data(s6, 'users', 'email'))
    outputs.extend(util.get_data(s6, 'users', 'first_name'))
    outputs.extend(util.get_data(s6, 'users', 'last_name'))
    s7 = util.do_sql(conn, "SELECT 'users'.* FROM 'users'", ()
    outputs.extend(util.get_data(s7, 'users', 'id'))
    outputs.extend(util.get_data(s7, 'users', 'email'))
    outputs.extend(util.get_data(s7, 'users', 'first_name'))
    outputs.extend(util.get_data(s7, 'users', 'last_name'))
    outputs.extend(util.get_data(s7, 'users', 'username'))
else:
    s3 = util.do_sql(conn, "SELECT 'users'.* FROM 'users' WHERE 'users'.'id' = :x0 LIMIT 1", ('x0': util.get_one_data(s3, 'users', 'id')))
    outputs.extend(util.get_data(s3, 'users', 'id'))
    outputs.extend(util.get_data(s3, 'users', 'email'))
    outputs.extend(util.get_data(s3, 'users', 'first_name'))
    outputs.extend(util.get_data(s3, 'users', 'last_name'))
    s4 = util.do_sql(conn, "SELECT 'users' FROM 'users'", ()
    outputs.extend(util.get_data(s4, 'users', 'id'))
    outputs.extend(util.get_data(s4, 'users', 'email'))
    outputs.extend(util.get_data(s4, 'users', 'first_name'))
    outputs.extend(util.get_data(s4, 'users', 'last_name'))
else:
    pass
return util.add_warnings(outputs)
```

A.33 Kandan Chat Room Command get_users_id

```python
def get_users_id (conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT 'users'.'* FROM 'users' WHERE 'users'.'username' = :x0 LIMIT 1", ('x0': inputs[0]))
    outputs.extend(util.get_data(s0, 'users', 'id'))
    outputs.extend(util.get_data(s0, 'users', 'email'))
    outputs.extend(util.get_data(s0, 'users', 'first_name'))
    outputs.extend(util.get_data(s0, 'users', 'last_name'))
    if util.has_rows(s0):
```
s7 = util.do_sql(conn, "SELECT 'users' FROM 'users' WHERE 'users'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
outputs.extend(util.get_data(s1, 'users', 'id'))
outputs.extend(util.get_data(s1, 'users', 'first_name'))
outputs.extend(util.get_data(s1, 'users', 'last_name'))
outputs.extend(util.get_data(s1, 'users', 'username'))
if util.has_rows(s2):
s4 = util.do_sql(conn, "SELECT 'activities' FROM 'activities' WHERE 'activities'.'channel_id' IN :x0", {'x0': util.get_data(s2, 'channels', 'id')})
if util.has_rows(s4):
s6 = util.do_sql(conn, "SELECT 'users' FROM 'users' WHERE 'users'.'id' IN :x0", {'x0': util.get_data(s4, 'activities', 'user_id')})
s7 = util.do_sql(conn, "SELECT 'users' FROM 'users' WHERE 'users'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
outputs.extend(util.get_data(s7, 'users', 'id'))
outputs.extend(util.get_data(s7, 'users', 'email'))
outputs.extend(util.get_data(s7, 'users', 'first_name'))
outputs.extend(util.get_data(s7, 'users', 'last_name'))
outputs.extend(util.get_data(s7, 'users', 'username'))
else:
s5 = util.do_sql(conn, "SELECT 'users' FROM 'users' WHERE 'users'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
outputs.extend(util.get_data(s5, 'users', 'id'))
outputs.extend(util.get_data(s5, 'users', 'email'))
outputs.extend(util.get_data(s5, 'users', 'first_name'))
outputs.extend(util.get_data(s5, 'users', 'last_name'))
outputs.extend(util.get_data(s5, 'users', 'username'))
else:
s3 = util.do_sql(conn, "SELECT 'users' FROM 'users' WHERE 'users'.'id' = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
outputs.extend(util.get_data(s3, 'users', 'id'))
outputs.extend(util.get_data(s3, 'users', 'email'))
outputs.extend(util.get_data(s3, 'users', 'first_name'))
outputs.extend(util.get_data(s3, 'users', 'last_name'))
outputs.extend(util.get_data(s3, 'users', 'username'))
else:
    pass
return util.add_warnings(outputs)

A.34 Enki Blogging Application Command get_home

For this command we use the original version of Enki, but disregard all queries on the taggings table.

def get_home(conn, inputs):
    util.clear_warnings()
    outputs = []
s0 = util.do_sql(conn, "SELECT COUNT(count_column) FROM (SELECT 1 AS count_column FROM "posts WHERE (1=1) LIMIT 15) subquery_for_count", { })
if util.has_rows(s0):
s4 = util.do_sql(conn, "SELECT 'posts' FROM 'posts' WHERE (1=1) ORDER BY published_at DESC LIMIT 15", { })
outputs.extend(util.get_data(s4, 'posts', 'id'))
outputs.extend(util.get_data(s4, 'posts', 'title'))
outputs.extend(util.get_data(s4, 'posts', 'slug'))
outputs.extend(util.get_data(s4, 'posts', 'body_html'))
s4_all = s4
for s4 in s4_all:
s5 = util.do_sql(conn, "SELECT COUNT(*) FROM 'comments' WHERE 'comments'."post_id" = :x0", {'x0': util.get_one_data(s4, 'posts', 'id')})
s4 = s4_all
s6 = util.do_sql(conn, "SELECT 'pages' FROM 'pages' ORDER BY title", { })
outputs.extend(util.get_data(s6, 'pages', 'title'))
outputs.extend(util.get_data(s6, 'pages', 'slug'))
s7 = util.do_sql(conn, "SELECT 'posts' FROM 'posts' WHERE ('posts'.'published_at' IS NOT NULL)", { })
s8 = util.do_sql(conn, "SELECT \"tags\".* FROM \"tags\" WHERE \"tags\".\"name\" IS NULL", 
    {}}
else:
    s1 = util.do_sql(conn, "SELECT \"pages\".* FROM \"pages\" ORDER BY title", 
    {}}
outputs.extend(util.get_data(s1, \"pages\", \"title\"))
outputs.extend(util.get_data(s1, \"pages\", \"slug\"))
s2 = util.do_sql(conn, "SELECT \"posts\".* FROM \"posts\" WHERE \"(\"posts\".\"published_at\" 
    IS NOT NULL)\", 
    {}}
s3 = util.do_sql(conn, "SELECT \"tags\".* FROM \"tags\" WHERE 1=0", 
    {}}
return util.add_warnings(outputs)

A.35 Enki Blogging Application Command get_archives

def get_archives (conn, inputs):
    util.clear_warnings()
    outputs = []
s0 = util.do_sql(conn, "SELECT \"posts\".* FROM \"posts\" WHERE (1=1) ORDER BY published_at 
    DESC", 
    {}}
outputs.extend(util.get_data(s0, \"posts\", \"title\"))
outputs.extend(util.get_data(s0, \"posts\", \"slug\"))
s0_all = s0
for s0 in s0_all:
    s1 = util.do_sql(conn, "SELECT 1 AS one FROM \"tags\" INNER JOIN \"taggings\" ON \"tags 
    \".\"id\" = \"taggings\".\"taggable_id\" = :x0 AND \"taggings 
    \".\"taggable_type\" = \"Post\" AND \"taggings\".\"context\" = \"tags\" LIMIT 1", 
    \'{\"x0\": 
    util.get_one_data(s0, \"posts\", \"id\")}
    if util.has_rows(s1):
        s2 = util.do_sql(conn, "SELECT \"tags\".* FROM \"tags\" INNER JOIN \"taggings\" ON \"tags 
            \".\"id\" = \"taggings\".\"taggable_id\" = :x0 AND \"taggings 
            \".\"taggable_type\" = \"Post\" AND \"taggings\".\"context\" = \"tags\"", 
    \'{\"x0 
    : util.get_one_data(s0, \"posts\", \"id\")}
    outputs.extend(util.get_data(s2, \"tags\", \"name\"))
else:
    pass
s0 = s0_all
s3 = util.do_sql(conn, "SELECT \"pages\".* FROM \"pages\" ORDER BY title", 
    {}}
outputs.extend(util.get_data(s3, \"pages\", \"title\"))
outputs.extend(util.get_data(s3, \"pages\", \"slug\"))
s4 = util.do_sql(conn, "SELECT \"posts\".* FROM \"posts\" WHERE \"(\"posts\".\"published_at\" 
    IS NOT NULL)\", 
    {}}
s5 = util.do_sql(conn, "SELECT \"tags\".* FROM \"tags\" WHERE 1=0", 
    {}}
return util.add_warnings(outputs)

A.36 Enki Blogging Application Command get_admin_comments_id

def get_admin_comments_id (conn, inputs):
    util.clear_warnings()
    outputs = []
s0 = util.do_sql(conn, "SELECT \"comments\".* FROM \"comments\" WHERE \"comments\".\"id\" = :x0 
    LIMIT 1", 
    \'{\"x0\": inputs[0]}
    outputs.extend(util.get_data(s0, \"comments\", \"id\"))
outputs.extend(util.get_data(s0, \"comments\", \"author\"))
outputs.extend(util.get_data(s0, \"comments\", \"author_url\"))
outputs.extend(util.get_data(s0, \"comments\", \"author_email\"))
return util.add_warnings(outputs)

A.37 Enki Blogging Application Command get_admin_pages

def get_admin_pages (conn, inputs):
    util.clear_warnings()
    outputs = []
s0 = util.do_sql(conn, "SELECT COUNT(*) FROM \"pages\"", 
    {}}
if util.has_rows(s0):
    s1 = util.do_sql(conn, "SELECT \"pages\".* FROM \"pages\" ORDER BY created_at DESC 
    LIMIT 30 OFFSET 0", 
    {}}
A.38 Enki Blogging Application Command get_admin_pages_id

```python
def get_admin_pages_id(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT `pages`.* FROM `pages` WHERE `pages`.
    `id` = $x0 LIMIT 1", ($x0: inputs[0]))
    outputs.extend(util.get_data(s0, 'pages', 'id'))
    outputs.extend(util.get_data(s0, 'pages', 'title'))
    outputs.extend(util.get_data(s0, 'pages', 'slug'))
    outputs.extend(util.get_data(s0, 'pages', 'body'))
    return util.add_warnings(outputs)
```

A.39 Enki Blogging Application Command get_admin_posts

```python
def get_admin_posts(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT COUNT(*) FROM `posts`", ())
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT * FROM `posts` ORDER BY coalesce(`published_at`, `updated_at`) DESC LIMIT 30 OFFSET 0", ())
        outputs.extend(util.get_data(s1, 'posts', 'id'))
        outputs.extend(util.get_data(s1, 'posts', 'title'))
        outputs.extend(util.get_data(s1, 'posts', 'body'))
        s1_all = s1
        for s1 in s1_all:
            s2 = util.do_sql(conn, "SELECT * FROM `comments` WHERE `comments`.
            `post_id` = $x0 ORDER BY created_at DESC LIMIT 1", ($x0: util.get_one_data(s1, 'posts', 'id'))) 
            s1 = s1_all
        pass
    else:
        pass
    return util.add_warnings(outputs)
```

A.40 Enki Blogging Application Command get_admin

For this command we use a trimmed version of Enki, where we removed a `<ul>` element that displays recent comments in the file app/views/admin/dashboard/show.html.erb.

```python
def get_admin(conn, inputs):
    util.clear_warnings()
    s0 = util.do_sql(conn, "SELECT * , max(`comments`.created_at) FROM `posts` INNER JOIN `comments` ON `comments`.post_id = `posts`.id GROUP BY `posts`.id ORDER BY `comments`.created_at DESC LIMIT 5", ())
    s0_all = s0
    for s0 in s0_all:
        s1 = util.do_sql(conn, "SELECT * FROM `comments` WHERE `comments`.
        `post_id` = $x0 ORDER BY created_at DESC LIMIT 1", ($x0: util.get_one_data(s0, 'posts', 'id'))) 
        s0 = s0_all
        s2 = util.do_sql(conn, "SELECT COUNT(*) FROM `posts`", ())
        s3 = util.do_sql(conn, "SELECT COUNT(*) FROM `comments`", ())
        s4 = util.do_sql(conn, "SELECT COUNT(*) FROM `tags`", ())
```
```
s5 = util.do_sql(conn, "SELECT `posts`.* FROM `posts` WHERE (1=1) ORDER BY published_at DESC LIMIT 8", ({}))
outputs.extend(util.get_data(s5, 'posts', 'id'))
outputs.extend(util.get_data(s5, 'posts', 'slug'))
s5_all = s5
for s5 in s5_all:
    s6 = util.do_sql(conn, "SELECT COUNT(*) FROM `comments` WHERE `comments`.post_id = :x0", ('x0': util.get_one_data(s5, 'posts', 'id')))
return util.add_warnings(outputs)
```

**A.41 Blog Command get_articles**

```
def get_articles (conn, inputs):
    util.clear_warnings()
outputs = []
s0 = util.do_sql(conn, "SELECT `articles`.* FROM `articles` WHERE `articles`.*id = :x0 LIMIT 1", ('x0': inputs[0]))
outputs.extend(util.get_data(s0, 'articles', 'id'))
outputs.extend(util.get_data(s0, 'articles', 'title'))
outputs.extend(util.get_data(s0, 'articles', 'text'))
if util.has_rows(s0):
    s1 = util.do_sql(conn, "SELECT `comments`.* FROM `comments` WHERE `comments`.article_id = :x0", ('x0': inputs[0]))
    outputs.extend(util.get_data(s1, 'comments', 'commenter'))
    outputs.extend(util.get_data(s1, 'comments', 'body'))
else:
    pass
return util.add_warnings(outputs)
```

**A.42 Blog Command get_article_id**

```
def get_article_id (conn, inputs):
    util.clear_warnings()
outputs = []
s0 = util.do_sql(conn, "SELECT `articles`.* FROM `articles` WHERE `articles`.*id = :x0 LIMIT 1", ('x0': inputs[0]))
outputs.extend(util.get_data(s0, 'articles', 'id'))
outputs.extend(util.get_data(s0, 'articles', 'title'))
outputs.extend(util.get_data(s0, 'articles', 'text'))
if util.has_rows(s0):
    s1 = util.do_sql(conn, "SELECT `comments`.* FROM `comments` WHERE `comments`.article_id = :x0", ('x0': inputs[0]))
    outputs.extend(util.get_data(s1, 'comments', 'commenter'))
    outputs.extend(util.get_data(s1, 'comments', 'body'))
else:
    pass
return util.add_warnings(outputs)
```

**A.43 Student Registration Command liststudentcourses**

```
def liststudentcourses (conn, inputs):
    util.clear_warnings()
outputs = []
s0 = util.do_sql(conn, "SELECT * FROM student WHERE id = :x0", ('x0': inputs[0]))
if util.has_rows(s0):
    s1 = util.do_sql(conn, "SELECT * FROM student WHERE id=:x0 AND password=:x1", ('x0': inputs[0], 'x1': inputs[1]))
    if util.has_rows(s1):
        s2 = util.do_sql(conn, "SELECT * FROM course c JOIN registration r on r.course_id = c.id WHERE r.student_id = :x0", ('x0': inputs[0]))
        outputs.extend(util.get_data(s2, 'course', 'id'))
        outputs.extend(util.get_data(s2, 'course', 'teacher_id'))
        outputs.extend(util.get_data(s2, 'registration', 'course_id'))
        s2_all = s2
        for s2 in s2_all:
            s3 = util.do_sql(conn, "Select firstname, lastname from teacher where id = :x0", ('x0': util.get_one_data(s2, 'course', 'teacher_id')))
            s4 = util.do_sql(conn, "SELECT count(*) FROM registration WHERE course_id = :x0", ('x0': util.get_one_data(s2, 'course', 'id')))
            s2 = s2_all
```
A.44 Synthetic Program repeat_2

```python
def repeat_2(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT * FROM t1", ()),
    s1 = util.do_sql(conn, "SELECT * FROM t2", ()),
    s2 = util.do_sql(conn, "SELECT * FROM t2", ()),
    return util.add_warnings(outputs)
```

A.45 Synthetic Program repeat_3

```python
def repeat_3(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT * FROM t1", ()),
    s1 = util.do_sql(conn, "SELECT * FROM t2", ()),
    s2 = util.do_sql(conn, "SELECT * FROM t2", ()),
    s3 = util.do_sql(conn, "SELECT * FROM t2", ()),
    return util.add_warnings(outputs)
```

A.46 Synthetic Program repeat_4

```python
def repeat_4(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT * FROM t1", ()),
    s1 = util.do_sql(conn, "SELECT * FROM t2", ()),
    s2 = util.do_sql(conn, "SELECT * FROM t2", ()),
    s3 = util.do_sql(conn, "SELECT * FROM t2", ()),
    s4 = util.do_sql(conn, "SELECT * FROM t2", ()),
    return util.add_warnings(outputs)
```

A.47 Synthetic Program repeat_5

```python
def repeat_5(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT * FROM t1", ()),
    s1 = util.do_sql(conn, "SELECT * FROM t2", ()),
    s2 = util.do_sql(conn, "SELECT * FROM t2", ()),
    s3 = util.do_sql(conn, "SELECT * FROM t2", ()),
    s4 = util.do_sql(conn, "SELECT * FROM t2", ()),
    s5 = util.do_sql(conn, "SELECT * FROM t2", ()),
    return util.add_warnings(outputs)
```

A.48 Synthetic Program nest

```python
def nest_2(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT * FROM t0", ()),
    s0_all = s0
    for s0 in s0_all:
        s1 = util.do_sql(conn, "SELECT * FROM t1", ()),
        s1_all = s1
```
for $s_1$ in $s_1\_all$:
    $s_2 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t2"}, ())
    s_1 = s_1\_all
    pass
    s_0 = s_0\_all
    pass
return \text{util}.\text{add}\_warnings(\text{outputs})

A.49 Synthetic Program after_2

def after_2 (\text{conn}, \text{inputs}):
    \text{util}.\text{clear}\_warnings()
    outputs = []
    $s_0 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t1"}, ())
    s_0\_all = s_0
    for $s_0$ in $s_0\_all$:
        $s_1 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t0"}, ())
        s_0 = s_0\_all
        $s_2 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t2"}, ())
        s_2\_all = s_2
        for $s_2$ in $s_2\_all$:
            $s_3 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t0"}, ())
            s_2 = s_2\_all
            s_4 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t3"}, ())
            s_4\_all = s_4
            for $s_4$ in $s_4\_all$:
                $s_5 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t0"}, ())
                s_4 = s_4\_all
                pass
return \text{util}.\text{add}\_warnings(\text{outputs})

A.50 Synthetic Program after_3

def after_3 (\text{conn}, \text{inputs}):
    \text{util}.\text{clear}\_warnings()
    outputs = []
    $s_0 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t1"}, ())
    s_0\_all = s_0
    for $s_0$ in $s_0\_all$:
        $s_1 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t0"}, ())
        s_0 = s_0\_all
        $s_2 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t2"}, ())
        s_2\_all = s_2
        for $s_2$ in $s_2\_all$:
            $s_3 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t0"}, ())
            s_2 = s_2\_all
            s_4 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t3"}, ())
            s_4\_all = s_4
            for $s_4$ in $s_4\_all$:
                $s_5 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t0"}, ())
                s_4 = s_4\_all
                pass
return \text{util}.\text{add}\_warnings(\text{outputs})

A.51 Synthetic Program after_4

def after_4 (\text{conn}, \text{inputs}):
    \text{util}.\text{clear}\_warnings()
    outputs = []
    $s_0 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t1"}, ())
    s_0\_all = s_0
    for $s_0$ in $s_0\_all$:
        $s_1 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t0"}, ())
        s_0 = s_0\_all
        $s_2 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t2"}, ())
        s_2\_all = s_2
        for $s_2$ in $s_2\_all$:
            $s_3 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t0"}, ())
            s_2 = s_2\_all
            s_4 = \text{util}.\text{do}\_sql(\text{conn}, \text{"SELECT * FROM t3"}, ())
            s_4\_all = s_4
for s4 in s4_all:
    s5 = util.do_sql(conn, "SELECT * FROM t0", ())
    s4 = s4_all
    s6 = util.do_sql(conn, "SELECT * FROM t4", ())
    s6_all = s6
    for s6 in s6_all:
        s7 = util.do_sql(conn, "SELECT * FROM t0", ())
        s6 = s6_all
        pass
    return util.add_warnings(outputs)

A.52 Synthetic Program after_5

def after_5(conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT * FROM t1", ())
    s0_all = s0
    for s0 in s0_all:
        s1 = util.do_sql(conn, "SELECT * FROM t0", ())
        s0 = s0_all
        s2 = util.do_sql(conn, "SELECT * FROM t2", ())
        s2_all = s2
        for s2 in s2_all:
            s3 = util.do_sql(conn, "SELECT * FROM t0", ())
            s2 = s2_all
            s4 = util.do_sql(conn, "SELECT * FROM t3", ())
            s4_all = s4
        for s4 in s4_all:
            s5 = util.do_sql(conn, "SELECT * FROM t0", ())
            s4 = s4_all
            s6 = util.do_sql(conn, "SELECT * FROM t4", ())
            s6_all = s6
        for s6 in s6_all:
            s7 = util.do_sql(conn, "SELECT * FROM t0", ())
            s6 = s6_all
            s8 = util.do_sql(conn, "SELECT * FROM t5", ())
            s8_all = s8
        for s8 in s8_all:
            s9 = util.do_sql(conn, "SELECT * FROM t0", ())
            s8 = s8_all
        pass
    return util.add_warnings(outputs)

A.53 Synthetic Program example (Section 2)

def example_3 (conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT * FROM tasks WHERE id = :x0", {'x0': inputs[0]})
    outputs.extend(util.get_data(s0, 'tasks', 'title'))
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT * FROM comments WHERE task_id = :x0", {'x0': inputs[0]})
        outputs.extend(util.get_data(s1, 'comments', 'content'))
        s1_all = s1
        for s1 in s1_all:
            s2 = util.do_sql(conn, "SELECT * FROM users WHERE id = :x0", {'x0': util.get_one_data(s1, 'comments', 'commenter_id')})
            outputs.extend(util.get_data(s2, 'users', 'name'))
        s3 = util.do_sql(conn, "SELECT * FROM tasks WHERE creator_id = :x0", {'x0': util.get_one_data(s1, 'comments', 'commenter_id')})
        outputs.extend(util.get_data(s3, 'tasks', 'title'))
        s1 = s1_all
        s4 = util.do_sql(conn, "SELECT * FROM users WHERE id = :x0", {'x0': util.get_one_data(s0, 'tasks', 'assignee_id')})
        outputs.extend(util.get_data(s4, 'users', 'name'))
s5 = util.do_sql(conn, "SELECT * FROM tasks WHERE creator_id = :x0", {'x0': util.get_one_data(s0, 'tasks', 'assignee_id'))
outputs.extend(util.get_data(s5, 'tasks', 'title'))
else:
    pass
return util.add_warnings(outputs)